Lecture Notes in Computer Science 5482

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

Carlos Cotta Peter Cowling (Eds.)

Evolutionary Computation in Combinatorial Optimization

9th European Conference, EvoCOP 2009
Tübingen, Germany, April 15-17, 2009
Proceedings

Volume Editors

Carlos Cotta
Universidad de Málaga, Dept. de Lenguajes y Ciencias de la Computación
ETSI Informática, Campus Teatinos, 29071 Málaga, Spain
E-mail: ccottap@lcc.uma.es

Peter Cowling
University of Bradford, Department of Computing
Bradford BD7 1DP, UK
E-mail: p.i.cowling@bradford.ac.uk

Cover illustration: "You Pretty Little Flocker" by Alice Eldridge (www.ecila.org and www.infotech.monash.edu.au/research/groups/cema/flocker/flocker.html)

Library of Congress Control Number: Applied for

CR Subject Classification (1998): F.1, F.2, G.1.6, G.2.1, J.3

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

ISSN 0302-9743
ISBN-10 3-642-01008-3 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-01008-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12651580 06/3180 5 4 3 2 1 0

Preface

Metaheuristics have been shown to be effective for difficult combinatorial optimization problems appearing in a wide variety of industrial, economic, and scientific domains. Prominent examples of metaheuristics are evolutionary algorithms, tabu search, simulated annealing, scatter search, memetic algorithms, variable neighborhood search, iterated local search, greedy randomized adaptive search procedures, ant colony optimization, and estimation of distribution algorithms. Problems solved successfully include scheduling, timetabling, network design, transportation and distribution, vehicle routing, the travelling salesman problem, packing and cutting, satisfiability, and general mixed integer programming.

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

This volume contains the proceedings of EvoCOP 2009, the 9th European Conference on Evolutionary Computation in Combinatorial Optimization. It was held in Eberhard Karls Universität Tübingen, Germany, April 15-17, 2009, jointly with EuroGP 2009, the 12th European Conference on Genetic Programming, EvoBIO 2009, the 7th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, and EvoWorkshops 2009, which consisted of the following 12 individual workshops: 6th European Workshop on the Application of Nature-Inspired Techniques for Telecommunication Networks and Other Parallel and Distributed Systems; First European Workshop on Nature-Inspired Methods for Environmental Issues; Third European Workshop on Evolutionary Computation in Finance and Economics; First European Workshop on Bio-inspired Algorithms in Games; 5th European Workshop on Bio-Inspired Heuristics for Design Automation, 11th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing; Third European Workshop on Interactive Evolution and Humanized Computational Intelligence; 7th European Workshop on Evolutionary and Biologically Inspired Music, Sound, Art and Design; Second European Workshop on Bio-inspired Algorithms for Continuous Parameter Optimization; 4th European Graduate Student Workshop on Evolutionary Computation; 6th European Workshop on Evolutionary Algorithms in Stochastic and Dynamic Environments; and Third European Workshop on Evolutionary Computation in Transportation and Logistics. Since 2007, all these events have been grouped

under the collective name EvoStar, and constitute Europe's premier co-located meetings on evolutionary computation.

Accepted papers of previous EvoCOP editions were published by Springer in the series *Lecture Notes in Computer Science* (LNCS – Volumes 2037, 2279, 2611, 3004, 3448, 3906, 4446, 4972).

EvoCOP	Submitted	Accepted	Acceptance Rate
2001	31	23	74.2%
2002	32	18	56.3%
2003	39	19	48.7%
2004	86	23	26.7%
2005	66	24	36.4%
2006	77	24	31.2%
2007	81	21	25.9%
2008	69	24	34.8%
2009	53	21	39.6%

The rigorous, double-blind reviewing process of EvoCOP 2009 resulted in a strong selection among the submitted papers; the acceptance rate was 39.6%. Each paper was reviewed by at least three members of the international Program Committee. All accepted papers were presented orally at the conference and are included in this proceedings volume. We would like to acknowledge the members of our Program Committee, to whom we are very grateful for their thorough work. EvoCOP 2009 contributions consist of new algorithms together with important new insights into how well these algorithms can solve prominent test problems from the literature or real-world problems.

We would like to express our sincere gratitude to the two internationally renowned invited speakers, who gave the keynote talks at the conference: Stuart Hameroff, Professor Emeritus Departments of Anesthesiology and Psychology and Director, Center for Consciousness Studies, The University of Arizona, USA, and Peter Schuster, Institute of Theoretical Chemistry at Universität Wien and President of the Austrian Academy of Sciences.

The success of the conference resulted from the input of many people to whom we would like to express our appreciation. The local organizers, led by Marc Ebner, from Wilhelm Schickard Institute for Computer Science at Universität Tübingen, have done an extraordinary job for which we are very grateful. Also grateful thanks for local support from Andreas Zell, Chair of Computer Architecture at the Wilhelm Schickard Institute for Computer Science, Universität Tübingen, and Peter Weit, Vice Director of the Seminar for Rhetorics at the New Philology Department, Universität Tübingen. Thanks also to Tübingen Info Office (especially Marco Schubert) for local tourism and information, and to the German Research Foundation (DFG) for financial support for the EvoStar conference.

We thank Marc Schoenauer from INRIA in France for continued assistance in providing the MyReview conference management system. Thanks are also due to

Jennifer Willies and the Centre for Emergent Computing at Edinburgh Napier University, UK for administrative support and event coordination. Last, but not least, we would like to thank Jens Gottlieb, Jano van Hemert, and Günther Raidl for their hard work and dedication in past editions of EvoCOP, which contributed to making this conference one of the reference events in evolutionary computation and metaheuristics.

April 2009

Carlos Cotta
Peter Cowling

Organization

EvoCOP 2009 was organized jointly with EuroGP 2009, EvoBIO 2009, and EvoWorkshops 2009.

Organizing Committee

Chairs	Carlos Cotta, Universidad de Málaga, Spain
	Peter Cowling, University of Bradford, UK
Local Chair	Marc Ebner, Universität Tübingen, Germany
Publicity Chair	Ivanoe de Falco, ICAR, National Research Council of Italy, Italy

EvoCOP Steering Committee

Carlos Cotta	Universidad de Málaga, Spain
Peter Cowling	University of Bradford, UK
Jens Gottlieb	SAP AG, Germany
Jano van Hemert	University of Edinburgh, UK
Günther Raidl	Vienna University of Technology, Austria

Program Committee

Adnan Acan	Middle East Technical University, Ankara, Turkey
Hernán Aguirre	Shinshu University, Nagano, Japan
Enrique Alba	Universidad de Málaga, Spain
Mehmet Emin Aydin	University of Bedfordshire, UK
Ruibin Bai	University of Nottingham, UK
Thomas Bartz-Beielstein	Cologne University of Applied Sciences, Germany
Christian Bierwirth	University of Bremen, Germany
Maria Blesa	Universitat Politècnica de Catalunya, Spain
Christian Blum	Universitat Politècnica de Catalunya, Spain
Peter Brucker	University of Osnabrück, Germany
Rafael Caballero	University of Málaga, Spain
Pedro Castillo	Universidad de Granada, Spain
Konstantin Chakhlevitch	City University, UK
Carlos Coello Coello	National Polytechnic Institute, Mexico
Carlos Cotta	Universidad de Málaga, Spain
Peter Cowling	University of Bradford, UK

Bart Craenen	Napier University, UK
Keshav Dahal	University of Bradford, UK
Karl Doerner	Universität Wien, Austria
Marco Dorigo	Free University of Brussels, Belgium
Jeroen Eggermont	Leiden University Medical Center, The Netherlands
Anton V. Eremeev	Omsk Branch of Sobolev Institute of Mathematics, Russia
Richard F. Hartl	University of Vienna, Austria
Antonio J. Fernández	Universidad de Málaga, Spain
Francisco Fernández de Vega	University of Extremadura, Spain
Bernd Freisleben	University of Marburg, Germany
José Enrique Gallardo	University of Málaga, Spain
Jens Gottlieb	SAP, Germany
Walter Gutjahr	University of Vienna, Austria
Jin-Kao Hao	University of Angers, France
Geir Hasle	SINTEF Applied Mathematics, Norway
Juhos István	University of Szeged, Hungary
Mario Köppen	Kyushu Institute of Technology, Japan
Graham Kendall	University of Nottingham, UK
Joshua Knowles	University of Manchester, UK
Jozef Kratica	University of Belgrade, Serbia
Arne Løkketangen	Molde College, Norway
Rhyd Lewis	Cardiff University, UK
Andrea Lodi	University of Bologna, Italy
José Antonio Lozano	University of the Basque Country, Spain
Vittorio Maniezzo	University of Bologna, Italy
Dirk C. Mattfeld	Technische Universität Braunschweig, Germany
Barry McCollum	Queen's University Belfast, UK
Juan Julián Merelo	University of Granada, Spain
Daniel Merkle	University of Southern Denmark, Denmark
Peter Merz	Technische Universität Kaiserslautern, Germany
Martin Middendorf	Universität Leipzig, Germany
Julian Molina	University of Málaga, Spain
Jose Marcos Moreno	University of La Laguna, Spain
Pablo Moscato	The University of Newcastle, Australia
Christine L. Mumford	Cardiff University, UK
Nysret Musliu	Vienna University of Technology, Austria
Gabriela Ochoa	University of Nottingham, UK
Francisco J. B. Pereira	Universidade de Coimbra, Portugal
Jakob Puchinger	Arsenal Research, Vienna, Austria
Günther Raidl	Vienna University of Technology, Austria

Marcus Randall	Bond University, Queensland, Australia
Marc Reimann	Warwick Business School, UK
Andrea Roli	Università degli Studi di Bologna, Italy
Michael Sampels	Université Libre de Bruxelles, Belgium
Marc Schoenauer	INRIA, France
Marc Sevaux	Université de Bretagne-Sud, France
Christine Solnon	University Lyon 1, France
Giovanni Squillero	Politecnico di Torino, Italy
Thomas Stützle	Université Libre de Bruxelles, Belgium
El-ghazali Talbi	Université des Sciences et Technologies de Lille, France
Kay Chen Tan	National University of Singapore, Singapore
Jorge Tavares	MIT, USA
Jano van Hemert	University of Edinburgh, UK

Table of Contents

A Critical Element-Guided Perturbation Strategy for Iterated Local Search

Zhipeng Lü and Jin-Kao Hao

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
zhipeng.lui@gmail.com, hao@info.univ-angers.fr

Abstract. In this paper, we study the perturbation operator of Iterated Local Search. To guide more efficiently the search to move towards new promising regions of the search space, we introduce a Critical Element-Guided Perturbation strategy (CEGP). This perturbation approach consists of the identification of critical elements and then focusing on these critical elements within the perturbation operator. Computational experiments on two case studies—graph coloring and course timetabling—give evidence that this critical element-guided perturbation strategy helps reinforce the performance of Iterated Local Search.

Keywords: iterated local search, perturbation operator, critical element-guided perturbation, graph coloring, course timetabling.

1 Introduction

Local search based metaheuristics are known to be an effective technique for solving a large number of constraint satisfaction and combinatorial optimization problems [14]. However, they may sometimes be trapped into a poor local optimum and it becomes extremely difficult to jump out of it even with more computing efforts. Therefore, diversification mechanisms play an important role in designing such kinds of algorithms.

In order to obtain a tradeoff between intensification and diversification in local search metaheuristics, many kinds of high-level diversification mechanisms have been proposed in the literature to avoid the search to fall into local optima. Typical examples include tabu list in Tabu Search [11], random acceptance criteria in Simulated Annealing, perturbation operator in Iterated Local Search [16], multiple neighborhoods in Variable Neighborhood Search [12] and so on. In particular, it is of significance to utilize low-level problem specific knowledge for constructing strong diversification mechanisms.

In this paper, we study the main diversification mechanism of Iterated Local Search (ILS) [16], i.e., the perturbation operator. ILS is a popular metaheuristic which is mainly composed of two basic components: one is a local search procedure and the other is a perturbation operator. When a local optimum solution cannot be improved any more using the local search, a perturbation operator is employed to produce a new solution, from which a new round of local search

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 1–12, 2009.

starts. It is desirable that the perturbation should be able to guide the search to a promising region of the search space.

As for other components of an ILS procedure, it is useful to integrate problem specific knowledge to make a perturbation operator informative and effective. One fundamental question is then what kind of problem knowledge should be used in the design of an effective perturbation operator.

In this paper, we put forward that the structure of the local optimum solution found so far itself can be used for constructing the perturbation operator. For this purpose, we introduce a new perturbation strategy, called Critical Element-Guided Perturbation (CEGP) for Iterated Local Search (Section 2). We illustrate this on two typical hard combinatorial optimization problems—graph coloring (Section 3) and course timetabling (Section 4), showing its importance in the design of a powerful ILS algorithm.

2 Critical Element-Guided Perturbation (CEGP)

Iterated Local Search can be described by a simple computing schema (see Section 2.3 for a general template). A fundamental principle of ILS is to exploit the tradeoff between diversification and intensification. Intensification focuses on optimizing the objective function as far as possible within a limited search region while diversification should be able to drive the search to explore new promising regions of the search space.

The diversification mechanism of ILS—perturbation operator—has two aims: one is to jump out of the local optimum trap just visited; the other is to lead the search procedure to a new promising region. A commonly-used perturbation operator is to destruct partially the previous local optimum solution in a *random* way, not necessarily guided by an evaluation function [16]. Zhang and Sun used the idea of estimation of distribution algorithms to construct perturbation, which combines global statistical information and the location information of good individual solutions for generating new trial solutions [20]. However, we think that the solution structure of the previously obtained local optimum itself can be used in the design of more intelligent and informative perturbation operators.

2.1 CEGP Procedure

For the two purposes just mentioned previously, one more elaborated perturbation should take into account the specific problem structure. Given a local optimum solution obtained by the local search procedure, if one can identify the contribution of each *element* to the cost function or constraint violations, then it is reasonable that a perturbation by changing the values of these critical *elements* would be helpful to jump out of local optimum trap. In its simplest form, an *element* can be a decision variable. For example, in the knapsack problem, an element might be an object while in university course timetabling problem, an element can be a lecture of a course.

Generally speaking, our critical element-guided perturbation strategy is composed of three phases: 1. ***Scoring***: give each element a score; 2. ***Selection***: choose

a certain number of highly-scored elements; 3. ***Perturbing***: randomly perturb the solution using the chosen critical elements.

In order to score an element e_i in the ***Scoring*** phase, it is indispensable to define a scoring function and its parameters. Generally, the parameters include the current element e_i and those elements which are strongly related to e_i. More formally, an element e_i can be scored as $Score(e_i) = h(e_i, \widehat{e_i})$, where $\widehat{e_i}$ are the elements set related to e_i and $h(\cdot)$ is a scoring function. For a specific problem, in order to score an element e_i, it is necessary to define its related elements set $\widehat{e_i}$ and the appropriate scoring function $h(\cdot)$ according to the problem specific knowledge. We will show two examples in Sections 3.5 and 4.4.

The ***Selection*** phase consists of choosing a certain number of elements according to their scores. For the ***Selection*** phase, it is implemented in an adaptive and random way, i.e., the higher score an element has, the more possibly this element is chosen. Note that this selection procedure is problem independent as shown in Algorithm 1 line 6.

After a certain number of critical elements are chosen, the ***Perturbing*** phase randomly perturbs the chosen elements. The perturbation operator can employ the moves in the local search procedure or quite different moves. In Sections 3.5 and 4.4, we respectively use these two kinds of perturbation operators.

2.2 CEGP Framework

Given a general constraint satisfaction and optimization problem (CSOP) [19] and a local optimum solution, the proposed critical element-guided perturbation operator is described in Algorithm 1, where ϕ is a positive real number and in this paper we empirically set $\phi \in [1.5, 3.0]$. One observes that the value of ϕ determines the intensity of the selection procedure: the larger the value of ϕ is, the higher is the possibility that the high-score elements are selected. Note that the commonly-used random perturbation is a special case of our CEGP strategy with $\phi = 0$. In this case, the selection probability becomes $P(k) = 1/n$. On the other hand, setting $\phi = \infty$ will select always the first η elements.

Algorithm 1. Critical Element-Guided Perturbation Strategy

1: **Input**: a local optimum solution s
2: **Output**: a perturbed solution s'
3: **Scoring**: score each element $e_i, i = 1, \ldots, n$: $Score(e_i) = h(e_i, \widehat{e_i})$
4: sort all the elements in a non-increasing order according to their scores
5: determine a perturbation strength η
6: **Selection**: randomly select η elements to be perturbed. The rth critical element is selected according to the following probability:

$$P(r) = r^{-\phi} / \sum_{i=1}^{n} i^{-\phi} \tag{1}$$

7: **Perturbing**: randomly perturb the selected η elements
8: get a perturbed solution s'

It should be noted that the scoring method (Algorithm 1, line 3) is essential in the CEGP strategy. It must be combined with low level problem specific knowledge, as shown in Sections 3.5 and 4.4. In addition, the perturbation strength η should also be determined according to the given problem.

2.3 ILS with CEGP

Iterated local search starts with an initial solution and performs local search until a local optimum is found. Then, the current local optimum solution is perturbed and another round of local search is performed to the perturbed solution. The perturbation procedure is implemented by the CEGP strategy just described above. Finally, an acceptance criterion is used to decide whether the new local optimum solution is accepted as the initial solution for the next run of local search. Algorithm 2 shows the pseudocode of CEGP-based ILS. The detailed description of the general ILS procedure can be found in [16].

Algorithm 2. Iterated Local Search with CEGP Strategy

1: $s_0 \leftarrow$ Initial Solution
2: $s' \leftarrow$ Local Search(s_0)
3: **repeat**
4: $\quad s^* \leftarrow$ Critical Element-Guided Perturbation(s')
5: $\quad s^{*'} \leftarrow$ Local Search(s^*)
6: $\quad s' \leftarrow$ Acceptance Criterion($s^{*'}$,$s^{'}$)
7: **until** stop condition met

In order to implement the CEGP strategy for a given problem, we just need to define what an element e_i is, how the related elements of each e_i are identified, how the scoring function is designed and what is the perturbation moves for the chosen elements. In the following two sections, we show two case studies of applying the ILS algorithm with the proposed CEGP strategy to two difficult problems—graph coloring and course timetabling.

3 Case Study 1: Graph Coloring

3.1 Problem Description

Given an undirected graph $G = (V, E)$ with a set V of n vertices and an edge set E as well as the number of colors to be used k, a legal k-coloring of graph G is a partition of V into k independent sets where an independent set is a subset of nonadjacent vertices of G. In a formal way, let c_i be the color of vertex v_i ($c_i \in [1, k]$, $i = 1, \ldots, n$), a legal k-coloring of graph G is a coloring $C = \{c_1, \ldots, c_n\}$ such that $\forall\{v_i, v_j\} \in E$, $c_i \neq c_j$.

Graph coloring aims at finding the smallest k for a given graph G (the chromatic number χ_G of G) such that G has a legal k-coloring.

3.2 General Solution Procedure

The graph coloring problem can be solved from the point of view of constraint satisfaction by solving a series of k-coloring problems. We starts from an initial number of k colors ($k = |V|$ is certainly sufficient) and solve the k-coloring problem. As soon as the k-coloring problem is solved, we decrease k by setting k to k-1 and solve again the k-coloring problem. This process is repeated until no legal k-coloring can be found.

3.3 Initial Solution and Evaluation Function

For a k-coloring problem with a given k, we generate an initial solution by means of a greedy algorithm presented in [10]. It can be considered as an improved version of the famous DSATUR algorithm [2]. Note that this greedy heuristic generally generates an illegal k-coloring.

Once an initial solution is obtained where each vertex has been assigned a color, our CEGP-based ILS algorithm is used to minimize the number of edges having both endpoints with a same color (or the conflict number). Therefore, our evaluation function is just the number of conflicts $f(C)$ such that

$$f(C) = \sum_{\{v_i, v_j\} \in E} \delta_{ij} \tag{2}$$

where

$$\delta_{ij} = \begin{cases} 1, \text{ if } c_i = c_j; \\ 0, \text{ otherwise.} \end{cases} \tag{3}$$

Accordingly, any coloring C with $f(C) = 0$ corresponds to a legal k-coloring which presents a solution to the k-coloring problem.

3.4 Local Search Procedure

In this paper, we employ the Tabu Search algorithm presented in [8] as our local search procedure. This TS algorithm is an improved version of the TABUCOL algorithm in [13]. Here a neighborhood of a given configuration is obtained by changing the color c_i of a conflicting vertex v_i to another color c_j ($c_j \neq c_i$), denoted by (v_i, c_j). The cost deviation of the move (v_i, c_j) is denoted by $\Delta f_{(v_i,c_j)}(C)$. More details can be found in [8].

For the tabu list, once a move (v_i, c_j) is performed, vertex v_i is not allowed to receive again the color c_i for the next tt iterations. The tabu tenure tt is empirically determined by $tt = f + random(10)$ where f is the conflict number of the current solution and $random(10)$ takes a random number in $\{1, \ldots, 10\}$. The stop condition of our tabu search is just the maximal number of iterations during which the best solution has not been improved. In this work, we set this number to be 1,000,000 for all the tested instances.

3.5 Perturbation

Once the local search procedure stops with a local optimum solution, a Critical Element-Guided Perturbation operator is performed to reconstruct the obtained local optimum solution. Given the general CEGP strategy described in Section 2, one just needs to know how to score each vertex (element) and what is the perturbation operator for the chosen vertices.

Let us first consider the ***Scoring*** phase. For a vertex (an element) v_i, its related element set $\widehat{v_i}$ is defined as the set of v_i's adjacent vertices, i.e., $\widehat{v_i} = \{v_j | \{v_i, v_j\} \in E\}$. Based on $\widehat{v_i}$, the following three sets can be derived:

$$V_i^1 = \{v_j | v_j \in \widehat{v_i}, c_i = c_j\} \tag{4}$$

$$V_i^2 = \{v_j | v_j \in \widehat{v_i}, |V_j^1| > 0\} \tag{5}$$

$$K_i = \{c_j | c_j \neq c_i, \Delta f_{(v_i,c_j)}(C) = 0\} \tag{6}$$

Given these notations, the score of a vertex v_i is calculated as:

$$Score(v_i) = h(v_i, \widehat{v_i}) = \omega_1 \cdot |V_i^1| + \omega_2 \cdot |V_i^2| + \omega_3 \cdot |K_i| \tag{7}$$

where ω_1, ω_2 and ω_3 are the associated weights for these three kinds of scores and we set empirically $\omega_1 = 5$ and $\omega_2 = \omega_3 = 1$ respectively.

In the above formulations, $|V_i^1|$ denotes the total number of vertices conflicting with vertex v_i: $|V_i^1| = 0$ means that vertex v_i is conflict-free while $|V_i^1| > 0$ implies that vertex v_i is conflicting. $|V_i^2|$ denotes the number of v_i's adjacent vertices which themselves are conflicting. It is easy to observe that V_i^1 is a subset of V_i^2. The larger $|V_i^1|$ and $|V_i^2|$ are, the higher score vertex v_i has. The rationale behind this is that a conflicting vertex should naturally change its color while the vertex adjacent to a number of conflicting vertices should also be recolored since it would be impossible for the conflicting vertex to become conflict-free in the next round of the local search if its adjacent vertices are not reassigned.

Furthermore, it is reasonable to give a higher priority to a vertex with a large number of *side walks*, where a *side walk* of vertex v_i denotes a move (v_i, c_j) $(c_j \neq c_i)$ that will not change the total cost function, i.e., $\Delta f_{(v_i,c_j)}(C) = 0$. $|K_i|$ represents the number of *side walks* for vertex v_i. Note that changing the color of a vertex having a large number of *side walks* will not worsen the solution quality to a large extent, thus the reassignment of these vertices might help the search to jump out of local optimum solution while keeping the solution quality at a good level.

Once all the vertices are scored in accordance with Eq. (7), they are sorted in a decreasing order according to their scores. The perturbation strength is empirically determined by $\eta = 0.33 \cdot n + random(100)$. We observed that a weaker perturbation strength did not allow the search to escape from the local optima. According to Eq. (1) in Algorithm 1, η vertices are randomly chosen. After that, we remove all these η vertices and reassign them using the greedy heuristic as shown in Section 3.3. This is the ***Perturbing*** phase in our CEGP strategy. Thus, a perturbed solution is obtained, from which a new round of local search starts.

3.6 Experimental Results and Comparisons

To evaluate the efficiency of this CEGP-based ILS algorithm, we carry out experiments on a set of 23 non-trivial DIMACS coloring benchmarks. We contrast the results of our CEGP-based ILS algorithm with the uniformly random perturbation strategy (URP) in order to highlight the impact of the CEGP strategy. To make the comparison as fair as possible, the only difference between URP-based and CEGP-based ILS algorithms is that the perturbed vertices are selected in a blindly uniform way with the URP strategy, as described in Section 2.2. Moreover, we compare our coloring results with those of some best performing reference algorithms. Our algorithm is run on a PC with 3.4GHz CPU and 2.0Gb RAM. To obtain our computational results, the total CPU time limit for each instance is limited to 8 hours. Note that the time limit for the reference algorithms is from several hours to several tens of hours.

Table 1 gives our computational results. Column 2 presents the best known k^* ever reported in the literature. Columns 3 and 4 give the results of the CEGP-based and URP-based ILS algorithms respectively, together with the CPU time in minutes (in brackets). Columns 5 to 11 give the results of seven reference algorithms, including four local search algorithms [1,3,6,13] as well as three hybrid algorithms [7,8,9].

One finds that the CEGP-based ILS algorithm obtains smaller k than the URP-based ILS algorithm for 4 out of 23 instances while larger k for only 1 instance, which would suggest the effectiveness of the proposed CEGP strategy. Furthermore, the CEGP-based ILS algorithm obtains comparable results with these famous reference algorithms.

Table 1. Computational results of CEGP-based ILS algorithm on graph coloring problem

		ILS		Local Search Algorithms				Hybrid Algorithms		
Instances	k^*	CEGP(t)	URP(t)	[1]	[3]	[6]	[13]	[7]	[8]	[9]
dsjc250.5	28	28 (1)	28 (2)	—	28	28	—	29	28	28
dsjc500.1	12	12 (1)	12 (1)	12	12	13	—	—	—	12
dsjc500.5	48	48 (76)	48 (98)	49	49	50	51	49	48	48
dsjc500.9	126	126 (10)	126 (16)	127	126	127	—	—	—	126
dsjc1000.1	20	21 (3)	21 (4)	20	—	21	—	—	20	20
dsjc1000.5	83	87 (35)	88 (29)	89	89	90	94	84	83	84
dsjc1000.9	224	224 (46)	225 (32)	228	—	226	—	—	224	224
r125.5	36	36 (6)	36 (8)	—	—	36	—	—	—	—
r250.5	65	65 (4)	65 (4)	66	—	66	—	69	—	—
r1000.1c	98	98 (7)	98 (6)	98	—	98	—	—	—	—
r1000.5	234	253 (38)	252 (51)	249	—	242	—	—	—	—
dsjr500.1c	85	85 (14)	85 (19)	85	—	—	—	—	—	86
dsjr500.5	122	125 (29)	125 (38)	126	124	—	—	130	—	127
le450_15c	15	16 (1)	16 (1)	15	15	—	18	15	15	15
le450_15d	15	15 (24)	15 (34)	15	15	—	18	15	—	15
le450_25c	25	25 (28)	26 (1)	25	26	—	—	25	26	26
le450_25d	25	25 (34)	26 (1)	25	26	—	—	25	—	26
flat300_26_0	26	26 (3)	26 (3)	—	26	26	32	26	—	26
flat300_28_0	28	30 (18)	30 (15)	28	31	31	32	33	31	31
flat300_50_0	50	50 (6)	50 (8)	50	—	50	—	90	—	50
flat300_60_0	60	60 (10)	60 (17)	60	—	60	—	90	—	60
flat300_76_0	82	87 (46)	87 (69)	88	—	89	93	84	83	84
latin_square_10	98	100 (37)	100 (34)	—	—	—	—	—	—	104

4 Case Study 2: Course Timetabling

The course timetabling problem consists of scheduling all lectures of a set of courses into a weekly timetable, where each lecture of a course must be assigned a period and a room in accordance with a given set of constraints. In this problem, all hard constraints must be strictly satisfied and the weighted soft constraint violations should be minimized. In this paper, we study the curriculum-based course timetabling problem (CB-CTT), which is one of the three topics of the second international timetabling competition (ITC–2007)[1].

4.1 Problem Description

In the CB-CTT problem, a feasible timetable is one in which all lectures are scheduled at a timeslot and a room, such that the hard constraints H_1-H_4 (see below) are satisfied. In addition, a feasible timetable satisfying the four hard constraints incurs a penalty cost for the violations of the four soft constraints S_1-S_4 (see below). Then, the objective of CB-CTT is to minimize the weighted soft constraint violations in a feasible solution. The four hard constraints and four soft constraints are:

- H_1. **Lectures:** Each lecture of a course must be scheduled in a distinct period and a room.
- H_2. **Room occupancy:** Any two lectures cannot be assigned in the same period and the same room.
- H_3. **Conflicts:** Lectures of courses in the same curriculum or taught by the same teacher cannot be scheduled in the same period, i.e., no period can have an overlapping of students nor teachers.
- H_4. **Availability:** If the teacher of a course is not available at a given period, then no lectures of the course can be assigned to that period.
- S_1. **Room capacity:** For each lecture, the number of students attending the course should not be greater than the capacity of the room hosting the lecture.
- S_2. **Room stability:** All lectures of a course should be scheduled in the same room. If this is impossible, the number of occupied rooms should be as few as possible.
- S_3. **Minimum working days:** The lectures of a course should be spread into the given minimum number of days.
- S_4. **Curriculum compactness:** For a given curriculum, a violation is counted if there is one lecture not adjacent to any other lecture belonging to the same curriculum within the same day, which means the agenda of students should be as compact as possible.

[1] http://www.cs.qub.ac.uk/itc2007/

4.2 Initial Solution, Search Space and Evaluation Function

Starting from an empty timetable, we generate first an initial feasible solution by means of a greedy graph coloring heuristic. We simply mention that for all the 21 competition instances, this greedy heuristic can easily obtain feasible solutions. Once a feasible timetable that satisfies all the hard constraints is reached, our ILS algorithm is used to minimize the soft constraint violations while keeping hard constraints satisfied. Therefore, the search space of our algorithm is limited to the feasible timetables. The evaluation function of our algorithm is just the weighted soft constraint violations as defined for the ITC–2007.

4.3 Local Search Procedure

For this problem, we use a Tabu Search algorithm with two distinct neighborhoods defined by two *move*s denoted as *SimpleSwap* and *KempeSwap*. *SimpleSwap* move consists in exchanging the hosting periods and rooms assigned to two lectures of different courses while a *KempeSwap* move produces a new feasible assignment by swapping the period labels assigned to the courses belonging to two specified Kempe chains. Our Tabu Search algorithm explores these two neighborhoods in a token-ring way. More details about these neighborhoods and the TS algorithm are given in [17].

4.4 Perturbation

For the ***Scoring*** phase of this problem, an element is just one lecture of a course and the related elements of a lecture are the lectures involved in the calculation of the lecture's soft constraint violations. When the current TS phase terminates with a local optimum solution, the scoring function of a lecture is just the weighted sum of soft constraint violations involving the lecture.

Then, all the lectures are ranked in a decreasing order according to their scores and a number of lectures are randomly selected in accordance with Eq. (1). Finally, the ***Perturbing*** phase consists of randomly selecting a series of *SimpleSwap* or *KempeSwap* moves involving the chosen lectures. Thus, a perturbed solution is obtained from which a new round of Tabu Search starts.

4.5 Experimental Results

In this section, we report computational results on the set of 21 competition instances using two formulations of the CB-CTT problem [4]. The first formulation is previously studied by Di Gaspero *et al* in [5] and the second one is just the topic of the ITC–2007. Like in Section 3.5, we contrast the results of our CEGP-ILS algorithm with that of URP-ILS. We also compare our results with the best known results obtained by other algorithms from the literature. To obtain our computational results, each instance is solved 100 times independently and each ILS run is given a maximum of 2,000,000 local search steps.

Table 2 shows the best results of the CEGP-based and URP-based ILS algorithms on the 21 competition instances for both formulations as well as the

Table 2. Computational results and comparison on the 21 course timetabling competition instances

	Old Formulation					ITC-2007 Formulation					
Instance	best	CEGP(t)	URP(t)	[4]	[15]	best	CEGP(t)	URP(t)	[4]	[15]	[18]
comp01	4	4 (0)	4 (0)	4	—	5	5 (0)	5 (0)	5	—	5
comp02	24	**20** (35)	22 (28)	24	—	33	**29** (64)	35 (52)	56	33	35
comp03	39	**38** (29)	40 (20)	39	—	66	66 (35)	68 (28)	79	—	66
comp04	18	18 (36)	18 (48)	18	—	35	35 (9)	35 (13)	38	—	35
comp05	240	**219** (8)	224 (4)	240	—	298	**292** (3)	310 (5)	316	—	298
comp06	**16**	18 (72)	20 (60)	25	16	37	37 (99)	38 (112)	55	—	37
comp07	3	3 (28)	3 (36)	7	3	**7**	13 (67)	14 (75)	26	—	7
comp08	20	20 (18)	21 (12)	22	20	**38**	39 (51)	41 (40)	42	—	38
comp09	59	**54** (42)	56 (50)	59	—	99	**96** (29)	102 (38)	104	99	100
comp10	**2**	3 (15)	4 (17)	6	2	**7**	10 (14)	14 (10)	19	—	7
comp11	0	0 (0)	0 (0)	0	—	0	0 (0)	0 (0)	0	—	0
comp12	241	**239** (45)	242 (31)	241	—	320	**310** (42)	315 (61)	342	—	320
comp13	33	**32** (24)	32 (17)	36	33	60	**59** (70)	60 (78)	72	60	61
comp14	28	**27** (34)	28 (38)	29	28	51	51 (66)	51 (48)	57	51	53
comp15	39	**38** (14)	38 (19)	39	—	70	**68** (156)	69 (145)	79	—	70
comp16	21	**16** (95)	18 (76)	21	—	28	**23** (171)	30 (176)	46	28	30
comp17	41	**34** (42)	36 (48)	41	—	70	**69** (47)	72 (68)	88	—	70
comp18	37	**34** (21)	34 (38)	37	—	75	**65** (125)	68 (118)	75	—	75
comp19	33	**32** (70)	33 (52)	33	—	57	57 (164)	57 (143)	64	—	57
comp20	14	**11** (16)	11 (9)	14	—	**17**	22 (146)	30 (130)	32	17	22
comp21	56	**52** (31)	53 (46)	56	—	**89**	93 (82)	96 (70)	107	—	89

previous best known results available in the literature. The average CPU time for our CEGP-ILS and URP-ILS algorithms is also indicated in brackets (in minutes). The reference algorithms include the tabu search algorithm in [4], the integer programming algorithm in [15] and the hybrid algorithm in [18]. Interestingly, these reference best known results are from a web site maintained by the organizers of ITC–2007 (track 3)[2], which provides a complete description about the CB-CTT problem and the continuously updated best known results uploaded by researchers (the column "best" in Table 2).

From Table 2, one easily observes that the CEGP-based ILS algorithm reaches quite competitive results. First of all, it performs better than the URP-based ILS algorithm for the majority of the 21 competition instances and no worse result is observed for any instance. In order to testify the influence of the proposed CEGP perturbation operator, we performed a 95% confidence t-test to compare CEGP-ILS with URP-ILS for both formulations. We found that for 13 (respectively 15) out of the 21 instances of the old formulation (respectively ITC-2007 formulation), the difference of the computational results obtained by CEGP-ILS and URP-ILS is statistically significant. In addition, the CEGP-based ILS algorithm improves the previous best known solutions for 14 and 9 out of the 21 instances respectively for the two formulations, showing the strong search power of the CEGP-based ILS algorithm.

5 Conclusion and Discussion

The purpose of this paper is to investigate the diversification scheme of Iterated Local Search. To this end, we proposed a general Critical Element-Guided

[2] http://tabu.diegm.uniud.it/ctt/index.php

Perturbation strategy for jumping out of local optimum solution. The essential idea of this perturbation strategy lies in identifying the critical elements in the local optimum solution and adaptively perturbing the solution using these critical elements. This perturbation approach provides a mechanism for diversifying the search more efficiently compared with the commonly-used uniformly random perturbation strategy.

An ILS algorithm with this CEGP strategy was tested on two case studies—graph coloring and course timetabling, showing clear improvements over the traditional blindly uniform perturbation strategy. The results also show that the CEGP-based ILS algorithm competes well with other reference algorithms in the literature.

The practical effectiveness of this strategy depends mainly on the problem-specific scoring method and the perturbation moves that will be performed. For constrained problems, the score for each element can be simply based on the number of violations involved. For optimization problems, such as Job Shop Scheduling (JSS) and TSP problems, the problem-specific knowledge must be explored in order to score each element. For JSS, the elements in the bottleneck machine or the critical path should reasonably have high scores. For TSP, the scores for elements might be marked according to the tour length involved.

To conclude, we believe that the Critical Element-Guided Perturbation strategy helps design high performance ILS algorithm. At the same time, it should be clear that for a given problem, it is indispensable to realize specific adaptations by considering problem-specific knowledge in order to obtain high efficiency.

Acknowledgment

The authors would like to thank the anonymous referees for their helpful comments. This work was partially supported by "*Angers Loire Métropole*" and the Region of "*Pays de la Loire*" within the MILES and RADAPOP Projects.

References

1. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research 35(3), 960–975 (2008)
2. Brélaz, D.: New methods to color the vertices of a graph. Communications of the ACM 22(4), 251–256 (1979)
3. Chiarandini, M., Stützle, T.: An application of iterated local search to graph coloring. In: Johnson, D.S., Mehrotra, A., Trick, M.A. (eds.) Proceedings of the Computational Symposium on Graph Coloring and its Generalizations, Ithaca, New York, USA, pp. 112–125 (2002)
4. De Cesco, F., Di Gaspero, L., Schaerf, A.: Benchmarking curriculum-based course timetabling: Formulations, data formats, instances, validation, and results. In: Proceedings of the 7th PATAT Conference (2008), `http://tabu.diegm.uniud.it/ctt/DDS2008.pdf`
5. Di Gaspero, L., Schaerf, A.: Neighborhood portfolio approach for local search applied to timetabling problems. Journal of Mathematical Modeling and Algorithms 5(1), 65–89 (2006)

6. Dorne, R., Hao, J.K.: Tabu search for graph coloring, T-colorings and set Tcolorings. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization, ch. 6, pp. 77–92. Kluwer, Dordrecht (1998)
7. Fleurent, C., Ferland, J.A.: Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability. In: Johnson, D.S., Trick, M.A. (eds.) DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 619–652. American Mathematical Society, Providence (1996)
8. Galinier, P., Hao, J.K.: Hybrid evolutionary algorithms for graph coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
9. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the K-colouring problem. Discrete Applied Mathematics 156(2), 267–279 (2008)
10. Glover, F., Parker, M., Ryan, J.: Coloring by tabu branch and bound. In: Johnson, D.S., Trick, M.A. (eds.) DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 26, pp. 285–307. American Mathematical Society, Providence (1996)
11. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
12. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. European Journal of Operational Research 130, 449–467 (2001)
13. Hertz, A., De Werra, D.: Using tabu search techniques for graph coloring. Computing 39(4), 345–351 (1987)
14. Hoos, H.H., Stützle, T.: Stochastic local search: foundations and applications. Morgan Kaufmann, San Francisco (2004)
15. Lach, G., Lübbecke, M.E.: Curriculum based course timetabling: optimal solutions to the udine benchmark instances. In: Proceedings of the 7th PATAT Conference (2008), http://www.math.tu-berlin.de/~luebbeck/papers/udine.pdf
16. Lourenco, H.R., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Meta-heuristics, pp. 321–353. Springer, Heidelberg (2003)
17. Lü, Z., Hao, J.K.: Adaptive tabu search for course timetabling. European Journal of Operational Research (2009), doi:10.1016/j.ejor.2008.12.007
18. Müller, T.: ITC2007 solver description: A hybrid approach. In: Proceedings of the 7th PATAT Conference (2008), http://www.unitime.org/papers/itc2007.pdf
19. Tsang, E.: Foundations of constraint satisfaction. Academic Press, London (1993)
20. Zhang, Q., Sun, J.: Iterated local search with guided mutation. In: Proceedings of IEEE Congress on Evolutionary Computation, pp. 924–929 (2006)

Binary Exponential Back Off for Tabu Tenure in Hyperheuristics*

Stephen Remde, Keshav Dahal, Peter Cowling, and Nic Colledge

MOSAIC Research Group, University of Bradford, Bradford, BD7 1DP, United Kingdom
{s.m.remde,p.i.cowling,k.p.dahal,n.j.colledge}@bradford.ac.uk
http://mosaic.ac/

Abstract. In this paper we propose a new tabu search hyperheuristic which makes individual low level heuristics tabu dynamically using an analogy with the Binary Exponential Back Off (BEBO) method used in network communication. We compare this method to a reduced Variable Neighbourhood Search (rVNS), greedy and random hyperheuristic approaches and other tabu search based heuristics for a complex real world workforce scheduling problem. Parallelisation is used to perform nearly 155 CPU-days of experiments. The results show that the new methods can produce results fitter than rVNS methods and within 99% of the fitness of those produced by a highly CPU-intensive greedy hyperheuristic in a fraction of the time.

1 Introduction

Hyperheuristics [1] [2] reflect problem knowledge using a number of (usually simple) low level heuristics (LLHs) and objective measure(s). The hyperheuristic uses information about the performance of each low level heuristic (CPU time and objective measures) to determine which low level heuristics to apply at each decision point. The hypothesis of the hyperheuristic method is that some combination of these low level heuristics will prove effective in escaping any poor quality local optimum/basin of attraction [1]. However, the method which decides which low level heuristics to choose needs not be problem specific, and hence a single hyperheuristic method can work generally across many problem models and instances. In some cases low level heuristics are parameterised, or composed by "multiplying" together components [3] [4], which can give rise to hundreds or even thousands of heuristics [5]. In such a case, deciding in reasonable time which heuristics to use may be difficult. However, there is evidence that having such a rich selection of low level heuristics may yield better results for complex problems in the long run [4].

In collaboration with Trimble we look at a workforce scheduling problem, which is a resource constrained scheduling problem similar to but more complex than many other well-studied scheduling problems such as the Resource Constrained Project Scheduling Problem (RCPSP) [6] and job shop scheduling problem [7]. Trimble

* This work was funded by EPSRC and Trimble under an EPSRC CASE studentship, made available through the Smith Institute for Industrial Mathematics and System Engineering.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 109–120, 2009.

develops scheduling solutions for very large, complex mobile workforce scheduling problems in a variety of industries, particularly telecommunications and utilities.

The workforce scheduling problem that we consider consists of four main components: Tasks, Resources, Skills and Locations. Each Task has a location and a value which indicates priority relative to other tasks. Resources are engineers and large pieces of equipment. They are mobile, travelling at a variety of speeds to geographically dispersed task Locations. Tasks and resources have time windows, and each time window has an associated cost per unit time (which models customer inconvenience, overtime etc.). Tasks require a pre-specified quantity of certain skills. Each resource possesses one or more of these skills and a competence level which models the resources speed at executing the skill. A major source of complexity in our problem arises since a task's duration is unknown until resources are assigned to it. In this paper, the fitness of a schedule is given by a single weighted objective function [3], $f = SP - 4SC - 2TT$, where SP is the sum of the priority of scheduled tasks, SC is the sum of the time window costs in the schedule (both resource and task) and TT is the total amount of travel time. This objective is to maximise the total priority of tasks scheduled while minimising travel time and cost. The weights in this expression are sensible in a large class of the problems encountered in practice.

Using the hyperheuristic framework of [3], we create more low level heuristics and implement a new tabu based hyperheuristic with dynamic tabu tenures designed with large neighbourhoods in mind. *HyperRandom* and *HyperGreedy* heuristics from [3] try significant numbers of bad low level heuristics and hence waste CPU time, which may be highly significant for complex instances requiring CPU-hours or -days to solve. Analysis of the low level heuristics used in the *HyperGreedy* method showed that 26.4% of the low level heuristics were never used and 51.2% of the low level heuristics were used less than one percent of the time. An approach which more aggressively prunes poor low level heuristics could result in large CPU savings in this case, with little or no impact on solution quality. This impact would increase with increasing numbers of low level heuristics.

The method we propose is a tabu based hyperheuristic with dynamically adapting tabu tenures designed for very large neighbourhoods. In a variety of computer networks, binary exponential back off or truncated binary exponential back-off is a randomized protocol for regulating transmission on a multiple access broadcast channel [8]. This algorithm is used to spread out repeated retransmissions of the same block of data and to increase overall efficiency. Data needs to be retransmitted when a collision occurs. This happens when two (or more) computers try to transmit information on the same medium (a wire, a wireless frequency, etc) at the same time. When a computer transmits information, it also "listens" to see whether received information is what has been transmitted. If it detects anomalies it assumes that there is interference (probably from another computer trying to transmit at the same time). When a collision occurs, the computer will increase its *backoff* value by 1, and wait a random amount of time between 0 and $2^{backoff}$-1 before trying to retransmit. If the transmission is successful, the back off value is reset to 0, otherwise the back-off value is increased by 1 again and the process is repeated. Truncated binary exponential back-off [9] works in a similar way, but also sets a ceiling on the maximum back-off time. This is the industry standard for many computer networks including Ethernet.

Binary exponential back off seems like a promising approach to adjust individual tabu tenures in a tabu-search hyperheuristic [10] or metaheuristic, where the neighbourhood size is large. For low level heuristics that perform poorly, the tabu tenure would increase exponentially with each poor application and thus minimise wasted CPU time. For low level heuristics that perform badly at the start of the search and well at the end, little time would be wasted at the start but when the low level heuristic starts performing well, it will not be penalised for doing badly at the start. This could focus a very large set of low level heuristics down to a smaller set of low level heuristics which are effective at a particular point in the search.

This paper is structured as follows: we present related work in section 2 and propose our new hyperheuristic approach in section 3. In section 4 we empirically investigate the new technique and compare it to Variable Neighbourhood Search, Greedy, Random and Tabu based heuristics in terms of solution quality and computational time. We present conclusions in section 5.

2 Related Work

The workforce scheduling problem we study is complex and [3] proposes a method to break down this problem by splitting it into smaller parts and solving each part using exact enumerative approaches. These smaller parts are the combination of a method to select a task and a method to select resources, including time, for the task. Reduced Variable Neighbourhood Search (*rVNS*) [11] and hyperheuristics are then used to decide the order in which to solve them. Analysis of results for the CPU-intensive *HyperGreedy* show that the success of the hyperheuristics was due to there being a rich set of low level heuristics and not just a "silver bullet". However, a lot of time was wasted trying bad low level heuristics.

Many hyperheuristic approaches have been investigated, based on generalisation of metaheuristic methods, including early work in [12] where a genetic algorithm evolves a chromosome which determined how jobs were scheduled in open shop scheduling. Learning approaches based on the "choice function" were considered in [13,14], where an estimate of how well a low level heuristic is likely to perform is used to decide which low level heuristic to choose next. Related tabu search hyperheuristics [10] are discussed in detail later. In [15] simulated annealing is used to decide whether to accept the solution resulting from a randomly applied low level heuristic. [16] uses a Genetic Algorithm to evolve good sequences of low level heuristics. [4] use a method called step by step reduction (SSR) and Warming Up (WU) approach to reduce the number of low level heuristics and show that SSR produced better results. SSR removes bad heuristics early on, but, unlike the method in this paper, does not allow the reintroduction of these heuristics later in the search.

Tabu Search [10] can be used to make undesirable moves unusable for a certain number of iterations (the tabu tenure). This is usually used to stop poorly performing moves being tried in succession or to stop the undoing of good moves. The optimal duration of a tabu tenure has been tested in several papers and it is most likely a function of the neighbourhood size and the problem size [17]. Random tabu tenures are used in [18] where a move is made tabu for a period randomly chosen between 1 and the maximum tabu tenure and was found to be superior to fixed tabu tenures. Much work has been done on Tabu Search and related choice function methods [1].

Several papers have investigated tabu-based and related choice function [13] based hyperheuristics. [19] uses a simple tabu mechanism where good and bad low level heuristics are made tabu for a fixed tabu tenure. A small number of low level heuristics are used (13) with short tabu tenures (1-4 iterations) and good results are obtained in a large amount of CPU time. This is extended in [20] where the low level heuristic is repeated until no further improvements can be found before being made tabu and random tabu tenures are utilised. Random tabu tenures provide results of similar quality to those with fixed tenure equal to the expected random tenure on the two problem instances they consider. The repeated application of a low level heuristics does not increase solution quality considerably. [10] uses a ranking system for non tabu low level heuristics. When a non-tabu low level heuristic performs well its rank is increased, when it doesn't make a positive change its rank is decreased and the low level heuristic is put in the tabu list on a first in first out basis. If it makes a negative change the tabu list is emptied. At each iteration the low level heuristic with the highest rank that is not tabu is used. The number of low level heuristics is again small, and the maximum size of the tabu list is between 2 and 4. [5] use a tabu hyperheuristic to manage a large set (95) of low level heuristics. The hyperheuristic allows the use of tabu low level heuristics if it makes the best improvement (and then stops the low level heuristic from being tabu). If no improving low level heuristics are available, a non improving non tabu low level heuristic is used and made tabu. Fixed tabu tenures of 10, 30, 60 and 100 and adaptive tabu tenures are investigated, but results provide no clear advantage of using adaptive tabu tenures over fixed ones.

3 Hyperheuristic Approaches

Papers such as [3] [4] generate possible LLHs by considering separately (1) selecting a task to be scheduled and (2) allocating potential resources (including time) for that task. The task selector chooses a task and the resource allocator assigns resources for each skill required by the task, so that the total number of LLHs is the number of task selectors multiplied by the number of resource allocators. Fig. 1 shows an example of this. The task selector has chosen Task 8 (which requires Skill 5 and Skill 4). The resource allocator has chosen R_a or R_b for Skill 5 and R_a or R_c for Skill 4. The different combinations are then tried and the best one is accepted (R_b, R_a).

In [3] resource selectors order the resources by their competency at the skill (as more competent resources can complete the task quicker) and then pick a range of these resources (Top 5, Top 10, etc). In addition to these approaches, here we add more low level heuristics in an attempt to yield better results, by improving the likelihood of there being at least one good LLH in every situation. Table 1 describes the new resource allocators. Combining each of the 9 task selector with each of the 27 resource allocators gives a total of 243 Low Level Heuristics. Note that for this problem it is usually better to choose a group of uniformly poor competence resources for a task (so that they complete at about the same time) rather than a heterogeneous set (where fast, effective, resources have to wait for slower resources to finish when they could be completing other tasks).

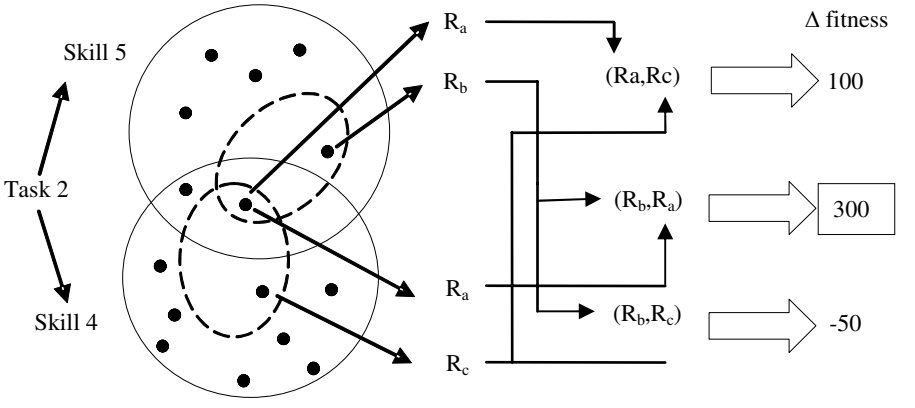


Fig. 1. The dotted subset of resources possessing the required skill is chosen by a Resource Selector. The assignment (R_a, R_b) is chosen as the best insertion.

Table 1. New Resource Selectors

Name	Description
Deviation *x*	Resources complete a skill in a time dependent upon their competence. This selector attempts to find resources that will complete the different skills of task in the same amount of time by selecting resources with competencies that deviate *x={50%, 25%, 12.5%, 6.25%}* from the task's skill requirement.
x^{th} Quarter	This picks the *x={1,2,3,4}* quarter of task ranked by skill. Unlike the "Top x" task selectors, the number chosen is proportionate to the number of resources who can do the task.
x^{th} Eighth	This picks the *x={1…8}* eighth of task ranked by skill.
Dynamic *x*	This selector picks larger sets of resources for the skills requiring more effort and less to those requiring less effort. It will create *x={10, 50, 100, 1000}* combinations when enumerating the resulting sets.
All Resources	Considers all possible resources (and hence is very slow).

The hyperheuristic *HyperRandom*, selects at random a Low Level Heuristic (i.e. a (task order, resource selector) pair) to use at each iteration and applies it if the application will result in a positive improvement. This continues until no improvement has been found for a certain number of iterations. *HyperGreedy* evaluates all the Low Level Heuristics at each iteration and applies the best if it makes an improvement. This continues until no improvement is found. As might be expected, *HyperGreedy* is very CPU-intensive, and generates good quality results, but is inefficient. For example, over one quarter of the low level heuristics were never applied in experimental trials [3] and over half of them were only applied once.

Here we propose a tabu based hyperheuristic with dynamically adapting tabu tenures designed for very large neighbourhoods, inspired by the binary exponential back-off algorithm used in networking [8]. We use an analogous backing off method to exponentially increase the tabu tenure of low level heuristics which repeatedly yield

no improvements, meaning the time between trials of bad heuristics gets exponentially greater. The heuristic is given in fig. 2. We use two methods to decide which of the low level heuristics, which were tried, to back off (those "deemed bad"):

1) "Best x": only the best x improving low level heuristics are not backed off.
2) "Prop x": all non improving low level heuristics and those improving low level heuristics not in the top x% of the range of the fitness are backed off.

Define:
$backoff\_min$ is the minimum backoff value (we choose 4)
LLH_i is Low level heuristic i
$tabu_i$ is the Tabu value of LLH_i ($0 \leq Tabu_i \leq Backoff_i$)
$backoff_i$ is the backoff value of LLH_i ($Backoff\_min \leq Backoff_i$)
$Eligible = \{LLH_i: tabu_i = 0\}$
$\Delta(S,LLH_i)$ is the change in the objective function which would result from applying low level heuristic LLH_i to solution S.
$apply(S,LLH_i)$ is the new solution we get after applying low level heuristic LLH_i to solution S.

Initialise:
create an initial solution S (often the solution S is the empty solution).
for all i:
  $backoff_i \leftarrow backoff\_min$
  choose $tabu_i$ uniformly at random in $\{0,1,2,\ldots, backoff_i\}$
Iterate:
while $(Eligible \neq \{\})$
  $best\Delta = 0$
  for each low level heuristic $LLH_i \in Eligible$
    if $\Delta(S,LLH_i) > 0$
      $backoff_i \leftarrow backoff\_min$
      if $\Delta(S,LLH_i) > best\Delta$
        $best\Delta \leftarrow \Delta(S,LLH_i)$
        $besti \leftarrow i$
    else
      if LLH_i is "deemed bad" (see text)
        $backoff_i \leftarrow 2 * backoff_i$
          choose $tabu_i$ uniformly at random in $\{0,1,2,\ldots, backoff_i\}$
  for each low level heuristic $LLH_i \notin Eligible$
      $tabu_i \leftarrow tabu_i - 1$
  if $best\Delta > 0$
    $S \leftarrow apply(S,LLH_{besti})$
Terminate:
  for each low level heuristic LLH_i
    if $\Delta(S,LLH_i) > 0$
      $S \leftarrow apply(S,LLH_{besti})$
      go to **Iterate**

Fig. 2. The Binary Exponential Back Off (BEBO) hyperheuristic

4 Computational Experiments

We compare several hyperheuristic methods (*rVNS*, *HyperGreedy*, *HyperGreedyMore*, *HyperRandom*, *HyperRandomMore*, 9 BEBO *"Best x"*, 7 BEBO *"Prop x"*, and *15* "standard" *Tabu Hyperheuristics*) ten times on five different problem instances and averaged the 50 results. The five problem instances require the scheduling of 400 tasks using 100 resources over one day using five different skills. Tasks require between one and three skills and resources possess between one and five skills. The problems reflect realistic problems Trimble have identified and are generated using the problem generator used in [21]. The complexity of dealing with these real-world problem instances mean that these experiments require over 155 CPU days to complete, so they were run in parallel on 88 cores of 22 identical 4 core 2.0 GHz Machines. Implementation was in C# .NET under Windows.

rVNS is best rVNS method taken from [3]. *HyperRandom* and *HyperRandomMore* are the random hyperheuristics from [3] with the latter including the additional low level heuristics introduced in this paper. *HyperGreedy* and *HyperGreedyMore* are the greedy hyperheuristics from [3] with the latter including the additional low level heuristics introduced in this paper. *HyperGreedyMore* will be the benchmark for all the tests as this is the most CPU-intensive approach and produces the best result.

The *BEBO* hyperheuristics are described in the above section. We try both of the proposed back-off methods with various sets of parameters. We also compare with a "standard" *Tabu* hyperheuristic, setting the tabu tenure to *t=5, 7, 10, 25, 50* each time a low level heuristic is tried and fails to give an improvement. We also investigate different methods of deciding which LLHs to make tabu. *TabuBest y t=x* signifies that all but the top *y* improving low level heuristics will not be made tabu with tenure *x* at each iteration. This is similar to the method used in [10] however we experiment with larger tabu tenures as we use more low level heuristics. We also investigated making all non improving low level heuristics tabu however the results for these were very poor in terms of CPU time (as nearly all of the low level heuristics make a positive improvement early in the search even if this improvement is very small) and these results are not reported below. In addition to these fixed tenures, we try random tenures as used in [20]: *rTabu Best y t=x* is similar to *Tabu Best y t=x*, but with a random tenure between 0 and *x* each time a low level heuristic is made tabu.

The results are presented in Table 2. We see that the availability of additional LLHs significantly improves the performance of *HyperGreedyMore* relative to *HyperGreedy* and *HyperRandomMore* relative to *HyperRandom.* The best BEBO method in terms of fitness is *BEBO Best 20*, which is also one of the slowest since it maintains a relatively large set of *Eligible* low level heuristics. Even so, the worst performing hyperheuristic *BEBO Best 1* got a result 97.64% as good as *HyperGreedyMore* in only 18.52% of the CPU time. Unsurprisingly, the size of the set of heuristics considered bad appears to determine the trade-off between solution quality and time reduction, although the reduction in solution quality is modest given the large reduction in CPU time. In all cases, randomisation improved tabu search, further supporting previous work [18]. The fastest "standard" tabu hyperheuristic *rTabu* was not as quick as the fastest BEBO method and also resulted in poorer quality solutions. The best *rTabu* hyperheuristic (in terms of solution quality) took much more CPU time than BEBO methods which gave similar quality.

Table 2. Fitness and Time of the tested hyperheuristic

Method	Average Fitness	Average Time (s)	Fitness % of *HyperGreedyMore*	Time % of *HyperGreedyMore*
rVNS	21974.9	19.3	88.21%	0.25%
HyperRandom	19326.7	18.5	77.58%	0.24%
HyperGreedy	22084.0	233.0	88.65%	2.98%
HyperRandomMore	20553.8	176.3	82.51%	2.26%
HyperGreedyMore	24911.3	7807.2	100.00%	100.00%
BEBO Best 1	24324.6	1446.1	97.64%	**18.52%**
BEBO Best 2	24588.6	1775.4	98.70%	22.74%
BEBO Best 3	24774.3	2043.6	99.45%	26.18%
BEBO Best 4	24693.9	2077.5	99.13%	26.61%
BEBO Best 5	24734.6	2209.5	99.29%	28.30%
BEBO Best 10	24782.8	2572.5	99.48%	32.95%
BEBO Best 15	24869.3	2825.4	99.83%	36.19%
BEBO Best 20	24993.8	3150.9	**100.33%**	40.36%
BEBO Best 25	24927.1	3261.1	100.06%	41.77%
BEBO Prop 0.01%	24756.3	2341.1	99.38%	29.99%
BEBO Prop 0.05%	24737.2	2260.3	99.30%	28.95%
BEBO Prop 0.1%	24670.5	2295.6	99.03%	29.40%
BEBO Prop 0.5%	24753.3	2434.1	99.37%	31.18%
BEBO Prop 1%	24685.3	2278.6	99.09%	29.19%
BEBO Prop 5%	24543.7	2453.7	98.52%	31.43%
BEBO Prop 10%	24429.5	2507.3	98.07%	32.12%
rTabu Best 5 t=5	24420.5	4818.5	98.03%	61.72%
rTabu Best 5 t=7	24307.0	4151.0	97.57%	53.17%
rTabu Best 5 t=10	24121.2	3572.1	96.83%	45.75%
rTabu Best 5 t=25	23976.3	2663.8	96.25%	34.12%
rTabu Best 5 t=50	22872.2	2271.5	91.81%	29.10%
rTabu Best 10 t=5	24415.1	5305.6	98.01%	67.96%
rTabu Best 10 t=7	24459.0	4834.1	**98.18%**	61.92%
rTabu Best 10 t=10	24235.2	4251.3	97.29%	54.45%
rTabu Best 10 t=25	24149.5	3448.7	96.94%	44.17%
rTabu Best 10 t=50	24014.8	3047.5	96.40%	39.03%
Tabu Best 5 t=5	18104.2	2641.1	72.67%	33.83%
Tabu Best 5 t=7	19141.0	2577.4	76.84%	33.01%
Tabu Best 5 t=10	19364.5	2419.1	77.73%	30.99%
Tabu Best 5 t=25	18714.3	1968.9	75.12%	25.22%
Tabu Best 5 t=50	19139.1	1784.8	76.83%	**22.86%**
Tabu Best 10 t=5	20085.0	3443.8	80.63%	44.11%
Tabu Best 10 t=7	19246.5	3028.4	77.26%	38.79%
Tabu Best 10 t=10	19648.4	2675.3	78.87%	34.27%
Tabu Best 10 t=25	20143.4	2534.1	80.86%	32.46%
Tabu Best 10 t=50	20087.3	2351.1	80.64%	30.12%

To see how the time reduction scales with the number of low level heuristics, *HyperGreedy* and *Best 10* experiments were repeated 10 times with randomly chosen subsets of low level heuristics. The results, shown in fig. 3, compare *BEBO Best 10* with *HyperGreedy* with different numbers of low level heuristics. We can see that fitness of the two approaches remains very close (none of the results for *BEBO Best 10* dropped below 99.3% of the *HyperGreedy* fitness). As the number of low level heuristics decreases, the time savings for *BEBO Best 10* reduce. When 80 or 90% of

heuristics have been removed, results are erratic (since the small set of low level heuristics is not guaranteed to be rich enough to yield good results). In this case *BEBO Best 10* deals better with the erratic nature and produces better fitness that *HyperGreedy* on average.

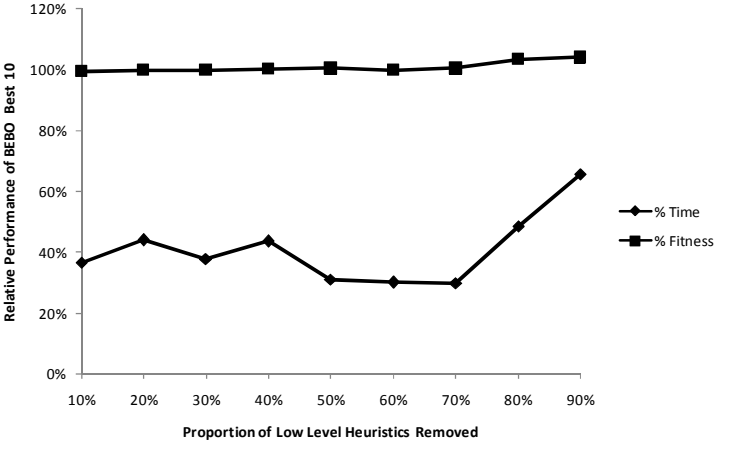


Fig. 3. Graph showing the relative performance of *BEBO Best 10* to *HyperGreedyMore* with different neighborhood sizes

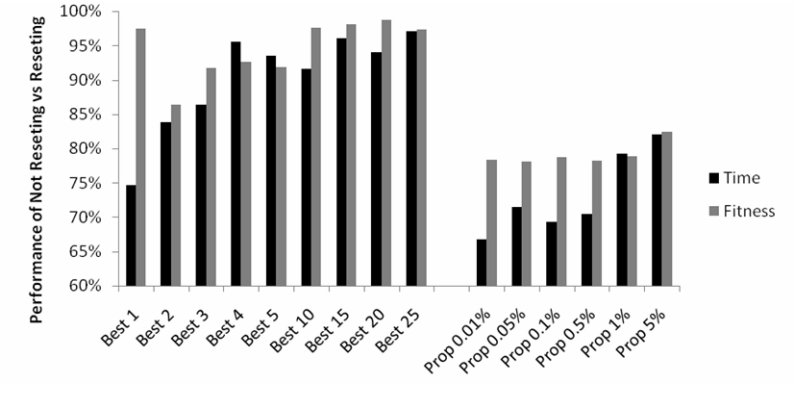


Fig. 4. Graph showing the relative performance of each hyperheuristic without resetting compared to the performance of using the same hyperheuristic with resetting

When the search finds no positive moves in the *Eligible* set of low level heuristics the tabu tenures of the low level heuristics are reset and all low level heuristics are tried again. This is potentially a waste of time, if after resetting and trying all the low level heuristics again no improvement is found. To see the impact this has on fitness and CPU time, the methods were tried with and without the reset. Fig 4 shows the relative performance of each hyperheuristic without resetting compared to the performance of using the same hyperheuristic with resetting in terms of time and fitness.

Here 100% would denote that not resetting was equally as good as resetting – since all values are below 100%, and some are well below 100% these results indicate that resetting is essential, and its effects on the time used are modest.

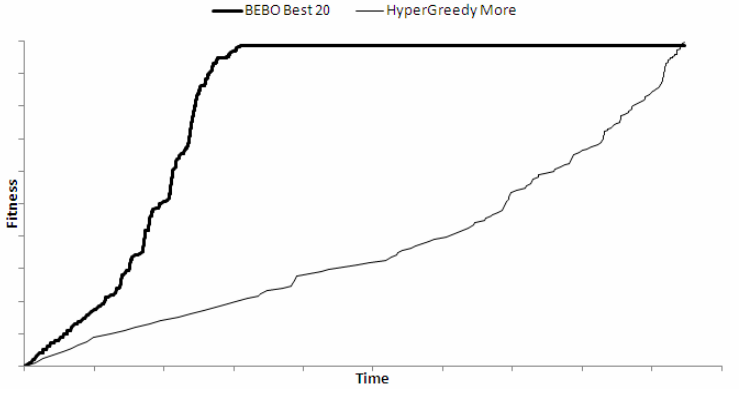


Fig. 5. Graph showing the fitness against CPU time of the *BEBO Best 20* and *HyperGreedy-More* heuristics

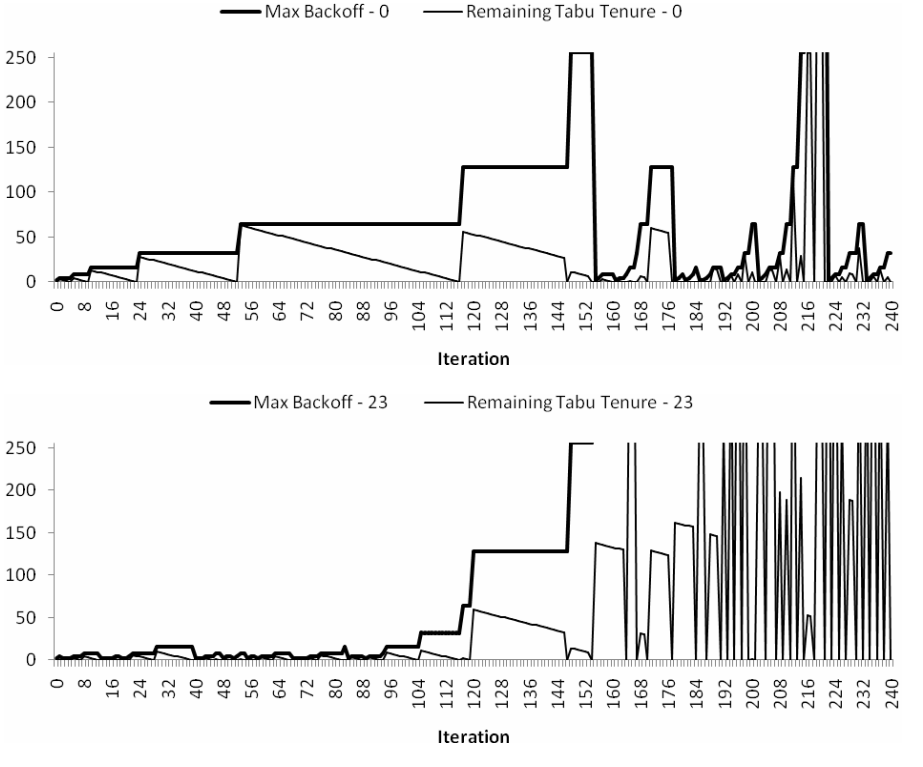


Fig. 6. Graph showing the Tabu Tenure and Back off of a two different low level heuristics over the iterations of the search

Fig. 5 shows the difference in the evolution of fitness through time for *BEBO Best 20* and *HyperGreedy More*. The curve for BEBO Best 20 stays well ahead of the corresponding curve for *HyperGreedy More*, until very late in the search process. There is good evidence here of the efficiency savings resulting from exponential back-off, which waste far less function calls to poor LLHs.

Fig. 6 shows how the BEBO tabu tenures change over the search for two typical low level heuristics with very different properties. The plot shows $Backoff_i$ and $Tabu_i$ at each iteration of a *BEBO Best 20* run for LLH_0 and LLH_{23}. LLH_0 is used towards the end of the search and as we can see, BEBO is very efficient only trying the low level heuristic about 7 times in the first 150 iterations. Later in the search LLH_0 maintains a low back-off value, and produces useful solution improvements. LLH_{23} performs well at the beginning of the search but after about 96 iteration it no longer serves a useful purpose. Hence fig. 6 clearly shows the back off value and the tabu tenure increasing so that little time is wasted with the low level heuristic later in the search. These behaviours are typical of the BEBO approach, as is the (less interesting, and very common) behaviour where a consistently poor heuristic is called only rarely right throughout the search.

5 Conclusions

This paper investigates the use of Binary Exponential Back-Off (as used in computer and telecommunications networks) to set the tabu tenure for low level heuristics in a hyperheuristic framework. The approach has been empirically tested on a complex, real-world workforce scheduling problem. The results have shown the potential of the new method to generate good solutions much more quickly than exhaustive (greedy) approaches and standard tabu approaches. In particular, the benefits of the approach increase with increasing neighbourhood size (i.e. with an increased number of low level heuristics). Binary Exponential Back-Off is able to produce results very close to a highly CPU intensive greedy heuristic which investigates a much larger set of low level heuristics at each iteration, in terms of fitness, and is able to do so in a fraction of the CPU time. BEBO performs much better than "standard" fixed and random tabu tenure hyperheuristics. Different method for deciding which heuristics to back off were tested, since adjusting the number of low level heuristics backed off determines the trade-off between CPU time used and solution quality. We have shown that modifying the number of low level heuristics backed off may be used to adjust the search and trade off time available against solution quality.

In principle our exponential back-off methods could be used in any tabu implementation with large neighbourhoods, which provides a promising possible direction for further research.

References

1. Chakhlevitch, K., Cowling, P.I.: Hyperheuristics: Recent Developments. Metaheuristics. In: Cotta, C., Sevaux, M., Sorensen, K. (eds.) Adaptive and Multilevel. Studies in Computational Intelligence, vol. 136, pp. 3–29. Springer, Heidelberg (2008)
2. Burke, E., et al.: Hyper-Heuristics: An Emerging Direction in Modern Search Technology. In: Handbook of Metaheuristics, pp. 457–474. Springer, Heidelberg (2003)

3. Remde, S., Cowling, P., Dahal, K., Colledge, N.: Exact/Heuristic Hybrids using rVNS and Hyperheuristics for Workforce Scheduling. In: Proc. Evolutionary Computation in Combinatorial Optimization. LNCS, vol. 4464, pp. 188–197. Springer, Heidelberg (2007)
4. Chakhlevitch, K., Cowling, P.I.: Choosing the Fittest Subset of Low Level Heuristics in a Hyperheuristic Framework. In: Raidl, G.R., Gottlieb, J. (eds.) EvoCOP 2005. LNCS, vol. 3448, pp. 23–33. Springer, Heidelberg (2005)
5. Cowling, P.I., Chakhlevitch, K.: Hyperheuristic for managing a large collection of low level heuristics to schedule personnel. In: Proc. of the 2003 IEEE Congress on Evolutionary Computation (CEC 2003), pp. 1214–1221. IEEE Press, Los Alamitos (2003)
6. Kolisch, R., Hartmann, S.: Experimental Investigations of Heuristics for RCPSP: An Update. European Journal Of Operational Research 174(1), 23–37 (2006)
7. Pinedo, M., Chao, X.: Operations scheduling with applications in manufacturing and services. McGraw-Hill, New York (1999)
8. Metcalfe, R.M., Boggs, D.R.: Ethernet: Distributed Packet Switching for Local Computer Networks. Coms. of the ACM 19(5), 395–404 (1976)
9. Kwak, B.J., Song, N.O., Miller, L.E.: Performance Analysis of Exponential Backoff. IEE-ACM Transactions on Networking 13(2), 343–355 (2005)
10. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. Journal of Heuristics 9(6), 451–470 (2003)
11. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers & Operational Research 24(11), 1097–1100 (1997)
12. Fang, H., Ross, P., Corne, D.: A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. In: 11th European Conf. on Artificial Intelligence (1994)
13. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 176–190. Springer, Heidelberg (2001)
14. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. Applied Optimization 86, 523–544 (2003)
15. Bai, R., Kendall, G.: An Investigation of Automated Planograms Using a Simulated Annealing Based Hyper-heuristics. In: Proc. of The Fifth Metaheuristics Int. Conference (MIC 2003), Kyoto International Conference Hall, Kyoto, Japan (August 2003)
16. Kendal, G., Han, L., Cowling, P.: An Investigation of a Hyperheuristic Genetic Algorithm Applied to a Trainer Scheduling Problem. In: CEC, pp. 1185–1190. IEEE Press, Los Alamitos (2002)
17. Laguna, M., Marti, R., Campos, V.: Intensification and Diversification with elite tabu search solutions for the linear ordering problem. Computers & Operations Research 26(12), 1217–1230 (1999)
18. Rolland, E., Schilling, D.A., Current, J.R.: An efficient tabu search procedure for the p-median problem. European Journal of Operational Research 96, 329–342 (1996)
19. Kendal, G., Modh Hussain, N.: An investigation of a tabu search based hyper heuristic for examination timetabling. In: Proc. MISTA, pp. 309–328. Springer, Heidelberg (2005)
20. Kendall, G., Mohd Hussain, N.: Tabu search hyperheuristic approach to the examination timetabling problem at the University of Technology MARA. In: Proc. of the 5th Int. Conf. on Practice and Theory of Automated Timetabling, pp. 199–217 (2004)
21. Cowling, P., Colledge, N., Dahal, K., Remde, S.: The Trade Off between Diversity and Quality for Multi-objective Workforce Scheduling. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 13–24. Springer, Heidelberg (2006)

Diversity Control and Multi-Parent Recombination for Evolutionary Graph Coloring Algorithms

Daniel Cosmin Porumbel[1], Jin-Kao Hao[1], and Pascale Kuntz[2]

[1] LERIA, Université d'Angers, 2 Bd Lavoisier, 49045 Angers, France
[2] LINA, Polytech'Nantes, rue Christian Pauc, 44306 Nantes, France

Abstract. We present a hybrid evolutionary algorithm for the graph coloring problem (Evocol). Evocol is based on two simple-but-effective ideas. First, we use an enhanced crossover that collects the *best* color classes out of *more than two* parents; the best color classes are selected using a ranking based on both class fitness and class size. We also introduce a simple method of using distances to assure the population diversity: at each operation that inserts an individual into the population or that eliminates an individual from the population, Evocol tries to maintain the distances between the remaining individuals as large as possible. The results of Evocol match the best-known results from the literature on almost all difficult DIMACS instances (a new solution is also reported for a very large graph). Evocol obtains these performances with a success rate of at least 50%.

1 Introduction

The graph coloring problem is one of the first problems proved to be NP-complete in the early 70's. It has a very simple formulation: label the vertices of a graph with the minimum number of colors such that no adjacent vertices share the same color. Many other problems and practical applications can be reduced to graph coloring: scheduling and timetabling problems, frequency assignment in mobile networks, register allocation in compilers, air traffic management, to name just a few.

The second DIMACS Implementation Challenge [13] introduced a large set of graphs for benchmarking coloring algorithms that has been extensively used since 1996. The most popular coloring algorithms belong to three main solution approaches: (i) sequential construction (very fast methods but not particularly efficient), (ii) local search methods (many different techniques [1, 2, 9, 11, 12, 15] can be found in the literature), and (iii) the population-based evolutionary methods that traditionally dominate the tables with the best results [4, 6, 7, 8, 14, 15].

We present in this work-in-progress paper a new hybrid evolutionary algorithm (Evocol) that makes contributions in two directions: the recombination operator (Section 3) and the population diversity control (Section 4). The recombination

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 121–132, 2009.

operator picks up the best color classes out of many (and *surely diverse*) parents; the classes are ranked according to their fitness and size. The diversity control uses a set-theoretic distance measure between colorings to strictly control which individuals are inserted or deleted from the population. As such, it keeps the distances between the population individuals as large as possible and it permanently guarantees global diversity. The resulting algorithm is quite simple and lightweight, as it incorporates no important additional elements. However, it obtains very competitive results (Section 5), with plenty of room for further improvement.

2 Problem Statement and Generic Hybrid Algorithm

Given a graph $G(V, E)$, the graph coloring problem requires finding the minimal number of colors χ (the chromatic number) such that there exists a vertex coloring (using χ colors) with no adjacent vertices of the same color (with no conflicts). One could determine the chromatic number by iteratively solving the following k-coloring problem: given a number of colors $k \geq \chi$, find a k-coloring (a coloring using k colors) without conflicts. This method starts with a sufficiently large k (e.g. $k = |V|$ is surely enough) and iteratively decrements k each time the corresponding k–coloring problem is solved. The k–coloring problem becomes increasingly difficult until the algorithm can no longer solve it.

A common coloring representation consists in a function $I : V \rightarrow \{1, 2, \cdots, k\}$, usually encoded as an array of colors $I = [I(1), I(2), \ldots, I(|V|)]$. While we also encoded this representation in our programs, it is very useful to interpret a coloring as a vertex set partition.

Definition 1. *(Partition representation) A k-coloring I of V is denoted by a partition $\{I^1, I^2, \ldots, I^k\}$ of V—i.e. a set of k disjoint subsets (color classes) of V covering V such that $\forall x \in V$, $x \in I^i \Leftrightarrow I(x) = i$.*

We say that I is an individual (a candidate solution for a k-coloring problem); I^i is the color class i induced by the coloring I, i.e. the set of vertices having color i in I. This partition based definition is particularly useful to avoid symmetry issues arising from the color based encoding. As such, it is used for the crossover operator (see Section 3) and also to define a meaningful distance between colorings (see Section 4.1). Moreover, I is a legal or conflict-free coloring (a solution) if and only no color class of I contains adjacent vertices.

Definition 2. *(Conflict number and fitness function) Given an individual I, we call conflict (or conflicting edge) any edge having both ends in the same class. The set of conflicts is denoted by $C(I)$ and the number of conflicts (i.e. $|C(I)|$—also referred to as the conflict number of I) is the fitness function $f(I)$. A conflicting vertex is a vertex $v \in V$, for which there exists an edge $\{v, u\}$ in $C(I)$.*

In this paper, we deal with the k-coloring problem as an optimization problem: given a pair (G, k), our algorithm searches the search space (call it Ω) for a k-coloring I^* such that $f(I^*) = Minf(I)$; if $f(I^*) = 0$, a legal coloring is found.

2.1 General Design of the Evolutionary Algorithm

The generic algorithmic template of Evocol shares some basic ideas with other evolutionary algorithms from the graph coloring literature [2, 4, 6, 7, 8, 14, 15], but we enriched the traditional template with new features: (i) the possibility to combine $n \geq 2$ parents to generate an offspring, (ii) the possibility to reject an offspring if it does not fit some diversity criteria (with respect to the existing individuals). To be specific, the skeleton of Evocol is presented in Algorithm 1. The *stopping condition* is either to find a legal coloring or to reach a predefined time limit. In our experiments, most of the CPU time is spent by the local search operator. Depending on the graph, a time limit is equivalent to a limit on the number of local search iterations—which is a constant multiple of the number of crossovers (step 2.A.2 and 2.A.3 are always performed together).

Algorithm 1. Evocol: Evolutionary Hybrid Algorithm for Graph Coloring

```
Input: the search space Ω
Result: the best configuration ever found
1. Initialize (randomly) parent population Pop = (I_1, I_2, ..., I_|Pop|)
2. While a stopping condition is not met
   A. For i = 1 to p (p = number of offspring)
      Repeat
        1. (I_1, I_2, ..., I_n)=SelectParents(Pop,n) /* n ≥ 2 */
        2. O_i =Crossover(I_1, I_2, ... I_n)
        3. O_i =LocalSearch(O_i, maxIter)
      Until AcceptOffspring(Pop, O_i)
   B. Pop=UpdatePopulation(Pop, O_1, O_2, ..., O_p)
```

The performance of Evocol closely depends on several *independent* components, most notably: the crossover operator (function `Crossover`), the population management (functions `AcceptOffspring` and `UpdatePopulation`), and the local search algorithm. The `SelectParents` and `LocalSearch` procedures are quite classical and we only briefly describe them.

The parent selection simply consists in choosing n different individuals uniformly at random from the population. Such a selection favors diversity of the chosen parents in comparison with the roulette wheel or tournament selection that favors the selection of the fittest individuals.

The `LocalSearch` procedure is an improved version of a classical Tabu Search algorithm for graph coloring: Tabucol [12]. Basically, this algorithm iteratively moves from one coloring to another by modifying the color of a conflicting vertex until either a legal coloring is found, or a predefined number of iterations (i.e. $maxIter = 100000$) is reached. Each performed move (or color assignment) is marked Tabu for a number of iterations referred to as the Tabu tenure T_ℓ; in this manner, Tabucol cannot re-perform a move that was already performed during the last T_ℓ iterations.

Numerous versions of this algorithm can be found in the literature, and one of the most notable differences between them lies in the way they set the Tabu tenure. In our case, $T_\ell = \alpha * f(C) + random(A) + \left\lfloor \frac{M}{M_{\max}} \right\rfloor$, where α, A and $M_{\max}$ are predefined parameters, M is number of the last consecutive moves that kept the fitness function constant, $random(A)$ is a random integer in $[1..A]$. Concerning the parameters values, we use: $\alpha = 0.6$, $A = 10$ (as previously published in [7]), and $M_{\max} = 1000$. The last term is a new reactive component only introduced to increment T_ℓ after each series of $M_{\max}$ iterations with no fitness variation. This situation typically appears when the search process is completely blocked cycling on a plateau; an extended Tabu list can more easily trigger the search process diversification that is needed in this case.

3 New Multi-Parent Crossover

As already indicated in [4,7], effective graph coloring crossovers can be designed by considering a coloring as a partition of V (Definition 1). Here, we propose a Multi-Parent Crossover (MPX) for k-coloring that collects in the offspring the *best color classes* from several parents. To formally define the notion of "best class", each class in each parent receives a score based on two criteria: (i) the number of conflicts (generated only by the class vertices) and (ii) the size of the class. Note that, in comparison with other crossovers in the literature, MPX also takes into account the class conflict number.

The MPX operator (see Algorithm 2) actually searches (Steps 2.A and 2.B) for the largest class among those with the minimum class conflict number (i.e. minimum number of conflicting edges *in* the class). After assigning the class to the offspring (Step 2.C), it chooses the next best class and repeats. At each step, all class scores are calculated only after erasing the vertices that already received a color in the offspring (Step 2.A.1). It stops when k colors classes are assigned and a simple greedy procedure then fills any remaining unassigned vertex (Step 3).

The only risk of this crossover is to inherit most classes only from one parent, especially if there is a (very fit) parent whose classes "eclipse" the others. However, the similarity between the offspring and the parents is implicitly checked afterward by the `AcceptOffspring` procedure (see Section 4.2) that rejects the offspring if it is too similar to any existing individual.

The complexity of MPX is $O(k^2 \times n \times \overline{|I_i^j|}^2)$ where $\overline{|I_i^j|}$ is the average class size and n is the number of parents—the term $k^2 \times n$ is due to the three `For/Foreach` loops in step 2 and $\overline{|I_i^j|}^2$ is due to step 2.A.2. Since $\overline{|I_i^j|}$ is about $\frac{|V|}{k}$, this complexity is roughly equivalent to $O(|V|^2 \times n)$. In our practical case $n = 3$, and thus, the crossover takes much less time than the $maxIter = 100000 \geq |V|^2$ iterations of the local search procedure (in which, each iteration takes at least $O(|V|)$).

Notice that the authors of [5] report multi-parent crossover for the 3-coloring problem. Contrary to MPX, their crossover operates on an order-based

Algorithm 2. The multi-parent crossover MPX

Input: parents $I_1, I_2, \ldots, I_n$
Result: offspring O
1. O =empty, i.e. start with no vertex color assigned
2. For *currentColor*= 1 **To** k
A. **Foreach** parent $I_i \in \{I_1, I_2, \ldots, I_n\}$
Foreach color class I_i^j in I_i
1. Remove from I_i^j all vertices already assigned in O
2. *conflicts* = $|\{(v_1, v_2) \in I_i^j \times I_i^j \ : \ (v_1, v_2) \in E\}|$
3. *classSize* = $|I_i^j|$
4. *score*$[I_i^j]$ = *conflicts*$\times|V|$-*classSize*
B. **Set** (i^*, j^*) = $arg\min_{(i,j)} score[I_i^j]$
C. **Foreach** $v \in I_{i*}^{j^*}$
$O[v]$ =*currentColor*
3. Foreach unassigned $v \in O$
$O[v]$ =a color that generates the least number of conflicts

representation of colorings. Experimental results are reported on two small random graphs of 90 vertices.

4 Population Management

It is well known that the population diversity is a key element of an effective evolutionary algorithm [7, 16]. In fact, a low diversity constitutes a stopping condition for numerous practical algorithms—it usually indicates a premature convergence on poor solutions. By using a distance metric on the search space, Evocol *strictly* controls diversity with two mechanisms:

- It rejects a new offspring if it is too close to an existing individual of the population;
- It introduces a diversity criterion in the selection of the individuals to be eliminated (diversity-based replacement strategy).

4.1 Search Space Distance Metric

Let us first describe the distance metric on which the diversity control is based. We define the distance function between individuals I_A and I_B using the partition coloring representation (see Definition 1). As such, we view two colorings as two partitions of V and we apply the following set-theoretic partition distance (call it d): the minimum number of elements that need to be moved between classes of the first partition so that it becomes equal to the second partition. This distance was defined several times since the 60's and the currently used computation methodology was first described in the 80's (see [3], or, more recently [10]); it was also already used for graph coloring [7, 9].

The distance $d(I_A, I_B)$ is determined using the formula $d(I_A, I_B) = |V| - s(I_A, I_B)$, where s denotes the (complementary) similarity function: the maximum number of elements in I_A that do not need change their class in order to transform I_A into I_B. This similarity function reflects a structural similarity: the better the I_A classes can be mapped to the I_B classes, the higher the value of $s(I_A, I_B)$ becomes; in case of equality, this mapping is an isomorphism and $s(I_A, I_B)$ is $|V|$. Both the distance and the similarity take values between 0 and $|V|$ and this is why we usually report them in terms of percentages of $|V|$.

To compute these values, we define the $k \times k$ matrix S with elements $S_{ij} = |I_A^i \cap I_B^j|$; thus, s can be determined by solving a classical assignment problem: find a S assignment (i.e. a selection of S cells with no two cells on the same row or column) so that the sum of all selected cell values is maximized. This assignment problem is typically solved with the Hungarian algorithm of complexity $O(k^3)$ in the *worst* case. However, in our practical application, there are very few situations requiring this worst-case time complexity. We did not observe any significant slow-down caused by distance computations; the most time consuming procedure is still the local search.

4.2 Offspring Reject Mechanism

As we are committed to maintaining population diversity, we insert an offspring in the population only if its distance to each existing individual is greater than a predefined threshold; denote it by R. Consequently, if an offspring O is situated at a distance of less than R from an individual I, the `AcceptOffspring` procedure (see Algorithm 1) either (i) rejects O or, (ii) directly replaces I with O if $f(O) \leq f(I)$ (i.e. if O is better than I). However, in both cases, a new offspring is generated by starting with the parent selection—see the `Repeat-Until` loop in step 2.A of Algorithm 1.

The only delicate issue in the application of this simple mechanism is to determine a suitable R value. Let us denote by $S_R(I)$ the closed sphere of radius R centered at I, i.e. the set of individuals $I' \in \Omega$ such that $d(I, I') \leq R$. If I is a local minimum, an appropriate value of R should imply that all other local minima from $S_R(I)$ share important color classes with I, i.e. they bring no new information into the population (or they are structurally related to I). We have to determine the maximum value of R such that all local minima, that are structurally *unrelated* to I, are situated outside $S_R(I)$.

Since all individuals in the population are local minima obtained with Tabu Search, we determine R from an analysis of its exploration path. Consider this classical scenario: start from an initial local minima I_0, and let Tabu Search visit a sequence of neighboring colorings as usually; we denote by I_0, I_1 ,$I_2, \ldots I_N$ all visited individuals satisfying $f(I_i) \leq f(I_0)$ $(\forall i \in [1..N])$. After recording all these individuals up to $N = 40000$, we computed the distance for each pair (I_i, I_j) with $1 \leq i, j \leq N$ and we constructed a histogram to show the number of occurrences of each distance value.

This histogram directly showed that the distribution of the distance value is bimodal, with numerous occurrences of small values (around $5\%|V|$) and of some

much larger values. This provides evidence that the $I_i's$ are arranged in distant groups of close points (clusters); the large distances correspond to inter-cluster distances and the small ones to intra-cluster distances. If we denote a "cluster diameter" by C_d, we can say that C_d varies from $7\%|V|$ to $10\%|V|$ depending on the graph, such that: (i) there are numerous pairs (i, j) such that $d(I_i, I_j) < C_d$, (ii) there are very few (less than 1%) pairs (i, j) such that $C_d < d(I_i, I_j) < 2C_d$ and, (iii) there are numerous occurrences of some larger distance values.

To determine a good value of R, it is enough to note that any two local minima situated at a distance of more than $10\%|V|$ (approximately the highest possible C_d value) are not in the same cluster—because (ideally) they have some different essential color classes. We assume that this observation holds on all sequences of colorings visited by Tabu Search and we set the value of R to $10\%|V|$ for all subsequent runs.

4.3 Diversity-Based Replacement Strategy

The `UpdatePopulation` procedure determines which existing individual is eliminated for each offspring that needs to be inserted. While most previous algorithms take into account only the fitness values of the population (e.g. by replacing the least fit individual), we also take interest into the population diversity. To control diversity, this procedure encourages the elimination of individuals that are too close to some other individuals; in this manner, it gets rid of small distances in the population.

Generally speaking, the procedure (see Algorithm 3 bellow) selects two very close individuals that candidate for elimination and only the least fit of them is eliminated. The first candidate C_1 is chosen by a random function using some fitness-based guidelines (via the `AcceptCandidate` function). The second candidate C_2 is chosen by introducing the following *diversity criterion*: C_2 is the closest individual to C_1 respecting the same fitness-based guidelines as C_1.

The `AcceptCandidate` function makes a distinction between the first half of the population (the individuals with a fitness value lower than the *median*), the second half of the population and the best individuals. As such, this function always accepts a candidate C_i for elimination if C_i belongs to the second half, but it accepts C_i only with 50% probability if C_i belongs to the first half. Only the best individual is fully protected; it can never become a candidate for elimination—unless there are too many best individuals (more than half of the population) in which case *any* individual can be eliminated. As such, the role of the first half of the population is to permanently keep a sample of the best individuals ever discovered. The first half of the population stays quite stable in comparison with the second half that is changing very rapidly.

5 Experimental Results

The experimental studies are carried out on the most difficult instances from the well-known DIMACS Benchmark [13]: (i)$dsjcA.B$—classical random graphs

Algorithm 3. The replacement (elimination) function

```
Input: population Pop = (I_1, I_2, ..., I_|Pop|)
Result: the individual to be eliminated
1. Repeat
     C_1 = RandomIndividual(Pop)
   Until AcceptCandidate(C_1) (fitness-based acceptance)
2. minDist = maximum possible integer
3. Foreach I ∈ Pop − {C_1}
     If d(I, C_1) <minDist
        If AcceptCandidate(I)
          • minDist = d(I, C_1)
          • C_2 = I
4. If f(C_1) < f(C_2)
     Return C_2
   Else
     Return C_1
```

with unknown chromatic numbers (A denotes $|V|$ and B denotes the density), (ii)$le450.25c$ and $le450.25d$—the most difficult "Leighton graphs" with $|V| = 450$ and $\chi = 25$ (they have at least one clique of size χ), (iii)$flat300.28$ and $flat1000.76$—the most difficult "flat" graphs with χ denoted by the last number (generated by partitioning the vertex set in χ classes, and by distributing the edges only between vertices of different classes), (iv) $r1000.1$, $r1000.5$ and $dsjr500.5$—random geometric graphs, generated by picking points (vertices) uniformly at random in the square and by adding edges between each two vertices situated within a certain distance, (v) $C2000.5$—a very large graph (2.000 vertices and 1.000.000 edges).

We report in Table 1 the general results[1] obtained by Evocol with the following settings: $|Pop| = 15$ (population size), $n = 3$ (number of parents), $p = 3$ (number of offspring constructed each generation), $maxIter = 100000$ (the maximum number of iterations of the Tabu Search procedure), $R = 10\%|V|$ (the sphere radius, the minimum imposed distance between two individuals in the population). For each important value of k, this table reports the success rate over 10 independent runs (Column 3), the average number of generations required to solve each problem (Column 4), the average number of crossovers (Column 5) and the average CPU time in seconds (last column). The reported times are measured on a 2.8GHz Xeon processor using the C++ programming language compiled with the $-O3$ optimization option (gcc version 4.1.2 under Linux).

The total number of local search iterations is in close relation with the number of crossovers because the local search procedure (with $maxIter = 100000$) is applied once for each crossover. The algorithm performs at least $p = 3$ crossovers per generation, but it can perform many more (e.g. for the $dsjc500.1$ instance)

[1] The best colorings reported in this paper are publicly available at: www.info.univ-angers.fr/pub/porumbel/graphs/evocop/

Table 1. The results of Evocol with a CPU time limit of 5 hours. The algorithm finds most of the best known solutions with a success rate of more than 50% (see Column 3)—the minimal value of k for which a solution was ever reported in the literature (i.e. k^*) is given in the parentheses of Column 1.

Graph (best known k)	k	successes/runs	generations	crossovers	time[s]
$dsjc250.5$ ($K^* = 28$)	28	10/10	9	33	20
$dsjc500.1$ ($K^* = 12$)	12	10/10	96	1573	928
$dsjc500.5$ ($K^* = 48$)	48	10/10	258	827	1428
$dsjc500.9$ ($K^* = 126$)	126	10/10	222	985	1804
$dsjc1000.1$ ($K^* = 20$)	20	8/10	301	3350	4688
$dsjc1000.5$ ($K^* = 83$)	84	10/10	274	839	4729
	83	6/10	798	2722	13251
$dsjc1000.9$ ($K^* = 224$)	225	10/10	268	857	4328
	224	8/10	500	1702	8487
$le450.25c$ ($K^* = 25$)	26	10/10	1	3	2
	25	7/10	1102	8232	5690
$le450.25d$ ($K^* = 25$)	26	10/10	1	3	2
	25	5/10	650	4479	3152
$flat300.28.0$ ($K^* = 28$)	31	10/10	16	56	44
$flat1000.76.0$ ($K^* = 82$)	83	10/10	261	802	4539
	82	5/10	583	1940	9956
$r1000.1c$ ($K^* = 98$)	98	7/10	80	1939	4591
$r1000.5$ ($K^* = 234$)	248	10/10	326	1088	5368
	247	8/10	524	1836	8698
	246	7/10	448	1417	7497
	245	3/10	682	2242	11049
$dsjr500.5$ ($K^* = 122$)	125	10/10	265	1207	1764
	124	6/10	612	3113	4508
$C2000.5$ ($K^* = 153$)	152	5/5	380	1163	27262[a]
	151	4/5	433	1368	32520[a]

[a] Only for this very large graph, we used an exceptional time limit of 10 hours.

if many offspring are rejected by the `AcceptOffspring` procedure—see more discussions in the next section.

6 Discussion

In this section, we investigate the algorithm evolution, placing a special emphasis on the number of parents (in the recombination) and on the diversity control. Figure 1 compares the *running profile* of Evocol (i.e. the graph of the function $t \mapsto f_*(t)$, where t is the time and $f_*(t)$ is the best known fitness value at time t) for different values of the number of parents n. We first notice that the two-parent recombination ($n = 2$) always gives poor results in comparison with any value $n > 2$. This confirms that the multi-parent recombination has more potential; however, it seems more difficult to determine which is the exact optimum number of parents. We set $n = 3$ in this paper because this is the most stable choice: it always produces reasonable results on all graphs—the choice $n = 7$, even if it seems surprisingly competitive on some random instances, has great difficulties in solving the Leighton graphs.

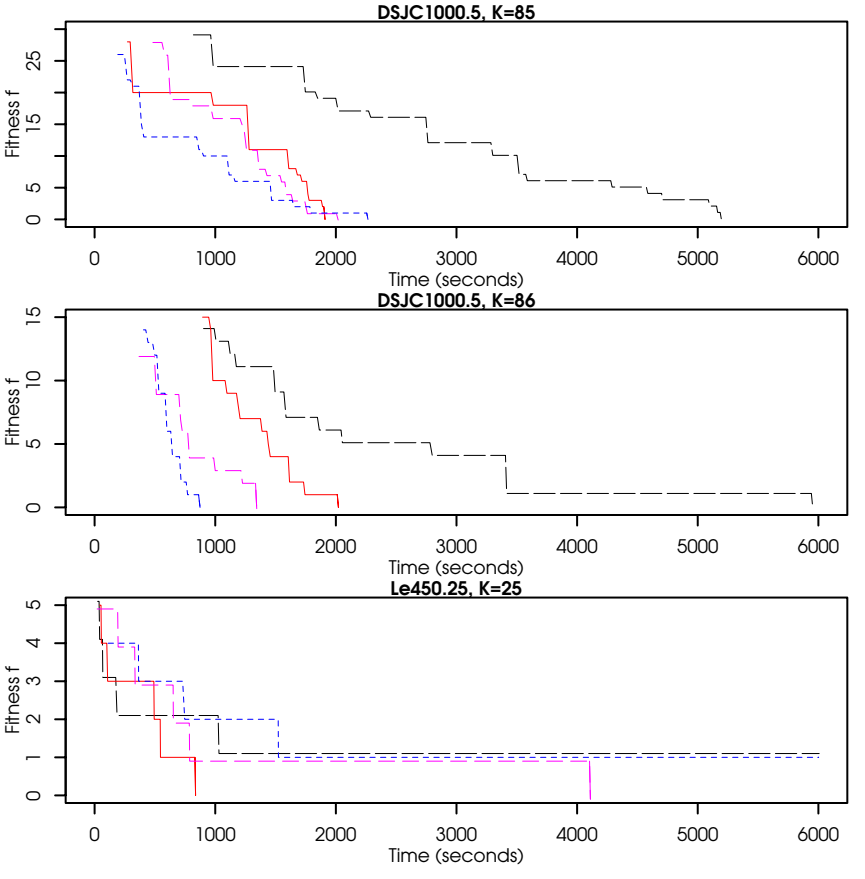


Fig. 1. The running profile (i.e. the evolution of the fitness of the best individual in the population) for several values of the number of parents: 3 parents (red, continuous line), 5 parents (magenta, with long dashes), 7 parents (blue, with normal dashes), 2 parents (black, with very long dashes)

The population management also plays a very important role in the evolution of the algorithm. In all situations from Figure 1 where the 7-parent crossover operator is not effective, we observed that the `AcceptOffspring` procedure rejects a very large proportion of the offspring. Table 1 shows important information on this issue: if we denote by g the number of generations and by c the number of crossovers, the number of rejected offspring is $c - 3g$. As such, the probability to reject an offspring can vary from 0 (e.g. for $G = flat1000.76$ and $k = 83$, we obtain $\frac{c-3g}{c} = \frac{802-3\times261}{802} \approx 0.02$) to 90% (e.g. for $G = r1000.1$ $\frac{c-3g}{c} = \frac{1939-3\times80}{1939} \approx 0.88$). In the cases where the offspring rejection rate is high, the population management accounts for the most important performance gain.

7 Conclusions

We described a new hybrid evolutionary algorithm (Evocol) that distinguishes itself from other population-based heuristics by introducing a strict population diversity control and by employing a multi-parent recombination operator (MPX). Compared with six state-of-the-art algorithms from the literature (see Table 2), the results of Evocol are very encouraging. Evocol finds most of the best-known colorings with at least 50% success rate (see also Table 1, column 3).

Table 2. Comparison of the best values of k for which a legal coloring is found by Evocol (Column 3) and by the best algorithms (Columns 4-9). Column 2 reports the chromatic number (? if unknown) and the best k for which a solution was ever reported.

Graph	χ, k^*	*Evocol*	VSS [11] 2008	PCol [1] 2008	ACol [8] 2008	MOR [15] 1993	GH [7] 1999	MMT [14] 2008
$dsjc250.5$	?, 28	*28*	–	–	28	28	28	28
$dsjc500.1$	?, 12	*12*	12	12	12	12	–	12
$dsjc500.5$	?, 48	*48*	48	49	48	49	48	48
$dsjc500.9$	?, 126	*126*	127	126	126	126	–	127
$dsjc1000.1$	?, 20	*20*	20	20	20	21	20	20
$dsjc1000.5$	?, 83	*83*	87	88	84	88	83	83
$dsjc1000.9$	?, 224	*224*	224	225	224	226	224	225
$le450.25c$	25, 25	*25*	26	25	26	25	26	25
$le450.25d$	25, 25	*25*	26	25	26	25	26	25
$flat300.28$	28, 28	*31*	28	28	31	31	31	31
$flat1000.76$	76, 82	*82*	86	87	84	89	83	82
$r1000.1c$	?, 98	*98*	–	98	–	98	–	98
$r1000.5$	?, 234	*245*	–	247	–	241	–	234
$dsjr500.5$	?, 122	*124*	125	125	125	123	–	122
$C2000.5$	?, 153[a]	*151*	–	–	–	165	–	–

[a] This graph was colored with $k = 153$ [6] by first removing several independent sets.

Indeed, for 11 out of the 15 difficult graphs, Evocol matches the previously best results: only for 3 DIMACS instances the results of Evocol are worse. For the largest graph $C2000.5$, it is remarkable that Evocol manages to find a 151-coloring (i.e. with 2 colors less than the best coloring known today) with a 4/5 success rate within 10 hours —for such a large instance, other algorithms might need several days. Note that for most unlisted DIMACS instances, all modern algorithms report the same k because these instances are not difficult; they can be easily colored by Evocol using the same number of colors k reported by most algorithms (like in the case $k = 12$ for $dsjc500.1$).

The general principles behind Evocol are quite simple and natural; moreover, its practical implementation only consists in relatively lightweight programming procedures. Nevertheless, it can quite quickly find the best known k-colorings and leaves plenty of room for further development.

Acknowledgments. This work is partially supported by the CPER project "Pôle Informatique Régional" (2000-2006) and the Régional Project MILES (2007-2009). We thank the referees for their useful suggestions and comments.

References

1. Blöchliger, I., Zufferey, N.: A graph coloring heuristic using partial solutions and a reactive tabu scheme. Computers and Operations Research 35(3), 960–975 (2008)
2. Costa, D., Hertz, A., Dubuis, C.: Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. Journal of Heuristics 1(1), 105–128 (1995)
3. Day, W.H.E.: The complexity of computing metric distances between partitions. Mathematical Social Sciences 1, 269–287 (1981)
4. Dorne, R., Hao, J.K.: A new genetic local search algorithm for graph coloring. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 745–754. Springer, Heidelberg (1998)
5. Eiben, A.E., Raué, P.E., Ruttkay, Z.: Genetic algorithms with multi-parent recombination. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 78–87. Springer, Heidelberg (1994)
6. Fleurent, C., Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. Annals of Operations Research 63(3), 437–461 (1996)
7. Galinier, P., Hao, J.K.: Hybrid Evolutionary Algorithms for Graph Coloring. Journal of Combinatorial Optimization 3(4), 379–397 (1999)
8. Galinier, P., Hertz, A., Zufferey, N.: An adaptive memory algorithm for the k-coloring problem. Discrete Applied Mathematics 156(2), 267–279 (2008)
9. Glass, C.A., Pruegel-Bennett, A.: A polynomially searchable exponential neighbourhood for graph colouring. Journal of the Operational Research Society 56(3), 324–330 (2005)
10. Gusfield, D.: Partition-distance: A problem and class of perfect graphs arising in clustering. Information Processing Letters 82(3), 159–164 (2002)
11. Hertz, A., Plumettaz, A., Zufferey, N.: Variable space search for graph coloring. Discrete Applied Mathematics 156(13), 2551–2560 (2008)
12. Hertz, A., Werra, D.: Using tabu search techniques for graph coloring. Computing 39(4), 345–351 (1987)
13. Johnson, D.S., Trick, M.: Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. DIMACS series in Discrete Mathematics and Theoretical Computer Science, vol. 26. American Mathematical Society, Providence (1996)
14. Malaguti, E., Monaci, M., Toth, P.: A Metaheuristic Approach for the Vertex Coloring Problem. INFORMS Journal on Computing 20(2), 302 (2008)
15. Morgenstern, C.: Distributed coloration neighborhood search. In: [13], pp. 335–358
16. Sörensen, K., Sevaux, M.: MA—PM: Memetic algorithms with population management. Computers and Operations Research 33(5), 1214–1225 (2006)

A Genetic Algorithm for Net Present Value Maximization for Resource Constrained Projects

Mario Vanhoucke[1,2]

[1] Ghent University, Faculty of Economics and Business Administration,
Tweekerkenstraat 2, 9000 Ghent, Belgium
[2] Vlerick Leuven Gent Management School, Operations & Technology Management Centre,
Reep 1, 9000 Ghent, Belgium
mario.vanhoucke@ugent.be

Abstract. In this paper, we present a new genetic algorithm for the resource-constrained project scheduling problem with discounted cash flows and investigate the trade-off between a project's net present value and its corresponding makespan. We consider a problem formulation where the pre-specified project deadline is not set as a hard constraint, but rather as a soft constraint that can be violated against a certain penalty cost. The genetic algorithm creates children from parents taken from three different populations, each containing relevant information about the (positive or negative) activity cash flows. We have tested various parent selection methods based on four crossover operators taken from literature and present extensive computational results.

1 Introduction

Since the introduction of cash flows in project scheduling problems by [16], the maximization of the net present value (npv) has gained increasing attention throughout the literature. The majority of the contributions assume a completely deterministic project setting, in which all relevant problem data (project activities, activity durations, the various cash flows, etc.) are assumed to be known. Numerous efforts aim at providing exact or heuristic solutions to the project scheduling problem under various types of resource constraints, using a rich variety of assumptions with respect to network representation (activity-on-the-node versus activity-on-the-arc), cash flows patterns (positive and/or negative cash flows; event-oriented or activity-based; time-dependent and –independent cash flows; single-mode versus multi-mode formulations; etc…), and resource constraints (capital constrained; different resource types; materials considerations; time/cost trade-offs). In addition, a number of efforts focus on the simultaneous determination of both the amount and timing of payments, known as the so-called payment scheduling problem. For a recent extensive review of the literature and a categorization of the solution procedures, we refer the reader to [12].

This paper presents a genetic algorithm to solve the single-mode resource-constrained project scheduling problem with discounted cash flows (RCPSPDC), which is an extension of the basic RCPSP within the presence of renewable resources with a constant availability and where no activity pre-emption is allowed. This problem formulation aims at the construction of a resource feasible schedule subject to a

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 13–24, 2009.

pre-specified (hard or soft) project deadline while maximizing the total net present value of the (positive or negative) activity cash flows. We present a genetic algorithm and investigate the relation between a project's net present value and its corresponding total duration or makespan, by testing the algorithm on a large and very diverse set of problem instances.

In the remainder of this paper, we represent a project by an activity-on-the-node network G = (N, A) where the nodes in the set N represent the project activities and the arcs of set A the finish-start precedence relations with a time-lag of zero. The activities are numbered from a dummy start node 0 to a dummy end node $n + 1$. Each activity i has a duration d_i and its performance involves a series of cash flow payments and receipts throughout this duration. When cf_{it} denotes the pre-specified cash flow of activity i in period t of its execution, a terminal value c_i upon completion can be calculated by compounding cf_{it} to the end of the activity as $c_i = \sum_{t=1}^{d_i} cf_{it} e^{\alpha(d_i - t)}$ with α the discount rate. Since we assume that all activity cash flows occur at predefined time points during execution of the corresponding activity, we exclude more general problem formulations with, for example, progress payments, time-dependent cash flows or payments associated with events. If the non-negative integer variable s_i represents the starting time of activity i, its discounted value at the beginning of the project is $c_i e^{-\alpha(s_i + d_i)}$. Each activity requires r_{ik} units of renewable resource k which is available for the project within a_k units.

Most RCPSPDC problems have been formulated within the presence of a hard pre-specified project deadline. In this case, each project mush finish on or before this pre-specified project deadline δ_{n+1}, and there is no violation possible whatsoever. This problem formulation can be represented as $m,1|cpm,\delta_{n+1},c_j|npv$ following the classification scheme of [7] or as $PS|prec|\sum C_j^F \beta^{C_j}$ following the classification scheme of [3] and is known to be NP-hard. In the current paper, we relax this constraint and allow a project deadline violation at a certain penalty cost. This allows the investigation of a trade-off between a project's best net present value and its corresponding makespan, which may be smaller than, equal to or larger than the pre-specified deadline. In literature, only a few research papers deal with the trade-off between cash flow optimization and project makespan. An iterative forward/backward scheduling algorithm has been proposed by [18] based on the principle of [8] which simultaneously optimizes the project duration and the net present value. This iterative forward/backward generation scheme has been extended by [14] with a local constraint based analysis which evaluates the resource and precedence constraints in determining the necessary sequence of conflicting activities fighting for the same resources. Likewise, [13] use an iterative forward/backward scheduling algorithm while optimizing a project's net present value and tardiness.

A conceptual formulation for the RCPSPDC considered in this paper can be given as follows:

$$\text{Maximize} \sum_{i=1}^{n} c_i e^{-\alpha(s_i + d_i)} - p_{n+1} \sum_{j=0}^{v-1} (1+r)^j e^{-\alpha(\delta_{n+1} + j + 1)} \tag{1}$$

Subject to

$$s_i + d_i \leq s_j \qquad \forall (i, j) \in A \tag{2}$$

$$\sum_{i \in S(t)} r_{ik} \leq a_k \qquad k = 1, \ldots, K \text{ and } t = 1, \ldots, \max(s_{n+1}, \delta_{n+1}) \tag{3}$$

$$s_{n+1} - v \leq \delta_{n+1} \tag{4}$$

where S(t) denotes the set of activities in progress in period $]t - 1, t]$.

Eq. (1) maximizes the net present value of the project. A project's total net present value consists of the discounted value of all activity cash flows, decreased with the unit penalty cost $p_{n+1} \geq 0$ which increases at a penalty rate r for each additional period that the project finishes later than the negotiated project deadline δ_{n+1}. Eq. (2) takes the finish-start precedence relations with a time-lag of zero into account. The renewable resource constraints are satisfied thanks to eq. (3). Eq. (4) imposes a soft pre-specified deadline to the project with a extra slack variable v allowing project deadline violations.

The reader should note that the objective function of Eq. (1) optimizes both the net present value and the project makespan, and can therefore be considered as a multi-objective optimisation model. Recent state-of-the-art work on multi-objective optimisation can be found in [1].

2 Genetic Algorithm

A genetic algorithm is a population based improvement heuristic in which an initial set of solutions is gradually improved during the search process. In the project scheduling literature, an improvement heuristic does not operate directly on a schedule, but on some indirect schedule representation (i.e. a schedule represented in a particular way) which can be transformed into a project schedule using a schedule generation scheme. The *representation of a schedule* can be done in various ways, among which the random key (RK) and the activity list (AL) representation are the most common ones [10]. In our genetic algorithm, we rely on a random key representation to represent each solution (i.e. schedule) as a point in a Euclidian space, so that mathematical operations can be performed on its components during the child generation method. For a critical discussion and the comparative analyses of the (dis)advantages of the RK and AL representation, see [6]. A *schedule generation scheme* transforms the encoded representation of a schedule into a resource feasible schedule following the pre-defined principles of the scheme, among which the serial and the parallel schedule generation schemes are the most widely known [9]. In our current manuscript, we rely on the schedule generation approach of [21]. This generation scheme aims at the construction of a resource feasible schedule and is a combination of the well-known serial schedule generation scheme and an adapted version of the bi-directional forward/backward generation scheme [17], and has been tested on a large and diverse instance set under various conditions. Consequently, the constructed schedules of our genetic algorithm have the following characteristics:

- The project schedule might end before, on or behind the pre-specified project deadline: although the generation scheme approach aims at the construction of a schedule that ends on or before the negotiated project deadline, it might fail to meet the pre-specified project deadline (due to e.g. tight resources), and hence, a project schedule with a longer makespan might be constructed.
- The objective function is the maximization of the net present value: the generation scheme approach combines forward and backward steps, to incorporate the basic idea of the net present value maximization for positive and negative cash flows.
- After each schedule generation, information from the obtained schedule is used to transform the random key RK into a standardized random key (SRK) in order to fulfil the topological order condition [20]. To that purpose, the genetic algorithm replaces the original RK values by the starting times of each activity to obtain the SRK values.

The algorithmic details of the different components of the genetic algorithm can be represented by the following the pseudo-code:

```
Algorithm Genetic Algorithm
  Initialize pool of random solutions
  Construct three populations
  While Stop Criterion not met
    Parent selection method
    Child generation method
    Fitness evaluation method
    Mutation method
    Update population
  End While
```

Initialize pool of random solutions. The algorithm creates an initial pool of solution elements by randomly generating random key (RK) vectors and constructing the corresponding schedule using the generation approach presented in [21].

Construct three populations. The algorithm creates three different populations containing solutions ranked according to its corresponding total net present value of (a subset of) its activities, as follows:

- Population 1 contains solution elements ranked according to decreasing values of the net present value of the activities with a positive cash flow. Since the net present value of all positive cash flows is always positive, we refer to this population set as set P^+.
- Population 2 contains solution elements ranked according to decreasing (negative) net present values of the negative cash flow activities. This population set is referred to as set P^-.
- Population 3 contains solution elements ranked according to decreasing values for the total net present value of the project (taking all activities with positive *and* negative cash flows into account). Since this set contains the best solutions for each generation run, we refer to this set as set P^b.

Note that the net present values of solutions from set P^+ and set P^- are only calculated on a subset of activities, without taking the penalty cost into account. The net present values for solutions of the set P^b, on the contrary, are calculated for all project activities, decreased with a penalty cost when violating the project deadline.

Parent selection method. The parent selection method can consist of various solution elements from identical and/or different subpopulations. In the algorithm, we have implemented the parent selection based on selections of the three population sets as given in Fig. 1. The set P^b consists of the $|P^b|$ best solutions for each generation run, and plays a central role in the parent selection method. Each population element (i.e. the father) of this set is combined with three other randomly selected solution elements (i.e. the mothers): (a) one solution element from the set P^-, (b) one from the set P^+ and (c) one solution element from the same set as the father solution element (P^b). Moreover, the algorithm also considers the combination of all solutions elements from P^- and P^+, as shown in Fig. 1(d). In the computational results section, we test alternative parent selection combinations and show that the selection approach of Fig. 1 outperforms all others.

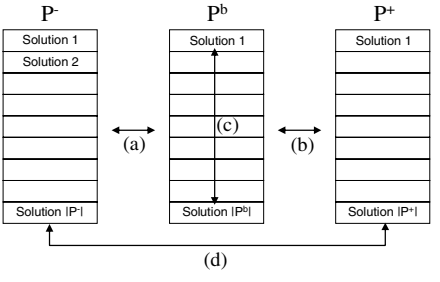


Fig. 1. The parent selection method with a central population set P^b

Child generation method. The exchange of information between two parent solutions based on a crossover operators results in child solutions and (hopefully) improves the quality of both parents. In our genetic algorithm, we have implemented and tested four different crossover operators, as follows:

The *two-point crossover operator* randomly selects two crossover points $c_1 \in [0, C_{max} / 2]$ and $c_2 \in]C_{max} / 2, C_{max}]$ with C_{max} the minimum of the makespans of both parents. Two child solutions are constructed by exchanging all SRK values between c_1 and c_2 between the parents. Activities that are not subject to a change are modified in order to preserve the relative ranking of these activities. More precisely, they get an SRK value equal to its original SRK value plus (minus) a large constant when its original value is higher (lower) than c_2 (c_1).

The *electromagnetic crossover operator* is based on the principles proposed by [2] and follows the law of Coulomb. The crossover operator calculates a charge as the relative difference in objective functions between the two parent solutions compared to the difference between the objectives of both parents to the best population element of the subpopulation of each parent. Next, a force is calculated based on the calculated charges such that the parent solution with the lowest net present value is moved

towards the better parent. This attraction mechanism of the forces results in a modification of (a subset of) the RK values of the parent solution with the lowest net present value towards the other parent, and is a simplification of the attraction/repulsion mechanism originally proposed by [2]. Details about the specific implementation of the operators can by found in [5], who have successfully implemented the electromagnetic algorithm to solve the resource-constrained project scheduling problem.

The *RUR based crossover operator* calculates two crossover points t_1 and t_2 for the mother solution based on resource information from the father solution, and runs as follows: First, the algorithm calculates the resource utilization ratio (RUR) of the father for each time instance t as the the average resource use for all activities active at that time instance. Second, the algorithm calculates a time window $[t_1, t_2[\in [0, C_{max}]$ with a given randomly selected length l for which the sum of the RUR is maximal and with C_{max} the minimal makespan of the father. Finally, the crossover operator copies the father RK values of all activities starting in the interval $[t_1, t_2[$ of the mother solution, while the RK values of the remaining activities are copied from the mother. This crossover operator is an adapted version of the peak crossover operator [19] and has been implemented by [4] for the resource-constrained project scheduling problem.

The *cash flow crossover operator* combines information from both parents into a single child solution based on the sign of the cash flows, and has been implemented in two versions according to the origin of the subpopulation, as follows:

- Combination of two solutions within a population: the crossover operator scans all SRK values of the father and the mother and copies the lowest (largest) SRK value in to the child solution when the cash flow of the corresponding activity is positive (negative). This approach aims at combining the best characteristics from two solution elements and has been implemented in the scatter search algorithm of [21].
- Combination of two solutions from two separate populations: the crossover operator combines good elements from parents from different populations into a single child solution. The good elements are defined as the RK values for the positive (negative) cash flow activities for population set (P^+) and P^-. Hence, a combination between P^+ and P^- results in a child for which all positive cash flow activities have an RK value copied from P^+ while the negative cash flow activities have an RK value copied from P^-. A child constructed from parents taken from P^+ and P^b have all RK values from P^+ for the positive cash flow activities while the remaining RK values come from P^b.

Fitness evaluation method. the net present value of the generated child solution is calculated after construction of a resource-feasible schedule with the generation approach presented in the subroutine “Initialize pool of random solutions”. When the project makespan exceeds the pre-defined project deadline, the net present value is decreased by a discounted penalty cost as given in eq. (1).

Mutation method. The mutation method diversifies the population to avoid the creation of a set of homogeneous population elements. Each time a mutation is performed, the algorithm randomly swaps a small fraction of the SRK values to obtain a new SRK vector. Computational results revealed that less than 10% of the SRK values need to be swapped to lead to the best results.

Update population. The newly obtained child solutions replace the parent solutions in the subset P^+, P^b and P^-, as follows. First, the P^b subset contains the best found solutions during each generation run, and hence, a parent solution is only replaced by a newly generated child solution if the parent is not the currently best found solution so far. In doing so, we prevent the loss of high quality solutions. Second, all solution elements of the subsets P^+ and P^- are automatically replaced by their best generated child solutions, even if this leads to a deterioration of the net present value, as long as the newly obtained solution has not yet been incorporated in the P^b subset. In doing so, we prevent duplication of solution elements among subsets. This is particularly important for problem instances where most activity cash flows have the same sign (positive or negative). In this case, the best solution elements of the P^b subset are very similar to the solutions of the P^- (in case of many negative cash flows) or the P^+ (in case of many positive cash flows) subset, which only takes the negative or positive cash flows into account during the net present value calculation.

The various crossover operators are illustrated on a project network example displayed in Fig. 2 with a pre-specified project duration of $\delta_{n+1} = 15$, an interest rate $\alpha = 0.01$ and a fixed resource availability for a single renewable resource $a_1 = 5$. Fig. 2 also displays a father solution (left) and a mother solution (right). We have run the GA without penalty cost, but instead we have assigned a large positive cash inflow (a lump sum cash inflow of 300) at the dummy end node, which gives the genetic algorithm an incentive to finish the project earlier than infinity.

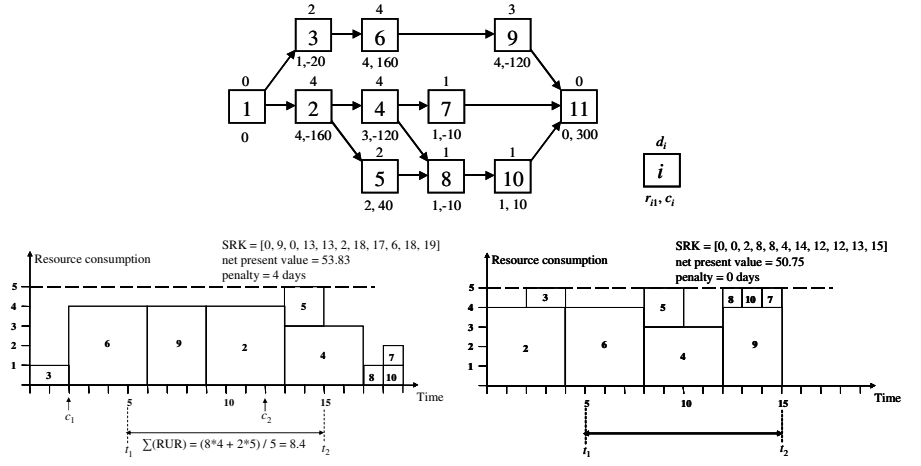


Fig. 2. An example network with two parent solutions (schedules)

The two-point crossover operator randomly selects the crossover points $c_1 = 2$ en $c_2 = 12$ from the interval [0, 9] and [10, 19], respectively, and generates a new RK vector as [0 - Δ, 0, 0 - Δ, 13 + Δ, 13 + Δ, 4, 18 + Δ, 17 + Δ, 12, 18 + Δ, 19 + Δ], with Δ a large constant to preserve the relative ranking. The RUR based crossover randomly selects a length $l = 10$ from the interval [0, 19] and calculates the maximal total RUR value on the parent solutions resulting in two crossover point $t_1 = 5$ en $t_2 = 15$. The constructed RK vector equals [0 - Δ, 0 - Δ, 2 - Δ, 13, 13, 4 - Δ, 18, 17, 6, 18, 15 + Δ].

The application of the two crossover operations results in a new child solution given in Fig. 3 (left). The electromagnetic crossover can not be illustrated on this simple example, since information about the objectives of all population elements is needed to calculate the charges for the father and mother solutions. The general philosophy is that the resulting charges will change a subset of the mother solution (lowest net present value) such that the RK values are modified towards the RK values of the father solution (highest net present value). The cash flow based crossover copies the RK values from the mother based on cash flow information, resulting in the RK vector equal to [0, 9, 2, 13, 8, 2, 18, 17, 12, 13, 15] (under the assumption that both parent solutions come from the same subset). The new child solution is displayed in Fig. 3 (right).

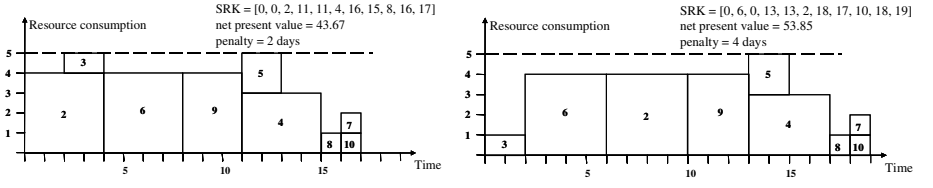


Fig. 3. The child solutions obtained by the two-point crossover, RUR based crossover and EM crossover (left) and the cash flow based crossover (right)

3 Computational Tests

We have coded the genetic algorithm in Visual C++ 6.0 and performed computational tests on a Dell Dimension DM051 with a Pentium D with a 2.80 GHz processor. Several algorithmic parameters have been fine-tuned to the best found values based on computational results on a small randomly generated dataset containing 1,000 instances with up to 100 activities. Computational results revealed that a distance-based mutation approach outperforms a random mutation approach. Hence, the algorithm only uses the mutation method in case a child has been constructed from two not mutually diverse parents. The algorithm calculates the distance between two parent solutions as the sum of the differences of the standardized random key values for each solution. We use a fixed minimal threshold value of n / 5 to distinguish between mutually equal and mutually diverse parent solutions and the number of swaps has been set fixed at n / 3 for each mutation. The population size has been set to 250 and the search has been truncated after 5,000 generated schedules. This population sizes is split into three disjoint subsets with each a population size of $|P^-|$, $|P^+|$ and $|P^b|$ (where $|P^-| + |P^+| + |P^b|$ is equal to the size of the total population). In all our computational tests, we have set the size of the subset P^b to a value equal to 65% of the population size. The remaining part is divided between subsets P^- and P^+, according to the percentage negative (for set P^-) and positive (for set P^+) cash flow activities in the project network.

Table 1 displays the results of the tests where all possible combinations of parent selection methods (see Fig. 1) and the four crossover operators of the child generation method have been tested. We have tested our algorithm on the N25 dataset presented

in [21] which can be downloaded from www.projectmanagement.ugent.be/npv.php which contains 180 problem instances with varying levels for the order strength OS [11] and the resource constrainedness RC [15]. . We extended these instances with activity cash flows where 0%, 25%, 50%, 75% or 100% of the activities have a negative cash flow and with four varying project deadlines from C_{max} * (1 + 5%) to C_{max} * (1 + 20%) in steps of 5%. We have extended each problem instance with two penalty costs: 10 + 10% (low) and 100 + 20% (high). Consequently, we test 180 * 5 * 4 * 2 = 7,200 project instances in total.

Table 1. Computational results for various combinations of parent selection and child generation methods

$P^c \times P$ and $P^c \times P^c$	$P^h \times P^h$	$P^c \times P^c$	$P^c \times P^h$ and $P^c \times P^h$	Penalty = low				Penalty = high			
				TPC	CFC	EMC	RUC	TPC	CFC	EMC	RUC
×				10.70%	0.73%	11.42%	10.25%	11.57%	0.81%	12.38%	11.08%
	×			**10.54%**	0.61%	**11.41%**	**9.97%**	**11.41%**	0.66%	**12.35%**	**10.79%**
		×		11.70%	0.84%	12.53%	11.37%	12.65%	0.93%	13.57%	12.29%
			×	10.86%	1.23%	12.31%	11.67%	11.77%	1.35%	13.34%	12.64%
×	×			11.34%	1.18%	11.79%	10.36%	12.26%	1.27%	12.78%	11.21%
×		×		11.25%	0.48%	11.94%	10.71%	12.18%	0.52%	12.94%	11.58%
×			×	11.80%	0.82%	12.20%	11.36%	12.77%	0.89%	13.21%	12.30%
	×	×		10.92%	0.38%	11.62%	10.08%	11.83%	0.40%	12.60%	10.93%
	×		×	10.58%	0.19%	11.72%	10.07%	11.46%	0.20%	12.70%	10.88%
		×	×	11.95%	1.10%	12.41%	11.64%	12.96%	1.24%	13.46%	12.59%
×	×	×		11.24%	0.34%	11.81%	10.41%	12.15%	0.36%	12.81%	11.27%
×	×		×	11.08%	0.15%	11.90%	10.60%	11.98%	0.17%	12.91%	11.48%
×		×	×	11.53%	0.42%	12.27%	11.32%	12.49%	0.46%	13.30%	12.27%
	×	×	×	10.64%	**0.00%**	11.69%	10.04%	11.53%	**0.00%**	12.67%	10.87%
×	×	×	×	11.24%	0.61%	12.10%	11.02%	12.16%	0.66%	13.12%	11.92%
Avg. Dev.				11.16%	0.61%	11.94%	10.73%	12.08%	0.66%	12.94%	11.61%

Table 1 displays the results for each penalty setting as an average deviation from the best average solution found by a particular combination of parent selection/child generation methods. The table clearly shows that the cash flow based crossover operator outperforms all other crossover operators for all child generation methods. Moreover, the table also reveals that the best parent selection method consists of a combination of all subpopulations, as presented in Fig. 1. However, the other crossover operators that do not take cash flow information into account perform best under a single population genetic algorithm, where only the best found solutions are saved in the population set.

Fig. 4 shows partial results of a large computational experiment where each set of solutions has been split up by network structure (OS) and resource tightness (RC). The average best found solutions have been displayed for each (OS, RC) combination (9 combinations in total), split up in the net present value excluding the penalty cost (y-axis) and the total amount of days exceeding the C_{max} (x-axis), and under three different settings for the %Neg and two penalty cost settings. The figure shows the intuitively clear result that a higher penalty cost leads to a lower project makespan and project delays (above the C_{max}) occur more often when the number of negative cash flows increases.

The table shows a clear positive effect between a project's net present value and its corresponding RC value when %Neg >=50. Indeed, projects with low RC value have

relative low lead times, and hence, can not fully benefit from the delay of negative cash flow activities. Hence, in order to obtain high net present values, the project lead time need to be increased a lot compared to the minimal project makespan C_{max}, resulting in a huge extra penalty cost. However, projects with a high RC value have a relatively high lead time, which automatically delays the negative cash flow activities resulting in a higher net present value. Hence, no large penalty cost need to be substracted compared to the minimal makespan C_{max} to obtain a relatively good (positive) net present value. A similar, but opposite effect can be observed for project instances with %Neg = 20%. In these cases, low RC value instances have low project deadlines, which is beneficial for the net present value of positive cash flows.

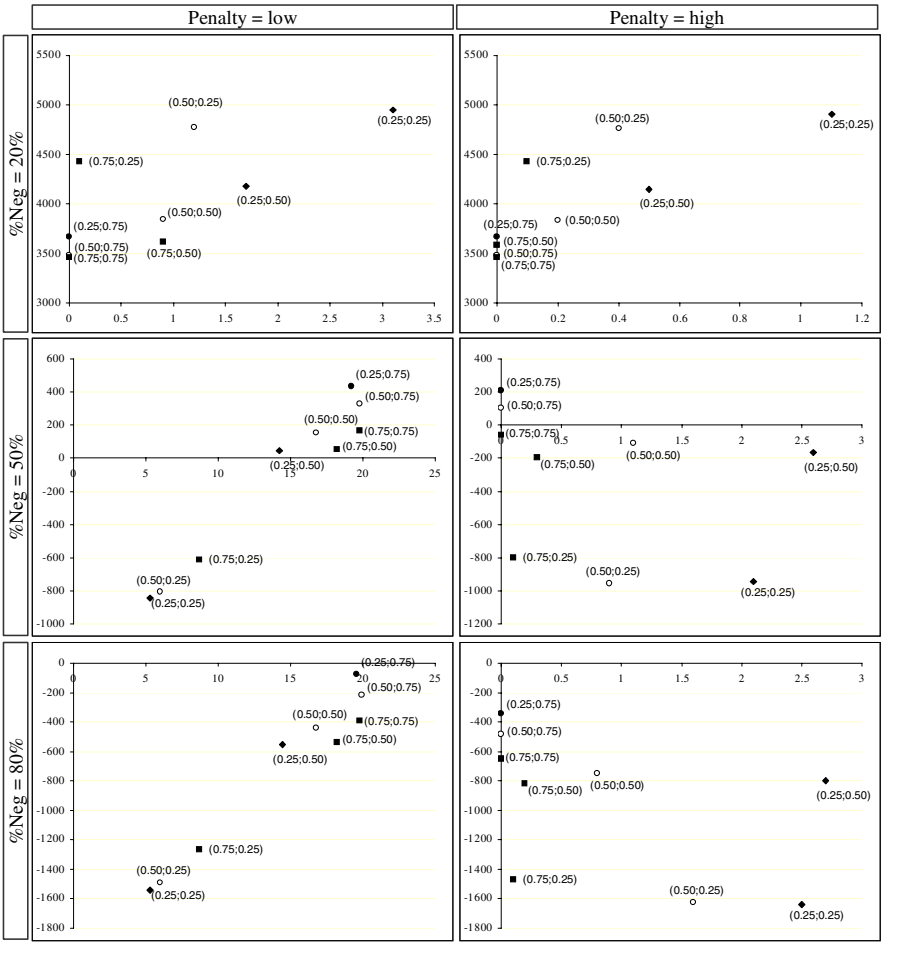


Fig. 4. The net present value (excl. penalty cost) versus the number of days above C_{max} (Each dot number represents a (OS, RC) value)

The influence of the OS follows a similar reasoning, although less outspoken. Project instances with %Neg = 20 show that lower OS values result in higher net present values as an immediate result of the relatively low project deadlines. A similar, but opposite effect can be observed for project instances with %Neg ≥ 50% and RC = 25 project instances: larger OS value instances have more precedence relations resulting in a larger project deadline. In case of negative cash flows (%Neg ≥ 50%), this has a beneficial effect on the net present value. However, all other instances (i.e. %Neg ≥ 50 and RC > 25) show a reverse OS effect: in these cases, the high resource tightness (RC > 25) automatically results in larger project deadlines (and higher corresponding net present values), and the positive influence of high OS values on the project deadline is no longer relevant. In these cases, lower OS values results in project instances with more scheduling degrees of freedom, allowing the postponement of negative cash flows further in time within the project duration. Note that the y-axis of Fig. 4 displays a project's net present value without the penalty cost. However, in order to detect which (OS, RC) combination has the highest *total* net present value (including the penalty cost), one can draw iso-npv lines (i.e. lines with identical total net present values) with a slope equal to the penalty cost.

4 Conclusions

In this paper, we presented a genetic algorithm using three different subpopulations. Each population sorts the individual solutions according to their corresponding net present value for a subset of activities.

We have tested three crossover operators that perform well for the resource-constrained project scheduling problem and compare them with a simple cash-flow based crossover operator. This operator relies on (positive and/or negative) cash flow characteristics and is based on the straightforward net present value philosophy, and the specific implementation slightly differs in function of the combination of the subpopulations. We also have tested various mutation operators and the influence of the problem structure (measured by the order strength) and the resource tightness (measured by the resource constrainedness) on the total net present value of a project. The computational results revealed that the combination of three populations outperforms a classical genetic algorithm (with one population) and the cash flow crossover performs far better than the other three operators.

References

1. Branke, J., Deb, K., Miettinen, K., Slowinski, R. (eds.): Multi-objective optimization: interactive and evolutionary approaches. LNCS, vol. 5252, p. 470. Springer, Heidelberg (2008)
2. Birbil, S.I., Fang, S.C.: An electromagnetism-like mechanism for global optimization. sssJournal of Global Optimization 25, 263–282 (2003)
3. Brücker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: notation, classification, models and methods. European Journal of Operational Research 112, 3–41 (1999)
4. Debels, D., Vanhoucke, M.: A decomposition-based genetic algorithm for the resource-constrained project scheduling problem. Operations Research 55, 457–469 (2007)

5. Debels, D., Vanhoucke, M.: The electromagnetism meta-heuristic applied to the resource-constrained project scheduling problem. In: Talbi, E.-G., Liardet, P., Collet, P., Lutton, E., Schoenauer, M. (eds.) EA 2005. LNCS, vol. 3871, pp. 259–270. Springer, Heidelberg (2006)
6. Debels, D., De Reyck, B., Leus, R., Vanhoucke, M.: A hybrid scatter search/electromagnetism meta-heuristic for project scheduling. European Journal of Operational Research 169, 638–653 (2006)
7. Herroelen, W., Demeulemeester, E., De Reyck, B.: A classification scheme for project scheduling. In: Weglarz, J. (ed.) Project Scheduling – Recent Models, Algorithms and Applications. International Series in Operations Research and Management Science, vol. 14, pp. 77–106. Kluwer Academic Publishers, Boston (1999)
8. Li, K.Y., Willis, R.J.: An iterative scheduling technique for resource-constrained project scheduling. European Journal of Operational Research 56, 370–379 (1992)
9. Kolisch, R.: Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. European Journal of Operational Research 43, 23–40 (1996)
10. Kolisch, R., Hartmann, S.: Heuristic algorithms for solving the resource-constrained project scheduling problem: classification and computational analysis. In: Weglarz, J. (ed.) Project Scheduling – Recent Models, Algorithms and Applications, pp. 147–178. Kluwer Academic Publishers, Boston (1999)
11. Mastor, A.A.: An experimental and comparative evaluation of production line balancing techniques. Management Science 16, 728–746 (1970)
12. Mika, M., Waligora, G., Weglarz, J.: Simulated annealing and tabu search for multi-mode resource-constrained project scheduling with positive discounted cash flows and different payment models. European Journal of Operational Research 164, 639–668 (2005)
13. Özdamar, L., Ulusoy, G., Bayyigit, M.: A heuristic treatment of tardiness and net present value criteria in resource-constrained project scheduling. International Journal of Physical Distribution and Logistics 28, 805–824 (1998)
14. Özdamar, L., Ulusoy, G.: An iterative local constraint based analysis for solving the resource-constrained project scheduling problem. Journal of Operations Management 1996, 193–208 (1996)
15. Patterson, J.H.: Project scheduling: the effects of problem structure on heuristic scheduling. Naval Research Logistics 23, 95–123 (1976)
16. Russell, A.H.: Cash flows in networks. Management Science 16, 357–373 (1970)
17. Selle, T., Zimmermann, J.: A bidirectional heuristic for maximizing the net present value of large-scale projects subject to limited resources. Naval Research Logistics 50, 130–148 (2003)
18. Ulusoy, G., Özdamar, L.: A heuristic scheduling algorithm for improving the duration and net present value of a project. International Journal of Operations and Production Management 15, 89–98 (1995)
19. Valls, V., Ballestín, F., Quintanilla, S.: A hybrid genetic algorithm for the Resource-constrained project scheduling problem with the peak crossover operator. In: Eighth International Workshop on Project Management and Scheduling, pp. 368–371 (2002)
20. Valls, V., Quintanilla, S., Ballestín, F.: Resource-constrained project scheduling: a critical activity reordering heuristic. European Journal of Operational Research 149, 282–301 (2003)
21. Vanhoucke, M.: A scatter search heuristic for maximizing the net present value of a resource-constrained project with fixed activity cash flows. International Journal of Production Research (2009) (to appear)

Divide-And-Evolve Facing State-of-the-Art Temporal Planners during the 6^{th} International Planning Competition

Jacques Bibai[1,2], Marc Schoenauer[1], and Pierre Savéant[2]

[1] Projet TAO, INRIA Saclay & LRI, Université Paris Sud, Orsay, France
firstname.lastname@inria.fr
[2] Thales Research & Technology, Palaiseau, France
firstname.lastname@thalesgroup.com

Abstract. *Divide-and-Evolve* (DAE) is the first evolutionary planner that has entered the biennial International Planning Competition (IPC). Though the overall results were disappointing, a detailed investigation demonstrates that in spite of a harsh time constraint imposed by the competition rules, DAE was able to obtain the best quality results in a number of instances. Moreover, those results can be further improved by removing the time constraint, and correcting a problem due to completely random individuals. Room for further improvements are also explored.

1 Introduction

An artificial intelligence planning problem is specified by the description of an initial state, a set of desired goals to reach and a set of possible actions. An action modifies the current state, and can be applied only if certain conditions in the current state are met. A solution to a planning problem is an ordered set of actions forming a valid plan, whose execution in the initial state transforms it into a state where the problem goals are satisfied. In temporal planning problems a given goal is reached by taking a number of durative actions which may temporally overlap.

Although researchers have investigated a variety of methods for solving the temporal planning problems submitted to the biennial International Planning Competition (IPC) since 1998 (e.g. extended planning graph: TGP [13], LPG [6,7]; reduction to linear programming: LPGP [8]; reduction to constraint programming: CPT [14,15]; partitioning planning problems into subproblems by parallel decomposition: SGPlan [3]), none of them rely on an evolutionary algorithm. Recently, *Divide-and-Evolve* (DAE), an Evolutionary Planner was proposed by the authors [11,12], and entered the IPC-6 competition, becoming, to the best of our knowledge, the first evolutionary temporal planning system that participated to such competition.

DAE hybridises evolutionary algorithms with classical planning methods by dividing the initial problem into a sequence of subproblems, solving each subproblem in turn, and building a global solution from the subproblem solutions.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 133–144, 2009.

This approach is both original and generic for the planning community. However, the results of DAE in the IPC-6 competition were altogether disappointing, even though it performed best on several problems.

In this paper, we investigate the reasons for such results, and propose several ways to enhance DAE performances. Section 2 briefly introduces Temporal Planning Problems (TPP); Section 3 details the *Divide-and-Evolve* approach, representation, fitness function and variation operators; section 4 validates DAE beyond [11,12] by presenting experimental results on previous IPC benchmarks; Section 5 details the results of DAE during the IPC-6 competition, then proposes some enhancements that are validated on the IPC-6 benchmarks.

2 Temporal Planning Problem

Domain-independent planners rely on the Planning Domain Definition Language (PDDL) [10], inherited from the STRIPS model [4], to standardise and represent a planning problem. It was developed mainly to make the International Planning Competition (IPC) series possible. The planning competition compares the performance of candidate planners on a set of benchmark problems, thus requiring a common language for specifying them. The language has been extended for representing temporality and action concurrency in PDDL2.1 [5].

The description of a planning problem splits in two separate parts: the generic domain theory on one hand and a specific instance scenario on the other hand. The domain definition specifies object types, predicates and actions which capture the possible moves, whereas the instance scenario declares the objects of interest, the initial state and the goal description. A state is described by a set of atomic formulae, or atoms. An atom is defined by a predicate symbol from the domain followed by a list of object identifiers: *(PREDICATE_NAME* OBJ_1 *...* OBJ_N*)*. The initial state is complete, i.e. it gives a unique status of the world, whereas the goal is only partial and can be true in many different states. An action is composed of a set of preconditions and a set of effects, and applies to a list of variables given as arguments. Preconditions are logical constraints which apply domain predicates to the arguments an trigger the effects when they are satisfied. Effects enable state translations by adding or removing atoms.

A solution to a temporal planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state where the problem goal is satisfied. The total duration of the solution plan is called the *makespan* and its minimisation is the usual objective of the optimisation.

A planning problem defined on domain D with initial state I and goal G will be denoted $\mathcal{P}_D(I, G)$ in the following.

3 Divide-And-Evolve

Divide-and-Evolve (DAE) is an evolutionnary computation technique that searches the space of state decompositions [11,12]. In order to solve a planning problem $\mathcal{P}_D(I, G)$, the basic idea is to find a sequence of states $S_1, \ldots, S_n$, and to

use some 'local' planner to solve the series of planning problems $\mathcal{P}_D(S_k, S_{k+1})$, for $k \in [0, n]$ (with the convention that $S_0 = I$ and $S_{n+1} = G$). The concatenation of the plans solving all subproblems is then a plan that solves the original global problem. The rationale is that all sub-problems $\mathcal{P}_D(S_k, S_{k+1})$ can be made simpler for the local planner than the original problem $\mathcal{P}_D(I, G)$, thus allowing DAE to find solutions of instances that the local planner alone cannot solve. Very preliminary validation on the IPC-3 `zeno` benchmarks validated this idea in [11,12]. However, many improvements have been brought to DAE since those early results, and, more importantly, comparison of DAE results with state-of-the-art planners remained to be done.

We shall now present DAE, specifically detailing the problem-specific representation and associated initialisation and variation operators, as well as the fitness function.

3.1 Representation

Following the rationale above, an individual in DAE is a sequence of states. And as described in Section 2, a state is a list of boolean atoms. However, searching the space of complete states would result in a very fast combinatorial explosion of the search space size. Moreover, goals of TPPs need only to be defined as partial states. It thus seemed practical to search only sequences of partial states. However, this raises the issue of the **choice of the atoms** to be used to represent individuals, among all possible atoms. The choice made in the first versions of DAE [11,12] was to use only the predicates that appear in the goal – this variant of DAE will be termed DAE1 in the following. Another option is to use the results of the grounding step of CPT, that generates all possible atoms, and to choose the predicates (at most 3) that are more frequent. This variant is termed DAE2 in the following.

Nevertheless, even when restricted to specific choices of atoms, the choice of random atoms can lead to inconsistent partial states, because some sets of atoms can be *mutually exclusive* (`mutex` in short). Whereas it could be possible to allow `mutex` atoms in the partial states generated by DAE, and to let evolution discard them, it seems more efficient to a priori forbid them, as much as possible. Because the embedded planner CPT computes and maintains a list of all `mutex` pairs of atoms, it is thus possible to exclude such pairs a priori, even though this still does not guarantee that the partial state is consistent – but determining if a state is consistent amounts to solving the complete planning problem!

An individual in DAE is hence represented as a variable length ordered list of partial states, and each state is a variable length list of atoms involving only predicates that are present in the goal G that are not pairwise `mutex`.

3.2 Fitness, and CPT

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling a local planner to solve the sequence of problems $\mathcal{P}_D(S_k, S_{k+1})$ $(k = 0, \ldots, n)$. Any existing planner could be used here, and DAE uses CPT, an exact planning

system for temporal STRIPS planning. CPT combines a branching scheme based on Partial Order Causal Link (POCL) Planning with powerful and sound pruning rules implemented as constraints [14,15].

For any given k, if CPT succeeds in solving $\mathcal{P}_D(S_k, S_{k+1})$, the final complete state is computed, and becomes the initial state of next problem: initial states need to be complete (and denoting each problem as $\mathcal{P}_D(S_k, S_{k+1})$ is indeed an abusive notation). If all problems, $\mathcal{P}_D(S_k, S_{k+1})$ are solved by CPT, the individual is called *feasible*, and the concatenation of all solutions plans for all $\mathcal{P}_D(S_k, S_{k+1})$ is a global solution plan for $\mathcal{P}_D(S_0 = I, S_{n+1} = G)$. However, this plan can in general be optimised by parallelising some of its actions, in a step call *compression* (see [2,12] for detailed discussion). The fitness of a feasible individual is the makespan of the compressed plan.

However, as soon as CPT fails to solve one $\mathcal{P}_D(S_k, S_{k+1})$ problem, the following problem $\mathcal{P}_D(S_{k+1}, S_{k+2})$ cannot be even tackled by CPT, as its complete initial state is in fact unknown, and no makespan can be given to that individual. All such plans receive a fixed penalty cost such that the fitness of any infeasible individual is higher than that of any feasible individual. In order to nevertheless give some selection pressure toward feasible individuals, the relative rank of the first problem that CPT fails to solve is added to the fixed penalty, so infeasible individuals which solve the more subproblems are favoured by selection.

Finally, because the initial population contains randomly generated individuals, some of them might contain some subproblems that are in fact more difficult than the original global problems. Because CPT can sometimes take months to solve very difficult problems, it was necessary to limit the **maximal number of backtracks** that CPT is allowed to use to solve any of the subproblems. And because, ultimately, it is hoped that all subproblems will be easy to solve, such limitation should not harm the search for solutions – though setting this limit might prove difficult (see Section 5.1).

3.3 Initialisation and Variation Operators

The **initialisation** of an individual is the following: First, the number of states is uniformly drawn between one and the number of atoms in the goal of the problem, divided by the number of atoms per state; the number of atoms per state is chosen uniformly in $[1, 4]$. Atoms are then chosen one by one, uniformly in the allowed set of atoms (i.e. built on the goal predicate in most results here), and added to the individual if not `mutex` with any other atom already there (thanks to CPT list of pairwise exclusions).

A 1-point **crossover** is used, adapted to variable-length representation in that both crossover points are uniformly independently chosen in both parents.

Four different mutation operators have been designed, and once an individual has been chosen for mutation (according to a population-level mutation rate), the choice of which mutation to apply is made according to user-defined relative weights (see Section 3.4). Two mutation operators act at the individual level, either removing or inserting a partial state at a uniformly chosen position in the list, and two act at the state level, removing or modifying an atom (while still avoiding

pairwise `mutex` atoms) in a uniformly chosen partial state of the individual. A mutation that adds an atom to an existing partial state was used in early experiments, but soon was found to have no influence whatsoever on the results, and hence was abandoned. The reason for that is probably that when a new partial-state is inserted in the individual, it is created using some atoms of its two neighbors, plus some new atoms, thus bringing in diversity at the state level.

3.4 Evolution Engine and Parameter Settings

One of the main weaknesses of Evolutionary Algorithms today is the difficulty in tuning their numerous parameters, for which there exist no theoretical guidelines. Users generally rely on their previous experience on similar problems, or use standard but expensive statistical methods, e.g. Design of Experiments (DOE) and Analysis of Variance (ANOVA).

However, because there are here many parameters to tune, some of them were set once and for all based on preliminary experiments [11,12]. This is the case for the **evolution engine**, chosen to be a (10+70)-ES: 10 parents generate 70 offspring using variation operators, and the best of those 80 individuals become the parents of the next generation. The same stopping criterion has also been used for all experiments: after a minimum number of 10 generations, evolution is stopped if no improvement of the best fitness in the population is made during 20 generations, with a maximum of 100 generations altogether.

The remaining parameters concern the variation operators: the probabilities of individual-level application of crossover and mutation (p_{cross} and p_{mut}) and the relative weights of the 4 mutation operators ($w_{addStation}$, $w_{delStation}$, $w_{changeAtom}$, $w_{delAtom}$). A two-stage DOE was used: first, the relative weights were set to (4, 1, 4, 1) (from preliminary experiments), and an incomplete factorial DOE was done on p_{cross} and p_{mut} on `zeno 10-12` problems (11 runs per parameter set). The differences were then validated using both Kolmogorov and Wilcoxon non-parametric tests at 95% confidence levels. Three pairs for (p_{cross}, p_{mut}) were found significantly better than the others, and another DOE on the 4 weights and those 3 pairs yielded the final setting: (0.25, 0.75) for (p_{cross}, p_{mut}), and (35, 3, 35, 7) for the relative mutation weights.

4 Comparing DAE and CPT on IPC-3 Problems

This section presents experimental results that further validate the DAE approach beyond the initial results in [11] using several other domains from the 3^{rd} International Planning Competition.

Significantly, DAE can solve several problems that CPT alone cannot. For instance, for the `zeno` domain (not shown on the figure 1), DAE solved the first 19 instances when CPT failed after instance 14. Same thing is true, though not as clearly related to the instance number, for the `satellite` and `drivers` domains, and to a lesser extent `depot`.

The other observation concerns the quality of the results, compared to the optimal values found by CPT, an exact planner. For all instances of the `rovers`,

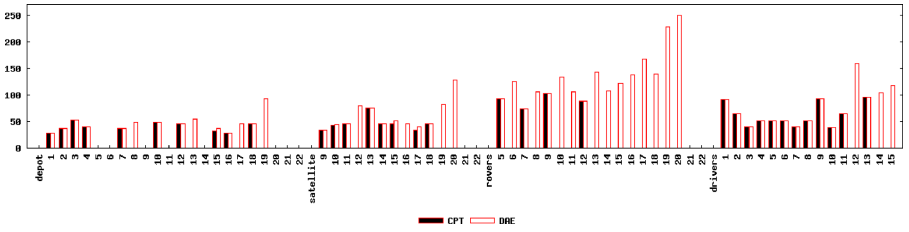


Fig. 1. Optimal makespans for CPT (1 month time limit) and DAE (best of 100 runs, 4000 backtracks limit) on `depot, satellite, rovers and drivers` IPC-3 domains. Each column represents an instance of the domain.

`zeno` and `drivers` domains, DAE has found optimal values. For the `satellite` and `depot` domains, DAE always found either the optimal value, or a best makespan very close to the optimum (more than 93%). Moreover, when CPT fails to find a solution, the values found by DAE are very close to (and sometimes better than) those found by LPG [2].

5 DAE at the IPC-6 Competition

The International Planning Competition (IPC) is a biennial event organised within the International Conference on Planning and Scheduling, aiming at analysing and advancing the state-of-the-art in automated planning systems. The experimental conditions imposed by this competition are very simple: all planners have to solve several instances of different planning domains in a completely automated way, and within 30min of CPU time.

5.1 Maximal Number of Backtracks

Divide-and-Evolve entered the 6^{th} edition in the deterministic track [1] with DAE1 and DAE2, the two versions described in Section 3.1 (DAE1 uses the predicates of the goal to represent the individuals, while DAE2 uses those having the most instances amongst all possible atoms).

Whereas all parameters described in Section 3.4 were chosen as default, giving a fixed value to the maximum number of backtracks allowed for CPT had two possible major drawbacks: on the one hand, a too small limit could completely prevent DAE from finding solutions even to problems that CPT alone could solve (using more backtracks); on the other hand, a too large limit would allow CPT to spend a lot of time on poor individuals in the early generations, slowing down the evolution and forbidding any good solution to be found within the 30min limit.

Based on numerous experiments made on IPC-3 benchmarks, the maximum number of backtracks allowed for CPT was set to a linear combination of the ratio of the number of causal links plus the number of actions to the number of

atoms generated by those actions, and the ratio of the number of nodes to the number of conflicts:

$$bks_{state} = \#Ga * (\frac{\#nodes}{\#conflicts} + 2 * (\frac{\#causals + \#actions}{\#atoms})) \quad (1)$$

where $\#Ga$ is the number of goal atoms, $\#causals$ the number of causal links, $\#actions$ the number of actions and $\#atoms$ the number of atoms generated after action grounding.

Further experiments have also demonstrated the need for more backtracks when solving the last subproblem (reaching the global goal). Hence a specific formula was designed for this case: $bks_{goal} = 7 * bks_{state}$

5.2 Detailed DAE Results at IPC-6

Using the above automated parameter settings, the raw global result of DAE are the 4^{th} and 5^{th} ranks among 6 participants. Those poor results can however be refined when considered domain by domain: In two of the 6 domains, CPT could not even complete its grounding step in less than 30min: in such domain, of course, there is no hope that DAE can solve any of the instances. Furthermore, though in the 4 other domains DAE could not solve all instances, it almost always found a better makespan than *SGPlan6*, the winner of the competition, on instances it did solve, as witnessed by Figures 2, 3, 4 and 5 for, respectively, the `peg solitaire`, the `openstack`, the `parcprinter` and the `crewplanning` domains.

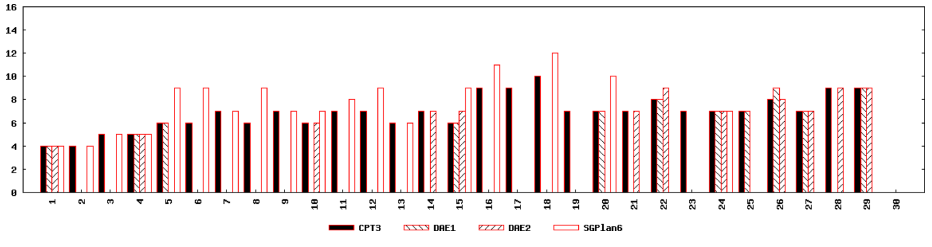


Fig. 2. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `peg solitaire` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance)

Surprisingly, however, DAE failed to solve instances that CPT alone could solve, as `peg` 11 to 13 and 16 to 19, or `parcprinter` 7 to 15 (and even 3 to 5 for DAE2). First investigations of this strange behaviour showed that it was not due to the time limit, nor to a too small value for the maximal number of backtracks. It finally turned out that, for some of the random individuals of the first generation, CPT entered some infinite loop. This is again a case where Evolutionary Algorithms can be seen as "fitness function debuggers" [9].

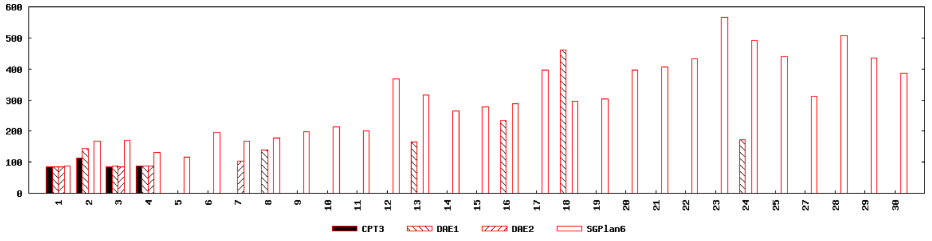


Fig. 3. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `openstack` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance)

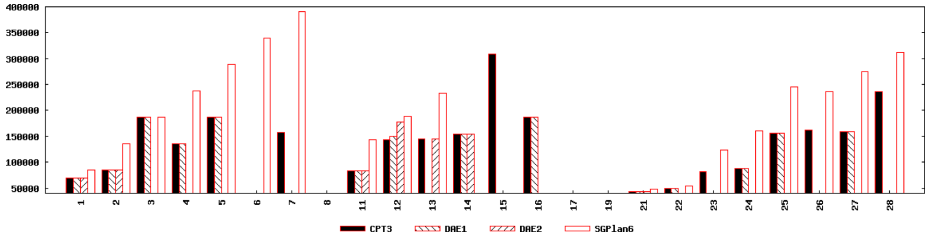


Fig. 4. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `parcprinter` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance)

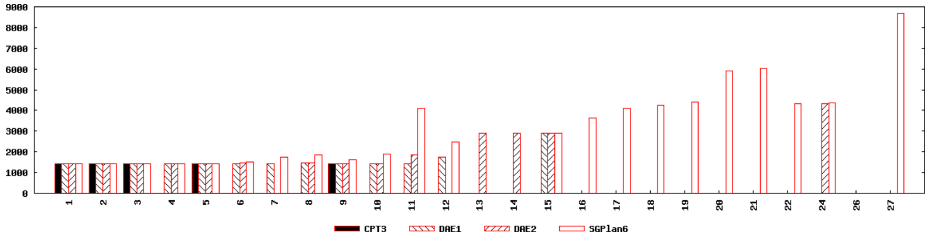


Fig. 5. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and SGPlan6 (the winner of the competition, white) on `crewplanning` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance)

5.3 A New Version of DAE

A patch was applied to CPT, and used within the DAE1 version, now termed DAE-p. Note that this patch modifies CPT internal consistency check, and could result in inconsistent plans. Hence all the solutions found by DAE-p were validated with the IPC solution checker (`http://planning.cis.strath.ac.uk/VAL/`), thus ensuring their consistency.

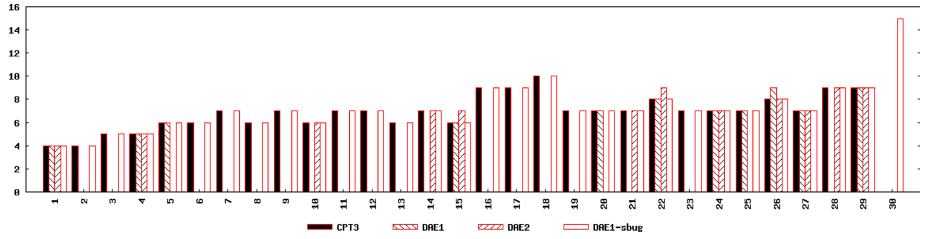


Fig. 6. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `peg solitaire` domain (one column per instance). No time limit.

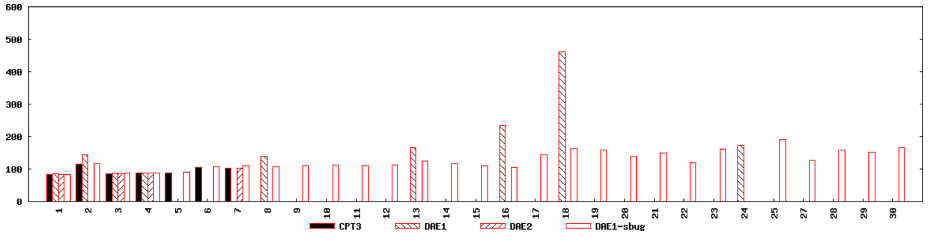


Fig. 7. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `openstack` domain (one column per instance). No time limit.

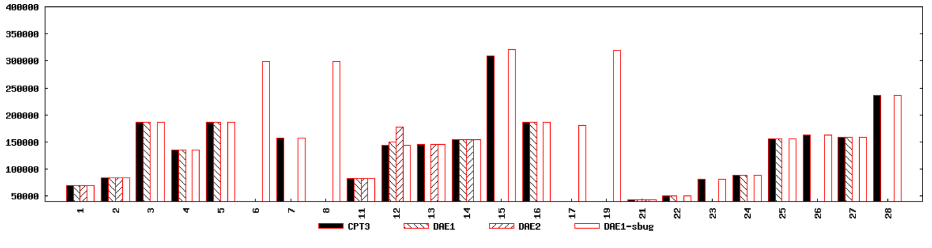


Fig. 8. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `parcprinter` domain (one column per instance). No time limit.

DAE-p was able to solve almost all instances of the 4 domains of the IPC-6 competition that CPT could handle, though demanding more than 30 min for the large instances: See Figures 6, 7, 8, and 9 for the comparative results of CPT, DAE1, DAE2 and DAE-p, and Figure 10 for examples of running times of the new DAE-p algorithm, where the horizontal bars show the 30min limit of IPC-6 competition: even DAE-p would have failed to solve the most difficult instances in the competition conditions.

5.4 Time-Based Atom Choice

Another conclusion that can be drawn from the results of IPC-6 (Section 5.2) is that none of the planners DAE1 or DAE2 outperforms the other one. Indeed,

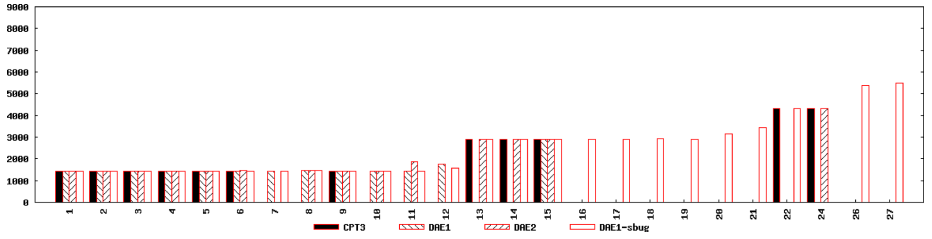


Fig. 9. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines) and DAE-p (white) on `crewplanning` domain (one column per instance). No time limit.

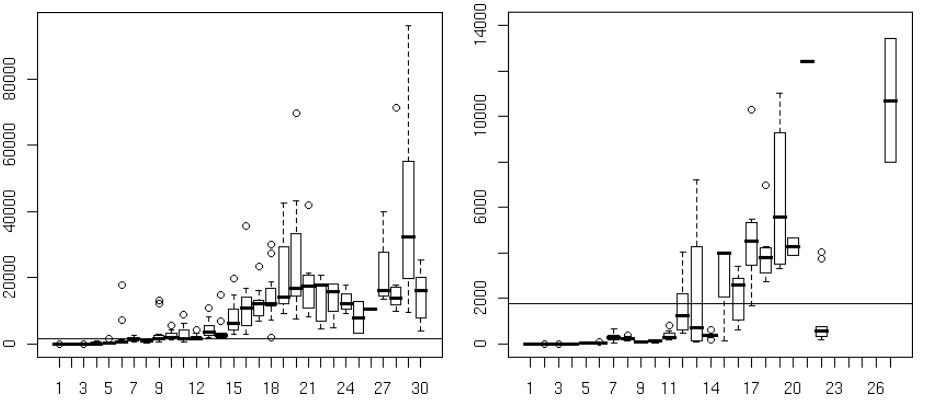


Fig. 10. Run times (s) of DAE-p on `openstack` (left) and `crewplanning` (right) domains. For each instance (column), the central box is the 25% – 75% quartile with median bar, the end of the upper (lower) dashed lines indicate the highest (lowest) values that are not considered as outliers, while the circles outside these lines are outliers.

though DAE1 outperforms DAE2 on most instances (e.g. for `parcprinter` domain Figure 8, or for instances `peg` 5, 20 and 25 Figure 6), the reverse is true on a large minority of other instances (e.g. `crewplanning` 24 Figure 9, `peg` 10, 14, 21, and 28 Figure 6). No satisfactory explanation could be found for this unpredictable behaviour. However, it demonstrates the need for a very careful choice of the atoms that are used to build the partial states.

This lead to propose a new method to actually build the partial states, based on a lower bound on the earliest time an atom can become true. Such lower bounds can easily be computed using a relaxation of the initial problem. The time before the earliest time all atoms from the goal can become true is then discretized into the number of partial states that are needed, and a partial state is build at each value of this discretized time by randomly choosing atoms between 1 and the number of subset of `mutex` atoms that are possible true at this time. Variations operators (cf. Section 3.3) are modified changed in order take into account those earliest time for all atoms.

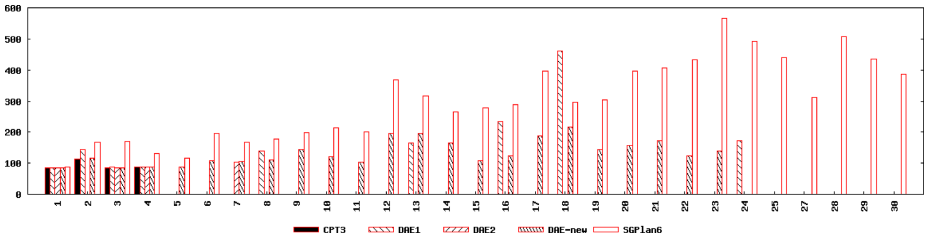


Fig. 11. Results of CPT (black), DAE1 (downward lines), DAE2 (upward lines), **DAE**$_{new}$ (grey), and SGPlan6 (the winner of the competition, white) on `openstack` domain (one column per instance) with IPC-6 conditions (30 min CPU per instance). To be compared with Figure 3.

Preliminary results using this new algorithm, termed at the moment DAE$_{new}$, can be seen on Figure 11, and should be compared to those of Figure 3: DAE$_{new}$ solves all instances up to instance 23, and clearly outperforms SGPlan6 – though deeper and more intensive statistical tests are still on-going.

6 Conclusion and Further Work

Divide-and-Evolve is an original "memeticization" of Evolutionary and Operational Research algorithms in the area or Temporal Planning. A lot of progress has been made since its original inception [11], and it has now reached a level of performance high enough to be able to compete with the state-of-the-art planners of all kind.

First of all, DAE concept when using CPT as the embedded planner has been validated on many benchmarks of both IPC-3 and IPC-6 competitions, as DAE did solve many problems that CPT alone did not. Whereas this demonstrate the ability of DAE to overcome the curse of time complexity, DAE still fails when space-complexity is the issue, and CPT cannot even initialise the problem data.

DAE is the first evolutionary planner to have entered the IPC-6 competition, where CPT allowed DAE to tackle 4 out of 6 domains. A very positive result is that DAE gave a better makespan than SGPlan6, winner of the competition, whenever it could find a feasible solution (i.e. reach the goal). Unfortunately, there remained several instances that DAE could not solve – sometimes even without the harsh time limit of the competition, and even on instances that CPT alone could solve. This gave us an opportunity to detect a weird behaviour of CPT, that is unlikely to take place with 'standard' instances, but did happen with the random initial individuals of some evolutionary runs. The patched version of DAE was then able to solve most IPC-6 instances - though sometimes requiring much more CPU time than allowed during the official runs.

But there is still room for large improvements for DAE. First, the choice of the atoms that are used to represent individuals is still an open issue, and preliminary experiments using more information from the planning domain are very

promising. But another critical parameter is the maximum number of backtracks that we allow to CPT runs. The empirical formulae need to be refined, and this puts some light on the more general issue of parameter tuning: what is needed now are some descriptors of temporal planning instances, that would allow us to learn the best parameters based on the instance description. Such promising directions will be the subject of further research.

References

1. Bibai, J., Schoenauer, M., Savéant, P., Vidal, V.: DAE: Planning as Artificial Evolution (Deterministic part). In: IPC 2008 Competition Booklet (2008)
2. Bibai, J., Schoenauer, M., Savéant, P., Vidal, V.: Evolutionary Planification by decomposition. INRIA Research Report No. RT-0355 (2008), http://hal.inria.fr/inria-00322880/en/
3. Chen, Y., Hsu, C., Wah, B.: Temporal Planning using Subgoal Partitioning and Resolution in SGPlan. Artificial Intelligence 26, 323–369 (2006)
4. Fikes, R., Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 2(3-4), 189–208 (1971)
5. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research 20 (2003)
6. Gerevini, A., Saetti, A., Serina, I.: Planning through Stochastic Local Search and Temporal Action Graphs in LPG. Journal of Artificial Intelligence Research 20, 239–290 (2003)
7. Gerevini, A., Saetti, A., Serina, I.: On Managing Temporal Information for Handling Durative Actions. In: LPG. AI*IA 2003: Advances in Artificial Intelligence. Springer, Heidelberg (2003)
8. Long, D., Fox, M.: Exploiting a Graphplan Framework in Temporal Planning. In: Proceedings of ICAPS 2003, pp. 51–62 (2003)
9. Mansanne, F., Carre, F., Ehinger, A., Schoenauer, M.: Evolutionary Algorithms as Fitness Function Debuggers. In: Raś, Z.W., Skowron, A. (eds.) ISMIS 1999. LNCS, vol. 1609. Springer, Heidelberg (1999)
10. McDermott, D.: PDDL – The Planning Domain Definition Language (1998), http://ftp.cs.yale.edu/pub/mcdermott
11. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: a New Memetic Scheme for Domain-Independent Temporal Planning. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 247–260. Springer, Heidelberg (2006)
12. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: a Sequential Hybridisation Strategy using Evolutionary Algorithms. In: Michalewicz, Z., Siarry, P. (eds.) Advances in Metaheuristics for Hard Optimisation, pp. 179–198. Springer, Heidelberg (2007)
13. Smith, D., Weld, D.S.: Temporal Planning with Mutual Exclusion Reasoning. In: Proc. IJCAI 1999, pp. 326–337 (1999)
14. Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In: Proc. of AAAI 2004, pp. 570–577 (2004)
15. Vidal, V., Geffner, H.: Branching and Pruning: Optimal Temporal POCL Planner based on Constraint Programming. Artificial Intelligence 170(3), 298–335 (2006)

Exact Solutions to the Traveling Salesperson Problem by a Population-Based Evolutionary Algorithm*

Madeleine Theile

Institut für Mathematik, TU Berlin, Germany
theile@math.tu-berlin.de

Abstract. This articles introduces a $(\mu + 1)$-EA, which is proven to be an exact *TSP* problem solver for a population of exponential size. We will show non-trivial upper bounds on the runtime until an optimum solution has been found. To the best of our knowledge this is the first time it has been shown that an $\mathcal{NP}$-hard problem is solved exactly instead of approximated only by a black box algorithm.

1 Introduction

Evolutionary algorithms (EA) are based on the principle of bio-inspired computing using principles like mutation, crossover, fitness of individuals and various types of selection. Evolutionary algorithms together with other algorithm classes like e. g. memetic algorithms and ant colony optimization algorithms belong to the very general class of randomized search heuristics (RSH) (cf. [20] for an introductory article). They enable an engineer to attack difficult optimization problems in a straightforward manner as all of these algorithm classes consist of a modular framework. Naturally, such generic approaches cannot compete with a custom-tailored algorithm, which explicitely uses problem-specific knowledge. Thus, it is not surprising that early hopes that RSH might make notoriously hard problems become tractable did not fulfill. While most research on evolutionary computation is experimental, the last ten years produced a growing interest in a theory-founded understanding of the success of these algorithms.

The challenge to gain additional theoretical insights complementary to experimental results is that RSH have not been designed with the aim to make them analyzable in terms of runtime or solution quality. A theoretical understanding of such methods - in contrast to experimental successes - is still in its infancy. Although search heuristics are mainly applied to problems whose structure is not well known, the analysis of evolutionary algorithms - as a starting point - concentrates on problems whose structure is well understood. The recent years produced some very nice theoretical results, mostly on convergence phenomena

* This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 145–155, 2009.

and runtime analyses. But nevertheless the overall goal of these theoretical considerations is to provide practitioners with insights of when and why randomized search heuristics work well.

The problem we are dealing with in this article is to find optimal solutions to the *Traveling Salesperson Problem (TSP)*, which is well known to be $\mathcal{NP}$-hard. When faced with an $\mathcal{NP}$-hard optimization problem the typical approach to compute a solution is to find a polynomial algorithm which computes an approximate solution. In the case of the general TSP dealing with arbitrary distances between the cities of the input, it is not even possible to find an approximation algorithm[1]. Thus, the TSP is one of the hardest combinatorial problems to tackle. Nevertheless, we will show, how it is possible to solve a given arbitrary TSP instance by a population-based evolutionary algorithm in a non-trivial runtime competitive to the best known upper bounds for the exact solution of the problem.

1.1 Previous Work

There are some exact algorithms for the solution of the TSP based on linear programming and dynamic programming, see [1], [10], which are typically not the best algorithms in terms of performance. Consequently, there is still the need for experimental solution of large TSP instances. This demand has produced a lot of clever ideas using local optimization techniques, such as tabu search, particle swarm and genetic algorithms (cf. [9], [14]). Possibly the most famous one is the algorithmic scheme of Lin and Kernighan [12] which implements a variable k-opt neighborhood in order to find good solutions to real-world instances. However, there is a recent result proving for "a family of Euclidean instances" that "2-opt can take an exponential number of steps", see [4]. The last result shows that there is always the need to theoretically understand the guarantees of any practically successful heuristic.

Evolutionary Algorithms have been found to be efficient optimizers for a lot of problems, see, e.g., the Proceedings of the conferences EvoCop, GECCO, PPSN. Among the theory-driven results are combinatorial optimization problems which are typical components of real-world optimization problems like e. g. sorting and shortest paths [19], spanning trees [17], maximum matchings [7], matroids [18], all-pairs shortest path [3] and minimum cuts [15,16]. There are also some results on evolutionary algorithms acting as approximation algorithms for $\mathcal{NP}$-hard problems like partition [21] and covering problems [6]. RSH in their pure form are considered to be black box algorithms. This means that they do not gain any explicit knowledge about the structure of the problem instance. However, the analysis of a problem explicitely calls for the exploitation of the problem structure. Thus, it is common to all of these articles that they use structural knowledge of the problem gained by the analysis of classical algorithms to prove how a black-box algorithms is capable of solving the problem.

[1] Under the reasonable assumption of $\mathcal{P} \neq \mathcal{NP}$.

1.2 Our Contributions

In this article we will prove for the first time, that evolutionary algorithms - in their classical sense of being black box algorithms - are able to act as exact solvers for $\mathcal{NP}$-hard problems. We will show how a population-based evolutionary algorithm is able to find an optimal solution to an arbitrary TSP instance in non-trivial worst-case optimization time $O(n^3 \cdot 2^n)$ with a population of size less than $(n-1) \cdot 2^{n-1}$. I. e. we prove that the evolutionary algorithm is significantly faster even in the worst-case than to search the whole search space of size $n! = 2^{\Theta(n \log n)}$ in order to find an optimum solution. Clearly, this does not yield a practical algorithm, but the aim of this article is to take the logical step from analyzing evolutionary algorithms yielding exact solutions on "easy" problems in $\mathcal{P}$ to notoriously hard problems in $\mathcal{NP}$.

The results will help to gain a deeper understanding of how evolutionary algorithms can use the underlying structure of the problem to find an optimum solution without knowing the structure of the problem itself. We find, that the evolutionary algorithm is capable of acting as a randomized variant of a dynamic program. Such a behavior has recently been proven in [3] based on a population-based evolutionary algorithm, though the authors do not explicitely state it.

2 An EA for the TSP

In the traveling salesperson problem *TSP* we are given a set of cities $v_1, \ldots v_n$ and a distance $w(v_i, v_j)$ for each pair of cities v_i, v_j. The goal is to find an ordering π of the cities that minimizes the total cost of the round-trip $\sum_{i=1}^{n} w(v_{\pi(i)}, v_{\pi(j)}) + w(v_{\pi(n)}, v_{\pi(1)})$. The input to the TSP problem induces a complete graph $G = (V, E)$ with $V = \{v_1, \ldots, v_n\}$ having weighted edges according to the distances between cities.

The two problems Hamiltonian cycle and Hamiltonian path are two closely related problems to the TSP. The task to find an Hamiltonian cycle in a given graph is equivalent to finding a TSP tour in G regardless of its length. A Hamiltonian path in a given graph is a simple path having $n-1$ vertices with each vertex lying on the path exactly once.

The remainder of this section introduces all necessary modules of the $(\mu+1)$-EA for the solution of the TSP problem. The $(\mu+1)$-EA is finally defined in section 2.4.

2.1 Individuals and Population

An individual $x = (S, p)$ is defined on a set of vertices $S \subseteq V \backslash \{v_1\}$ with a dedicated vertex p taken from S. x is a permutation π_S of the nodes from $S \cup \{v_1\}$ with $\pi_S(|S|) := p$ and $\pi_S(i) \in S$. The TSP tour C_x on x is given by $C_x = (c_0 = v_1, c_1 = v_{\pi(1)}, \ldots, c_k = p, c_{k+1} = v_1)$. That is, the tour starts in a fixed vertex v_1, then runs over all nodes from $S \backslash \{p\}$ in an arbitrary order ending in a given vertex p before returning to vertex v_1. The definition of the mutation

operator will make sure that only feasible individuals being Hamiltonian paths on their specific ground set S are created.

The full population $\mathcal{P}$ consists of every possible individual $x = (S, p)$ with vertex set $S \subseteq V \backslash \{v_1\}$ and vertex $p \in S$, i.e. each individual represents a Hamiltonian path on S. As there are exactly $k \cdot \binom{n-1}{k}$ individuals of size k, for $k \in \{1, \ldots, n-1\}$ the population has size $\mu = \sum_{k=1}^{n-1} k \cdot \binom{n-1}{k}$. We will use $\mu \leq (n-1) \cdot 2^{n-1}$ as an upper bound on the size of the population.

2.2 Fitness and Selection

A fitness function $f : \mathcal{P} \to \mathbb{R}$ assigns each individual a non-negative real value which is called its fitness. The fitness of an individual $x = (S, p)$ is given by the weight of the Hamiltonian path $f(x) := w(v_1, v_{\pi(1)}) + \sum_{i=1}^{|S|-1} w(v_{\pi(i)}, v_{\pi(i+1)})$. The individual is assigned a fitness $+\infty$ if it does not represent a Hamiltonian path.

The aim of the selection modules in an evolutionary algorithm is twofold. First, individuals need to be selected from the population in order to be changed by a variation operator (cf. line 2 of Algorithm 1). And second, the population needs to be prevented from growing too large while at the same time the "right" individuals have to remain in the population (cf. line 12 and 13 in Algorithm 1). In our case the selection operator will choose an individual from the population uniformly at random for the application of the mutation operator. It will also act as a diversity mechanism to ensure that each possible individual is contained in the population at most once. For each pair $x = (S, p)$ only the fittest will stay in the population.

In fact this means that the second selection step (line 13 of algorithm 1) of the $(\mu + 1)$-EA only needs to compare the fitness of individuals which are identical with respect to their ground set S and vertex p.

2.3 Mutation

A typical mutation operator changes the representation of an individual only slightly. For example an individual which is represented by a bitstring of length n is changed by flipping each bit uniformly at random with probability $\frac{1}{n}$. This is obviously not feasible for a path representation. However, the behavior of the standard bit mutation can be simulated by a mechanism, that was proposed in [19]. A number s is chosen at random according to a Poisson distribution with parameter $\lambda = 1$, i. e. Pois($\lambda = 1$). An elementary mutation is then applied $s+1$ times to an individual which has been selected. The mutation works as follows: Let $p \in V$ be the end vertex of the Hamiltonian path P of individual $x = (S, p)$. A node q is chosen uniformly at random from the set $V \backslash \{v_1\}$. If the edge (p, q) is already contained in the path, remove it from the individual, otherwise append it at the corresponding end of the individual. It is reasonable to use the Poisson distribution with $\lambda = 1$ because it is the limit of the binomial distribution for n trials each having probability $\frac{1}{n}$ in the bitstring model. Please note that the mutation operator is able to insert a vertex into an individual which is already

present and thus creating an infeasible individual. Such an individual will have fitness $+\infty$ as assigned by the fitness function. Producing infeasible individuals does not pose a problem, because we will see later that there always is a mutation creating a feasible individual.

2.4 $(\mu + 1)$-EA

The aim of the optimization process is to minimize the fitness of individuals having size $n - 1$. That is the optimization process has finished after every individual $(\{v_2, \ldots, v_n\}, p)$ $\forall p \in \{v_2, \ldots, v_n\}$ has been minimized.

For the analysis of the optimization time of an evolutionary algorithm it is important to note that we will only count the number of fitness evaluations, that is the number of iterations of the algorithm. This is a common measure in the algorithm community because it neglects the number of steps for the evaluation of the problem-specific fitness function.

We now bring together the above defined modules in the definition of a concrete $(\mu + 1)$-EA in Algorithm 1.

Algorithm 1. $(\mu + 1)$-EA

Initialize the population $\mathcal{P}$ with μ different individuals $x_1, \ldots, x_\mu$ with $x_i = (S, p)$, $S \subseteq V \backslash \{v_1\}$ and $p \in S$
while true **do**
  Mutation
  Select individual $z = (S, v) \in \mathcal{P}$ uniformly at random
  Choose $s := Pois(\lambda = 1)$.
  Generate new individual $z' = (S', v')$ by $s + 1$ times applying the
  mutation operator
  Selection
  Let $z'' = (S', v') \in \mathcal{P}$ be the individual defined on the same ground set having the same end vertex if such an individual exists in the population
  if $w(z') \leq w(z'')$ **then**
    Add z' to $\mathcal{P}$ and remove z'' from $\mathcal{P}$
  end if
end while

In the first step of the algorithm the initialization of the population takes places. In each iteration the evolutionary algorithm using only mutation selects an individual uniformly at random. It then applies the above described mutation operator by choosing a Poisson-distributed variable s.

The newly created individual $z' := (S', v)$ is inserted into the population. But as it is not reasonable to have more than one individual (S', v) in the population, the next selection step ensures the above discussed diversity by only keeping the fitter individual in the population.

Formally this algorithm is an infinite loop, but we aim at the analysis of the first time when the $(n - 1)$ possible Hamiltonian paths on $(\{2, \ldots, n\}, p)$ with

$p \in \{2, \ldots, n\}$ have been minimized with respect to their fitness. Taking the fittest individual we get an optimal solution of the TSP problem.

3 Analysis of the $(\mu + 1)$-EA

In this section we show the worst case optimization time of the $(\mu + 1)$-EA is $O(n^3 \cdot 2^n)$ with probability at least $(1 - e^{\Omega(n)})$- this is also known to be "with overwhelming probability".

The initialization of the population in step 1 of Algorithm 1 takes time $O(n^2 \cdot 2^n)$ which is linear in the number of individuals and their size. This step of the algorithm poses no problem in the proof of the runtime.

Proposition 1. *The $(\mu + 1)$-EA works correctly, that is it will always find an optimal solution to the TSP problem.*

Proof. After the initialization there is an individual in the population for every possible pair (S, p), $S \subseteq V \backslash \{v_1\}$ with $p \in S$. Due to the way the selection works in line 10 of Algorithm 1, there is always at least an individual defined on a pair (S, p), and its fitness can never increase. An individual of optimal fitness $OPT(S, p)$ is then defined by the Bellman principle $OPT(S, p) = \min\{OPT(S \backslash \{p\}, q) + w(q, p), \forall q \in S \backslash \{p\}\}$. Every path has the the so-called optimal substructure property, meaning that in order to have a Hamiltonian path on (S, p) with minimal length, every sub-path also has to be a Hamiltonian path of minimal length. Thus, starting with all optimal individuals $(\{2\}, 2), \ldots, (\{n\}, n)$, there always is a possibility to successively create optimal individuals defined on larger subsets by the right mutation step. The overall optimal TSP tour can easily be read off by taking an individual $x^* = \arg\min_{p \in \{2,\ldots,n\}} f((\{v_2, \ldots, v_n\}, p)) + w(p, 1)$, for which the associated TSP tour is optimal. □

Bounding the optimization time of the algorithm from above, we can make some pessimistic assumptions on the way the algorithm works. An optimal individual (S, p) of size $|S| = k$ with $(v_0 = 1, v_1, v_2, \ldots, v_{k-1}, v_k = p, v_{k+1} = 1)$ being the associated Hamiltonian cycle can be created if the evolutionary algorithm performs the following sequence of steps: At first individual $(\{v_1\}, v_1)$ is chosen in the selection step and then is extended in the mutation step by at least vertex v_2. The created individual $(\{v_1, v_2\}, v_2)$ is an optimal one with respect to the set $\{v_1, v_2\}$ with vertex v_3 being the end vertex of the Hamiltonian path on set $\{v_1, v_2\}$. This holds because it is a sub-path of the optimal Hamiltonian path $(v_1, v_2, \ldots, v_{k-1}, v_k = p)$, that is we are using the optimal substructure-property of paths. This procedure is repeated for the subsequent vertices until the path has been created. The described procedure is again a pessimistic view on the way the evolutionary algorithm is capable of optimizing an individual of size k, because an individual can be extended by more than one vertex in a mutation step.

The proof of the runtime will be based on mutations with $s = 0$ only, which only slows down the considered process. For the proof we will need Chernoff

bounds (cf. [13, Chapter 4]) and the Union Bound(cf. [13, Chapter 3.1]), which we state here for reasons of simplicity.

Theorem 3.1. *Let $X_1, \ldots, X_t$ be mutually independent random variables with $X_i \in \{0,1\}$ for all $i \in \{1,\ldots,t\}$. Let $X := X_1 + \ldots + X_n$.*

$$\forall 0 < \delta < 1: \ \Pr(X < (1-\delta) \cdot E(X))) < \exp\left(\frac{-E(X) \cdot \delta^2}{2}\right)$$

Theorem 3.2. *Let $\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n$ be arbitrary events.*

$$\Pr\left(\bigcup_{i=1}^{n} \mathcal{E}_i\right) \leq \sum_{i=1}^{n} \Pr(\mathcal{E}_i)$$

We will now upper bound the optimization time of all individuals of length k. For this we look at the expected optimization time for an arbitrary but fixed individual of length $2 \leq k \leq n-1$. We then find sharp bounds for the success probability to optimize this individual within a given time-interval. This bound in turn is used for a Union Bound argument, which bounds the overall failure probability not to create all individuals of size $2 \leq k \leq n-1$ within the time interval. Taking $k = n-1$ leads to the wanted optimization time to compute an optimal TSP tour.

Lemma 3.3. *Let $2 \leq k \leq n-1$ be the size of the ground set S an individual (S,p) is defined on. Then with probability at least $(1 - e^{\Omega(n)})$ all individuals of size k are optimized with respect to their fitness in $O(k \cdot n^2 \cdot 2^n)$ iterations of the $(\mu+1)$-EA.*

Proof. To create an arbitrary but fixed individual (S,p) of size $|S| = k$ with associated Hamiltonian cycle $(v_0 = 1, v_1, v_2, \ldots, v_k = p, v_{k+1} = 1)$, the evolutionary algorithm needs an expected number of $O(k \cdot n^2 \cdot 2^n)$ iterations. This can be seen as follows: The algorithm needs to select the appropriate path of length 1 from the population and then choose an appropriate vertex for the extension of the individual (and the associated path). The probability to select a specific individual from the population is bounded below by $\frac{1}{\mu} \geq \frac{1}{(n-1)\cdot 2^{n-1}}$. The probability to choose a specific vertex for the extension is depending on the outcome of the experiment to get $s = 0$ and then to choose the correct vertex out of at most $n-1$ vertices. The probability to have a mutation with $s = 0$ by definition of the Poisson distribution with $\lambda = 1$ equals e^{-1}. Thus, the probability to get a successful extension is bounded below by $\frac{1}{e\cdot(n-1)}$. We call a successful mutation of the type described above *necessary*. The probability to have a necessary mutation is thus $p := \frac{1}{e\cdot(n-1)^2\cdot 2^{n-1}}$ Altogether the expected number of iterations until the fitness of an element of size k has been optimized is upper bounded by $e \cdot k \cdot (n-1)^2 \cdot 2^{n-1} = O(k \cdot n^2 \cdot 2^n)$.

We will now show that a time interval of $O(k \cdot n^2 \cdot 2^n)$ suffices to optimize all individuals of size k with probability at least $(1 - e^{\Omega(n)})$.

By using Chernoff bounds we are able to bound the number of iterations sharply around the expectation value. Let $t := \alpha \cdot e \cdot k \cdot (n-1)^2 \cdot 2^{n-1}$ with $\alpha := c \cdot \frac{n}{k}$ and $c > 0$ constant be the time interval the $(\mu+1)$-EA is given to perform k necessary mutations. For every iteration i from within that time interval define the random variable $X_i \in \{0,1\}$ to be $X_i = 1$ if a necessary mutation has occurred in the i-th iteration of the algorithm. Then all the X_i are mutually independent, and we have $\Pr[X_i = 1] = p$. Let $X := X_1 + \ldots + X_t$, then its expectation value is $E[X] = t \cdot p = c \cdot n$.

As argued above the algorithms needs at least k necessary mutations to create an arbitrary but fixed optimal individual of size k. Thus, the algorithm will fail to create this individual if $X < k$. We will now bound the associated failure probability Pr[x was not optimized within t steps] $\leq \Pr(X < k)$ by using Chernoff bounds. This is possible because the event that the considered x has not been optimized within t iterations is composed of the conjunction of several possibilities that the algorithm fails, among which $X < k$ is just one.

Let $\delta := 1 - \frac{k}{E[X]}$ and it holds that $0 < \delta < 1$ and $1 \leq k \leq n-1$.

$$
\begin{aligned}
\Pr[X < k] &\overset{(1-\delta)=\frac{k}{E[X]}}{=} \Pr[X < (1-\delta)E[X]] \\
&\overset{\text{Theorem 3.1}}{<} \exp\left(-p \cdot t \cdot \left(1 - \frac{k}{p \cdot t}\right)^2 \cdot 2^{-1}\right) \\
&= \exp\left(-\left(c \cdot n - 2k + \frac{k^2}{n}\right) \cdot 2^{-1}\right) \\
&\leq \exp\left(-\frac{c \cdot n}{2} + k\right) \\
&\leq \exp(-c' \cdot n)
\end{aligned}
$$

Where in the last line we bounded $\frac{c}{2} + k$ by an appropriate constant $c' > c$, because $k < n-1$ holds.

Now using Theorem 3.2 we can upper bound the probability, that one of the $\ell := k \cdot \binom{n-1}{k}$ individuals $x_1, \ldots, x_\ell$ of size k is not optimized within t iterations.

$$
\begin{aligned}
&\Pr[x_1 \vee \ldots \vee x_\ell \text{ not optimized within } t \text{ iterations}] \\
\overset{\text{Theorem 3.2}}{\leq}\ & \sum_{i=1}^{\ell} \Pr[x_i \text{ not optimized within } t \text{ iterations}] \\
\leq\ & \sum_{i=1}^{\ell} \exp(-c' \cdot n) \\
=\ & k \cdot \binom{n-1}{k} \cdot \exp(-c' \cdot n) \\
\leq\ & \exp\left(\ln k + \ln\left(\frac{(n-1)^k}{k!}\right) - c' \cdot n\right) \\
\leq\ & \exp(\ln k + k \ln(n-1) - k \ln k - c' \cdot n \\
\leq\ & e^{-\Omega(n)}
\end{aligned}
$$

This term is dominated by $\exp(-c' \cdot n)$ for both cases of $k = o(n)$ and $k = O(n)$.

We have shown, that the wanted optimization time is at most $O(k \cdot n^2 \cdot 2^n)$ with probability at least $\left(1 - e^{\Omega(n)}\right)$. This finishes the proof. □

For $k = n$ we immediately get an upper bound of $O(n^3 \cdot 2^n)$ on the number of iterations until all possible Hamiltonian paths have been optimized with overwhelming probability.

Theorem 3.4. *The $(\mu + 1)$-EA finds all shortest Hamiltonian paths (S, p) with $S \subseteq V \backslash \{1\}$ and $p \in S$ in $O(n^3 \cdot 2^n)$ number of iterations with probability $(1 - e^{\Omega(n)})$.*

4 Conclusions and Outlook

In this article we proved for $\mu \leq O(n \cdot 2^n)$ that a $(\mu + 1)$-EA is able to solve a given TSP instance to optimality. The proven upper bound of $O(n^3 \cdot 2^n)$ is competitive to the best-known upper bound of $O(n^2 \cdot 2^n)$ of the dynamical programming approach. This was the first time it could theoretically be shown that a black-box algorithm is able to solve an $\mathcal{NP}$-hard problem to optimality.

The importance of a theoretical result of this flavor is due to the fact that the TSP is *the* classical $\mathcal{NP}$-hard combinatorial optimization problem with numerous applications. Every algorithmic idea is sooner or later put to test on the TSP (cf. [2, Chapter 4]). Many important techniques in integer and linear programming were developed because they proved valuable for the solution of the TSP, as the prototypical combinatorial optimization problem.

The development of practical algorithms for the solution of large TSP instances is an important topic in combinatorial optimization. We pose the question under which circumstances it is possible to speed up the proposed evolutionary algorithm by the use of a crossover operator. Moreover, we conducted some preliminary experiments which show that the EA is still able to find the optimal solution to the given instance if the size of the population is reduced - however, at the cost of an increased runtime. These questions will be dealt with in the future by an experimental evaluation of the evolutionary algorithm.

Acknowledgments

The author would like to thank Martin Skutella and Andreas Wiese for many useful discussions.

This article is dedicated to the memory of Ingo Wegener.

References

1. Applegate, D., Bixby, R., Chvátal, V., Cook, W.: Implementing the dantzig-fulkerson-johnson algorithm for large traveling salesman problems. Mathematical Programming 97, 91–153 (2003)
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: The Traveling Salesman Problem: A Computational Study (2007)
3. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. In: GECCO 2008: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 539–546. ACM, New York (2008)
4. Englert, M., Röglin, H., Vöcking, B.: Worst case and probabilistic analysis of the 2-opt algorithm for the TSP: extended abstract. In: Bansal, N., Pruhs, K., Stein, C. (eds.) SODA, pp. 1295–1304. SIAM, Philadelphia (2007)
5. Forrest, S.: Genetic algorithms. ACM Computing Surveys 28, 77–80 (1996)
6. Friedrich, T., Hebbinghaus, N., Neumann, F., He, J., Witt, C.: Approximating covering problems by randomized search heuristics using multi-objective models. In: GECCO 2007: Proceedings of the 9th annual conference on Genetic and evolutionary computation, pp. 797–804. ACM, New York (2007)
7. Giel, O., Wegener, I.: Evolutionary algorithms and the maximum matching problem. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 415–426. Springer, Heidelberg (2003)
8. Giel, O., Wegener, I.: Maximum cardinality matchings on trees by randomized local search. In: GECCO 2006: Proceedings of the 8th annual conference on Genetic and evolutionary computation, pp. 539–546. ACM, New York (2006)
9. Goldbarg, E.F.G., de Souza, G.R., Goldbarg, M.C.: Particle swarm for the traveling salesman problem. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 99–110. Springer, Heidelberg (2006)
10. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. In: Proceedings of the 1961 16th ACM national meeting, pp. 71.201–71.204. ACM Press, New York (1961)
11. Johnson, D.S., McGeoch, L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization, pp. 215–310. Wiley, Chichester (1997)
12. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the travelling-salesman problem. Operations Research 21, 498–516 (1973)
13. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
14. Nagata, Y.: Fast eax algorithm considering population diversity for traveling salesman problems. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 171–182. Springer, Heidelberg (2006)
15. Neumann, F., Reichel, J.: Approximating minimum multicuts by evolutionary multi-objective algorithms. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 72–81. Springer, Heidelberg (2008)
16. Neumann, F., Reichel, J., Skutella, M.: Computing minimum cuts by randomized search heuristics. In: Proc. of the 10th Genetic and Evolutionary Computation Conference (GECCO 2008), pp. 779–786 (2008)
17. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theoretical Computer Science 378(1), 32–40 (2007)

18. Reichel, J., Skutella, M.: Evolutionary algorithms and matroid optimization problems. In: Proc. of the 9th Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 947–954 (2007)
19. Scharnow, J., Tinnefeld, K., Wegener, I.: The analysis of evolutionary algorithms on sorting and shortest paths problems. Journal of Mathematical Modelling and Algorithms, 349–366 (2004)
20. Wegener, I.: Randomized search heuristics as an alternative to exact optimization. In: Lenski, W. (ed.) Logic versus Approximation. LNCS, vol. 3075, pp. 138–149. Springer, Heidelberg (2004)
21. Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 44–56. Springer, Heidelberg (2005)

Finding Balanced Incomplete Block Designs with Metaheuristics

David Rodríguez Rueda[1], Carlos Cotta[2], and Antonio J. Fernández[2]

[1] Universidad Nacional Experimental del Táchira (UNET), Laboratorio de Computación de Alto Rendimiento (LCAR), San Cristóbal, Venezuela
drodri@unet.edu.ve
[2] Universidad de Málaga, ETSI Informática, Campus de Teatinos, 29071 Málaga, Spain
{ccottap,afdez}@lcc.uma.es

Abstract. This paper deals with the generation of balanced incomplete block designs (BIBD), a hard constrained combinatorial problem with multiple applications. This problem is here formulated as a combinatorial optimization problem (COP) whose solutions are binary matrices. Two different neighborhood structures are defined, based on bit-flipping and position-swapping. These are used within three metaheuristics, i.e., hill climbing, tabu search, and genetic algorithms. An extensive empirical evaluation is done using 86 different instances of the problem. The results indicate the superiority of the swap-based neighborhood, and the impressive performance of tabu search. This latter approach is capable of outperforming two techniques that had reported the best results in the literature (namely, a neural network with simulated annealing and a constraint local search algorithm).

1 Introduction

The generation of block designs is a well-known combinatorial problem, which is very hard to solve [1]. The problem has a number of variants, among which a popular one is the so-called Balanced Incomplete Block Designs (BIBDs). Basically, a BIBD is defined as an arrangement of v distinct objects into b blocks such that each block contains exactly k distinct objects, each object occurs in exactly r different blocks, and every two distinct objects occur together in exactly λ blocks (for $k, r, \lambda > 0$). The construction of BIBDs was initially attacked in the area of experiment design [2,3]; however, nowadays BIBD can be applied to a variety of fields such as cryptography [4] and coding theory [5], among others.

BIBD generation is a NP-hard problem [6] that provides an excellent benchmark since it is scalable and has a wide variety of problem instances, ranging from easy instances to very difficult ones. The scalability of the problem as well as its difficulty make it an adequate setting to test the behavior of different techniques/algorithms. As it will be discussed in Sect. 2.2, complete methods (including exhaustive search) have been applied to the problem although this remains intractable even for designs of relatively small size [7]. As a proof of

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 156–167, 2009.

the difficulty of the problem, there currently exist a number of open instances that have not been solved yet (of course, it might be the case that there is no solution for them; then again, insolvability could not be established by complete methods). The fact that, in the general case, the algorithmic generation of block designs is an NP-hard problem [6] makes complete methods be inherently limited by the size of the problem instances. The application of metaheuristics thus seems to be more appropriate to attack larger problem instances. This paper provides some steps in this direction and demonstrates empirically that these approaches (particularly, local search techniques) are effective methods in the design of balanced incomplete blocks. More specifically, the paper describes two local searchers –i.e., a steepest descent hill climbing (HC) algorithm and a tabu search (TS)– and a genetic algorithm (GA), each of them with two variants. A wide range of problem instances have been tackled by these metaheuristics, and the results have been compared with two techniques found in the literature that reported the best results [8,9]. In the following we will show that a particular TS algorithm outperforms the remaining approaches and even can find solutions to instances that the other methods could not solve.

2 Background

This section provides a brief overview of the problem, presents its classical formulation, and discusses how it has been tackled in the literature.

2.1 Formulation

A standard way of representing a BIBD is in terms of its incidence matrix $M \equiv \{m_{ij}\}_{v\times b}$, which is a $v \times b$ binary matrix with exactly r ones per row, k ones per column, a scalar product of λ between any pair of distinct rows, and where $m_{ij} \in \{0, 1\}$ is equal to 1 if the ith object is contained in the jth block, and 0 otherwise; in this context, m_{ij} represents the incidence of object i in block j of M. A BIBD is then specified by five parameters $\langle v, b, r, k, \lambda\rangle$, i.e., a $\langle v, b, r, k, \lambda\rangle$-BIBD consists of a set of v points that is divided into b subsets in such a way that each point in v is contained in r different subsets and any couple of points in v is contained in $\lambda < b$ subsets with $k < v$ points in each subset.

The five parameters defining a $\langle v, b, r, k, \lambda\rangle-$BIBD are related and satisfy the following two relations: $bk = vr$ and $\lambda(v-1) = r(k-1)$. In fact, the corresponding instance can be defined by just three parameters $\langle v, k, \lambda\rangle$ since b and r are given in terms of the other parameters:

$$b = \frac{v(v-1)\lambda}{k(k-1)} \qquad r = \frac{(v-1)\lambda}{k-1} \tag{1}$$

Clearly, these relations restrict the set of admissible parameters for a BIBD; however, the parameter admissibility is a necessary condition but it is not sufficient to guarantee the existence of a BIBD [10,11]. According to the Fisher's inequality theorem [12], $b > v$ in any block design; the case $b = v$ represents

$$\begin{array}{c}
0\,0\,0\,1\,0\,1\,1\,1\,0\,0\,0\,1\,1\,1\\
1\,1\,0\,1\,1\,0\,1\,0\,0\,1\,0\,0\,0\,1\\
0\,1\,1\,1\,1\,1\,0\,0\,1\,0\,0\,0\,1\,0\\
0\,0\,0\,0\,1\,1\,1\,1\,1\,1\,1\,0\,0\,0\\
1\,0\,1\,0\,1\,1\,0\,0\,0\,0\,1\,1\,0\,1\\
0\,1\,0\,0\,0\,0\,0\,0\,1\,1\,1\,1\,1\,1\\
1\,1\,1\,0\,0\,0\,1\,1\,0\,0\,1\,0\,0\,0\\
1\,0\,1\,1\,0\,0\,0\,1\,0\,1\,1\,0\,1\,0
\end{array}
\qquad
\begin{array}{c}
0\,1\,0\,1\,0\,1\,0\\
1\,0\,0\,1\,0\,0\,1\\
1\,1\,1\,0\,0\,0\,0\\
0\,0\,1\,0\,0\,1\,1\\
0\,1\,0\,0\,1\,0\,1\\
1\,0\,0\,0\,1\,1\,0\\
0\,0\,1\,1\,1\,0\,0
\end{array}$$

Fig. 1. (Left) a $\langle 8, 14, 7, 4, 3\rangle-$BIBD; (Right) a $\langle 7, 7, 3, 3, 1\rangle-$symmetric BIBD

an special design called *symmetric design*. A direct consequence of a symmetric design is that $r = k$. This kind of blocks are usually used with a maximum order of $v = b = 7$, although this is not a strict requirement. Figure 1 shows configurations of the incidence matrix M representing possible solutions to a $\langle 8, 14, 7, 4, 3\rangle-$BIBD and a symmetric $\langle 7, 7, 3, 3, 1\rangle-$BIBD, respectively.

2.2 Related Work

The BIBD problem has been tackled by a number of different techniques in the literature, with different success. Traditionally, the problem was dealt via deterministic, constructive and/or complete methods. For instance John *et al.* [13,14] used mathematical programming methods to look for an optimal incomplete block design. Also, Zergaw [15] considered the error correlation, and presented a sequential algorithm for constructing optimal block designs. Following this line of work, Tjur [16] incorporated interchange mechanisms via the addition of experimental units (blocks) one by one. Flener *et al.* [17] proposed a matricial model based on ECLIPSE to solve the problem of block generation. Also, constraint programming techniques have been used; this way, Puget [18] formulated the problem as a constraint satisfaction problem (CSP) where each instance was represented by a classical binary matrix of size $v \times b$. Puget proposed to combine methods for *symmetry breaking via dominance detection* and *symmetry breaking using stabilizers* in order to solve the problem. Also, [19] explored two strategies (namely, a heuristic for variable selection and a domain pruning procedure) for exploiting the symmetry of the problem. The underlying idea in this work was to use symmetries to guide the search for a solution. The objective of this work was not solving specific instances but being effective in reducing search effort. Be as it may, although all these methods can be used to design BIBDs, their applicability is limited by the size of the problem instances. A survey of known results can be found in [1].

Stochastic methods were also applied to the problem. For example, the generation of BIBDs is formulated in [8] as a COP tackled with a neural network. Several optimization strategies were considered as relaxation strategies for comparative purposes. A simulated annealing algorithm endowed with this neural network (NN-SA) was shown to offer better performance than an analogous hybridization with mean field annealing. These results were further improved by Prestwich [20,9], that considered different schemes for adding symmetry breaking constraints inside a constrained local search (CLS).

In general, most of the proposals to generate BIBDs are focused in unsolved problems and consider a small number of instances, and only a small number of papers provide an extensive experimentation on a large set of instances. Among these papers, the most interesting ones are [8,9,19]. To the best of our knowledge, [9] provides the best results published in many instances of the problem and therefore, represents the state-of-the-art in the generation of BIBDs. For these reasons the NN-SA and CLS proposals will be later considered in the experimental section of this paper for comparative purposes with the methods described in this paper.

Let us finally mention that there exist other variants of the BIBD problem, e.g., partially BIBDs, randomized block designs, pairwise balanced designs, regular graph designs, and maximally balanced maximally uniform designs, among others [21,22,23,24]. Although in some cases metaheuristic approaches have been used on some of them [25,26,27], to the best of our knowledge there exists no previous literature on this line of attack for the BIBD problem we consider in this work (save the SA approach mentioned before).

3 Solving the $\langle v, b, r, k, \lambda \rangle$-BIBD Problem

The BIBD problem exhibits a clear combinatorial structure, and can be readily transformed in an optimization task. We have approached this challenging resolution via two local search techniques (HC and TS) and a population-based technique (GA), which will be described below. To this end, let us firstly define the objective function, and possible neighborhood structures.

3.1 Objective Function

The generation of BIBDs is a CSP posed here as a COP. This is done by relaxing the problem (allowing the violation of constraints) and defining an objective function that accounts for the number and degree of violation of them. More precisely, for the general case of the instance $\langle v, b, r, k, \lambda \rangle$, the following objective function is defined:

$$f^{\langle v,b,r,k,\lambda \rangle}(M) = \sum_{i=1}^{v} \phi_{ir}(M) + \sum_{j=1}^{b} \phi'_{jk}(M) + \sum_{i=1}^{v-1} \sum_{j=i+1}^{v} \phi''_{ij\lambda}(M) \qquad (2)$$

where

$$\phi_{ir}(M) = \left| r - \sum_{j=1}^{b} m_{ij} \right| ;\ \phi'_{jk}(M) = \left| k - \sum_{i=1}^{v} m_{ij} \right| ;\ \phi''_{ij\lambda}(M) = \left| \lambda - \sum_{k=1}^{b} m_{ik} m_{jk} \right| \qquad (3)$$

Observe that, for a given incidence matrix M, the value returned by the objective function sums up all discrepancies with respect to the expected values of the row constraints, column constraints and scalar product constraints. Obviously, the aim is to minimize the value of the objective function. If the instance is satisfiable, a global optimum is a configuration M^* such that $f^{\langle v,b,r,k,\lambda \rangle}(M^*) = 0$.

3.2 Neighborhood Structures

Two neighborhood structures are considered. The first one arises naturally from the binary representation of solutions as the incidence matrix M. This neighborhood is based on the Hamming distance, and will be denoted as bit-flip. Let $H(M_1, M_2)$ be the Hamming distance between two incidence matrices M_1 and M_2; the bit-flip neighborhood is defined as $\mathcal{N}_{bit-flip}(M) = \{M' \mid \text{H(M,M')=1}\}$. Clearly the size of this neighborhood is $|\mathcal{N}_{bit-flip}(M)| = vb$, and the evaluation of any $M' \in \mathcal{N}_{bit-flip}(M)$ requires the incremental re-computation (with respect to the evaluation of M) of $v + 1$ constraints (i.e., 1 row constraint + 1 column constraint + $v - 1$ scalar products). Observe that evaluating a solution from scratch requires to compute exactly $v + b + v(v - 1)/2$ constraints, and thus the complete exploration of the neighborhood can be assimilated to $n_{eq} = \frac{vb(v+1)}{v+b+v(v-1)/2}$ full evaluations. This consideration will be useful in order to provide a fair basis for comparing local search and population-based techniques later on, i.e., by taking constraint checks as a measure of computational effort. While this measure can admit several nuances, it is more informative than the number of solutions generated, and more hardware-independent than, e.g., running time.

A second neighborhood structure –which we denote as swap– can be considered as well. The underlying idea here is to take an object from one block, and move it to a different one. This can be formulated in binary terms as permuting a 0 and a 1 within the same row. Notice that by doing so, if a configuration holds the row constraint for a specific row, then all its neighbors will also hold it. The swap neighborhood is defined as $\mathcal{N}_{swap}(M) = \{M' \mid \exists! i, j, k : \ m_{ij} = m'_{ik} = 0, m'_{ij} = m_{ik} = 1\}$. Clearly, the size of this neighborhood is $|\mathcal{N}_{swap}(M)| = vr(b-r)$, from which the number of evaluations to explore the complete neighborhood can be directly inferred. Note that row constraints do not have to be re-evaluated as the number of 1's per row remains constant. In any case, the impact of this consideration is minimal since the computing effort is dominated by the quadratic term in the denominator. Let us note as a final consideration that a symmetrical version of this latter neighborhood could be defined, substituting an object by another different one within a block. Notice however that the objective function is not symmetrical in this sense, and the cost of exploring this neighborhood is higher (and it exhibits other difficulties when deployed on a GA, as $\mathcal{N}_{swap}$ will be in Sect. 3.4). For this reason, it has not been considered in this work.

3.3 Local Search Techniques

Two different versions of a hill climbing (HC) approach and a tabu search (TS) algorithm were defined on the basis of the two neighborhood structures. These are denoted as HC_{bf}, HC_{sw}, TS_{bf} and TS_{sw} respectively. Besides the obvious differences in algorithmic aspects and neighborhood computation, there is an additional consideration regarding the choice of neighborhood: since the swap neighborhood does not alter the number of 1's per row, if the initial solution

does not fulfill all row constraints no feasible solution will be ever found. Hence, it is mandatory to enforce these constraints when generating the starting point for a swap-based local search algorithm. Their bit-flip-based counterparts do not require this, and can take a fully random solution as initial point for the search. Nevertheless, the effect that such a guided initialization can have on these latter algorithms has been empirically studied as well in Sect. 4.

The HC algorithms follow a steepest-descent procedure: the neighborhood of the current solution is completely explored, and the best solution is chosen unless this is worse than the current one; if this is the case, the current solution is a local optimum, and the process is re-started from a different point (randomly chosen) until the computational budget allocated is exhausted. Regarding the TS algorithms, they also conduct a full-exploration of the current neighborhood, moving to the best non-tabu neighbor even if it is worse than the current solution. In the case of TS_{bf}, a move is tabu if it modifies a specific bit m_{ij} stored in the tabu list. Similarly, in the case of TS_{sw}, a move is tabu if it attempts to reverse a previous swap $m_{ij} \leftrightarrow m_{ik}$ stored in the tabu list. To prevent cycling, the tabu tenure –i.e., the number of iterations tabu move stays in the list– is chosen randomly in the range $[\beta/2, 3\beta/2]$, where $\beta = vb$ in TS_{bf} and $\beta = vbr$ in TS_{sw}. The tabu status of a move can be overridden if the aspiration criteria is fulfilled, namely, finding a solution better than the current best solution found so far. After a number of n_ι evaluations (a parameter that we will set as a function of the total number of evaluations) with no improvement, the search is intensified, by returning to the best solution found so far.

3.4 Genetic Algorithm

Two versions of a steady state GA have been considered. Both of them use binary tournament selection and replacement of the worst individual in the population. They differ in the reproductive stage though. The first one, which we denote as GA_{bf}, is related to the bit-flip neighborhood, since it uses uniform crossover and bit-flip mutation. The second one is denoted as GA_{sw}, and is more related to the swap neighborhood. To be precise, this latter algorithm performs uniform crossover at row level (that is, it randomly selects entire rows from either of the parents), and uses swap mutation. Obviously, this implies that the unitation of each row is never changed, and therefore the initialization of the population has to be done with solutions fulfilling all row constraints, as it was the case with HC_{sw} and TS_{sw}. Again, this guided initialization can be optionally done in GA_{bf}, although it is not mandatory.

To keep diversity in the population, both GA variants ban duplicated solutions, i.e., if an offspring is a copy of an existing solution it is discarded. Furthermore, a re-starting mechanism is introduced to re-activate the search whenever stagnation takes place. This is done by keeping a fraction $f_\%$ of the top individuals in the current population, and refreshing the rest of the population with random individuals. This procedure is triggered after a number of n_ι evaluations with no improvement of the current best solution.

4 Experimental Results

The experiments have been done on 86 instances taken from [8,9] where $vb \leqslant 1000$ and $k \neq 3$. This corresponds to the hardest instances reported therein, since the cases where $k = 3$ were easily solvable. Table 1 shows the particular instances considered, along with an identification label, and their solvability status regarding the NN-SA [8] and CLS [9] algorithm. Although the data reported in [9] is limited to the best result out of 3 runs of CLS per instance, it must be noted that the number of instances solved by the latter algorithm is more than 3 times that of NN-SA.

All algorithms have been run 30 times per problem instance. To make the comparison with CLS as fair as possible, all runs of local search techniques are limited to explore $n_\nu = 2 \cdot 10^6$ neighbors. This number correspond to the maximum number of backtrack steps (fixing one entry of the incidence matrix) performed by CLS in [9]. The GAs consider the equivalent number of full evaluations in each case (see Sect. 3.2). The number of evaluations without improvement to trigger intensification in TS or re-starting in GA is $n_\iota = n_\nu/10$. Other parameters of the

Table 1. BIBD instances considered in this work, and their solvability status with respect to the simulated annealing/neural network hybrid algorithm (NN-SA) in [8], and the constrained local search algorithm (CLS) in [9]

ID	v	b	r	k	λ	vb	NN-SA	CLS	ID	v	b	r	k	λ	vb	NN-SA	CLS
1	8	14	7	4	3	112	yes	yes	44	25	25	9	9	3	625	no	no
2	11	11	5	5	2	121	yes	yes	45	15	42	14	5	4	630	no	yes
3	10	15	6	4	2	150	yes	yes	46	21	30	10	7	3	630	no	no
4	9	18	8	4	3	162	yes	yes	47	16	40	10	4	2	640	no	yes
5	13	13	4	4	1	169	yes	yes	48	16	40	15	6	5	640	no	no
6	10	18	9	5	4	180	yes	yes	49	9	72	32	4	12	648	no	yes
7	8	28	14	4	6	224	yes	yes	50	15	45	21	7	9	675	no	no
8	15	15	7	7	3	225	yes	yes	51	13	52	16	4	4	676	no	yes
9	11	22	10	5	4	242	yes	yes	52	13	52	24	6	10	676	no	yes
10	16	16	6	6	2	256	yes	yes	53	10	72	36	5	16	720	no	yes
11	12	22	11	6	5	264	no	yes	54	19	38	18	9	8	722	no	no
12	10	30	12	4	4	300	yes	yes	55	11	66	30	5	12	726	no	yes
13	16	20	5	4	1	320	yes	yes	56	22	33	12	8	4	726	no	no
14	9	36	16	4	6	324	yes	yes	57	15	52	26	7	12	780	no	no
15	8	42	21	4	9	336	no	yes	58	27	27	13	13	6	729	no	no
16	13	26	8	4	2	338	yes	yes	59	21	35	15	9	6	735	no	no
17	13	26	12	6	5	338	no	yes	60	10	75	30	4	10	750	no	yes
18	10	36	18	5	8	360	no	yes	61	25	30	6	5	1	750	no	yes
19	19	19	9	9	4	361	no	yes	62	20	38	19	10	9	760	no	no
20	11	33	15	5	6	363	no	yes	63	16	48	15	5	4	768	no	yes
21	14	26	13	7	6	364	no	no	64	16	48	18	6	6	768	no	no
22	16	24	9	6	3	384	no	yes	65	12	66	22	4	6	792	no	yes
23	12	33	11	4	3	396	yes	yes	66	12	66	33	6	15	792	no	yes
24	21	21	5	5	1	441	yes	yes	67	9	90	40	4	15	810	no	yes
25	8	56	28	4	12	448	no	yes	68	13	65	20	4	5	845	no	yes
26	10	45	18	4	6	450	no	yes	69	11	77	35	5	14	847	no	yes
27	15	30	14	7	6	450	no	no	70	21	42	10	5	2	882	no	no
28	16	30	15	8	7	480	no	no	71	21	42	12	6	3	882	no	no
29	11	44	20	5	8	484	no	yes	72	21	42	20	10	9	882	no	no
30	9	54	24	4	9	486	no	yes	73	16	56	21	6	7	896	no	no
31	13	39	12	4	3	507	no	yes	74	10	90	36	4	12	900	no	yes
32	13	39	15	5	5	507	no	yes	75	15	60	28	7	12	900	no	no
33	16	32	12	6	4	512	no	no	76	18	51	17	6	5	918	no	no
34	15	35	14	6	5	525	no	no	77	22	42	21	11	10	924	no	no
35	12	44	22	6	10	528	no	yes	78	15	63	21	5	6	945	no	yes
36	23	23	11	11	5	529	no	no	79	16	60	15	4	3	960	no	yes
37	10	54	27	5	12	540	no	yes	80	16	60	30	8	14	960	no	no
38	8	70	35	4	15	560	no	yes	81	31	31	6	6	1	961	no	yes
39	17	34	16	8	7	578	no	no	82	31	31	10	10	3	961	no	no
40	10	60	24	4	8	600	no	yes	83	31	31	15	15	7	961	no	no
41	11	55	20	4	6	605	no	yes	84	11	88	40	5	16	968	no	yes
42	11	55	25	5	10	605	no	yes	85	22	44	14	7	4	968	no	no
43	18	34	17	9	8	612	no	no	86	25	40	16	10	6	1000	no	no

Table 2. Results of HC algorithms (30 runs per instance). $\overline{x}$, σ, B and S denote, respectively, the fitness average value, the standard deviation, the best obtained result, and the number of times that a problem instance solution is obtained.

	HC_{bf}			HC_{sw}		
ID	$\overline{x} \pm \sigma$	B	S	$\overline{x} \pm \sigma$	B	S
1	3.50 ± 1.12	0	2	0.00 ± 0.00	0	30
2	5.13 ± 4.62	0	13	0.00 ± 0.00	0	30
3	5.83 ± 1.24	4	0	0.00 ± 0.00	0	30
4	5.73 ± 1.44	0	1	0.00 ± 0.00	0	30
5	3.13 ± 4.15	0	19	0.00 ± 0.00	0	30
6	11.27 ± 1.63	7	0	1.47 ± 1.93	0	19
7	7.80 ± 1.19	5	0	0.00 ± 0.00	0	30
8	36.60 ± 2.68	30	0	4.33 ± 5.17	0	17
9	14.67 ± 1.70	10	0	3.97 ± 0.84	0	1
10	37.23 ± 1.67	33	0	4.47 ± 5.67	0	18
11	22.27 ± 2.14	16	0	6.13 ± 1.26	4	0
12	11.30 ± 1.35	9	0	2.53 ± 1.93	0	11
13	20.67 ± 2.07	16	0	8.40 ± 4.12	0	5
14	11.20 ± 1.66	7	0	0.27 ± 1.00	0	28
15	12.07 ± 2.03	8	0	0.00 ± 0.00	0	30
16	16.17 ± 2.08	12	0	6.13 ± 1.06	4	0
17	28.53 ± 2.20	22	0	9.63 ± 1.58	6	0
18	19.67 ± 1.96	14	0	2.67 ± 1.89	0	10
19	78.00 ± 2.49	73	0	45.50 ± 3.38	35	0
20	19.17 ± 2.18	15	0	4.10 ± 1.35	0	2
21	38.77 ± 3.09	33	0	13.37 ± 2.26	6	0
22	40.47 ± 2.05	35	0	19.80 ± 2.17	15	0
23	16.30 ± 1.62	13	0	5.00 ± 1.18	4	0
24	48.87 ± 4.57	34	0	23.30 ± 7.20	0	1
25	19.20 ± 3.52	12	0	0.00 ± 0.00	0	30
26	15.37 ± 1.80	9	0	2.00 ± 2.00	0	15
27	46.27 ± 3.00	41	0	17.00 ± 2.42	11	0
28	59.83 ± 3.88	52	0	22.70 ± 2.52	16	0
29	24.00 ± 2.52	19	0	3.87 ± 1.73	0	4
30	17.63 ± 2.51	13	0	0.67 ± 1.49	0	25
31	20.63 ± 2.37	16	0	6.37 ± 1.94	4	0
32	26.97 ± 2.47	20	0	8.63 ± 1.74	5	0
33	43.53 ± 2.28	37	0	20.67 ± 2.56	15	0
34	40.87 ± 2.60	35	0	16.43 ± 2.68	10	0
35	36.20 ± 3.91	28	0	6.13 ± 1.67	4	0
36	135.73 ± 3.86	128	0	84.43 ± 4.10	72	0
37	27.83 ± 3.61	20	0	3.53 ± 1.43	0	4
38	29.00 ± 4.93	16	0	0.13 ± 0.72	0	29
39	67.13 ± 4.35	57	0	28.17 ± 2.00	23	0
40	19.40 ± 3.02	11	0	2.67 ± 1.89	0	10
41	19.23 ± 2.43	14	0	4.10 ± 1.60	0	3
42	26.90 ± 3.22	20	0	4.53 ± 1.67	0	2
43	85.57 ± 3.60	79	0	34.93 ± 3.22	28	0
44	139.67 ± 3.80	130	0	100.40 ± 3.08	95	0
45	35.33 ± 3.17	29	0	15.00 ± 2.14	11	0
46	84.60 ± 3.53	78	0	50.63 ± 3.22	45	0
47	29.63 ± 2.74	23	0	13.10 ± 2.41	8	0
48	47.57 ± 2.63	43	0	20.47 ± 2.60	11	0
49	25.93 ± 3.80	13	0	2.00 ± 2.00	0	15
50	54.90 ± 4.43	41	0	17.60 ± 2.09	13	0
51	24.67 ± 2.44	16	0	7.63 ± 1.62	4	0
52	42.20 ± 3.00	36	0	10.03 ± 2.76	4	0
53	38.87 ± 4.54	29	0	3.17 ± 2.18	0	9
54	96.27 ± 4.14	88	0	42.00 ± 3.75	33	0
55	31.93 ± 3.45	21	0	5.87 ± 2.17	0	2
56	104.97 ± 5.44	94	0	61.43 ± 4.26	50	0
57	72.53 ± 4.81	59	0	44.57 ± 1.65	41	0
58	214.20 ± 4.93	200	0	137.03 ± 6.52	111	0
59	110.00 ± 4.82	102	0	58.27 ± 3.85	51	0
60	25.50 ± 3.51	17	0	3.43 ± 1.84	0	6
61	73.53 ± 4.81	63	0	49.53 ± 4.11	41	0
62	116.73 ± 5.07	102	0	48.60 ± 3.42	42	0
63	41.77 ± 3.29	35	0	18.47 ± 2.92	13	0
64	50.37 ± 4.20	40	0	20.63 ± 3.01	13	0
65	24.13 ± 2.46	20	0	6.97 ± 2.51	4	0
66	51.70 ± 4.18	43	0	7.40 ± 2.56	0	1
67	33.90 ± 6.15	15	0	3.33 ± 2.36	0	9
68	26.67 ± 3.62	17	0	9.73 ± 2.35	4	0
69	38.47 ± 3.39	31	0	5.43 ± 2.06	0	1
70	64.90 ± 3.83	56	0	36.90 ± 3.16	29	0
71	76.73 ± 3.92	69	0	44.70 ± 3.28	37	0
72	125.93 ± 6.71	106	0	59.17 ± 5.88	46	0
73	53.83 ± 3.85	43	0	22.50 ± 3.28	17	0
74	28.00 ± 6.10	14	0	5.70 ± 2.58	0	3
75	66.43 ± 4.98	53	0	19.10 ± 2.75	13	0
76	60.73 ± 4.05	48	0	30.60 ± 4.57	21	0
77	153.00 ± 6.62	131	0	67.27 ± 5.53	54	0
78	43.30 ± 3.57	35	0	16.37 ± 3.02	11	0
79	39.20 ± 4.46	27	0	14.60 ± 3.53	9	0
80	83.07 ± 6.50	70	0	24.83 ± 3.22	17	0
81	134.73 ± 5.88	122	0	100.77 ± 5.04	87	0
82	231.17 ± 5.85	219	0	175.60 ± 6.37	160	0
83	312.40 ± 6.15	300	0	206.10 ± 6.14	194	0
84	42.53 ± 5.08	34	0	7.70 ± 2.64	0	1
85	102.07 ± 4.68	94	0	57.73 ± 4.63	44	0
86	167.60 ± 6.52	150	0	98.57 ± 5.78	88	0

GA are *population size*= 100, crossover and mutation probabilities $p_X = .9$ and $p_M = 1/\ell$ (where $\ell = vb$ is the size of individuals) respectively, and $f_{\%} = 10\%$.

Tables 2-4 show all results. Regarding the initialization procedure, bit-flip-based algorithms have been tested both with purely random initialization and guided initialization (i.e., enforcing row constraints). The guided initialization resulted in worse results for HC, indistinguishable results for TS, and better results for GA in most instances (in all cases with statistical significance at the standard 0.05 level according to a Wilcoxon ranksum test). This can be explained by the inferior exploration capabilities of HC_{bf} when starting from a solution satisfying row constraints (the search will be confined to a narrow path uphill). This consideration is unimportant for TS_{bf}, since it can easily make downhill moves to keep on exploring. The GA_{bf} benefits however from having a diverse population of higher quality than random. We have therefore opted for reporting the results of HC_{bf} and TS_{bf} with random initialization, and the results of GA_{bf} with guided initialization.

A summary of performance is provided in Table 5, showing the number of problem instances (out of 86) that were solved in at least one run by each of the algorithms, and the corresponding success percentage. As expected, TS

Table 3. Results of TS algorithms (30 runs per instance). $\overline{x}$, σ, B and S denote, respectively, the fitness average value, the standard deviation, the best obtained result, and the number of times that a problem instance solution is obtained.

	TS_{bf}			TS_{sw}		
ID	$\overline{x} \pm \sigma$	B	S	$\overline{x} \pm \sigma$	B	S
1	3.30 ± 2.62	0	10	0.00 ± 0.00	0	30
2	6.93 ± 6.06	0	12	0.00 ± 0.00	0	30
3	6.17 ± 2.40	0	2	0.00 ± 0.00	0	30
4	4.30 ± 1.68	0	3	0.00 ± 0.00	0	30
5	3.67 ± 4.81	0	18	0.00 ± 0.00	0	30
6	7.33 ± 2.53	4	0	0.00 ± 0.00	0	30
7	1.67 ± 2.10	0	18	0.00 ± 0.00	0	30
8	28.60 ± 5.37	15	0	0.00 ± 0.00	0	30
9	9.27 ± 2.86	4	0	0.00 ± 0.00	0	30
10	27.13 ± 7.10	0	1	0.00 ± 0.00	0	30
11	13.50 ± 3.39	5	0	0.00 ± 0.00	0	30
12	3.93 ± 2.73	0	8	0.00 ± 0.00	0	30
13	16.60 ± 4.10	8	0	0.00 ± 0.00	0	30
14	3.13 ± 2.05	0	8	0.00 ± 0.00	0	30
15	1.60 ± 2.39	0	19	0.00 ± 0.00	0	30
16	10.60 ± 2.24	4	0	0.00 ± 0.00	0	30
17	16.33 ± 3.25	8	0	2.93 ± 1.77	0	8
18	7.27 ± 2.79	0	1	0.00 ± 0.00	0	30
19	59.27 ± 5.28	50	0	0.37 ± 1.97	0	29
20	8.37 ± 2.44	4	0	0.00 ± 0.00	0	30
21	22.70 ± 3.94	15	0	5.77 ± 0.88	4	0
22	27.43 ± 3.23	23	0	4.70 ± 4.41	0	12
23	8.33 ± 2.29	4	0	0.00 ± 0.00	0	30
24	33.33 ± 12.10	0	2	0.00 ± 0.00	0	30
25	1.43 ± 1.99	0	19	0.00 ± 0.00	0	30
26	3.77 ± 2.35	0	7	0.00 ± 0.00	0	30
27	26.10 ± 4.04	18	0	8.13 ± 1.82	4	0
28	34.70 ± 3.91	27	0	11.70 ± 1.51	9	0
29	8.57 ± 2.67	4	0	0.00 ± 0.00	0	30
30	2.67 ± 2.34	0	12	0.00 ± 0.00	0	30
31	8.47 ± 2.40	4	0	0.00 ± 0.00	0	30
32	12.93 ± 2.05	9	0	0.27 ± 1.00	0	28
33	25.63 ± 3.40	20	0	9.37 ± 1.87	4	0
34	22.67 ± 2.66	18	0	6.70 ± 1.39	4	0
35	12.70 ± 3.25	6	0	0.00 ± 0.00	0	30
36	100.07 ± 6.39	85	0	49.47 ± 18.59	0	3
37	6.13 ± 2.63	0	2	0.00 ± 0.00	0	30
38	2.33 ± 2.83	0	17	0.00 ± 0.00	0	30
39	36.27 ± 3.98	29	0	15.47 ± 1.78	12	0
40	3.30 ± 2.69	0	11	0.00 ± 0.00	0	30
41	5.47 ± 1.65	4	0	0.00 ± 0.00	0	30
42	7.47 ± 3.31	4	0	0.00 ± 0.00	0	30
43	48.80 ± 5.53	39	0	20.47 ± 1.87	16	0
44	108.80 ± 6.48	96	0	66.70 ± 9.48	22	0
45	17.53 ± 2.64	9	0	4.30 ± 1.44	0	2
46	58.37 ± 4.32	50	0	32.37 ± 1.74	29	0
47	16.87 ± 4.13	8	0	2.43 ± 1.99	0	12
48	24.23 ± 3.50	18	0	8.70 ± 1.72	4	0
49	1.50 ± 2.01	0	19	0.00 ± 0.00	0	30
50	23.43 ± 3.96	16	0	6.03 ± 1.35	4	0
51	8.70 ± 2.42	0	1	0.00 ± 0.00	0	30
52	14.37 ± 2.95	10	0	0.40 ± 1.20	0	27
53	6.20 ± 3.26	0	3	0.00 ± 0.00	0	30
54	50.77 ± 5.97	39	0	24.53 ± 2.05	20	0
55	6.83 ± 2.19	4	0	0.00 ± 0.00	0	30
56	68.33 ± 4.83	51	0	40.47 ± 2.68	35	0
57	43.17 ± 2.57	38	0	40.13 ± 0.43	40	0
58	160.80 ± 8.58	145	0	100.43 ± 3.85	91	0
59	66.53 ± 4.08	56	0	35.90 ± 2.53	30	0
60	2.47 ± 2.12	0	12	0.00 ± 0.00	0	30
61	54.60 ± 5.22	43	0	14.40 ± 11.64	0	10
62	65.37 ± 6.74	54	0	29.77 ± 1.80	26	0
63	20.73 ± 3.59	13	0	5.17 ± 1.44	0	1
64	23.53 ± 2.70	19	0	8.13 ± 1.69	4	0
65	5.50 ± 2.03	0	1	0.00 ± 0.00	0	30
66	12.57 ± 4.11	5	0	0.00 ± 0.00	0	30
67	2.23 ± 2.56	0	16	0.00 ± 0.00	0	30
68	7.37 ± 2.11	4	0	0.00 ± 0.00	0	30
69	7.27 ± 2.54	0	1	0.13 ± 0.72	0	29
70	40.53 ± 3.87	32	0	18.23 ± 1.84	14	0
71	47.37 ± 4.42	38	0	24.17 ± 2.60	17	0
72	70.63 ± 5.49	61	0	36.77 ± 2.60	32	0
73	21.90 ± 3.18	17	0	8.30 ± 1.73	4	0
74	2.20 ± 2.70	0	17	0.00 ± 0.00	0	30
75	24.90 ± 4.72	15	0	5.63 ± 2.33	0	2
76	32.00 ± 3.53	24	0	14.13 ± 2.53	9	0
77	84.23 ± 8.92	68	0	43.87 ± 2.20	41	0
78	16.47 ± 2.80	11	0	3.17 ± 2.07	0	8
79	15.00 ± 3.17	10	0	1.43 ± 2.06	0	20
80	32.00 ± 5.14	22	0	10.23 ± 2.01	6	0
81	109.23 ± 7.99	89	0	22.90 ± 18.70	0	12
82	184.40 ± 6.34	173	0	134.13 ± 5.08	124	0
83	230.37 ± 10.02	207	0	159.63 ± 5.92	148	0
84	7.60 ± 3.59	0	1	0.50 ± 1.28	0	26
85	61.90 ± 4.47	54	0	34.80 ± 2.56	31	0
86	107.70 ± 5.54	99	0	65.70 ± 3.64	56	0

variants outperform their HC counterparts. Note also that results of local search algorithms are considerably improved when considering the swap neighborhood. The difference is not so marked in the case of GAs, although GA_{sw} still manages to solve more instances than GA_{bf}. In global terms, TS_{sw} outperforms the rest of techniques, including NN-SA and CLS. In fact, TS_{sw} can solve every instance solved by CLS (i.e., the technique that had reported the best results on the problem), as well as instances $\langle 23, 23, 11, 11, 5\rangle$ and $\langle 15, 60, 28, 7, 12\rangle$.

A more fine-grained comparison of the algorithms considered is provided in Table 6. This table shows the percentage of instances in which a certain algorithm performs better (again, with statistical significance at the 0.05 level according to a Wilcoxon ranksum test) than another certain one (note that entries (i, j) and (j, i) in this table do not necessarily sum 100%, since there are instances on which there is no significant difference between the algorithms compared). As it can be seen, swap-based algorithms are consistently better than bit-flip-based algorithms (above 75% in almost all cases). Regarding the GAs, note GA_{sw} is better than GA_{bf} in about 78% of the runs, a larger difference than the number of solved instances. Finally, TS_{sw} is the clear winner, beating the remaining algorithms in 78%-100% of instances.

Table 4. Results of GAs (30 runs per instance). $\overline{x}$, σ, B and S denote, respectively, the fitness average value, the standard deviation, the best obtained result, and the number of times that a problem instance solution is obtained.

	GA_{bf}			GA_{sw}		
ID	$\bar{x} \pm \sigma$	B	S	$\bar{x} \pm \sigma$	B	S
1	0.00 ± 0.00	0	30	0.00 ± 0.00	0	30
2	0.37 ± 1.97	0	29	0.00 ± 0.00	0	30
3	0.77 ± 1.75	0	25	0.00 ± 0.00	0	30
4	1.07 ± 1.77	0	22	0.27 ± 1.00	0	28
5	0.00 ± 0.00	0	30	0.00 ± 0.00	0	30
6	2.60 ± 2.17	0	12	0.13 ± 0.72	0	29
7	0.10 ± 0.54	0	29	0.00 ± 0.00	0	30
8	11.20 ± 8.82	0	10	4.20 ± 6.43	0	21
9	5.37 ± 1.76	0	1	3.60 ± 1.70	0	5
10	14.67 ± 7.11	0	4	8.47 ± 6.48	0	11
11	8.47 ± 2.36	0	1	4.87 ± 1.41	0	1
12	2.10 ± 2.13	0	15	1.07 ± 1.77	0	22
13	10.20 ± 4.46	0	4	4.40 ± 5.46	0	18
14	1.60 ± 1.96	0	18	0.63 ± 1.43	0	25
15	0.73 ± 1.48	0	24	0.00 ± 0.00	0	30
16	6.47 ± 1.71	4	0	5.60 ± 1.23	4	0
17	11.43 ± 2.26	7	0	7.17 ± 1.83	4	0
18	4.03 ± 2.06	0	5	1.40 ± 1.85	0	19
19	55.10 ± 3.94	47	0	31.27 ± 8.52	0	1
20	6.17 ± 2.05	4	0	2.87 ± 1.89	0	9
21	17.00 ± 2.71	11	0	9.97 ± 1.78	6	0
22	23.30 ± 3.63	12	0	15.57 ± 2.39	10	0
23	5.00 ± 2.03	0	3	2.87 ± 1.93	0	9
24	17.67 ± 12.38	0	8	7.47 ± 10.00	0	19
25	0.80 ± 1.62	0	24	0.00 ± 0.00	0	30
26	2.50 ± 2.39	0	14	1.13 ± 1.75	0	21
27	22.80 ± 3.33	17	0	13.80 ± 2.09	9	0
28	29.60 ± 3.59	23	0	17.43 ± 2.74	12	0
29	5.43 ± 2.80	0	4	3.40 ± 2.01	0	7
30	1.30 ± 1.86	0	20	0.27 ± 1.00	0	28
31	7.60 ± 2.44	4	0	4.77 ± 1.71	0	2
32	10.03 ± 2.77	4	0	6.70 ± 2.00	4	0
33	23.30 ± 2.93	19	0	15.93 ± 2.31	11	0
34	19.00 ± 3.11	13	0	12.63 ± 1.85	10	0
35	10.57 ± 2.51	5	0	4.43 ± 1.61	0	2
36	103.27 ± 5.50	94	0	73.53 ± 5.00	60	0
37	3.53 ± 2.59	0	9	2.00 ± 1.93	0	14
38	1.23 ± 2.25	0	22	0.37 ± 1.11	0	27
39	37.97 ± 4.28	29	0	22.27 ± 2.45	17	0
40	2.60 ± 2.33	0	13	1.27 ± 1.81	0	20
41	4.00 ± 2.13	0	5	2.33 ± 2.09	0	13
42	6.50 ± 2.26	0	1	3.00 ± 2.05	0	9
43	47.23 ± 5.68	38	0	28.93 ± 3.19	24	0
44	113.43 ± 5.44	104	0	86.80 ± 5.72	76	0
45	15.07 ± 3.17	8	0	11.83 ± 2.03	8	0
46	56.83 ± 5.41	47	0	47.23 ± 4.42	40	0
47	13.77 ± 3.29	7	0	14.00 ± 2.28	10	0
48	25.33 ± 3.65	19	0	21.00 ± 3.16	17	0
49	1.97 ± 2.36	0	17	4.00 ± 2.27	0	5
50	25.70 ± 4.13	17	0	18.63 ± 3.18	13	0
51	7.33 ± 2.51	4	0	8.60 ± 3.17	4	0
52	14.90 ± 3.34	9	0	12.03 ± 2.64	7	0
53	4.80 ± 2.87	0	6	6.00 ± 2.46	0	2
54	56.23 ± 6.01	44	0	39.83 ± 5.70	25	0
55	7.37 ± 3.02	0	1	6.90 ± 2.01	3	0
56	72.70 ± 7.34	61	0	60.27 ± 5.40	51	0
57	45.97 ± 2.95	40	0	43.73 ± 1.79	40	0
58	171.43 ± 8.58	154	0	131.93 ± 7.23	118	0
59	70.23 ± 6.72	58	0	56.30 ± 6.95	46	0
60	4.40 ± 2.39	0	5	5.43 ± 2.74	0	4
61	51.23 ± 5.17	41	0	46.57 ± 5.24	36	0
62	70.50 ± 8.35	50	0	49.63 ± 5.31	39	0
63	20.10 ± 3.62	13	0	19.13 ± 3.43	13	0
64	25.33 ± 4.41	16	0	20.67 ± 3.28	15	0
65	5.90 ± 2.33	0	1	8.23 ± 2.26	4	0
66	12.40 ± 3.24	7	0	10.60 ± 2.70	6	0
67	3.93 ± 2.28	0	6	5.20 ± 1.66	0	1
68	8.50 ± 3.29	4	0	10.53 ± 2.31	7	0
69	9.13 ± 2.73	4	0	9.20 ± 3.11	0	1
70	39.20 ± 4.83	32	0	36.23 ± 4.33	25	0
71	50.53 ± 5.04	36	0	41.63 ± 5.20	33	0
72	82.23 ± 7.28	69	0	58.20 ± 5.07	46	0
73	27.87 ± 4.57	17	0	23.03 ± 3.70	15	0
74	5.20 ± 2.54	0	1	7.17 ± 3.14	0	2
75	29.57 ± 4.51	22	0	20.27 ± 2.45	16	0
76	35.40 ± 4.42	25	0	29.30 ± 3.80	21	0
77	101.73 ± 10.25	76	0	65.90 ± 5.02	58	0
78	19.93 ± 4.30	11	0	18.33 ± 3.60	12	0
79	16.40 ± 3.57	9	0	17.00 ± 3.20	10	0
80	37.60 ± 6.40	26	0	26.33 ± 4.75	18	0
81	107.37 ± 10.77	80	0	97.93 ± 9.80	78	0
82	200.70 ± 9.84	185	0	174.17 ± 7.86	157	0
83	261.47 ± 7.67	244	0	200.63 ± 8.21	182	0
84	9.27 ± 3.76	0	2	9.73 ± 2.78	5	0
85	67.10 ± 6.23	55	0	58.53 ± 5.96	48	0
86	125.20 ± 11.65	107	0	94.90 ± 8.25	81	0

Table 5. Number and percentage of solved instances for each algorithm on the 86 instances considered

NN-SA	CLS	HC_{bf}	HC_{sw}	TS_{bf}	TS_{sw}	GA_{bf}	GA_{sw}
16	55	4	35	27	**57**	35	37
(18.60%)	(63.95%)	(4.65%)	(40.70%)	(31.40%)	**(66.28%)**	(40.70%)	(43.02%)

Table 6. Summary of statistical significance results. Each entry in the table indicates the percentage of instances in which the algorithm labelled in the row outperforms the algorithm labelled in the column, with a statistically significant difference according to a Wilcoxon ranksum test.

	HC_{bf}	HC_{sw}	TS_{bf}	TS_{sw}	GA_{bf}	GA_{sw}
HC_{bf}	–	0.00%	0.00%	0.00%	0.00%	0.00%
HC_{sw}	100%	–	76.64%	0.00%	61.63%	12.79%
TS_{bf}	95.35%	12.79%	–	0.00%	27.91%	11.63%
TS_{sw}	100%	97.67%	100.00%	–	86.05%	77.91%
GA_{bf}	100%	10.47%	43.02%	11.63%	–	5.81%
GA_{sw}	100%	47.67%	81.40%	17.44%	77.91%	–

5 Conclusions and Future Work

The application of metaheuristics to the design of balanced incomplete blocks has resulted in very encouraging and positive results. An empirical evaluation of three different techniques (i.e., a hill climbing method, a tabu search algorithm, and a genetic algorithm), with two variants each, has shown that highly competitive results can be achieved. Furthermore, a TS algorithm working on the swap neighborhood has been shown to be competitive to an ad-hoc constrained local search (CLS) method, the current incumbent for this problem.

In addition, our analysis also indicates the relevance of the neighborhood structure chosen. The swap neighborhood provides better navigational capabilities than the bit-flip neighborhood, regardless how initial solutions are chosen in the latter. However, this does not imply the bit-flip neighborhood is not appropriate for this problem. For example, we believe a hybrid approach that combine both neighborhoods –e.g., in a variable neighborhood search framework– would be of the foremost interest. Work is in progress in this line. This hybridization can be also done from the algorithmic point of view, i.e., a memetic combination of TS and GAs. The form of this combination is an issue of further work.

Acknowledgements

This work is partially supported by projects TIN2008-05941 (of MICIIN) and P06-TIC2250 (from Andalusian Regional Government).

References

1. Colbourn, C., Dinitz, J.: The CRC handbook of combinatorial designs. CRC Press, Boca Raton (1996)
2. van Lint, J., Wilson, R.: A Course in Combinatorics. Cambridge University Press, Cambridge (1992)
3. Mead, R.: Design of Experiments: Statistical Principles for Practical Applications. Cambridge University Press, Cambridge (1993)
4. Buratti, M.: Some (17q, 17, 2) and (25q, 25, 3)BIBD constructions. Designs, Codes and Cryptography 16(2), 117–120 (1999)
5. Lan, L., Tai, Y.Y., Lin, S., Memari, B., Honary, B.: New constructions of quasi-cyclic LDPC codes based on special classes of BIDBs for the AWGN and binary erasure channels. IEEE Transactions on Communications 56(1), 39–48 (2008)
6. Corneil, D.G., Mathon, R.: Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations. Annals of Discrete Mathematics 2, 1–32 (1978)
7. Gibbons, P., Östergård, P.: Computational methods in design theory. In: [1], pp. 730–740
8. Bofill, P., Guimerà, R., Torras, C.: Comparison of simulated annealing and mean field annealing as applied to the generation of block designs. Neural Networks 16(10), 1421–1428 (2003)
9. Prestwich, S.: A local search algorithm for balanced incomplete block designs. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 53–64. Springer, Heidelberg (2003)

10. Cochran, W.G., Cox, G.M.: Experimental Design. John Wiley, New York (1957)
11. Fisher, R.A., Yates, F.: Statistical Tables for Biological, Agricultural and Medical Research, 3rd edn. Oliver & Boy (1949)
12. Fisher, R.A.: An examination of the different possible solutions of a problem in incomplete blocks. Annals of Eugenics 10, 52–75 (1940)
13. Whitaker, D., Triggs, C.M., John, J.A.: Construction of block designs using mathematical programming. J. Roy. Statist. Soc. B 52(3), 497–503 (1990)
14. John, J.A., Whitaker, D., Triggs, C.M.: Construction of cyclic designs using integer programming. Journal of statistical planning and inference 36(2), 357–366 (1993)
15. Zergaw, D.: A sequential method of constructing optimal block designs. Australian & New Zealand Journal of Statistics 31, 333–342 (1989)
16. Tjur, T.: An algorithm for optimization of block designs. Journal of Statistical Planning and Inference 36, 277–282 (1993)
17. Flener, P., Frisch, A.M., Hnich, B., Kzltan, Z., Miguel, I., Walsh, T.: Matrix modelling. In: CP 2001 Workshop on Modelling and Problem Formulation. International Conference on the Principles and Practice of Constraint Programming (2001)
18. Puget, J.F.: Symmetry breaking revisited. In: Van Hentenryck, P. (ed.) CP 2002. LNCS, vol. 2470, pp. 446–461. Springer, Heidelberg (2002)
19. Meseguer, P., Torras, C.: Exploiting symmetries within constraint satisfaction search. Artif. Intell. 129(1-2), 133–163 (2001)
20. Prestwich, S.: Negative effects of modeling techniques on search performance. Annals of Operations Research 18, 137–150 (2003)
21. Street, D., Street, A.: Partially balanced incomplete block designs. In: [1], pp. 419–423
22. Mullin, C., Gronau, H.: PBDs and GDDs: The basics. In: [1], pp. 185–193
23. Wallis, W.D.: Regular graph designs. Journal of Statistical Planning and Inference 51, 272–281 (1996)
24. Bofill, P., Torras, C.: MBMUDs: a combinatorial extension of BIBDs showing good optimality behaviour. Journal of Statistical Planning and Inference 124(1), 185–204 (2004)
25. Chuang, H.Y., Tsai, H.K., Kao, C.Y.: Optimal designs for microarray experiments. In: 7th International Symposium on Parallel Architectures, Algorithms, and Networks, Hong Kong, China, pp. 619–624. IEEE Computer Society, Los Alamitos (2004)
26. Morales, L.B.: Constructing difference families through an optimization approach: Six new BIBDs. Journal of Combinatorial Design 8(4), 261–273 (2000)
27. Angelis, L.: An evolutionary algorithm for A-optimal incomplete block designs. Journal of Statistical Computation and Simulation 73(10), 753–771 (2003)

Guided Ejection Search for the Job Shop Scheduling Problem

Yuichi Nagata and Satoshi Tojo

Graduate School of Information Sciences,
Japan Advanced Institute of Science and Technology, Japan
{nagatay,tojo}@jaist.ac.jp

Abstract. We present a local search framework we term guided ejection search (GES) for solving the job shop scheduling problem (JSP). The main principle of GES is to always search for an incomplete solution from which some components are removed, subject to the constraint that a quality of the incomplete solution is better than that of the best (complete) solution found during the search. Moreover, the search is enhanced by a concept reminiscent of guided local search and problem-dependent local searches. The experimental results for the standard benchmarks for the JSP demonstrate that the suggested GES is robust and highly competitive with the state-of-the-art metaheuristics for the JSP.

Keywords: job shop scheduling, tabu search, ejection chain, guided local search, metaheuristics.

1 Introduction

The job shop scheduling problem (JSP) consists of a set of jobs to be processed on machines. Each job is composed of an ordered list of operations, each of which must be processed by a predetermined machine and has an associated processing time. The objective of the JSP (with the makespan criterion) is to find a schedule (job sequence on the machines) that minimizes the makespan (i.e., the duration needed for processing all jobs), subject to the constraint that each machine can process only one operation at a time.

The JSP has been intensively studied since the 1950s because it models practical production processes very well. Apart from its practical importance, the JSP is a challenging optimization problem of significant academic value as it is known as one of the most difficult NP-complete combinatorial optimization problems and is often used as a benchmark problem when new solution approaches are suggested. Therefore, a number of metaheuristic approaches as well as exact algorithms have been developed and tested in the JSP. For an extensive survey of scheduling techniques, the reader is referred to [1][2].

Recently, we developed a powerful route minimization heuristic for the vehicle routing problem with time windows (VRPTW) [3]. We found that the basic framework of this heuristic can be used for solving a wide variety of combinational optimization problems, and we refer to this framework as guided ejection search (GES). In this paper we formulate GES and hence apply it to the JSP.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 168–179, 2009.

GES for the JSP tries to improve a schedule step-by-step without allowing temporal deterioration of the makespan. Instead, we allow temporal removal (ejection) of operations from a schedule and store them in the so-called *ejection pool* (EP) [4]. In GES, "insertion-ejection" moves are iterated until the EP becomes empty, subject to the constraint that the makespan of the incomplete schedule is less than that of the best (complete) schedule found so far. Here, an insertion-ejection move is performed by inserting an operation in the EP and ejecting operations (if necessary) from the resulting schedule (ejected operations are added to the EP). Moreover, the ejection is directed by a concept reminiscent of the guided local search [5], and the insertion is enhanced by a local search.

The GES framework shares similarities with the shifting bottleneck heuristic [6], ejection chains [7], and large neighborhood search [8]. These approaches also temporarily remove some components from a current solution and then the removed components are re-inserted based on construction heuristics or variable depth methods to construct complete solutions. A major difference between GES and these approaches is that GES always allows incomplete solutions, while other approaches must construct complete solutions after prearranged procedures. This feature of GES makes the search more flexible.

The suggested GES algorithm was tested on the standard benchmark problems for the JSP and was compared with the state-of-the-art heuristics for the JSP [9][10][11]. Computational results showed that the proposed algorithm is robust and highly competitive. It found new best-known solutions in eight benchmark instances. The reminder of this paper is arranged as follows. First, the problem definition and notations are described in Section 2. The problem solving methodology based on GES are described in Section 3. The computational analysis of GES and its comparison with other algorithms are presented in Section 4. Conclusions are presented in Section 5.

2 Problem Definition and Notations

The JSP can be formulated as follows. Let $J = \{1, \ldots, |J|\}$ denote a set of jobs and $M = \{1, \ldots, |M|\}$ a set of machines. Each job consists of a sequence of operations that must be processed in a particular order (job order criteria). Let $V = \{0, 1, \ldots, n, n+1\}$ denote a set of operations (with dummy operations 0 and $n+1$ representing the start and end of a schedule, respectively) where n refers to the total number of operations in all the jobs. For each operation $i \in V$, let μ_i and p_i (with $p_0 = p_{n+1} = 0$) denote the machine and time, respectively, required for processing it. A schedule is represented by the processing order of operations on the machines, $\pi = (\pi_1, \ldots, \pi_{|M|})$, where $\pi_m = \langle \pi_m(1), \ldots, \pi_m(l_m) \rangle$ represents the processing order of operations on machine m (l_m is the number of operations). In addition, we define $\pi_m(0) = 0$ and $\pi_m(l_m + 1) = n + 1$.

For a given schedule π, representation as a directed graph $G(\pi) = (V, R \cup E(\pi))$ [9] is useful (see Fig. 1). Here, R is defined as a set of arcs, each of which represents a successive operation pair on the same job. Likewise, $E(\pi)$ is defined as a set of arcs, each of which represents a successive operation pair on the same

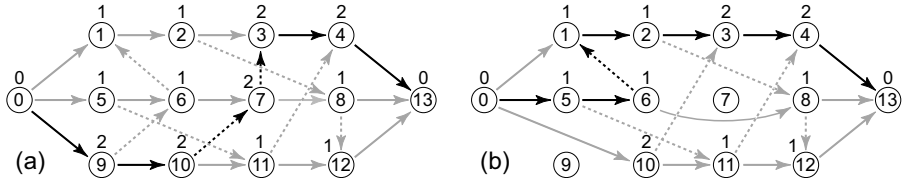


Fig. 1. Examples of directed graph $G(\pi)$ of 3-job, 4-machine instance; arcs in R and $E(\pi)$ are denoted by solid and dotted lines, respectively. The critical paths are illustrated by black lines. (a) A complete schedule has the makespan of 10. (b) An incomplete schedule (operations 7 and 9 are skipped) has the makespan of 8.

machine, i.e., $E(\pi) = \cup_{m=1}^{|M|} \cup_{s=1}^{l_m - 1} \{(\pi_m(s), \pi_m(s+1))\}$. Each node $i \in V$ has the associated weight p_i (the arcs have zero weight). A schedule π is called *feasible* if $G(\pi)$ does not contain any cycle and *infeasible* if otherwise. For a feasible schedule π, the makespan, denoted by $C(\pi)$, equals the length of the longest path from 0 to $n+1$ in $G(\pi)$ (called the critical path). Therefore, the JSP is equivalent to finding a feasible schedule π that minimizes $C(\pi)$. In addition, we define an incomplete schedule as a schedule in which some operations are skipped (i.e., some operations temporarily need not be processed). For an incomplete schedule, the directed graph and makespan are defined in the same manner as the complete schedule (see Fig. 1). In the sequel, a schedule refers to either complete or incomplete schedule unless otherwise stated.

For a given schedule π, let r_i denote the length of the longest path from 0 to i (including the node weight of i) in $G(\pi)$ and let q_i be the length of the longest path from i to $n+1$ (including the node weight of i). For each node $i \in \{1, \ldots, n\}$, we denote the job-predecessor by $JP[i]$ and the job-successor by $JS[i]$. Likewise, we denote the machine-predecessor by $MP[i]$ and the machine-successor by $MS[i]$. Let $f(s)$ $(s = 0, 1, \ldots, n, n+1)$ denote the *topological order* of the nodes in $G(\pi)$, i.e., if j must be processed after i, $f(i) < f(j)$ holds.

3 Guided Ejection Search for the JSP

3.1 GES Framework for the JSP

The pseudo-code of GES applied to the JSP is shown in Algorithm 1. Procedure MainGES() is the main function. GES starts with a randomly generated initial schedule π (line 2). Then the makespan is step-by-step reduced by repeating procedure RoutineGES(π, L), which searches a complete schedule whose makespan is less than or equal to the upper bound L. Here, L is set to $C(\pi^*) - 1$ (π^* refers to the current best complete schedule) (line 5).

Procedure RoutineGES(π, L) is started by ejecting operations from the input schedule so that the makespan (of the resulting incomplete schedule) is less than or equal to L. The ejection pool (EP) is then initialized with the ejected operations (lines 1, 13-16). The EP is to always hold the set of temporarily ejected operations, currently missing from an incomplete schedule π.

Here, $\mathcal{I}(\pi, L)$, which is obtained by procedure EJECTION(π, L), is defined as a set of all combinations consisting of at most k_{max} operations whose ejection from π results in $C(\pi) \leq L$. Therefore, a k-tuple ($k \leq k_{max}$) of operations selected from $\mathcal{I}(\pi, L)$ is ejected from the input schedule (the selection strategy is explained later). Then subsequent iterations are repeated to reinsert operations in the EP into π until EP becomes empty, subject to the constraint $C(\pi) \leq L$.

In each iteration, operation i_{in} is selected from the EP with the last-in first-out (LIFO) strategy and is removed from the EP (lines 4, 5). The selected operation is then inserted into all positions in the corresponding machine (some of which may result in infeasible schedules) (line 8). The resulting (feasible) schedules are then improved by a local search (line 9) to enhance the constraint satisfaction (i.e., $C(\pi_s) \leq L$). Here, one should note that the local search may performed on incomplete schedules (operations in the EP are ignored). Let π be the best of the schedules after invoking the local search (line 11). If $C(\pi) \leq L$ holds, we have succeeded in inserting operation i_{in} without violating the constraint. Otherwise, we eject at most k_{max} operations from π to satisfy the constraint and add the ejected operations to the EP (lines 13-16). Here, a k-tuple of operations, $\{i_{out}(1), \ldots, i_{out}(k)\}$ ($k \leq k_{max}$), is selected from $\mathcal{I}(\pi, L)$ so that the sum of the penalty counters of the ejecting operations, $P_{sum} = p[i_{out}(1)] + \ldots, p[i_{out}(k)]$, is minimized (line 14). Here, the penalty counter, $p[i]$ ($i \in V$), refers to how many times operation i is selected from the EP (which is almost the same as the number of ejections of i) over the iterations in Algorithm 1 (line 6). The general principle of this selection mechanism is to select a k-tuple, giving priority to fewer k. In addition, this selection mechanism is useful for avoiding *cycling*. If some operations conflict with each other because of insertion, cycling will occur (i.e., insertions and ejections are repeated within the conflicting operations) and the penalty counters of these operations will increase. Therefore, GES can escape from a cycling by ejecting a relatively large number of operations with relatively small penalty values when a cycling occurs. After each insertion-ejection move, the resulting schedule π is perturbed by procedure PERTURB(π, L) (line 19) to diversify the search where random local search moves are executed for a given number of times, subject to the constraint (i.e., $C(\pi) \leq L$).

However, $\mathcal{I}(\pi, L)$ potentially becomes empty, especially when k_{max} is small (e.g., $k_{max} = 1, 2$). We design two variants of GES to address this problem.

GES-1: If $\mathcal{I}(\pi, L)$ is empty, restore both π and EP to those before line 4 (we observed that $\mathcal{I}(\pi, L)$ was not empty for the input schedule (lines 1, 13)).

GES-2: If $\mathcal{I}(\pi, L)$ is empty, repeat ejections of k_{max} bottleneck nodes (operations causing the constraint violation (see Section 3.2)) with the fewest penalty counters until procedure EJECTION(π, L) generates a nonempty set $\mathcal{I}(\pi, L)$. Then, a k-tuple of operations is ejected according to lines 13-15. In addition, the ejected operations are restored to their original locations unless the constraint is violated because some operations will be excessively ejected. The ejected operations are then added to the EP.

Although the GES framework can be applied to a wide variety of combinational optimization problems, several procedures are problem-dependent. In particular, procedure EJECTION requires an efficient implementation because the number of possible combinations consisting of at most k_{max} operations is usually very large. As for procedures LOCALSEARCH and PERTURB, we can employ any local search algorithm that has been developed for the JSP.

Algorithm 1. Guided ejection search for the JSP

Procedure MAINGES()
1 :Set $iter := 0$;
2 :Generate an initial solution π;
3 :Initialize all penalty counters $p[i] := 1$ $(i \in V)$;
4 :**repeat**
5 : Set $\pi^* := \pi$ and $L := C(\pi^*) - 1$;
6 : $\pi :=$ ROUTINEGES(π, L);
7 :**until** $iter \geq maxIter$
8 :**return** π^*;

Procedure ROUTINEGES(π, L)
1 :Set $EP := \emptyset$ and **goto** line 13;
2 :**repeat**
3 : Set $iter := iter + 1$;
4 : Select i_{in} from EP with the LIFO strategy;
5 : Remove i_{in} from $EP \rightarrow EP$;
6 : Set $p[i_{in}] := p[i_{in}] + 1$;
7 : **for** $s := 0$ **to** l_m (where $m = \mu_{i_{in}}$) **do**
8 : Insert i_{in} between $\pi_m(s)$ and $\pi_m(s+1) \rightarrow \pi_s$;
9 : **if** π_s is a feasible schedule **then** $\pi_s :=$ LOCALSEARCH(π_s);
10: **end for**
11: Select best (feasible) schedule among $\{\pi_0, \pi_1, \ldots, \pi_{l_m}\} \rightarrow \pi$;
12: **if** $C(\pi) > L$ **then**
13: $\mathcal{I}(\pi, L) :=$ EJECTION(π, L);
14: Select $\{i_{out}(1), \ldots, i_{out}(k)\} \in \mathcal{I}(\pi, L)$
s.t. $p[i_{out}(1)]+, \ldots, +p[i_{out}(k)]$ is minimized;
15: Eject $\{i_{out}(1), \ldots, i_{out}(k)\}$ from $\pi \rightarrow \pi$;
16: Add $\{i_{out}(1), \ldots, i_{out}(k)\}$ to $EP \rightarrow EP$;
17: **end if**
18: $\pi :=$PERTURB(π, L);
19:**until** $EP = \emptyset$ or $iter \geq maxIter$
20:**return** π;

3.2 Algorithm of Procedure EJECTION

Procedure EJECTION(π, L), which computes $\mathcal{I}(\pi, L)$, is a core part of the GES. More formally, $\mathcal{I}(\pi, L)$ is defined by $\mathcal{I}(\pi, L) = \{T \in \mathcal{T}_{k_{max}} \mid C(\pi - T) \leq L\}$, where $\mathcal{T}_{k_{max}}$ is a set of all combinations consisting of at most k_{max} operations and $\pi - T$ refers to a schedule obtained by ejecting the operations in T from π.

First, we describe an outline of procedure EJECTION. Let the operations be labeled according to the topological order $f(\pi)$ in the directed graph $G(\pi)$ for

simplicity (see Fig. 2). To compute $\mathcal{I}(\pi, L)$[1], nodes in $G(\pi)$ are in principle ejected from π in the lexicographic order[2] and successful ejections T (i.e., $C(\pi - T) \leq L$) are stored. Indeed, most of the lexicographic ejections can be pruned. For node $i \in V$, $r_i + q_i - p_i$ gives the length of the longest path from 0 to $n+1$ that passes through i. We call node i a *bottleneck* node if $r_i + q_i - p_i > L$ holds. Obviously, we can limit ejecting nodes to the bottleneck nodes in the lexicographic ejection process. In addition, we define a *bottleneck* path as one whose length is greater than L in $G(\pi)$. We can see that any node in a bottleneck path is a bottleneck node. One should note that bottleneck nodes and paths will change after ejecting operations in the lexicographic ejection process (see Fig. 2). In the following, we use the term "temporal schedule" to refer to a temporal schedule obtained in the lexicographic ejection process.

The pseudo-code of procedure EJECTION(π, L) is shown in Algorithm 2. This algorithm has a hierarchical structure. Procedure MAINEJECTION(π, L) computes an initial state and procedure ROUTINE(k) tries to eject a k-th node from the temporal schedule (i.e., nodes $i_{out}(1), \ldots, i_{out}(k-1)$ are temporarily ejected in the higher-level ejections). In procedure ROUTINE(k), r_i^k and V_{bn}^k refer to r_i and a subset of the bottleneck nodes in the temporal schedule, respectively, which are dynamically updated in the lexicographic ejection process. As we can see from the update mechanism for V_{bn}^k (lines 2, 6-9, 17-18), all bottleneck paths in the temporal schedule certainly pass through at least one of the nodes in V_{bn}^k (see Fig. 2). Therefore, we can detect that the makespan of the temporal schedule is less than or equal to L as soon as V_{bn}^k becomes empty.

Now, we detail procedure ROUTINE(k). Lines 1[3] and 2 are required for backtracking. In the while loop (lines 3-20), the minimum node i in V_{bn}^k is selected as $i_{out}(k)$ and is removed from V_{bn}^k (lines 4-6). Then, we eject node i from the temporal schedule (modify the directed graph; delete i, link $JP(i)$ and $JS(i)$, and link $MP(i)$ and $MS(i)$) (line 7). If the length of the longest path that passes through arc $(JP(i), JS(i))$ is greater than L, $JS(i)$ is added to V_{bn}^k (line 8). The same applies to $MS(i)$ (line 9). If V_{bn}^k is empty, the k-tuple of currently ejected operations is stored to $\mathcal{I}$ (lines 10, 11). If this is not the case, proceed to the $(k+1)$-th ejection phase if $k < k_{max}$ (line 13). At lines 15-18, node i is re-inserted into the original position to shift the k-th ejection node (the directed graph is restored to that before line 7), r_i^k is then computed (line 16)[4], and V_{bn}^k is updated in the same manner as described above. After re-inserting node i, if the "finish" node $n+1$ becomes a bottleneck node, the subsequent k-th ejections can be pruned (line 19) because there exists a path $\langle 0, \ldots, i, n+1 \rangle$ whose length is greater than L.

[1] Actually, "excessive" ejections will be excluded. For example, if $C(\pi - \{1,2\}) \leq L$, combinations $\{1, 2, i\}$ $(i > 2)$ are excluded.

[2] For example, if $k_{max} = 3$, nodes are ejected in the following order: $\{1\}$, $\{1,2\}$, $\{1,2,3\}$, $\{1,2,4\}$, …, $\{1,2,n\}$, $\{1,3\}$, $\{1,3,4\}$, …, $\{1,n-1,n\}$, $\{2\}$, ….

[3] Actually, there is no need to copy all r_i^{k-1}. It is a simplified algorithm.

[4] Due to the lexicographic ejections in the topological order, when r_i^k is updated, $r_{JP(i)}^k$ $(r_{MP(i)}^k)$ has the true value if it is a bottleneck node or "-1" if otherwise.

Algorithm 2. Procedure EJECTION(π, L)

Procedure MAINEJECTION(π, L)
1 :$\mathcal{I} := \emptyset$;
2 :Calculate the topological order and q_i $(i \in V)$;
3 :$r_0^0 = 0,\ r_i^0 = -1$ $(i \in V \backslash \{0\})$;
4 :$V_{bn}^0 = \{i \in V \mid q_i > L,\ JP(i) = 0,\ MP(i) = 0\}$;
5 :ROUTINE(1);
6 :**return** $\mathcal{I}$;

Procedure ROUTINE(k)
1 :$r_i^k = r_i^{k-1}$ $(i \in V)$;
2 :$V_{bn}^k = V_{bn}^{k-1}$ (copy all elements);
3 :**while**
4 : Set $i :=$ the minimum node in V_{bn}^k;
5 : $i_{out}(k) = i$;
6 : $V_{bn}^k := V_{bn}^k \backslash \{i\}$;
7 : Modify the directed graph (eject i);
8 : **if** $r_{JP(i)}^k + q_{JS(i)} > L$ **then** Add $JS(i)$ to V_{bn}^k;
9 : **if** $r_{MP(i)}^k + q_{MS(i)} > L$ **then** Add $MS(i)$ to V_{bn}^k;
10: **if** $V_{bn}^k = \emptyset$ **then**
11: Add $\{i_{out}(1), \ldots, i_{out}(k)\}$ to $\mathcal{I}$;
12: **else**
13: **if** $k < k_{max}$ **then** ROUTINE$(k+1)$;
14: **end if**
15: Modify the directed graph (re-insert i to the original position);
16: $r_i^k = \max\{r_{JP(i)}^k, r_{MP(i)}^k\} + p_i$;
17: **if** $r_i^k + q_{JS(i)} > L$ **then** Add $JS(i)$ to V_{bn}^k;
18: **if** $r_i^k + q_{MS(i)} > L$ **then** Add $MS(i)$ to V_{bn}^k;
19: **if** $n + 1 \in V_{bn}^k$ **then break**;
20:**end while**

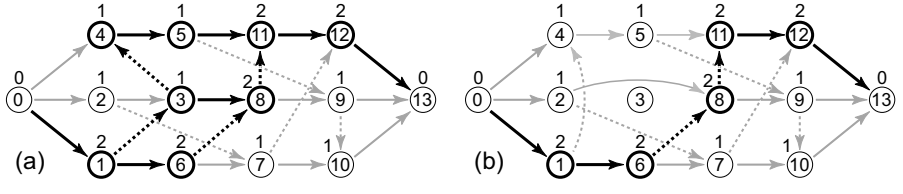


Fig. 2. Examples of lexicographic ejections where nodes are labeled according to the topological order of a schedule. Bottleneck nodes and bottleneck paths are highlighted $(L = 8)$. Figure (a) illustrates an example of 1st ejection phase (ROUTINE(1)); nodes 1, 3, 4, 5, 6, 8, 11, and 12 will be ejected as $i_{out}(1)$ in this order. Figure (b) illustrates an example of 2nd ejection phase (ROUTINE(2)); node 3 is ejected as $i_{out}(1)$, and nodes 6, 8, 11, and 12 will be ejected as $i_{out}(2)$ in this order. V_{bn}^k is updated as follows: (a) $V_{bn}^1 = \{1\}$ (line 2), $V_{bn}^1 = \{1\}$ before ejecting node 1 (line 4), $V_{bn}^1 = \emptyset$ after ejecting node 1 (lines 6-9), and $V_{bn}^1 = \{3, 6\}$ after re-inserting node 1 (lines 17-18). $V_{bn}^1 = \{3, 6\}$ before ejecting node 3 (line 4), $V_{bn}^1 = \{6\}$ after ejecting node 3 (lines 6-9), and $V_{bn}^1 = \{4, 6, 8\}$ after re-inserting node 3 (lines 17-18), ..., (b) $V_{bn}^2 = \{6\}$ (line 2), ...

3.3 Algorithms of Procedures LocalSearch and Perturb

First, we describe the local search algorithm (procedure LocalSearch(π)) embedded in the GES framework. Here, we should design a simple algorithm because the GES framework will call the local search a number of times. In this research, we used a TS algorithm developed by Zhang *et al.* [12] with a little modification.

The pseudo-code of the local search is shown in procedure LocalSearch(π). The neighborhood, denoted by $\mathcal{N}(\pi)$, is based on the well-known $N6$ neighborhood. For an extensive review of classical local search techniques for the JSP, refer to [1]. The suggested local search uses only short-term memory (tabu list). The tabu list, denoted by $Tabu$, stores the most recent $maxT$ solutions in the search history (line 5). For each neighbor solution π', if both a sequence of operations and their positions in a machine (i.e., $\pi_m(s_1), \ldots, \pi_m(s_2)$) modified by a move ($m$, s_1 and s_2 are defined) matches one of the solutions in the tabu list, such a solution is defined as a tabu-solution.

In our TS algorithm, tabu-solutions are not accepted. We used the first-acceptance strategy to accept a neighbor solution (line 3) to accelerate the iterations. If a solution better than π is not found in the neighborhood, the best non-tabu solution is accepted (line 4). The iterations are continued until the best solution is not improved for $stagLS$ iterations (line 6).

Procedure LocalSearch(π)
1 :Set the tabu list $Tabu := \emptyset$ and π^*(best solution) $:= \pi$;
2 :**repeat**
3 : With the first-acceptance strategy, randomly select non-tabu solution $\pi' \in \mathcal{N}(\pi)$ such that $C(\pi') < C(\pi)$;
4 : **if** π' does not exist **then** Set π' to the best non-tabu solution among $\mathcal{N}(\pi)$;
5 : Set $\pi := \pi'$, update π^* (if the best solution is improved), and add π to $Tabu$;
6 :**until** Improvement of π^* stagnates for $stagLS$ iterations
7 :**return** π^*;

The randomization procedure (procedure Perturb(π, L)) is rather simple. Here, random feasible moves are executed for a given number of times ($iterRand$) under the constraint that the makespan is less than or equal to L. Each move is executed by swapping two randomly selected successive operations in a machine. Here, one should note that the pair of operations can be selected from non-critical paths contrary to the standard neighborhood for the JSP.

4 Computational Experiments

The GES algorithm was implemented in C++ and was executed on AMD Opteron 2.4 GHz (4-GB memory) computers. Several computational experiments have been conducted to analyze the performance of GES. In this section we present the results along with a comparative analysis.

4.1 Experimental Settings

The suggested GES algorithms (GES-1 and GES-2) were tested on well-known and widely used benchmarks in the literature: SWV01-10, YN1-4, and TA01-50. All instances are available from the OR-library (http://people.brunel.ac.uk/mastjjb/jeb/orlib/) or Taillard's web-site (http://mistic.heig-vd.ch/taillard).

GES has two groups of parameters: k_{max} and $maxIter$ for the main framework (procedure RoutineGES) and $stagLS$ and $iterRand$ for the sub procedures (procedures LocalSearch and Perturb). Parameters for the sub-procedures were fixed as follows for simplicity: $stagLS = 50$ and $iterRand = 100$. As for the parameter k_{max}, we tested three values (1, 2, and 3) for both GES-1 and GES-2. Parameter $maxIter$ was set to 10,000 for relatively easy instances (YN1-4 and TA-1-10) or 20,000 for hard instances (SWV01-10 and TA11-50) to clarify the difference in performance between GES-1 and GES-2 with different k_{max}. GES with each configuration was applied to each instance ten times.

To evaluate the performance, we present the quality of solutions and the running time. We evaluate the quality of solution π with the relative percentage error to the known lower bound of the optimal solution defined as $RE = (C(\pi) - LB)/LB \times 100$ because this measure with lower bound LB taken from [2] has been widely used in recent literature on the JSP. In the following, b-RE and a-RE refer to the best and average values of RE, respectively, over ten runs. In addition, a-T refers to the average running time in seconds over ten runs.

4.2 Analysis of GES

The results obtained by both configurations GES-1 and GES-2 with different k_{max} are presented in Table 1. Due to space limitation, only a-RE and a-T averaged over selected groups of instances are presented. (See also Table 2 to refer to the problem size and RE for the best-known solutions (UB)).

First, let us focus on GES-1. The solution quality of GES-1 with $k_{max} = 1$ is clearly inferior to those of GES-1 with $k_{max} = 2$ and 3. The reason is simply because the number of operations in the EP is always one in this configuration and thus the strength of the GES framework is spoiled. In addition, we find that the solution quality of GES-1 improves with increasing k_{max}. However, the computation time of GES-1 is significantly increased, in particular for the large instances (e.g., TA31-50), by increasing k_{max} from 2 to 3. The reason apparently comes from the computation time for procedure Ejection. Next, let us focus on GES-2. GES-2 is superior to GES-1 in terms of the solution quality, indicating that GES-2 is better suited to address the problem of $\mathcal{I}(\pi, L)$ becoming empty. Note that when $k_{max} = 3$, a significant difference is not found in SWV01-10, YN1-4, and TA1-30 because $\mathcal{I}(\pi, L)$ becomes empty very occasionally in these instances.

In addition to the improvement in the solution quality, the computation time is also improved by GES-2. In our observation, when $\mathcal{I}(\pi, L)$ becomes empty, the lexicographic ejection process (see Section 3.2) tends not to be effectively pruned and the computation time per procedure Ejection will increase. Moreover, this

Table 1. Results of GES-1 and GES-2 with different k_{max}

	a-RE (%)						a-T (sec)					
Type	GES-1			GES-2			GES-1			GES-2		
Inst. \ k_{max}	1	2	3	1	2	3	1	2	3	1	2	3
SWV01-05	4.26	1.40	1.37	1.73	1.30	1.39	145	146	155	115	137	155
SWV06-10	12.20	8.56	8.15	8.45	8.12	8.20	200	205	267	157	170	255
YN01-04	8.90	6.98	6.85	6.93	6.93	6.95	113	105	158	75	83	131
TA01-10	1.23	0.09	0.09	0.17	0.08	0.08	55	47	48	42	44	48
TA11-20	4.36	2.80	2.71	2.92	2.73	2.70	183	168	204	126	135	173
TA21-30	7.82	6.01	5.84	5.95	5.83	5.89	207	201	327	150	160	241
TA31-40	3.09	1.29	0.90	0.86	0.83	0.80	403	406	2244	241	257	496
TA41-50	8.72	6.18	5.17	5.21	4.94	4.88	461	528	5098	273	313	1391

situation tends to continue in GES-1 because the insertion-ejection move is not performed in this situation (only the randomization procedure is applied). On the other hand, GES-2 can escape from this situation by ejecting more than k_{max} operations. In conclusion, GES-2 with $k_{max} = 2$ seems to be a good configuration for GES applied to the JSP.

Typical behaviors of GES are illustrated in Fig. 3. Due to space limitation, we present results of only GES-2 with $k_{max} = 2$ and 3 in instance SWV06. The graphs show both the makespan of the best complete schedule ($C(\pi^*)$) and the number of temporarily ejected operations (operations in the EP) against the number of iterations ($iter$). As we all know, the best solution is improved immediately after the EP becomes empty. The maximum number of temporarily ejected operations in both cases reaches 20 and more than 30, respectively.

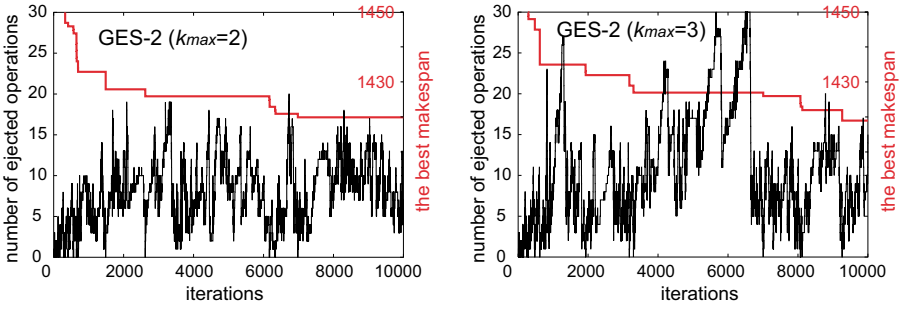


Fig. 3. Typical behaviors of GES-2 in instance SWV06 (20×15)

4.3 Comparisons with Other Algorithms

We compared the results of GES-2 ($k_{max} = 2$) with state-of-the-art metaheuristics for the JSP. Moreover, we tested GES-2 ($k_{max} = 2$) with a larger total number of iterations $maxIter$ (five times greater than those of the original values). The GES algorithms with these two configurations are referred to as GES-2 (small) and GES-2 (large). We selected three algorithms for comparison: i-TSAB by Nowicki and Smutnicki [9], TSSA by Zhang *et al.* [10], and i-STS/SGMPCS by Watson and Beck [11]. i-TSAB is a hybrid tabu search/path relinking algorithm that

Table 2. Comparisons with state-of-the-art algorithms

Inst.	$(\|M\| \times \|J\|)$	UB RE	GES-2(small) b-RE	GES-2(small) a-RE	GES-2(small) a-T	GES-2(large) b-RE	GES-2(large) a-RE	GES-2(large) a-T
SWV01-05	(20 × 10)	0.70	0.93	1.30	137	0.83	1.05	674
SWV06-10	(20 × 15)	6.86	7.05	8.12	170	6.62	7.07	834
YN01-04	(20 × 20)	6.42	6.56	6.93	83	6.39	6.64	411
TA01-10	(15 × 15)	0.00	0.01	0.08	44	0.00	0.01	214
TA11-20	(20 × 15)	2.28	2.38	2.73	135	2.29	2.46	655
TA21-30	(20 × 20)	5.36	5.44	5.83	160	5.39	5.55	791
TA31-40	(30 × 15)	0.48	0.60	0.83	257	0.55	0.66	1167
TA41-50	(30 × 20)	3.93	4.38	4.94	313	4.08	4.39	1494
Computer			Opteron 2.4 GHz			Opteron 2.4 GHz		

Inst.	i-TSAB RE	i-TSAB T	TSSA b-RE	TSSA a-RE	TSSA a-T	i-STS/SGMPCS b-RE	i-STS/SGMPCS a-RE	i-STS/SGMPCS a-T
SWV01-05	1.01	660	0.78	1.76	138			
SWV06-10	7.49	935	6.91	8.66	190			
YN01-04			6.44	6.99	109			
TA01-10	0.11	79	0.01	0.11	65			
TA11-20	2.81	390	2.37	2.92	235	2.26	2.45	1800
TA21-30	5.68	1265	5.43	5.97	433	5.52	5.71	1800
TA31-40	0.78	1225	0.55	0.93	370	0.50	0.68	1800
TA41-50	4.70	1670	4.07	4.84	846	4.22	4.63	1800
Computer	Penti. 900 M		Penti.IV 3.0 GHz			Opteron 2.0 GHz		

had dominated other algorithms since 2003 until very recently. TSSA is a hybrid tabu search/simulated annealing algorithm, and i-STS/SGMPCS is a hybrid tabu search/constraint programming algorithm. The latter two algorithms were proposed very recently and are competitive with i-TSAB. Here, one should note that these three algorithms dominate all other metaheuristics, although a number of algorithms have been proposed for the JSP.

The results are shown in Table 2. b-RE, a-RE, and a-T obtained over ten independent runs are presented for all algorithms except for i-TSAB. Because i-TSAB was executed once for each instance, results obtained by only a single run are presented (denoted as RE and T). In the table, a blank cell means that a corresponding result is not available. In addition, the column labeled "UB" provides RE for the best-known solutions (upper bound) taken from [9][10][11]. The column labeled "Computer" provides specifications on the computers used in the experiments; we applied GES algorithms to some instances on different machines and observed that our computer (Opteron 2.4 GHz) is about 1.6 times faster than a Pentium 2.8 GHz machine.

First, we compare GES-2 and i-TSAB. In general, a-RE of GES-2 (small) is slightly inferior to RE of i-TSAB, but the computational effort of GES-2 (small) is smaller than that of i-TSAB even if the difference of the computer speed is considered. In contrast, a-RE of GES-2 (large) is superior to RE of i-TSAB, but the computational effort is larger than that of i-TSAB. Next, GES-2 is compared with TSSA. The computational effort of GES-2 (small) is roughly equal to that of TSSA. GES-2 (small) tends to be superior to TSSA in terms of a-RE, but GES-2 (small) tends to be inferior to TSSA in terms of b-RE. The last comparison is made with i-STS/SGMPCS. i-STS/SGMPCS was executed with a fixed computation time (30 minutes plus a few minutes (not clearly described)). GES-2 (large) is highly competitive with i-STS/SGMPCS in terms of both

b-RE and a-RE even though the computational effort is not larger than that of i-STS/SGMPCS. In conclusion, these results show that GES-2 ($k_{max} = 2$) is highly competitive with the three state-of-the-art metaheuristics for the JSP.

The ten runs of GES-2 (large) improved the best-known upper bound in eight instances: SWV06 (1672), SWV07 (1594), SWV09 (1655), SWV10 (1751), YN02 (905), TA20 (1348), TA37 (1775), and TA42 (1949).

5 Conclusion

The concept of GES was originally developed as a route minimization heuristic for the VRPTW. In this paper we have formulated GES for other combinational optimization problems and hence apply it to the JSP. We have demonstrated that the suggested GES algorithm is highly competitive with the state-of-the-art metaheuristics for the JSP, improving eight best-known solutions in the well-known benchmarks. Given that these good results were obtained by the simple concept of GES, GES appears to have good potential for developing effective solution algorithms for other combinatorial optimization problems.

References

1. Blazewicz, J., Domschke, W., Pesch, E.: The job shop scheduling problem: Conventional and new solution techniques. European Journal of Operational Research 93, 1–33 (1996)
2. Jain, A.S., Meeran, S.: Deterministic Job-Shop Scheduling: Past, Present and Future. European Journal of Operational Research 113, 390–434 (1999)
3. Nagata, Y., Bräysy, O.: A Powerful Route Minimization Heuristic for the Vehicle Routing Problem with Time Windows (under submission)
4. Lim, A., Zhang, X.: A two-stage heuristic with ejection pools and generalized ejection chains for the vehicle routing problem with time windows. Informs Journal on Computing 19, 44–457 (2007)
5. Voudouris, C., Tsang, E.: Guided local search, Technical Report CSM-247, Department of Computer Science, University of Essex, UK (August 1995)
6. Adams, J., Balas, E., Zawack, D.: The shifting bottleneck procedure for job shop scheduling. Management Science 34, 391–401 (1988)
7. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. Discrete Applied Mathematics 65, 223–253 (1992)
8. Shaw, P.: Using constraint programming and local searchmethods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
9. Nowicki, E., Smutnick, C.: An Advanced Tabu Search Algorithm for the Job Shop Problem. Journal of Scheduling 8, 14–159 (2005)
10. Zhang, C., Li, P.G., Rao, Y.Q., Guan, Z.: A very fast TS/SA algorithm for the job shop scheduling problem. Computers and Operations Research 35, 282–294 (2008)
11. Watson, J.-P., Chistopher, J.: A Hybrid Constraint Programming / Local Search Approach to the Job-Shop Scheduling Problem. In: Perron, L., Trick, M.A. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 263–277. Springer, Heidelberg (2008)
12. Zhang, C., Li, P.G., Guan, Z.L., Rao, Y.Q.: A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem. Computers and Operations Research 34, 3229–3242 (2007)

Improving Performance in Combinatorial Optimisation Using Averaging and Clustering

Mohamed Qasem and Adam Prügel-Bennett

School of Electronics and Computer Science
University of Southampton, SO17 1BJ, UK

Abstract. In a recent paper an algorithm for solving MAX-SAT was proposed which worked by clustering good solutions and restarting the search from the closest feasible solutions. This was shown to be an extremely effective search strategy, substantially out-performing traditional optimisation techniques. In this paper we extend those ideas to a second classic NP-Hard problem, namely Vertex Cover. Again the algorithm appears to provide an advantage over more established search algorithms, although it shows different characteristics to MAX-SAT. We argue this is due to the different large-scale landscape structure of the two problems.

1 Introduction

One of the potential benefit of using an evolutionary algorithm (EA) is the possibility to learn about the large-scale structure of the fitness landscape from the whole population. However, there are few examples of EAs on real world problems where the algorithm unambiguously exploits this global knowledge of the landscape. Recently we proposed an algorithm for solving large MAX-SAT problems based on clustering good solutions which we argued does precisely this [1]. We review that work here and extend the idea to a second classic NP-Hard problem, Vertex Cover. The algorithm behaves differently on Vertex Cover to MAX-SAT, although again the algorithm provides a performance improvement. We argue that the reason for the difference in behaviour of the algorithm on the two problems reflects difference in the large-scale structure of the fitness landscape.

The algorithm we use here is a hybrid algorithm. We find many good solutions using a local neighbourhood search algorithm (a basic hill-climber, BHC). The solutions are either averaged or clustered using a K-means cluster algorithm. The solution closest to the mean solution or centroid of each cluster is then used as a starting position for applying a second round of the local neighbourhood search algorithm. This very simple algorithm finds remarkably good solutions—we describe our tests of the algorithm in section 2.1.

Our interpretation for the good performance of this algorithm is that in many combinatorial optimisation problems good quality solutions tend to surround the global maxima. Thus by averaging good solutions, we can find a position in the vicinity of a global maximum, or, at least, a very high quality solution. As

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 180–191, 2009.

the landscapes are often rugged, the average solution (or more accurately the nearest feasible solution to the average) may not itself be very good, although by continuing the search from the average solution we are likely to find an improved solution. It may be thought that crossover performs a similar operation to averaging, however, this is not the case, as, in expectation, a child is as far from the centroid of the parents as the parents themselves (this is even true when more than two parents are used in crossover). This observation is backed up empirically, where we find crossover does not perform at all like averaging.

For many combinatorial optimisation problems the landscape is more complex because there are often global optima a substantial distance apart. We can imagine the local optima being like stars clustered into galaxies. The high quality solutions typically being close to the centre of the galaxies. In this case, if we average solutions lying in different galaxies the mean solution may not lie close to the centre of any galaxy. However, by first clustering solutions and using the centroid of each cluster we can hope that at least some centroids might be close to the centre of a galaxy. For MAX-SAT we found clustering was superior to averaging all solutions (although, averaging all solutions was found to be superior to hill-climbing, suggesting that the galaxies themselves are correlated—this interpretation is supported by direct empirical studies). In Vertex Cover clustering provided no significant improvement over averaging suggesting that the global optima for Vertex Cover may not be so widely spread over the search space as MAX-SAT.

The rest of the paper is arranged as follows. In the next section, we recap on the behaviour of our new method for MAX-SAT. In section 3 we discuss the new results for Vertex Cover. Finally, in section 4 we draw conclusions.

2 Recap on MAX-SAT

MAX-SAT is a generalisation of the well-known SAT problem. A SAT problem consists of a set of m clauses $\{C_i | i = 1, 2, \ldots, m\}$, where each clause C_i is formed by the disjunction of Boolean variables, $\boldsymbol{X} = (X_1, X_2, \ldots, X_n)$ or their negation, where $X_i \in \{true, false\}$. The SAT decision problem is whether there exists a truth assignment $\boldsymbol{X}$ which satisfies all the clauses. If the number of literals in each clause is K then the problem is called K-SAT. If $K \geq 3$, then K-SAT is NP-Complete. In MAX-SAT we attempt to find an assignment of the variables which maximises the number of satisfied clauses. MAX-SAT is an NP-Hard problem.

MAX-SAT is one of the best studied optimisation problems—in part because of its association with SAT, which, besides from its theoretical importance, has a huge number of practical applications. A large amount of research has gone into characterising the typical behaviour of random instances. In this paper, we also concentrate on random *fixed-length clause* instances [2]. These are created by generating m clauses from n variables by randomly picking K variables for each clause. Then with a probability of 0.5 a variables is negated. Duplicate clauses are discarded.

For this class of problem there is a transition between the case where most instances are satisfiable to the case where most instances are unsatisfiable, which occurs at a ratio of clauses to variables of $\alpha = m/n \approx 4.3$. This transition becomes increasingly sharp as the problem size increases and is viewed as an example of a classic first-order phase transition. At around the same ratio of clauses to variables, there is an observed change in the difficulty of problem instances with most instances being easy to solve below the phase transition while above the phase transition most instances are hard to solve [2,3]. That is, the empirical time complexity for most complete SAT solvers grows dramatically around this phase transition. For larger ratios of m/n the 3-SAT decision problem typically becomes easy again because it is straightforward to prove unsatisfiability, however the MAX-SAT problem remains hard [4,5].

2.1 Experimental Results

We briefly present empirical results of our algorithm on large instances of MAX-3-SAT for $\alpha = m/n = 8$. More details can be found in [1]. We were unable to compare our algorithm with most other algorithms that appear in the literature since the other studied were performed on much smaller instances (typically around 100 variables). For such small instances we found that running the basic hill-climber a few times would almost always find a solution we were unable to improve on and which we believe to be the global optimum. This made our clustering approach redundant. The only work we are aware of which studied similar sized instances to those used here is by Zhang [6]. Our algorithm substantially out-performs the results given in that paper. To provide some comparators to the clustering approach we ran a number of variants of the algorithms. The main purpose of these comparators was to rule out other possible explanations of why the approach we are taking is successful.

Experimental Setup. We generated random MAX-3-SAT instances using the method described in section 2. We consider problem instances ranging in size from 6 000 to 18 000 variables in increments of 2000 variables, and with $\alpha = m/n = 8$. These are difficult problems since they are in the over-constrained region. For each increment we generated 100 problems instances.

In all tests we carried out we started by performing 1 000 hill-climbs using BHC starting from different random starting configurations. The number of iterations started from 20 000 for 6 000 variables and ended with 50 000 iterations for 18,000 variables in increments of 5 000. The number of iterations were increased with the number of variables so that BHC would be given more opportunities to find better quality solutions. With the growth of the number of variables it becomes more difficult for a local search algorithm to reach local maxima [7], although the goal was not necessarily to reach a local maximum, but only to find a good solution. The best result for the 1 000 hill-climbs averaged over all 100 problem instances is shown in the second column of table 1. We then tested a number of different strategies to boost the performance obtained from these initial 1 000 points. The testing procedure we carried out is shown schematically

Table 1. Comparison of different algorithms. The tests were carried out on random MAX-3-SAT problems with $\alpha = 8.0$. Each test was performed on 100 problem instances for each number of variables. The K-means algorithm performs best. The results are based on the number of unsatisfied clauses.

#Vars	First BHC	Second BHC (1)	K-Means/ BHC (2)	Average/ BHC (3)	hybrid-GA	Perturb/ BHC	(2) - (1)	(3) - (1)
6000	1971.77	1448.35	1370.61	1385.82	2429.5	1447.92	77.74	62.53
8000	2944.03	2037.26	1913.26	1943.38	3691.22	2038.78	124	93.88
10000	3464.7	2614.65	2456.67	2507.56	4908.87	2617.19	157.98	107.09
12000	4235.8	3247.74	3051.09	3125.79	6218.57	3247.4	196.65	121.95
14000	4999.14	3892.06	3652.23	3761.51	7533.33	3895.38	239.77	130.55
16000	5711.81	4496.69	4226.15	4368.23	N/A	N/A	270.54	128.46
18000	6551.83	5256.28	4932.41	5129.12	N/A	N/A	323.87	127.16

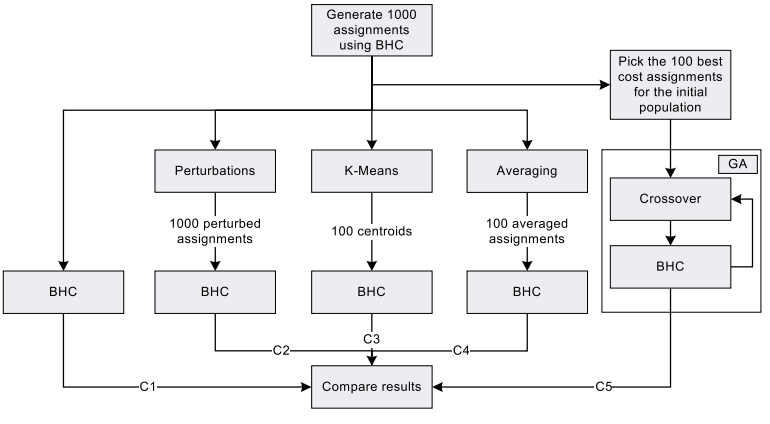


Fig. 1. Schematic diagram of the set of tests carried out and reported in table 1

in figure 1. As a baseline we repeated the basic hill-climber for same number of iterations on all 1 000 search points. These results are shown in the third column of table 1. This second round of hill-climbing shows that the solutions found in the first round were still some way away from being locally optimal.

***K*-Means and Averaging.** We next performed clustering using the K-means clustering algorithm [8] on the 1 000 search points found by the initial hill-climbing. This algorithm starts by assigning a random string on the n-cube to each of K initial "centres" (note that, in this section, K is used to denote the numbers of centres in K-means clustering and should not be confused with the number of variables in each clause). For a number of assignments $\boldsymbol{X}_k \in \{0,1\}^n$ with $k = 1, 2, \ldots, P$, the centre or average is defined to be

$$\boldsymbol{X} = \underset{\boldsymbol{X}\in\{0,1\}^n}{\operatorname{argmax}} \sum_k H(\boldsymbol{X}, \boldsymbol{X}_k)$$

where $H(\boldsymbol{X}, \boldsymbol{X}_k)$ is the Hamming distance between binary strings. Thus the average is a configuration which minimises the sum of distances to each assignment. Each of the 1 000 points is then assigned to the cluster with the nearest centre. The centres are then updated to be the centroid of the cluster. The points are reassigned to the nearest centroid and the process is repeated until there are no changes. This usually happens after five to ten iterations. Having computed the 100 centroids a second round of hill-climbing is then carried out. The results obtained after this procedure are shown in the forth column of table 1. In every case there is a considerable gain in performance compared to the baseline, although the K-means method used 100 points in contrast to the 1000 in the second round of hill-climbing (in consequence, the baseline method uses approximately 10 times as much CPU time in this second stage than the K-means clustering method). The gain in performance compared to the baseline in shown in column 8 of table 1.

We have compared clustering with 'averaging', where we randomly selected 10 points and averaged them to create a centroid. These were then rounded to give a valid assignment of the variables and a second cycle of hill-climbing carried out. This was repeated 100 times so as to give a fair comparison with the K-means clustering method. The results are shown in the fifth column of table 1. This again produced a substantial gain in performance compared with the baseline (the gain is shown in the last column of table 1), however, these gains are smaller than those obtained by K-means clustering. This seems to provide empirical support for the claim that the global maxima are clustered. It also shows that even the mean of all the good solutions provides a much better starting point than a random starting point.

To show that these results are *not* due to clustering or averaging acting as a macro-mutation which allows the search to escape out of local maxima we considered applying perturbations of 0.1%, 1%, 2%, 5% and 10% of the variables and then repeating hill-climbing. We found that doing this gave us worse performance than the baseline algorithm. Even with 0.1% the perturbation appears slightly detrimental (see column 7 of table 1).

Comparison with GAs. We also compared our algorithms against a hybrid genetic algorithm. This combined hill-climbing with selection and two-parent crossover. For selection we used scaled Boltzmann selection where we chose each member of the population with a probability proportional to $\exp(-\beta F_i/\sigma)$ where F_i is the fitness of individual, i; σ is the standard deviation of the fitness values in the population; and β controls the selection strength. Various values of β were tried, but this did not strongly affect the results. Uniform, single-point and multi-point crossovers were tried. The best results were obtained with single-point crossover. Column 6 of table 1 shows the best results we were able to obtain using a GA. Although we do not claim that all the parameters were optimally chosen, the results obtained by the hybrid-GA are extremely disappointing compared to the other algorithms.

The behaviour of the hybrid genetic algorithm appears particularly poor. This was due to the limited number of BHCs allowed for each algorithm. When given a

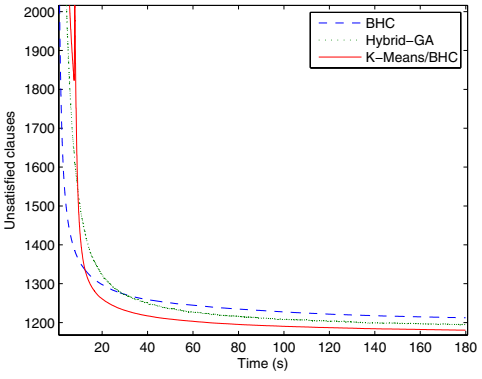


Fig. 2. Comparison of BHC, genetic algorithms and K-means clustering as a function of CPU time run on X instances of randomly generated MAX-3-SAT instances with 6000 variables at $\alpha = 8$. The large jump in fitness in the K-means algorithm after around 10 seconds marks the point where K-means clustering is carried out.

longer time the hybrid-GA performs considerably better. In figure 2 we show the average performance of parallel-BHC, K-means clustering and the hybrid-GA. Each algorithm was run for 3 minutes and the results were averaged over 100 instances of randomly generated MAX-3-SAT instances with 6000 variables at $\alpha = 8$. In parallel-BHC, we run 10 BHCs in parallel and show the best of these. K-means clustering was run starting with an initial population of 100 where we performed 27 000 BHCs before performing K-means clustering with $K = 10$ clusters and then running BHC starting from the 10 centroids. No tuning was performed on the K-means clustering algorithm. Finally we tested a hybrid-GA with a population of size 10 where we performed uniform crossover, Boltzmann selection with a selection strength of $\beta = 0.1$ and BHC. The parameters for the hybrid-GA were chosen after performing a large number of preliminary tests. As can be seen the GA outperforms BHC given enough time, but does not beat K-means on average, (although in some instances it does).

For larger problem instances the speed of K-means becomes more pronounced so that for problems with 18 000 variables run for 5 minutes K-means gave better performance than a hybrid-GA on every one of 50 instances that was tested.

2.2 Landscape of MAX-3-SAT

As described in the introduction we have attributed the performance of our algorithm to the clustering of the good quality solutions. Here we present some direct evidence in support of this picture obtained by extensive empirical observations on the fitness landscape of MAX-3-SAT for $\alpha = m/n = 8$. We studied instances up to size 100 by finding many local maxima. To achieve this we used the basic hill-climber. The algorithm was started from different, randomly-chosen, starting points. To ensure that we had found a local maximum, after running the

hill-climber with no improvements in many attempts we switched to an exhaustive search method that checked all neighbours at the same cost as the current point, and then checked their neighbours repeatedly, until either a fitter solution was found or else all neighbours at the current cost had been searched, in which case we could be sure that we were at a local maximum.

We postulate that the best local maxima we found are the global maxima, since if there were even a single maximum fitter than those we found then we would expect to find it with high probability given the number of hill-climbs we made (unless it had a very atypically small basin of attraction). We call our best maxima found in this way, *quasi-global maxima* as we believe them to be the true global maxima, although we have no proof of this[1].

We investigated the distribution of quasi-global maxima by examining the frequencies of Hamming distances between all quasi-global maxima in an instance. In figure 2.2, we show these frequencies averaged over 300 problem instances. To find the set of quasi-global maxima we ran BHC followed by exhaustive search 5000 times on each problem instance. The histogram has a large peak at a Hamming distance approximately equal to 5% of the total number of variables. This indicates a clustering of quasi-global maxima around each other. However, the histogram has a large tail with a second peak at a large Hamming distance away from the first. This is indicative of multiple clusters that are weakly correlated with each other (if there was no correlation then the clusters would be at a Hamming distance of $n/2$).

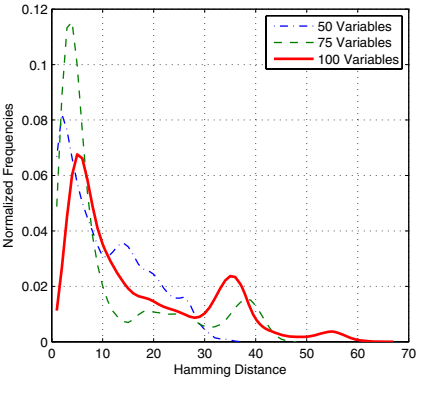


Fig. 3. Shows a histogram of the Hamming distance between quasi-global maxima for instances of size $n = 50$, 75, and 100 with $\alpha = 8$. There is a cluster of very close global maxima below a Hamming distance of 10. Also, a significant number of global maxima at a Hamming distances equivalent to 30–40% of the variables.

[1] For small problems, $n \leq 50$, we could find the true global maxima using a branch-and-bound algorithm. In every case, the best solution found by performing multiple BHC were true global maxima. We also tested problems with $n = 100$ from SATLIB and in every case we were able to find the best solution for the problem using BHC.

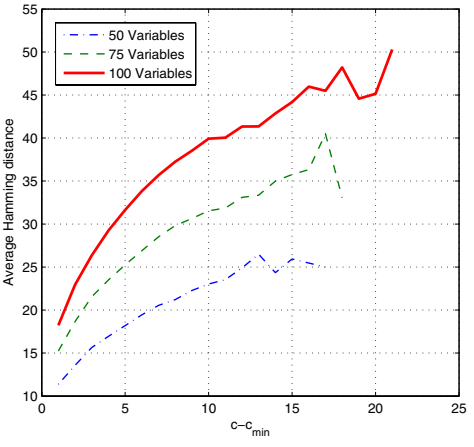


Fig. 4. The average Hamming distance between the quasi-global maxima and the local maxima. As the gap in fitness between the quasi-global maxima and local maxima decreases so does the average distance to the quasi-global minimum.

Figure 4 shows how the average Hamming distance between the local maximum and the nearest quasi-global maximum varies as a function of the difference in the cost between the local maxima and the quasi-global maxima. It is easy to understand why higher-cost solutions should be closely correlated on average with the quasi-global maxima as global-optimum solutions represent good ways of maximising the number of satisfied clauses. Therefore, nearby solutions are also likely to satisfy many clauses. However, what is perhaps more surprising is that the solutions whose cost differs by one from the quasi-global optima have a high average Hamming distance from any quasi-global optima. Even for relatively small problems with 100 variables this average Hamming distance is around 18 which is sufficiently large that the probability of a stochastic hill-climber reaching a global maximum from a local maximum is negligibly small.

Although it is always dangerous to rely on low-dimensional pictures to understand what happens in a high-dimensional space, nevertheless we offer the following caricature of our fitness landscape. We imagine the search space as being points on a 'world' where the height of the points representing the fitness values. This is schematically illustrated in figure 5. The good solutions lie in mountain ranges. The mountain ranges have hugely more foothills than high mountains. There are only a few mountain ranges in this world and they are slightly correlated (e.g. all the mountain ranges might lie in one hemisphere). The mountain ranges occupy only a very small proportion of the world. As with real mountain ranges, higher solutions tend to lie in the middle of the mountain ranges. Starting from a random position and hill-climbing we are likely to land up at a foothill, just because there are so many of them. Finding a good solution through hill-climbing alone will be very difficult. An alternative strategy is to perform a large number of hill-climbs starting from different

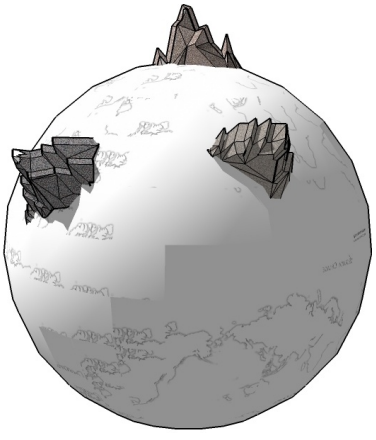


Fig. 5. Caricature of the Fitness Landscape showing the clustering of good solutions

randomly-chosen positions. We could then cluster the solutions we find after performing hill-climbing. If we are lucky, a cluster will correspond to a mountain range. The centres of the clusters corresponds to the regions with many high mountains so if we restart hill-climbing from the centre of a cluster we have a very good chance of finding a high quality solutions. Of course, this picture fails in many ways. The search space is not continuous, but is discrete. Furthermore, using a Hamming neighbourhood the topology of the search space is an n-dimensional hypercube. The high-dimensionality makes it harder for low-cost solutions to be local maxima since they have a large number of neighbours. Also the set of costs is discrete so that there is no gradient information. Nevertheless our algorithm based on clustering seems to perform very well which suggests that this simple picture might not be too misleading.

3 Vertex Covering

To further test our hypothesis that averaging or clustering is effective in other combinatorial problems we have tested the optimisation version of the Vertex Cover problem. A vertex cover for an undirected graph $G = (\mathcal{V}, \mathcal{E})$ (where $\mathcal{V}$ is the set of vertices, and $\mathcal{E}$ is the set of edges) is a subset of the vertices $\mathcal{S} \subseteq \mathcal{V}$ such that each edge has at least one end point in $\mathcal{S}$ [9]. The Vertex Cover decision problem is, given an integer R, does there exist a vertex cover. $\mathcal{S}$, such that $|\mathcal{S}| \leq R$? We consider an optimisation version of the Vertex Cover problem where we set $|\mathcal{S}| = R$ for a predetermine R and we seek a collection of elements of $\mathcal{S}$ which maximises the number of covered edges. Since the Vertex Cover problem is NP-complete, the optimisation version of the problem will be NP-hard.

To generate random problems we created 1 000 vertices, with an edge density of $p = 0.01$. That is an edge between two vertices is created with a probability of 0.01. For 1 000 vertices, the number of edges created was 4 995 on average. We

chose R to be 500 vertices, and used a basic hill-climber to find the 500 most covering vertices amongst the 1 000. (We tested our algorithm on instances with many different parameter values, but found very similar results in each case).

We applied three different algorithms to solve this problem. We applied a basic hill-climber (BHC), Averaging and a Hybrid-GA. We represented a solution by two sets $\mathcal{S}$ and $\bar{\mathcal{S}} = \mathcal{V} \backslash \mathcal{S}$. Initially the elements of $\mathcal{S}$ were randomly chosen with the constraint that $|\mathcal{S}| = R$. In (BHC) we chose a random element from $\mathcal{S}$ and another random element from $\bar{\mathcal{S}}$. These were exchanged provided the number of edges covered by the vertices after the exchange was, at least, as large at the number of edges covered by the vertices before the exchange. The "average" was taken to be the set $\mathcal{S}$ with the largest intersection with all members of the population $\mathcal{S}_k$ for $k = 1, 2, \ldots, P$. As with MAX-SAT averaging is used just once, in this case after 500 basic hill-climbing iterations. We also tried performing K-mean clustering in a similar manner to that described for MAX-SAT, but this gave us no noticeable benefit compared with averaging.

In the Hybrid-GA we combined BHC with selection and crossover. After a set number of hill-climbs (500 in the case shown), $2 \times P$ members were selected using scaled Boltzmann selection with a selection strength of $\beta = 4$. The selected members were paired up and crossed to create P children. A child is produced from two parents, $\mathcal{S}_k$ and $\mathcal{S}_l$ by randomly choosing R elements from $\mathcal{S}_k \cup \mathcal{S}_l$. We experimented with different parameter values for the Hybrid-GA but were unable to find parameters where it out-performed BHC.

The results averaged over 1000 different instances of the problem are shown in Figure 6. The performance of the Hill-Climber and the Hybrid-GA are indistinguishable on the graph. Averaging produces a rapid improvement in performance. Although not very clear in the graph, the results after averaging are significantly better than either Hill-Climbing or the Hybrid-GA even at the end of the run. Interestingly, in Vertex Cover averaging produces an immediate improvement in fitness. This is in contrast to MAX-SAT where initially averaging decreases the fitness (although in the longer term it leads to a more rapid increase in fitness), see Figure 2. This suggests that the landscape for Vertex Cover is much less rugged than MAX-SAT, at least, for the instances we chose.

An explanation of why clustering provides a performance advantage for Max-Sat, but not for Vertex Cover is that in Max-Sat many of the global optima are weakly correlated while in Vertex Cover the global optima lie in a much more confined part of the search space. To gain support for this view we have run independent hill-climbers on both problems and measured the correlation between solutions over time. Figure 7 shows the evolution of this correlation versus the number of hill-climbing steps. Initially, the hill-climbers are randomly chosen so on average they have zero correlation with each other. Over time the hill-climbers move towards fitter solutions. In both cases the solutions become correlated, however, this correlation is much more pronounced in Vertex Cover, suggesting that good solutions lie in a much smaller region of the search space than for MaxSat.

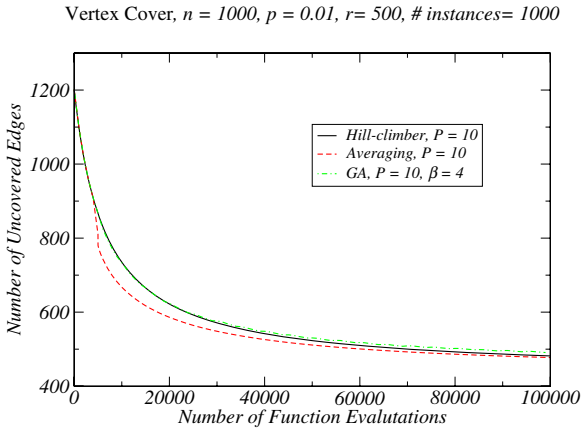


Fig. 6. Comparison of BHC, Hybrid-GA, and K-means clustering as a function of the number of evaluations. We see the the same jump in fitness we previously saw in the MAX-3-SAT problem.

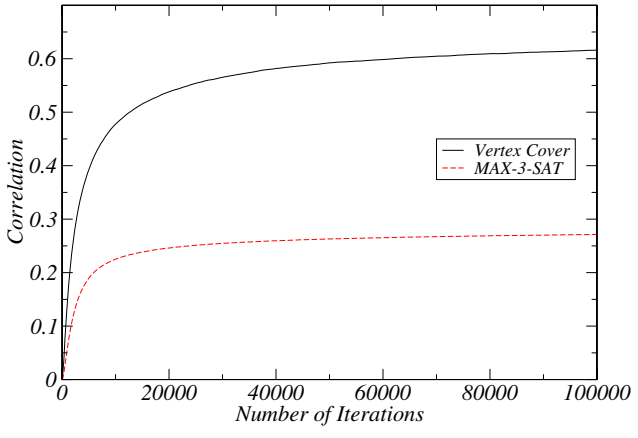


Fig. 7. Correlation between solutions found by hill-climbers versus number of iterations. The results are averaged over 100 instances of each problem. The Vertex Cover graphs are drawn from a distribution with $n = 1000$, $p = 0.01$ where $R = 500$. The MAX-3-SAT intances are for $n = 1000$ and $m = 8000$.

4 Conclusions

Clustering and averaging is shown to provide an important new search operator for solving hard optimisation problems. For MAX-3-SAT it provides a substantial improvement over other techniques. In Vertex Cover the improvement is less pronounced although it is still useful especially for finding good solutions quickly. From our studies of the landscape of both problems, the main difference would

appear to be that Vertex Cover is not particularly hard—or more accurately, the instances we chose are not particularly hard.

However, the success of averaging and clustering on these two classic combinatorial optimisation problems is encouraging. The success of these operators relies on two important features of a landscape. Firstly good solutions should be clustered around global optima (which is not difficult to imagine will often be the case). Secondly, these solutions should be isotropically clustered. If they were strongly biased in one direction then averaging may mislead the search. It is far from obvious that this isotropic clustering holds in real optimisation problems. The evidence from both MAX-3-SAT and Vertex Cover is that this is the case, which provides hope that this approach is applicable to more problems. We are currently extending this technique to new problems and particularly to problems which are known to by typically hard.

References

1. Qasem, M., Prügel-Bennett, A.: Learning the large-scale structure of the max-sat landscape using populations. IEEE Transactions on Evolutionary Computation (2008) (submitted)
2. Mitchell, D., Selman, B., Levesque, H.: Hard and easy distributions of SAT problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence, San Jose, CA, USA, pp. 459–465. AAAI, Menlo Park (1992)
3. Crawford, J.M., Auton, L.D.: Experimental results on the crossover point in random 3-SAT. Artificial Intelligence 81(1-2), 31–57 (1996)
4. Zhang, W.: Phase transitions and backbones of 3-SAT and maximum 3-SAT. In: Walsh, T. (ed.) CP 2001. LNCS, vol. 2239, pp. 153–167. Springer, Heidelberg (2001)
5. Prügel-Bennett, A.: Symmetry breaking in population-based optimization. IEEE Transactions on Evolutionary Computation 8(1), 63–79 (2004)
6. Zhang, W.: Configuration landscape analysis and backbone guided local search. part 1: satisfiability and maximum satisfiability. Artificial Intelligence 158(1), 1–26 (2004)
7. Qasem, M., Prügel-Bennett, A.: Complexity of max-sat using stochastic algorithms. In: GECCO (2008)
8. Hartigan, J.A., Wong, M.A.: A k-means clustering algorithm. Applied Statistics 28(1), 100–108 (1979)
9. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-completeness. Freeman, New York (1979)

Iterated Local Search for Minimum Power Symmetric Connectivity in Wireless Networks

Steffen Wolf and Peter Merz

Distributed Algorithms Group
University of Kaiserslautern, Germany
{wolf,pmerz}@informatik.uni-kl.de

Abstract. The problem of finding a symmetric connectivity topology with minimum power consumption in a wireless ad-hoc network is NP-hard. This work presents a new iterated local search to solve this problem by combining filtering techniques with local search. The algorithm is benchmarked using instances with up to 1000 nodes, and results are compared to optimal or best known results as well as other heuristics. For these instances, the proposed algorithm is able to find optimal and near-optimal solutions and outperforms previous heuristics.

1 Introduction

Wireless ad-hoc networks have received a lot of research attention recently [1]. Nodes in such networks usually carry their own power supply, which makes the wireless ad-hoc network a good choice for a first responders infrastructure, or as the main communications infrastructure in regions where installing a wired infrastructure would be too expensive or even infeasible.

Communication in such ad-hoc networks can be performed by adjusting the transmission power of the sending node to reach the recipient. However, because the transmission power is a polynomial function of the distance, the total energy consumption can often be reduced by using intermediate nodes [1]. E. g., if the power consumption is proportional to the squared distance (this is the case when there are no obstacles), sending to an intermediate node and having it relay the message to the final recipient can cost only half the energy of a direct connection to the destination.

Because of the limited battery power of each node, it is crucial to find communication topologies that minimize the energy consumption, leading to two opposing aims when setting up wireless ad-hoc networks: The network lifetime should be high, so settings with lower transmission powers are preferred. On the other hand, the network has to stay connected, so too low transmission ranges are discouraged. The connectivity of the network can be defined as a strong connectivity or as symmetric connectivity. Whereas symmetric connectivity only allows bidirectional links for communication, the strong connectivity allows different paths to be used for different directions. However, many of the low layer protocols in wireless networks require bidirectional links (e. g. CSMA/CA RTS/CTS in IEEE 802.11), so strong connectivity alone is not enough in those settings [1].

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 192–203, 2009.

In this work, we are searching for the wireless topology that minimizes the total energy consumption but still gives a symmetrically connected network. This problem is known as the Minimum Power Symmetric Connectivity Problem (MPSCP) [2,3,4,5]. We present a new Iterated Local Search for the MPSCP and compare its solutions against optimal solutions as well as other heuristics.

This paper is structured as follows. In the remainder of this section we give a formal definition of the MPSCP and summarize related work. In Section 2 we present our heuristic, and then give results of experiments carried out with this heuristic in Section 3. Section 4 summarizes our findings and gives an outline for future research.

1.1 Minimum Power Symmetric Connectivity Problem

The Minimum Power Symmetric Connectivity Problem (MPSCP) is an NP-hard optimization problem [2,6]. It is also known as the Strong Minimum Energy Topology (SMET) problem [6] or the Weakly Symmetric Range Assignment (WSRA) problem [1]. The MPSCP in an ad-hoc wireless network $G = (V, E, d)$ can be defined as the problem of finding a spanning tree $T = (V, E_T)$ (an undirected tree, defined by the set of edges $E_T \subseteq E$), that minimizes the necessary total transmission power $c(T)$ to connect all nodes of the network via bidirectional links:

$$c(T) = \sum_{i \in V} \underbrace{\max \{d(i,j)^\alpha | \{i,j\} \in E_T\}}_{\text{transmission power of node } i}$$

Here, the distance function $d : E \to \mathbb{R}^+$ refers to the Euclidean distance and the constant α is the distance-power gradient, which is 2 in an ideal environment, but can also vary from 1 to more than 6 in other environments [1]. Each node is required to set its transmission range to reach the farthest neighbouring node in the tree. Note that every node contributes to the total cost, as each node needs to reach at least one other node in the network.

1.2 Related Work

The MPSCP is first introduced by Călinescu *et al.* in [2] as a variant of another Range Assignment Problem searching for strong connectivity only [7,8]. They show that the Minimum Spanning Tree (MST) is a 2-approximation for the MPSCP, based on a similar result for the strong connectivity variant in [8]. This approximation ratio is shown to be tight using the example from Fig. 1. They also give an approximation scheme based on k-restricted decomposition yielding an approximation ratio of $1 + \ln 2 + \varepsilon \approx 1.69 + \varepsilon$.

The problem is rediscovered by Cheng *et al.* in [6], where the authors emphasize that the problem is different than the strong connectivity range assignment problem and continue to give a new NP-proof. They also rediscover the MST as a 2-approximation, and present a new greedy heuristic, later called Incremental Power: Prim (IPP). This heuristic builds the tree in a way that resembles Prim's algorithm for building MSTs. Since IPP explicitly exploits the fact that nodes

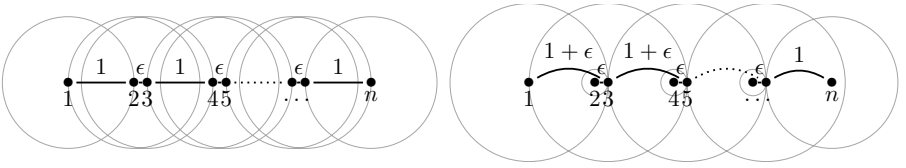


Fig. 1. Tight example for performance ratio of MST approximation. The MST on the left needs a total power of $n \cdot 1^\alpha$, each node needs to send over the larger link. The optimal assignment on the right needs $n/2 \cdot (1+\varepsilon)^\alpha + (n/2 - 1) \cdot \varepsilon^\alpha + 1^\alpha \overset{\varepsilon \to 0}{=} n/2 + 1$. Only about half the nodes need to send over a larger link. Example taken from [2].

can reach closer nodes for free when they already have to send to a farther node, it produces solutions with lower costs than the corresponding MST solutions.

Improving upon these construction heuristics, Althaus *et al.* [3,4] present two local search heuristics. In the edge-switching (ES) heuristic, edges from the tree are replaced by non-tree edges re-establishing the connectivity. In the edge-and-fork-switching (EFS) heuristic, not only edges, but also pairs of edges sharing one node, so-called forks, are inserted in the tree, and the resulting cycles are cut again by removing other edges. In both heuristics, the largest reduction in cost is chosen, and the process is repeated until a local optimum is reached. Forks were already used in [2] as 3-restricted decompositions, but the EFS produces better results. Average improvements over the MST of up to 6 % are achieved. The authors also try filtering out edges to reduce computation complexity, and concentrate on Delaunay edges only. This may also filter out edges that are part of the optimal solution, so the results using the filtered local searches are weaker.

Park *et al.* [9] give an experimental survey over different construction and local search heuristics. They name the greedy heuristics based on Prim or Kruskal Incremental Power: Prim and Kruskal (IPP and IPK), respectively, and present a new local search ES2 that applies the best double edge switch. They also show that the approximation ratio of 2 is not reduced by these heuristics. In the experiments, problem instances of up to 100 nodes are used, but no computation times are provided. Average improvements over the MST of up to 6 % are presented. For the ES2 local search, the average improvements are said to be as high as 14 %. We were unable to reproduce these values using similar random instances. In our instances, the *optimal* solution is about $6-7$ % below the MST. However, their instances may contain some properties that are unknown to us.

Several Mixed Integer Programming formulations (MIP) have been presented to compute optimal solutions. Althaus *et al.* [3,4] give a formulation using an exponential number of inequalities to ensure connectivity and to forbid cycles. A cutting plane algorithm is used to find optimal values for problem instances of up to 40 nodes (or up to 60 nodes when only Delaunay edges are considered). Montemanni *et al.* [5] give two MIP formulations based on incremental power. In the first formulation they avoid the problem of an exponential number of inequalities by defining n^3 flow variables. The second formulation is incorporated in an exact solver EX2, which iteratively solves the MIP and adds necessary

constraints. Also, the authors present a filtering technique (see Section 2), that greatly reduces the computation time. Problem instances of up to 40 nodes were solved with these formulations.

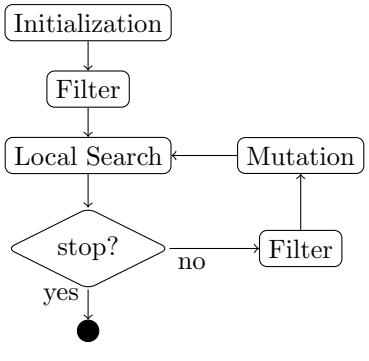
In these papers, many approximation algorithms have been presented, reducing the approximation ratio from 2 (MST, IPP, IPK) down to $1 + \ln 2 + \varepsilon$ [2] and $5/3 + \varepsilon$ [3,4]. Recent theoretical advances on approximation ratios for the more general Minimum Power k-Connected Subgraph problem can be found in [10].

Another problem in wireless networks is the Minimum Energy Broadcast Problem (MEB), where one node needs to send data to all other nodes. However, heuristics for the MEB (such as [11]) cannot be reused for the MPSCP, because of the bidirectional links in MPSCP. The local search and the mutation operator need to take the additional costs for the back-links into account.

2 Iterated Local Search

The MPSCP heuristic presented here is based on *iterated local search* [12]. It operates on a global view of the wireless ad-hoc network. It also incorporates a filtering technique first presented in [5] to exclude edges that cannot be part of the optimal solution. The general outline of the algorithm is shown in Fig. 2.

Filter. In order to reduce the number of edges to be considered by the heuristic, we apply a filtering technique as described in [5]. Since each node needs to have a transmission range at least high enough to reach its nearest neighbour, we can calculate a very weak lower bound. The filter then checks whether adding an edge $\{i, j\}$ increases this cost beyond an upper bound determined by the best solution found so far by the heuristic. Such edges are marked and are not used in the following operations. The filter is first applied after the initialization. Note that more edges are filtered out during the run of the heuristic when better solutions are found. Note also, that a larger edge might be allowed, although smaller edges starting from the same node are filtered out. This is the case, for example, when this node is the nearest neighbour of a remote node.



$T \leftarrow$ INITIALIZATION(E)
$F \leftarrow$ FILTER(E, T)
$T \leftarrow$ LOCALSEARCH(E, T, F)
while $\neg$ terminated **do**
  $F \leftarrow$ FILTER(E, T)
  $T_{\text{new}} \leftarrow$ MUTATION(E, T, F)
  $T_{\text{new}} \leftarrow$ LOCALSEARCH(E, T_{new}, F)
  if $c(T_{\text{new}}) < c(T)$ **then**
    $T \leftarrow T_{\text{new}}$
  end if
end while

Fig. 2. General overview of the Iterated Local Search

```
procedure ES(E, T, F)
repeat
  best savings ← 0
  Calculate all possible savings for removing edges {x, y} ∈ E_T
  for each edge {i, j} ∈ E \ (E_T ∪ F) do
    Temporarily insert {i, j} in E_T     // increases transmission ranges of i and j
    Find edge {x, y} on path from i to j in E_T \ {i, j} whose removal saves most
    savings ← saved cost from {x, y} − additional cost from {i, j}
    if savings > best savings then
      best savings ← savings
      (i*, j*, x*, y*) ← (i, j, x, y)
    end if
    Remove edge {i, j} again from E_T    // restores transmission ranges of i and j
  end for
  if best savings > 0 then
    Insert edge {i*, j*} in E_T and remove edge {x*, y*} from E_T
                                         // adjusts transmission ranges accordingly
  end if
until best savings = 0
```

Fig. 3. Edge switching search (ES), using filter F, implemented as an insert search

Initialization. The initial solution is created using MST or the Prim-like construction heuristic IPP. Using MST and IPP bears the advantage that there is an upper bound for the solutions of the Iterated Local Search heuristic, since both heuristics already give a 2-approximation for the MPSCP [2,9]. Although IPP tends to produce better solutions, in some cases its solutions are slightly worse than the MST. In our heuristic, we simply take the better solution as initialization.

Local Search. After initialization and mutation, a local search is applied to further improve the current solution. We use a modified edge switching search (ES), as well as an edge and fork switching search (EFS) as defined in [3,4]. We also use a faster subtree moving search (ST), as defined in [13]. Each local search heuristic follows a best improvement strategy. The local search is restarted whenever an improving step was found, until a local optimum has been reached.

The pseudocode for the modified edge switching search (ES) is shown in Fig. 3. The time complexity of one step of ES is $\mathcal{O}(n^3)$, since there are $\mathcal{O}(n^2)$ edges $\{i, j\}$ to be considered, and the path between any two nodes $i, j \in E$ contains at most $n - 1$ edges. In our implementation, the simple graph operations such as removing, finding or inserting an edge take constant time. Calculating the possible savings for every edge is done at the beginning of each step of the ES. Only edges adjacent to nodes i and j need recalculation, since the newly included edge $\{i, j\}$ changes the precalculated savings. The local search was modified not to insert edges that cannot be part of an optimal solution and are therefore filtered out (F) in a pre-processing step. The unmodified ES can be simulated by setting $F = \emptyset$.

```
procedure EFS(E, T, F)                    // F is ignored in this unfiltered version
repeat
  best savings ← 0
  for each edge {i, j} ∈ E_T do
    Temporarily remove {i, j} from E_T  // reduces transmission ranges of i and j
    for each edge {k, l} ∈ E_T do
      Temporarily remove {k, l} from E_T // reduces transmission ranges of k and
      l
      if saved costs > best savings then
        Find edges {x, y} and {y, z} reconnecting the separated trees, that mini-
        mize the additional cost
        if saved costs − additional costs > best savings then
          best savings ← saved costs − additional costs
          (i*, j*, k*, l*, x*, y*, z*) ← (i, j, k, l, x, y, z)
        end if
      end if
      Re-insert edge {k, l} in E_T        // restores transmission ranges of k and l
    end for
    Re-insert edge {i, j} in E_T          // restores transmission ranges of i and j
  end for
  if best savings > 0 then
    Remove edges {i*, j*} and {k*, l*} and insert edges {x*, y*} and {y*, z*}
                                          // adjusts transmission ranges accordingly
  end if
until best savings = 0
```

Fig. 4. Edge and fork switching search (EFS), implemented as a deletion search

Another local search is the edge and fork switching search (EFS), shown in Fig. 4. Here, two edges are removed from the tree. There are $n-1$ edges in E_T. Removing two edges splits the tree in three parts, each of size $\mathcal{O}(n)$. These three parts are reconnected by inserting a fork $\{x,y\}+\{y,z\}$ with x, y, z taken from different parts. Out of all possible deletion and insertion combinations, the one that reduces the total cost the most is taken. This process is repeated until no further improvements can be made. The time complexity of one step of EFS is $\mathcal{O}(n^5)$.

A third local search is the subtree moving search (ST). Here, an edge is removed from the tree. The node adjacent to the removed edge in one part of the tree is made the root of this subtree. The subtree is reconnected with the main tree by connecting its root node to another node in the main tree. This can be done by forcing $x = i$ in Fig. 3, reducing the time complexity of one step of this local search to $\mathcal{O}(n^2)$.

More expensive local searches are also possible. For example, EFS could be extended to allow two non-adjacent edges to reconnect the three parts, thus yielding the ES2 search from [9] with time complexity $\mathcal{O}(n^6)$.

Mutation. Since local search alone will get stuck in local optima, we use mutation to continue the search. The mutation operator should change the solution enough to leave the attraction basin of the local optimum, but it should also avoid

changing the solution too much and destroying already promising structures. Another criterion for the mutation operator is its relation to the local search. The mutation operator should not be easily reversible by the local search. In our heuristic, we use the following two mutation operators: random range increase and random edge exchange.

In the first mutation operation, the transmission power of a randomly chosen node is increased to a random level. All reached nodes adjust their transmission power to establish bidirectional links to the first node. Unnecessary connections, i. e. those that would introduce a cycle, are cut without adjusting the transmission power. The local search can later adjust the higher transmission powers or use them to reach other nodes. The cost of the solution increases according to the sum of the power level changes. Since this mutation operation could result in star-like topologies, we applied the same filters as for the local search. Also, with 20 % probability we increased the transmission range of the selected node to the largest level allowed by the filters.

In the second mutation operation, we remove a random edge from the tree and insert another random edge to reconnect the tree. The transmission ranges of the involved nodes are adjusted accordingly. However, new connections that could be established because of the increased ranges are not inserted in the tree. This can be done later by the local search heuristics.

Termination criterion. We stop the ILS after 200 iterations. For the smaller instances ($n \leq 50$), this setting is already too high. However, for the larger instances ($n \geq 500$) and especially for the weakest local search (ST), this setting should be increased in order to find optimal solutions instead of near-optimal solutions. Because of the large running times of the stronger local searches, we increased the number of iterations to 2000 only for the ST local search.

3 Experiments

For our experiments, we used two sets of well established test instances [11]. Each set contains 30 instances, where n nodes are randomly located in a $10\,000 \times 10\,000$ grid, using a uniform distribution. Euclidean distance was used and the distance-power gradient was set to $\alpha = 2$ as in an ideal environment. The sets use only $n \in \{20, 50\}$ nodes, so we added some larger sets ($n \in \{100, 200, 500, 1000\}$), as well as clustered instances ($n \in \{100, 200, 500\}$, with $c = \lfloor \sqrt{n} \rfloor$ clusters of size 1000×1000, the cluster centres placed at a random location in the $10\,000 \times 10\,000$ grid). All sets can be found at `http://dag.cs.uni-kl.de/research/rap/`.

Each experiment was repeated 30 times and average values are used for the following discussion. Calculation times refer to the CPU time on a 2.5 GHz Intel Xeon running 64 bit Linux 2.6; the algorithm was implemented in C, and only one CPU core was used for each experiment.

3.1 Obtaining Optimal Solutions

The commercial MIP solver CPlex 10.1 [14] was used together with the exact algorithm EX2 from [5] to obtain optimal solutions for the problem instances.

EX2 uses the same filtering technique as described in Section 2. We used the best known solution found by any of our experiments as an upper bound and filtered out all edges that would induce a larger cost. Since the MPSCP is NP-hard, EX2 was only able to provide optimal solutions for the 20 and 50 nodes problems, taking about 2 s and 10 mins, respectively. Three of the 100 nodes instances were solved to optimality, taking up to 30 days. Other instances of this set occupied the EX2 algorithm even longer, in one instance more than five months, without reaching a feasible solution.

We refrained from trying to solve all 100 nodes problems to optimality, or any of the larger problems. However, the intermediate solutions found by EX2 can still be used as lower bounds, as they represent topologies that need minimal power, but are not connected. EX2 is not suitable for clustered instances, though, as the intermediate solutions are even below the simple lower bound given by half the cost of the MST solution. The intermediate topologies tend to separate the clusters, thus leaving out the most expensive edges.

3.2 Results

We use the following table format to present our results: Each row gives the average values for all 30 problem instances of the same set. The first column denotes the problem size, where clustered instances are shown as 100c10, 200c14, and 500c22. The second column gives the improvements compared to the simple MST heuristic. The third column gives the calculation time in CPU seconds. The fourth column gives the excess over the optimum or the best known solutions. These best known solutions are taken from the experiments presented in this section. We believe only a few of these solutions up to $n = 500$ to be not optimal, but for $n = 1000$ the best known solutions can probably still be improved.

Table 1. Results for the simple heuristics ES and EFS starting from the MST

	edge switching (ES)			edge and fork switching (EFS)		
Size	Improvement to MST	CPU time	Excess over best known	Improvement to MST	CPU time	Excess over best known
20	4.18 %	0.00 s	0.250 %	4.33 %	0.00 s	0.092 %
50	6.04 %	0.00 s	0.541 %	6.32 %	0.07 s	0.243 %
100	5.58 %	0.00 s	0.542 %	5.95 %	1.67 s	0.143 %
200	6.05 %	0.08 s	0.518 %	6.40 %	48.07 s	0.140 %
500	6.08 %	1.94 s	0.593 %	6.46 %	4261.16 s	0.188 %
1000	6.07 %	23.71 s	0.326 %			
100c10	8.47 %	0.01 s	2.446 %	10.37 %	1.69 s	0.246 %
200c14	4.67 %	0.09 s	1.596 %	6.03 %	44.96 s	0.110 %
500c22	4.18 %	1.87 s	1.154 %	5.11 %	4244.00 s	0.150 %

Using the established edge switching heuristic (ES) starting from MST solutions already produces solutions that are less than 1 % above the optimum for the uniformly random networks (left side in Table 1). However, optimal solutions are found very rarely. Also, clustered instances pose a challenge for this simple heuristic. Using the stronger local search EFS reduces this gap (right side in Table 1). However, the high time complexity prohibits an application of this heuristic to larger instances. For the 1000 nodes problems we estimate running times of 200 hours, but already the running times for the 500 nodes problems are too high for practical use. The aim of our ILS heuristic is to produce results similar to EFS in less time.

Selected results for the experiments with ILS using different local searches and mutations are shown in Tables 2-5. In these tables, the last column shows how often the optimum or best known solution was found in all 900 runs.

Already the ILS using the weakest local search ST (Table 2) gives results that are better than the application of the simple ES heuristic to the MST. For the instances of up to 100 nodes, the results are even better than the EFS heuristic. Since ST has a time complexity of only $\mathcal{O}(n^2)$, it can easily be applied to much larger instances than the ones considered here.

However, better results can be found using the ILS with the stronger local search ES (Table 3). Optimal solutions are found with very high probability in instances of up to 100 nodes. This heuristic can be applied to the larger set of 1000 nodes instances, but the running times are still unacceptably high.

The effect of the filters can be seen when comparing Tables 3 and 4. Using filters, the ILS takes only a third of the time and still produces better results for the larger instances. For the clustered instances (100c10 and 200c14) the average solution is worse than without filters, but the best known solutions are still found more often. This is because of some runs ending in poor local optima.

Using the strongest EFS local search in the ILS produces the best results. Optimal solutions can be found with very high probability. However, the time complexity of $\mathcal{O}(n^5)$ prohibits the application of this local search to larger problem instances. In our environment, $n = 200$ was the largest setting that delivered results in acceptable time. Due to the high running times we only repeated the experiments 20 times for $n = 500$. Using filters would allow to work on slightly larger problem instances, but the time complexity remains the same. We used the ILS with EFS only for finding the best known solutions, and deactivated the filters in order to get the best results.

An analysis of the optimal solutions found by EX2 and the best known solutions for the larger problems shows that, on average, the MST solution for uniformly random graphs can only be improved by $6 - 7$ %. This observation was also made by Althaus *et al.* in [3,4]. However, for some of the instances, improvements as high as 12.25 % can be found.

Clustered instances allow larger improvements. E. g. in the clustered 100 nodes instances (100c10), the average improvement is as high as 10.58 %, the best improvement is 19.28 %. However, when the number of clusters increases, the possible improvements quickly reduce to the aforementioned $6 - 7$ %, due to

Table 2. Results for ILS using ST (filtered) and edge exchange mutation (unfiltered), 2000 iterations

Size	Improvement to MST	CPU time	Excess over best known	# best found
20	4.37 %	0.05 s	0.046 %	849
50	6.48 %	0.30 s	0.073 %	663
100	5.97 %	1.25 s	0.123 %	396
200	6.34 %	5.39 s	0.210 %	66
500	6.15 %	44.08 s	0.517 %	0
1000	5.86 %	238.68 s	0.546 %	0
100c10	10.48 %	1.07 s	0.120 %	380
200c14	5.96 %	4.72 s	0.191 %	47
500c22	4.94 %	41.08 s	0.331 %	0

Table 3. Results for ILS using ES (unfiltered) and edge exchange mutation (unfiltered)

Size	Improvement to MST	CPU time	Excess over best known	# best found
20	4.41 %	0.03 s	0.000 %	900
50	6.54 %	0.49 s	0.000 %	899
100	6.07 %	4.86 s	0.016 %	748
200	6.42 %	50.34 s	0.118 %	45
500	6.33 %	1291.41 s	0.330 %	0
1000	6.09 %	16452.88 s	0.300 %	0
100c10	10.51 %	5.06 s	0.085 %	413
200c14	5.97 %	50.43 s	0.176 %	47
500c22	5.04 %	1267.00 s	0.234 %	0

Table 4. Results for ILS using ES (filtered) and random increase mutation (filtered)

Size	Improvement to MST	CPU time	Excess over best known	# best found
20	4.41 %	0.01 s	0.000 %	900
50	6.54 %	0.15 s	0.005 %	871
100	6.07 %	1.44 s	0.016 %	797
200	6.47 %	14.39 s	0.071 %	216
500	6.45 %	342.88 s	0.200 %	1
1000	6.23 %	4421.94 s	0.153 %	30
100c10	10.39 %	2.27 s	0.222 %	433
200c14	5.95 %	21.96 s	0.196 %	162
500c22	5.06 %	384.31 s	0.206 %	0

Table 5. Results for ILS using EFS (unfiltered) and random increase mutation (filtered). Only 20 runs for instances marked with *.

Size	Improvement to MST	CPU time	Excess over best known	# best found
20	4.41 %	0.20 s	0.000 %	900
50	6.54 %	8.31 s	0.000 %	899
100	6.09 %	176.42 s	0.002 %	882
200	6.53 %	4106.83 s	0.003 %	847
500	6.62 %	130497.39 s	0.015 %	250*
100c10	10.58 %	305.10 s	0.001 %	875
200c14	6.13 %	7348.07 s	0.001 %	848
500c22	5.25 %	213787.02 s	0.009 %	144*

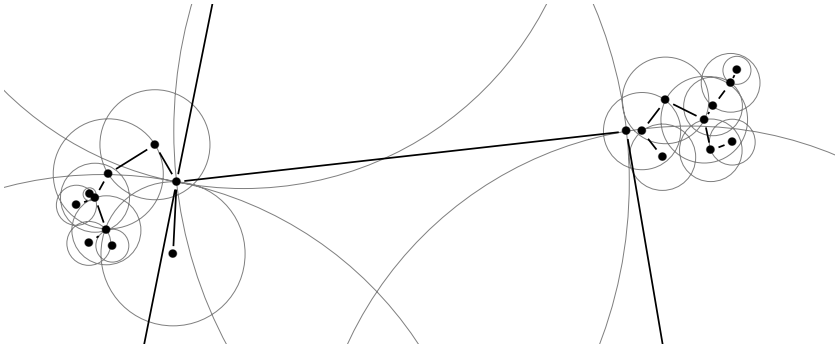


Fig. 5. Detail from the best known solution for the first instance of 100c10. From each of the shown clusters, only one node has a higher transmission range to reach another cluster. In the MST, at least two nodes from each cluster are chosen to build the inter-cluster connections.

closer or overlapping clusters. The higher improvements for clustered instances are a result of the edges connecting different clusters. In the MST, these edges connect the closest nodes from the two clusters, and thus often force multiple nodes in the same cluster to have a higher transmission range. In the optimal solutions, only a few nodes or even only one node from each cluster is required to have a large transmission range to connect its cluster to other clusters. An example for this effect is shown in Fig. 5. In a way, this is a similar effect as in the tightness example for the performance ratio of the MST in Fig. 1, where small clusters with only two nodes have to be connected.

The best heuristic in terms of running time and solution quality is the ILS using ES local search together with random increase mutation and filters. The fastest heuristic is the ILS with ST local search, and best results are found using ILS with the strongest local search EFS, ignoring the high calculation times. Unfortunately, the ILS cannot give a performance guarantee better than the MST approximation ratio. This can only be achieved by using better approximation algorithms for initialization, e. g. those from [2,3,4].

4 Conclusion

We have presented a new Iterated Local Search for the Minimum Power Symmetric Connectivity Problem. As local search we used ES, EFS, and a subtree search. The algorithm has been shown to find optimal or near-optimal solutions in short time for the considered test instances. Also, comparisons show that the proposed heuristic outperforms previous heuristics.

Future work focusses on an ant colonization heuristic incorporating local search for the same problem. We are also striving for a distributed algorithm that approximates the results of the heuristic proposed here.

References

1. Santi, P.: Topology Control in Wireless Ad Hoc and Sensor Networks. John Wiley & Sons, Chichester (2005)
2. Călinescu, G., Măndoiu, I.I., Zelikovsky, A.: Symmetric Connectivity with Minimum Power Consumption in Radio Networks. In: Baeza-Yates, R.A., Montanari, U., Santoro, N. (eds.) Proc. 2nd IFIP International Conference on Theoretical Computer Science. IFIP Conference Proceedings, vol. 223, pp. 119–130. Kluwer, Dordrecht (2002)
3. Althaus, E., Călinescu, G., Măndoiu, I.I., Prasad, S.K., Tchervenski, N., Zelikovsky, A.: Power Efficient Range Assignment in Ad-Hoc Wireless Networks. In: Proc. of the IEEE Wireless Communications and Networking Conference (WCNC 2003), pp. 1889–1894. IEEE Computer Society Press, Los Alamitos (2003)
4. Althaus, E., Călinescu, G., Măndoiu, I.I., Prasad, S.K., Tchervenski, N., Zelikovsky, A.: Power Efficient Range Assignment for Symmetric Connectivity in Static Ad Hoc Wireless Networks. Wireless Networks 12(3), 287–299 (2006)
5. Montemanni, R., Gambardella, L.M.: Exact algorithms for the minimum power symmetric connectivity problem in wireless networks. Computers & Operations Research 32(11), 2891–2904 (2005)
6. Cheng, X., Narahari, B., Simha, R., Cheng, M.X., Liu, D.: Strong Minimum Energy Topology in Wireless Sensor Networks: NP-Completeness and Heuristics. IEEE Transactions on Mobile Computing 2(3), 248–256 (2003)
7. Clementi, A.E.F., Penna, P., Silvestri, R.: Hardness Results for the Power Range Assignment Problem in Packet Radio Networks. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 197–208. Springer, Heidelberg (1999)
8. Kirousis, L.M., Kranakis, E., Krizanc, D., Pelc, A.: Power Consumption in Packet Radio Networks. Theoretical Computer Science 243(1-2), 289–305 (2000)
9. Park, J., Sahni, S.: Power Assignment For Symmetric Communication In Wireless Networks. In: Proceedings of the 11th IEEE Symposium on Computers and Communications (ISCC), Washington, pp. 591–596. IEEE Computer Society, Los Alamitos (2006)
10. Nutov, Z.: Approximating Minimum-Power k-Connectivity. In: Coudert, D., Simplot-Ryl, D., Stojmenovic, I. (eds.) ADHOC-NOW 2008. LNCS, vol. 5198, pp. 86–93. Springer, Heidelberg (2008)
11. Wolf, S., Merz, P.: Evolutionary Local Search for the Minimum Energy Broadcast Problem. In: van Hemert, J., Cotta, C. (eds.) EvoCOP 2008. LNCS, vol. 4972, pp. 61–72. Springer, Heidelberg (2008)
12. Lourenço, H.R., Martin, O., Stützle, T.: Iterated Local Search. In: Glover, F.W., Kochenberger, G.A. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 57, pp. 321–353. Springer, Heidelberg (2002)
13. Merz, P., Wolf, S.: Evolutionary Local Search for Designing Peer-to-Peer Overlay Topologies based on Minimum Routing Cost Spanning Trees. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) PPSN 2006. LNCS, vol. 4193, pp. 272–281. Springer, Heidelberg (2006)
14. ILOG S.A.: ILOG CPLEX User's Manual, Gentilly, France, and Mountain View, USA (July 2006), `http://www.cplex.com/`

Metropolis and Symmetric Functions: A Swan Song

Lars Kaden[1], Nicole Weicker[2], and Karsten Weicker[3]

[1] University of Stuttgart, Germany
[2] College of Education PH Heidelberg, Germany
[3] Applied University HTWK Leipzig, Germany
weicker@imn.htwk-leipzig.de

Abstract. The class of symmetric functions is based on the OneMax function by a subsequent assigning application of a real valued function. In this work we derive a sharp boundary between those problem instances that are solvable in polynomial time by the Metropolis algorithm and those that need at least exponential time. This result is both proven theoretically and illustrated by experimental data. The classification of functions into easy and hard problem instances allows a deep insight into the problem solving power of the Metropolis algorithm and can be used in the process of selecting an optimization algorithm for a concrete problem instance.

1 Motivation

For almost all combinatorial optimization problems there exists a wide spectrum of problem instances—from very hard to rather easy instances where the latter are already solvable by simple optimization algorithms. As a consequence the properties of the problem instances need to be considered when an optimization algorithm is chosen. Every time a simple problem solver suffices, the question arises how the easy problem subclass and especially the border between easy and hard problem instances might be characterized. With this motivation in mind, we investigate the symmetric functions and describe mathematically the borderline between polynomial and exponential runtime in expectation for the most simple stochastic optimizer, the Metropolis algorithm. In the remainder of the paper all statements concerning the runtime are in expectation.

Bit counting-based functions have a long history in evolutionary computation—from OneMax to the deceptive trap functions (Ackley, 1987) to a set of new functions by Droste et al. (2001). This problem class is referred to as *symmetric functions* $f : \{0,1\}^n \to \mathbb{R}$ which are defined by the fact that the function value depends only on the number of ones in the input string

$$f(x) = f_k \in \mathbb{R}, k = \|x\|_1, x \in \{0,1\}^n.$$

Furthermore, we assume the unique global maximum to be at bit string $(1,\ldots,1)$— the so-called *all-ones symmetric functions*.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 204–215, 2009.

Algorithm 1. (Metropolis algorithm with the parameter $\alpha \in (0,1)$ to control the selective pressure and the one-bit flipping mutation)

```
METROPOLIS( fitness function f : {0,1}^n → ℝ )
1  x ← choose random number uniformly from {0,1}^n
2  while f(x) is not optimal
3  do ⌜ i ← choose random number uniformly from {1,...,n}
4        y ← flip the ith bit in x
5        if (random number from (0,1)) ≤ min{α^(f(x)−f(y)), 1}
6     ⌟ then ⌞ x ← y
7  return x
```

The Metropolis algorithm (Metropolis et al., 1953, see Algorithm 1) with one-bit flipping mutation, and symmetric functions have been the subject of matter in a publication by Droste et al. (2001). They have shown that a function exists which can be solved by Metropolis in exponential time only but is polynomial for simulated annealing (Kirkpatrick et al., 1983). In the remainder of the paper the term *Metropolis algorithm* refers to Algorithm 1.

Based on some of the results of Droste et al. (2001) and the work of Kaden (2002) we derive a necessary and sufficient criterion for the polynomial runtime of Metropolis on symmetric functions. Furthermore we demonstrate the immediate practical relevance in a few experiments at the end of this contribution.

Since the borderline between polynomial and exponential time is determined exactly, we consider this work to rule off any runtime question concerning Metropolis and symmetric functions – as a consequence it is a swan song.

2 Criterion for Polynomial Time

As in Droste et al. (2001) the optimization process can be modeled as Markov chain where the state $0 \leq i \leq n$ corresponds to the number of ones in the current individual x. The transitions are defined by probability p_i^+ to advance from state i to state $i+1$ in one step ($0 \leq i < n$) and probability p_i^- for a step from state i to state $i-1$ ($0 < i \leq n$). The probability to stay in state i is $1-p_i^+-p_i^-$ ($0 \leq i \leq n$) where in addition $p_0^- = p_n^+ = 0$. Apparently the following equations hold:

$$p_i^+ = \frac{n-i}{n} \cdot \min\{\alpha^{f_i - f_{i+1}}, 1\} \tag{1}$$

$$p_i^- = \frac{i}{n} \cdot \min\{\alpha^{f_i - f_{i-1}}, 1\} \tag{2}$$

From Droste et al. (2001) the following notations and results are used within our analysis. The random variable T denotes the complete optimization time and T_i ($0 \leq i \leq$n) the optimization time when starting in state i. Then for the expected time

$$E(T) = \sum_{i=0}^{n} \frac{\binom{n}{i}}{2^n} \cdot E(T_i). \tag{3}$$

T_i^+ denotes the time until state $i+1$ is reached for the first time when starting from state i. Then for T_i

$$E(T_i) = \sum_{j=i}^{n-1} E(T_j^+) \tag{4}$$

And for the T_i^+ the following lemma holds. (For a proof we refer to Droste et al. (2001)).

Lemma 1. *When starting in state $0 \leq i < n$, the expected number of steps until we reach $i+1$ for the first time results as*

$$E(T_i^+) = \begin{cases} \dfrac{1}{p_0^+}, & \text{if } i = 0 \\ \dfrac{1}{p_i^+} + \dfrac{p_i^-}{p_i^+} \cdot E(T_{i-1}^+), & \text{otherwise} \end{cases} \tag{5}$$

$$E(T_i^+) = \sum_{k=0}^{i} \frac{1}{p_k^+} \cdot \prod_{l=k+1}^{i} \frac{p_l^-}{p_l^+} \tag{6}$$

To receive a general criterion concerning the polynomial runtime, a representation of the expected runtime in terms of the function values f_k is necessary. The function values and the selection parameter α are inserted into the formula in the following lemma.

Lemma 2

$$E(T) = \sum_{i=0}^{n} \frac{\binom{n}{i}}{2^n} \sum_{j=i}^{n-1} \sum_{m=0}^{j} n \cdot \frac{(n-j-1)!j!}{(n-m)!m!} \cdot \frac{\alpha^{f_j - f_m}}{\min\{\alpha^{f_j - f_{j+1}}, 1\}} \tag{7}$$

Proof First, we consider the term $(\prod_{l=a+1}^{b+1} p_l^-) \cdot (\prod_{l=a}^{b} p_l^+)^{-1}$ and apply equations (1) and (2). This leads to

$$\frac{\prod_{l=a+1}^{b+1} p_l^-}{\prod_{l=a}^{b} p_l^+} = \frac{(n-b-1)!(b+1)!}{(n-a)!a!} \cdot \alpha^{f_{b+1} - f_a}$$

which can be proved easily using induction. This result can be substituted into equation (3) using equations (6) and (4). □

To transform this result concerning the expected runtime into a general polynomial time-criterion, a notion for simplifying terms within the polynomial function class is introduced. The following lemma is crucial in the derivation of the criterion.

Definition 1 (P-equivalent). *Two functions $f : \mathbb{N} \to \mathbb{R}$ and $g : \mathbb{N} \to \mathbb{R}$ are called* P-equivalent

$$f \sim_P g :\Leftrightarrow \exists \text{ polynomials } p(n), q(n) : f(n) \leq p(n)g(n) \wedge g(n) \leq q(n)f(n).$$

Note, that the P-equivalence partitions the functions in the complement of P into an infinite number of classes. In the following we are only interested in distinguishing between functions with polynomially bound and non-polynomially bound runtime: As long as $f \in \mathrm{P}$ and $f \sim_P g$ holds, it is true that $g \in \mathrm{P}$. Also $f \notin \mathrm{P}$ and $f \sim_P g$ imply $g \notin \mathrm{P}$.

Lemma 3. *Let $f : \mathbb{N} \to \mathbb{R}$ be a sum of a polynomially bound number of terms $s(i,n)$ (with $a(n) \le i \le b(n)$ and $b(n) - a(n) + 1 \in P$). Then*

$$f(n) = \sum_{i=a(n)}^{b(n)} s(i,n) \sim_P \max_{a(n)\le i\le b(n)} \{s(i,n)\}$$

Proof. Without loss of generality we assume that $s(i,n) \ge 0$. Let $maxval(n)$ be the value denoted by the right hand side. Then, we have to show that there exist polynomials $p(n)$ and $q(n)$ such that $f(n) \le p(n)maxval(n)$ and $maxval(n) \le q(n)f(n)$ (according to definition 1). The second condition holds with $q(n) \equiv 1$ since $maxval(n)$ is an addend of $f(n)$. And the first condition holds with $p(n) = b(n) - a(n) + 1$.

Now the following result can be shown.

Theorem 1 (Polynomial time criterion)

$$E(T)(n) \in P \iff$$
$$\exists c \in \mathbb{R}: \quad \max\{n(\ln n - \ln 2) + \max_{0\le m\le j\le \lfloor \frac{n}{2}\rfloor} (g_m - f_j|\ln\alpha|),\ \max_{\substack{0\le m\le j\\ \lfloor\frac{n}{2}\rfloor < j\le n}} (g_m - g_j)\ \}$$
$$\le c \ln n,$$

where $g_x = f_x|\ln\alpha| - (n-x)\,\widetilde{\ln}\,(n-x) - x\,\widetilde{\ln}\,x$ *and*

$$\widetilde{\ln}\,x = \begin{cases} 1, & \text{iff } x \le 0 \\ \ln x, & \text{iff } x > 0. \end{cases}$$

Proof. By applying lemma 3 three times to (7), we receive

$$E(T)(n) \sim_P \max_{0\le i,m\le j\le n-1} \left\{ \frac{\binom{n}{i}}{2^n} \cdot n \cdot \frac{(n-j-1)!j!}{(n-m)!m!} \cdot \frac{\alpha^{f_j - f_m}}{\min\{\alpha^{f_j - f_{j+1}}, 1\}} \right\} \quad (8)$$

where the index $i = n$ is dropped since it results in a term $= 0$ and all terms are ≥ 0.
In order to simplify (8), we show that we can omit the dividend $\min\{\alpha^{f_j - f_{j+1}}, 1\}$ by proving

$$\max_{0\le i,m\le j\le n} \{F(i,m,j,n)\} \sim_P \max_{0\le i,m\le j\le n-1} \{F_{\min}(i,m,j,n)\}$$

with $F(i,m,j,n) = \dfrac{\binom{n}{i}}{2^n} \cdot n \cdot \dfrac{(n-j-1)!j!}{(n-m)!m!} \cdot \alpha^{f_j - f_m}$
and $F_{\min}(i,m,j,n) = F(i,m,j,n)\dfrac{1}{\min\{\alpha^{f_j - f_{j+1}}, 1\}}$, where $(-1)! := 1$.
This result follows directly, if we show

$$\max_{0\le i,m\le j\le n} \{F(i,m,j,n)\} \le_P \max_{0\le i,m\le j\le n-1} \{F_{\min}(i,m,j,n)\} \text{ and} \quad (9)$$
$$\max_{0\le i,m\le j\le n} \{F(i,m,j,n)\} \ge_P \max_{0\le i,m\le j\le n-1} \{F_{\min}(i,m,j,n)\} \quad (10)$$

with $f(n) \leq_P g(n) \iff \exists \text{polynomial } p(n) : f(n) \leq p(n)g(n)$.
For $j < n$, (9) follows immediately from

$$F(i,m,j,n) = \underbrace{\min\{\alpha^{f_j - f_{j+1}}, 1\}}_{\leq 1} \cdot F_{\min}(i,m,j,n). \tag{11}$$

And for $j = n$, the following equation holds

$$F(i,m,n,n) = n \cdot \underbrace{\frac{\min\{\alpha^{f_{n-1}-f_n}, 1\}}{\alpha^{f_{n-1}-f_n}}}_{\leq 1} \cdot F_{\min}(i,m,n-1,n).$$

For the proof of (10) the following cases are distinguished:

- case $f_j \leq f_{j+1}$: $F_{\min}(i,m,j,n) = F(i,m,j,n)$
- case $f_j > f_{j+1}$ and $j = n-1$: $F_{\min}(i,m,n-1,n) = \frac{1}{n} \cdot F(i,m,n,n)$
- case $f_j > f_{j+1}$ and $j < n-1$:

$$F_{\min}(i,m,j,n) = \frac{n-j-1}{j+1} \cdot F(i,m,j+1,n).$$

Now we have shown the simplified form

$$E(T)(n) \sim_P \max_{0 \leq i,m \leq j \leq n} \left\{ \frac{\binom{n}{i}}{2^n} n \cdot \frac{(n-j-1)!j!}{(n-m)!m!} \cdot \alpha^{f_j - f_m} \right\}. \tag{12}$$

The parameter i occurs in the binomial coefficient $\binom{n}{i}$ only. Using the condition $i \leq j$ the following result may be used

$$\max_{i \leq j} \binom{n}{i} = \binom{n}{\min\{j, \lfloor \frac{n}{2} \rfloor\}}.$$

If $j \leq \lfloor \frac{n}{2} \rfloor$ the term may be simplified

$$\frac{\binom{n}{i}}{2^n} n \cdot \frac{(n-j-1)!j!}{(n-m)!m!} \cdot \alpha^{f_j - f_m} = \frac{1}{2^n} \cdot \frac{n}{n-j} \cdot \frac{n!}{(n-m)!m!} \cdot \alpha^{f_j - f_m}.$$

If $j > \lfloor \frac{n}{2} \rfloor$ the observation $\frac{2^n}{n+1} \leq \binom{n}{\lfloor \frac{n}{2} \rfloor} \leq 2^n$ is used and the following criterion results:

$$E(T)(n) \sim_P \max \left\{ \max_{0 \leq m \leq j \leq \lfloor \frac{n}{2} \rfloor} \left\{ \frac{n}{n-j} \cdot \frac{n!}{2^n (n-m)!m!} \cdot \alpha^{f_j - f_m} \right\}, \right.$$
$$\left. \max_{0 \leq m \leq j \wedge \lfloor \frac{n}{2} \rfloor < j \leq n} \left\{ n \cdot \frac{(n-j-1)!j!}{(n-m)!m!} \cdot \alpha^{f_j - f_m} \right\} \right\}.$$

In order to get rid of the unhandy factorial terms we use Stirling's formula

$$k! \approx \sqrt{2\pi k} \left(\frac{k}{e} \right)^k$$

for $k \geq 1$. Therefore we get

$$E(T)(n) \sim_P \max\left\{ \max_{0\leq m\leq j\leq \lfloor \frac{n}{2}\rfloor}\left\{ \frac{n}{n-j}\cdot\sqrt{\frac{n}{2\pi m(n-m)}}\frac{n^n}{2^n(n-m)^{n-m}m^m}\cdot\alpha^{f_j-f_m}\right\}, \max_{0\leq m\leq j\wedge\lfloor\frac{n}{2}\rfloor<j\leq n}\left\{\frac{n}{n-j-1}\cdot\sqrt{\frac{(n-j-1)j}{(n-m)m}}\cdot\left(\frac{n-j-1}{n-j}\right)^{n-j}\cdot \frac{(n-j)^{n-j}j^j}{(n-m)^{n-m}m^m}\cdot\alpha^{f_j-f_m}\right\}\right\}.$$

where $m \neq 0$, $m \neq n$, $j \neq n-1$, and $j \neq n$. Those excluded cases are treated later.

It is easy to see that $\left(\frac{n-j-1}{n-j}\right)^{n-j} \in \Theta(1)$ and both terms

$$\frac{n}{n-j}\cdot\sqrt{\frac{n}{2\pi m(n-m)}} \sim_P 1 \quad \text{and} \quad \frac{n}{n-j-1}\cdot\sqrt{\frac{(n-j-1)j}{(n-m)m}} \sim_P 1$$

Those terms can be omitted in the criterion leading to

$$E(T)(n) \sim_P \max\left\{ \max_{0\leq m\leq j\leq \lfloor \frac{n}{2}\rfloor}\left\{ \frac{n^n}{2^n(n-m)^{n-m}m^m}\cdot\alpha^{f_j-f_m}\right\}, \max_{0\leq m\leq j\wedge\lfloor\frac{n}{2}\rfloor<j\leq n}\left\{\frac{(n-j)^{n-j}j^j}{(n-m)^{n-m}m^m}\cdot\alpha^{f_j-f_m}\right\}\right\}.$$

The cases $m = 0$, $m = n$, $j = n-1$, and $j = n$ require special treatment which is introduced by a modified logarithm function.

$$E(T)(n) \sim_P \max\{ \exp\big(n(\ln n - \ln 2) + \max_{0\leq m\leq j\leq\lfloor\frac{n}{2}\rfloor} g_m - f_j|\ln\alpha|\big), \exp\big(\max_{0\leq m\leq j\wedge\lfloor\frac{n}{2}\rfloor<j\leq n} g_m - g_j\big)\ \}.$$

Eventually, the theorem follows directly from the equation above by applying the logarithm to the equation. □

3 Interpretation

In this section, we want to describe the fitness functions that are solved within polynomial time as close as possible. Those functions must fulfill the condition in theorem 1.

Within the criterion a maximum of two terms is considered. For the maximum to be less than $c \ln n$ with a given $c \in \mathbb{R}$, this must be fulfilled for both terms which are examined separately.

In the remainder, we assume that the number n of bits is even.

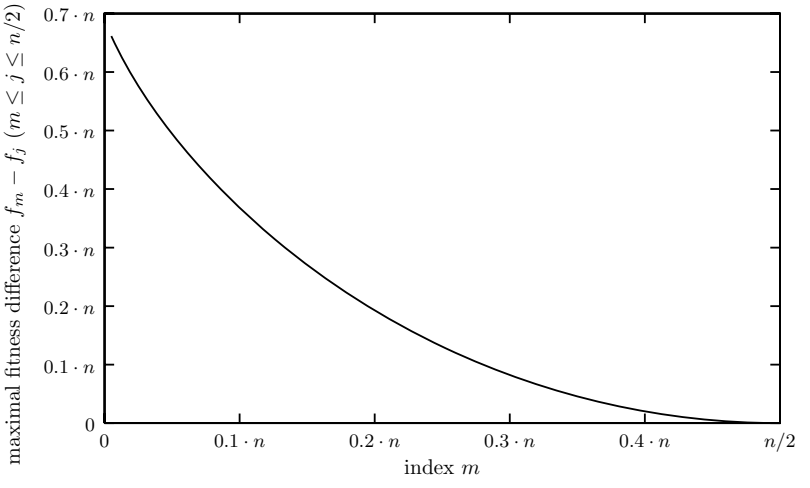


Fig. 1. First condition in equation (13). The given fitness difference needs to be scaled by the factor $|\ln \alpha|$.

The first resulting condition concerns the fitness values in the worse half of the search space:

$$n(\ln n - \ln 2) + \max_{0 \leq m \leq j \leq \frac{n}{2}} \left\{ (f_m - f_j)|\ln \alpha| - (n-m)\,\widetilde{\ln}\,(n-m) - m\,\widetilde{\ln}\,m \right\} \leq c \ln n. \quad (13)$$

Note that $-(n-m)\,\widetilde{\ln}\,(n-m) - m\,\widetilde{\ln}\,m$ has its maximal value for $m = \frac{n}{2}$ where it equals $-n(\ln n - \ln 2)$ and eliminates the first term in (13). As a consequence the difference between function values f_m and f_j is bound by $c \ln n$ plus a constant term if the maximum is not at $m = \frac{n}{2}$. The latter is shown in figure 1.

The second resulting condition relates the fitness value of each point in the better half of the search space with the fitness value of all points containing a smaller number of ones:

$$\max_{\substack{0 \leq m \leq j \\ \lfloor \frac{n}{2} \rfloor < j \leq n}} \left\{ (f_m - f_j)|\ln \alpha| + (n-j)\,\widetilde{\ln}\,(n-j) + j\,\widetilde{\ln}\,j - (n-m)\,\widetilde{\ln}\,(n-m) - m\,\widetilde{\ln}\,m \right\} \leq c \ln n \quad (14)$$

The maximal possible fitness differences are visualized in figure 2.

Note that the graphs in figures 1 and 2 omit the factor $|\ln \alpha|$. As a consequence the role of the selective pressure α is a mere rescaling of the acceptable fitness differences.

4 Practical Relevance

The proven theorem may be used to classify any symmetric function according to the runtime of the Metropolis algorithm in expectation. However it's

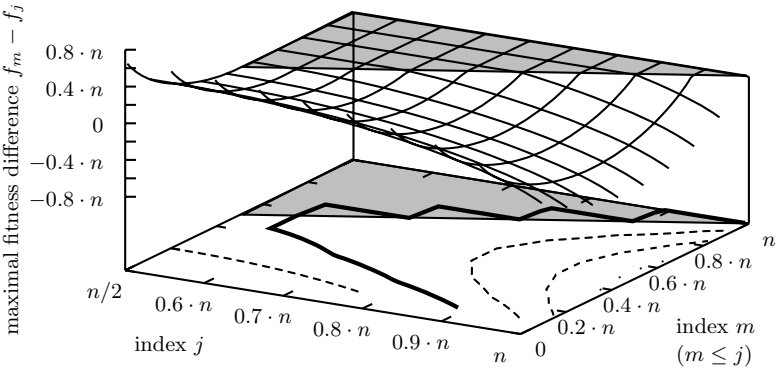


Fig. 2. Second condition in equation (14). The gray area marks the illegal combinations of (j, m). The thick contour line denotes the value 0. The given fitness difference needs to be scaled by the factor $|\ln \alpha|$.

application to well known functions like OneMax or the trap function is of minor interest: The polynomial runtime concerning OneMax is seen easily since there is no worsening of the function value with increasing number of ones—as a consequence the fitness difference is always below the limits in the theorem. Also the trap function is quickly classified as requiring exponential runtime in expectation since the linear declining in the worse half of the number of ones does not meet the condition in the theorem.

As a consequence it is more interesting to construct a function that lies exactly on the borderline of polynomial and exponential runtime in expectation. The fitness function is solved in polynomial runtime—but any linear modification of a single function value (e.g. $f_j + \varepsilon \cdot n$ with $\varepsilon > 0$) leads to exponential time. In fact $f_j + \varepsilon \cdot (\ln n)^2$ would suffice for exponential runtime.

The resulting function in the left part of figure 3 is generated by successive assignment of function values beginning for the optimum (n ones) down to the case without ones. For each number of ones the maximal possible value is chosen that is compatible with the values already assigned (see Algorithm 2). The value $c = 1$ and $\alpha = 0.95$ was chosen.

Theorem 1 shows that this function is solvable in polynomial runtime. It is modified in two ways. First, at position $3 \cdot n/4$ the value $10 \cdot \log n$ is added to the fitness value. Second, at the same position the value n is added to the function value (right part of figure 3). According to the theorem above the first modification should still be solvable in polynomial time, where the second modification should lead to exponential runtime. In the remainder of this section we examine whether this effect can be observed experimentally.

Now, 100 experiments were executed for each fitness function and each value $n = 10, 20, \ldots 160$ using the Metropolis algorithm with $\alpha = 0.95$. The starting point was chosen uniformly from all possible genotypes $\{0, 1\}^n$—as it is requested by the prerequisites of the theorem. The exact averaged results are shown in tabular 1 and in the left part of figure 4.

Algorithm 2. (Compute a borderline function $f : \{0,1\}^n \to \mathbb{R}$)

```
BORDERLINE-FUNCTION( equal problem size n ∈ ℕ, maximal function value maxf )
 1  f(n) ← maxf
 2  for m ← n − 1, ..., 1
 3  do ⌜ value ← maxf
 4        if m ≤ n/2
 5        then ⌜ for j ← m + 1, ..., n/2
 6                  do ⌜ testval ← f(j) + difference according to cond_1 (13) for j, m
 7                        if testval < value
 8                     ⌞ then ⌊ value ← testval
 9               ⌞ start ← n/2 + 1
10        else  ⌊ start ← m + 1
11        for j ← start, ..., n
12        do ⌜ testval ← f(j) + difference according to cond_2 (14) for j, m
13              if testval < value
14           ⌞ then ⌊ value ← testval
15     ⌞ f(m) ← value
16  f(0) ← f(1)
17  return f
```

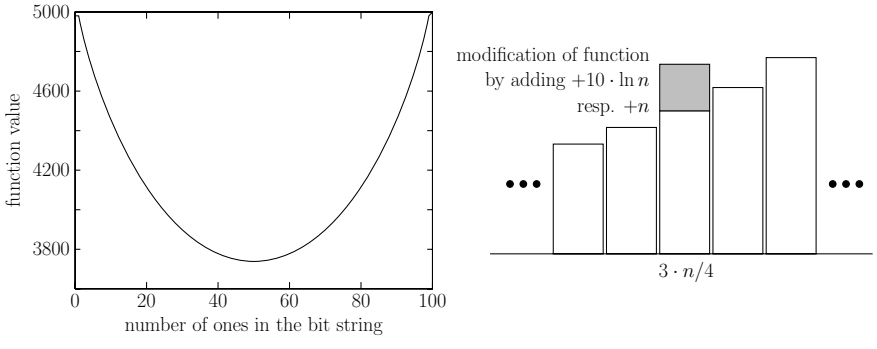


Fig. 3. The polynomially solvable function for $n = 100$ is shown in the left graph. The right graph illustrates how the function is modified to get exponential runtime (in the case of $+n$).

With a regression analysis we have estimated the empirical asymptotic runtime of the experiments. The unmodified constructed function results in the runtime $\mathcal{O}(x^{1.988})$. For the function modified using a $+n$ term at one point the runtime $\mathcal{O}(1.05^x)$ can be observed. The function with $+10 \cdot \ln n$ shows a runtime in $\mathcal{O}(x^{1.999})$. How well these estimates fit the experimental data is shown in figures 4 and 5 by dividing the observed runtime by the estimated asymptotic runtime. In case of the functions requiring polynomial time in expectation an almost constant factor verifies the theoretical runtime in the experiments. The function with exponential runtime in expectation can also be verified by a bound constant factor for $n = 50, \ldots, 150$.

Table 1. Number of iterations to find the optimum (average over 100 runs)

	constructed function with		
	no changes	$+10 \ln n$ at $f_{3n/4}$	$+n$ at $f_{3n/4}$
10	154.80	132.53	149.65
20	799.85	819.45	710.96
30	1678.52	2080.27	1949.65
40	3601.35	3761.74	4394.10
50	5308.06	7883.95	6533.16
60	8989.84	10593.79	10922.89
70	13339.12	13707.35	16755.72
80	17223.95	19183.02	23253.94
90	18746.36	21185.28	34986.41
100	24271.40	25827.67	56907.27
110	26423.45	36003.97	91038.00
120	34803.01	42738.73	136275.46
130	38848.24	45230.06	254373.45
140	48171.56	53457.71	335701.91
150	57042.62	58052.34	717697.53
160	68708.67	78813.22	1469201.78

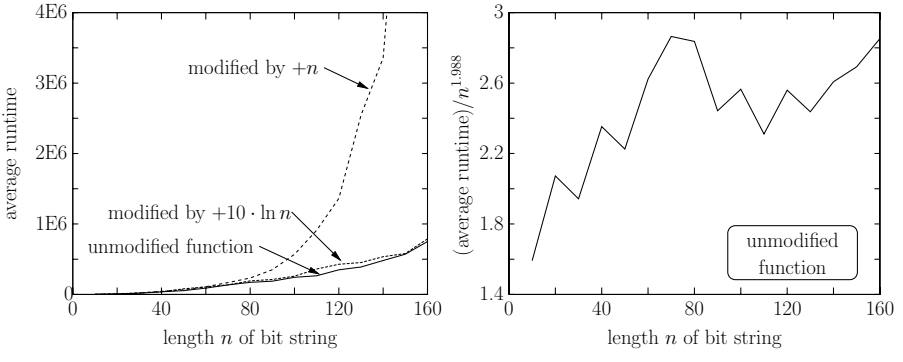


Fig. 4. The left graph shows a comparison of the experimentally deterimend runtime for the three functions. The right graph shows the average runtime scaled by the estimated asymptotic runtime for the unmodified constructed function.

These experimental data certainly cannot "prove" an asymptotic runtime behavior empirically because of the limited problem size—and this is not necessary since the theorem makes a clear statement in this regard. But the interesting fact is that this theoretical runtime can be observed immediately in our experements—even for functions that differ in one function value only. This shows how sharp the line between polynomial and exponential runtime is and underlines the practical relevance of the theorem.

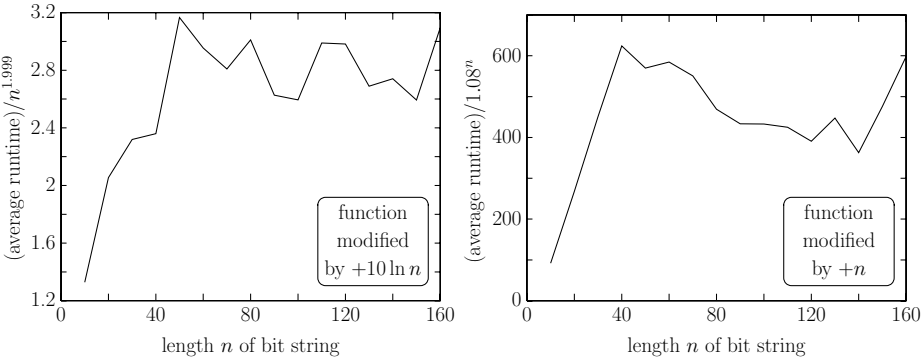


Fig. 5. The average runtime scaled by the estimated asymptotic runtime is shown for the function modified by $+10 \cdot \ln n$ on the left and for the function modified by the term $+n$ on the right.

5 Conclusion and Outlook

In this work, for the Metropolis algorithm and the one-bit flipping mutation, a sharp boundary between polynomially and exponentially solvable instances of symmetric functions is proven. The immediate impact of the result was shown experimentally—already for a small number of bits, a slight change in the symmetric function turns polynomial into exponential runtime. The asymptotic classification of the experimental runtime is demonstrated by a regression analysis. Modifications in terms of $\log n$ stay within the polynomial runtime class where any bigger change, like n, drops out of the class. By varying strong modifications in terms of $\log n$ for different numbers of ones, functions may be designed that do not seem to fall into the polynomial case at first look. Even very rugged functions are possible as long as the polynomial criterion derived above is true.

For the case of the symmetric functions a problem classification into easy and hard problems is possible and provides the most profound insight into the optimization power of the Metropolis algorithm. Whether these results can be transfered easily to other problems is an open question. However, if we stick to symmetric functions, similar results are possible even when moving from the simple Metropolis algorithm to a more competitive optimization algorithm like simulated annealing. We hope that a comparison between the result for the Metropolis algorithm and boundary for simulated annealing might produce a useful understanding concerning the scope of application of the latter algorithm. Such a result should be of considerable impact since simulated annealing is for many simple problem instances still a good choice in real-world applications.

Bibliography

Ackley, D.H.: A Connectionist Machine for Genetic Hillclimbing. Kluwer, Boston (1987)

Droste, S., Jansen, T., Wegener, I.: Dynamic parameter control in simple evolutionary algorithms. In: Martin, W.N., Spears, W.M. (eds.) Foundations of Genetic Algorithms, vol. 6, pp. 275–294. Morgan Kaufmann, San Francisco (2001)

Kaden, L.: Laufzeitkriterien für genetische Algorithmen mit und ohne dynamische Anpassung der Selektionsstrategie. Studienarbeit, University of Stuttgart, Germany (2002)

Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. Journal of Chemical Physics 21(6), 1087–1092 (1953)

Robustness Analysis in Evolutionary Multi-Objective Optimization Applied to VAR Planning in Electrical Distribution Networks

Carlos Barrico[1,2], Carlos Henggeler Antunes[1,3], and Dulce Fernão Pires[1,4]

[1] INESC Coimbra, Rua Antero de Quental 199, 3000-033 Coimbra, Portugal
[2] Department of Informatics, University of Beira Interior, 6200 Covilhã, Portugal
[3] Department of Electrical Engineering and Computers, University of Coimbra, Portugal
[4] School of Technology, Polytechnic Institute of Setúbal, 2910-761 Setúbal, Portugal
cbarrico@inescc.pt, ch@deec.uc.pt, dcosta@est.ips.pt

Abstract. In this paper an approach to robustness analysis in evolutionary multi-objective optimization is applied to the problem of locating and sizing capacitors for reactive power compensation (VAR planning) in electric radial distribution networks. The main goal of this evolutionary algorithm is to find a non-dominated front containing the most robust non-dominated solutions also ensuring diversity along the front. A concept of degree of robustness is incorporated into the evolutionary algorithm, which intervenes in the computation of the fitness value assigned to solutions. Two objective functions of technical and economical nature are explicitly considered in the mathematical model: minimization of system losses and minimization of capacitor installation costs. Constraints refer to quality of service, power flow, and technical requirements. It is assumed that some input data are subject to perturbations, both concerning the objective functions and the constraints coefficients.

Keywords: robustness analysis, multi-objective evolutionary algorithm, reactive power compensation problem.

1 Introduction

The purpose of the study of a multi-objective optimization problem may be either to support the decision maker (DM) in selecting a final compromise solution (or a reduced set of solution for further screening) or to characterize the whole set of non-dominated (Pareto optimal) solutions. However, some of these solutions, which could be of interest for a DM as adequate compromise solutions, in the sense they present a satisfactory balance between the multiple, conflicting and incommensurate objective functions, may be very susceptible to perturbations. Algorithms must strive for robust solutions, that is solutions that are relatively "immune" to perturbations, in the sense that their feasibility and objective function performances should not degrade significantly for small changes in the values estimated for the model data (coefficients of objective functions, coefficients of constraints, bounds of decision variables). Some studies have been devoted to compute robust solutions both in single-objective, [1]-[4], as well as in multi-objective evolutionary optimization, [5]-[11]. For more details about this topic see [12].

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 216–227, 2009.

The reactive power compensation (VAR planning) problem involves determining the number, location, and sizes for shunt capacitors (sources of reactive power) to be installed, in this case in a distribution feeder, to achieve a balance between costs (associated with installing new capacitors) and technical / quality of service evaluation aspects.

In this paper, an approach to robustness analysis in evolutionary multi-objective optimization is applied to the reactive power compensation problem. This approach is based on the concept of degree of robustness assigned to solutions and it is aimed at tackling perturbations in the objective function and constraint coefficients (i.e., drifting from their nominal values). This approach relies on the concept of scenario, which is a possible instantiation of the objective function and constraint coefficients around their nominal values. The set of initial (nominal) objective function and constraint coefficients is denoted by reference scenario. In this way, the approach used herein is based on the solution behavior in the neighborhood of the reference scenario. This concept of degree of robustness conveys more information to a DM than a simple robust/not robust classification and enables him/her to wield control on the level of robustness of solutions obtained through the setting of some parameters (see also [9]-[11]). The size of the reference scenario neighborhood can be specified, both regarding the objective function and constraint coefficients as well as the objective function space. The degree of robustness intervenes in the evaluation of a solution (individual) of a population and enables to classify the solutions accordingly.

The evolutionary approach used in this study has been developed encompassing this concept of degree of robustness, which is embedded in the evolutionary process, particularly in the fitness assessment of each individual. The underlying rationale is to bias the evolutionary process towards more robust solutions, that is solutions for which feasibility and the objective function performances are more insensitive to perturbations in the constraint and the objective function coefficients.

The interest and motivation of the study have been provided in this section. In section 2 the concept of degree of robustness is presented. The reactive power compensation problem in electrical distribution networks is presented in section 3. The main features of the evolutionary algorithm are described in section 4. Illustrative results are presented in section 5. In section 6 some conclusions are drawn.

2 The Degree of Robustness

The definition of robust solution is not harmonized in the literature. A robust solution must guarantee a good performance (regarding both feasibility and objective function values) even if (slightly) different model coefficients apply, vis-à-vis a nominal situation, due to the uncertainty associated with data gathering, estimates etc.

2.1 Perturbations of the Objective Function Coefficients

It is assumed that perturbations may occur in any coefficient of objective function f_r (c_{r1}, c_{r2},…, c_{rm}), for r = 1, …, R. The assessment of the degree of robustness of a solution x entails analyzing the neighborhood of the reference scenario s, where x is a solution to the problem and $f^s(x)$ is the point in the objective space for the reference scenario s. The underlying idea is to determine a set of neighborhoods $k\delta$ around the

reference scenario s, such that the images of x for these neighborhood scenarios are better than $f^s(x)$, for all objective functions, or still belong to a pre-specified neighborhood η around $f^s(x)$ in the objective space. The process begins by analyzing scenarios (coefficient instantiation) randomly generated inside a hyperbox of radius δ around s. This neighborhood (hyperbox) is then progressively enlarged, in multiples of δ (δ ,2δ ,…), until the percentage of scenarios for which the images of x in the objective space that are better than $f^s(x)$ or belong to the neighborhood of $f^s(x)$ is not greater than a pre-defined threshold. This enables to assign a degree of robustness to solutions according to the number of hyperbox enlargements for which that condition is fulfilled (see Fig. 1).

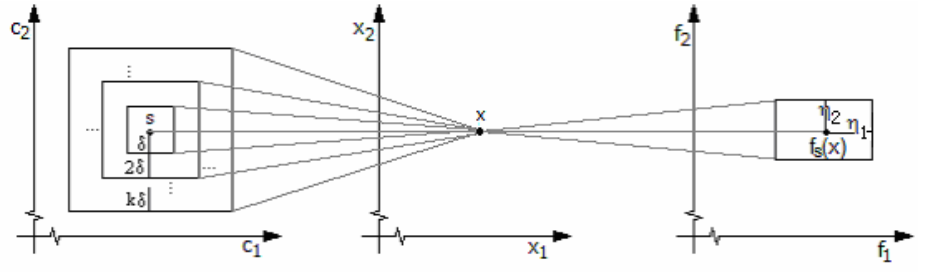


Fig. 1. Definition of neighborhoods in the scenario space and objective function space associated with a solution x (for 2-dimension spaces)

The degree of robustness depends on the size of a δ-neighborhood of scenario s and the percentage of the h neighboring points whose objective function values for x are better than $f^s(x)$ or belong to the η-neighborhood of $f^s(x)$. Those h neighboring points are randomly generated around scenario s (see also [5], [9]-[11]). The degree of robustness of solution x is a value k, such that (see Fig. 1):

a) the percentage of scenarios s' in the kδ-neighborhood of s, for which the objective function values $f^{s'}(x)$ that are better than $f^s(x)$ or that belong to the η-neighborhood of $f^s(x)$, is greater than or equal to a pre-specified threshold p1;
b) the percentage of scenarios s' in the (k+1)δ-neighborhood of s, for which the objective function values $f^{s'}(x)$ that are better than $f^s(x)$ or that belong to the η-neighborhood of $f^s(x)$, is lower than p1.

The degree of robustness k of a solution x determined in the reference scenario s is gradually computed as k increases (neighborhoods δ, 2δ, …, kδ), as well as the number of neighboring points of s (h, h+qh, …, h+(k-1)qh), such that h+(t-1)qh neighboring points ($t \in \{1, \ldots, k\}$) are analyzed in the tδ-neighborhood of s.

2.2 Perturbations of the Constraint Coefficients

For this case a scenario is a set of possible values for the constraint coefficients (it can include the decision variable bounds) subject to perturbations. The computation of the

degree of robustness of a solution x involves to analyze the feasibility of x for the neighborhood of the reference scenario s. f(x) is the point in the objective space for the reference scenario s, as well as for all possible scenarios. The feasibility of x varies for distinct scenarios, since each scenario (constraint coefficient instantiation) may correspond to distinct feasible regions. The aim is to compute a set of k neighborhoods around the reference scenario s, the radius of which is a multiple of a value (kδ), such that x still is a feasible solution for those coefficient instantiations around the nominal (reference scenario).

Randomly generated scenarios inside a hyperbox of radius δ around reference scenario s are analyzed. This neighborhood is then progressively enlarged (δ, 2δ, …) until the percentage of scenarios for which the solution x is a feasible solution is not greater than a pre-defined threshold. Again, this enables to assign a degree of robustness to solutions according to the number of hyperbox enlargements for which that condition is satisfied, which depends on the size of a δ-neighborhood of reference scenario s and the percentage of the h neighboring points for which the solution x is a feasible solution.

The degree of robustness of solution x is a value k, such that:

a) the percentage of scenarios s' in the kδ-neighborhood of s, for which the solution x is a feasible solution, is greater than or equal to a pre-specified threshold p2;
b) the percentage of scenarios s' in the (k+1)δ-neighborhood of s, for which the solution x is a feasible solution, is lower than p2.

The degree of robustness k of a solution x is determined in the same way as explained above for the case of perturbations of the objective function coefficients.

2.3 The Robustness Parameters

The radius of each neighborhood (around a solution or the reference scenario) is a multiple of the parameter δ (δ ,2δ , ...). This parameter reflects the DM's preferences about the base dimension of the neighboring solutions in the solutions space or the neighboring scenarios in the scenario space, such that he/she is indifferent for solutions or scenarios located therein.

The parameter h sets the number of neighboring points of a solution or the reference scenario that are generated and analyzed. The higher the value of h, more extensive is the analysis (however, higher will be the execution time of the algorithm).

The vector parameter η reflects the threshold of indifference for each objective function. This parameter is used as the upper bound for the distance between the images of a solution x in a given scenario and in the reference scenario. Increasing the values of η (meaning that the DM is more tolerant to the differences in the objective function values) tends to increase the number of solutions with a higher degree of robustness.

The thresholds p1 and p2 control the exigency of the degree of robustness.

The parameter q is a value between 0 and 1, which is associated with the increase of the number of neighboring points that are analyzed in successive enlargements of the neighborhood of a solution or the reference scenario.

The values of the parameters p1, p2, η and q may be different depending on the solution type, that is whether solutions are non-dominated, or they are dominated or non-feasible. Due to the significant run time of this approach, it is advisable that the value of p1 and p2 should be higher for dominated and infeasible solutions, and the values of η and q should be higher for non-dominated solutions. The underlying rationale is that the algorithm seeks for non-dominated solutions, which are more relevant than dominated and infeasible solutions, and therefore a more exhaustive analysis is necessary for those solutions.

The computation of the (absolute or relative) distance between the images of solution x according to the scenarios s and s' in the objective space, $f^{s'}(x)$ and $f^{s}(x)$, $f^{s'}(x)$ belonging to the η-neighborhood of $f^{s}(x)$, can be done using any metric (Manhattan, Euclidean, Chebycheff, etc.).

The amount of parameters required may be reduced by establishing some dependences between them.

3 The Reactive Power Compensation Problem in Electrical Distribution Networks

The compensation of reactive power (VAR planning) is an important issue in electric power systems, being directly related with efficient delivery of active power to loads (converted into "useful" energy, such as light or heat). Reactive power compensation contributes to releasing electric system capacity, improving voltage bus profile and reducing losses. The device generally used for reactive power compensation is the shunt capacitor (source of reactive power). Operational, economical and quality of service aspects for selecting a suitable deployment of capacitors need to be weighed by DMs (planning engineers) to select good solutions having in mind their practical implementation. The aim is to find the network nodes to install capacitors and the dimension of each capacitor to be installed to minimize costs and system losses while keeping an acceptable bus voltage profile.

Multi-objective mathematical models are then required to capture these multiple, conflicting, and incommensurate evaluation aspects of the merit of solutions. A multi-objective model has been developed considering two (conflicting) objective functions: minimizing (resistive) losses and minimizing the installation costs of new sources of reactive power. Constraints are related with requirements of acceptable node voltage profile (quality of service imposed by legislation), power flow (physical laws in electrical networks), and impossibility of capacitor locations at certain nodes (technical restrictions). For more details about this mathematical model see [13]-[15], which includes non-linearities in the losses objective function and in the power flow constraints as well as continuous, integer and binary decision variables.

Evolutionary algorithms (EAs) are quite adequate for dealing with multi-objective programming (MOP) models (particularly, of combinatorial nature) due to their capability of working with a population of solutions [16]-[19]. Since in those models the aim is generally the characterization of a Pareto optimal front rather than computing a single optimal solution, EAs endowed with techniques to maintain diversity of solutions present advantages with respect to the use of approaches based on scalarizing functions as in traditional mathematical programming approaches. These are surrogate scalar func-

tions that (temporarily) aggregate the multiple objective functions so that an optimal solution to the scalarizing function is a non-dominated solution to the MOP problem. Also, EAs are well-suited for solution representation in networks [20]-[25].

In this paper, it is assumed that the input data associated with costs and capacities of capacitors are unchanged and the remaining input data are subject to small perturbations, due to the uncertainties inherent to measurements and estimates. The changes in these input data imply that both the objective function values and the feasibility of the solutions are subject to variations.

4 The Evolutionary Algorithm

An EA has been developed aimed at characterizing the Pareto optimal (non-dominated) front and assessing the robustness of solutions therein taking into account changes in the input data. This EA includes an elitist strategy with a secondary population (with feasible non-dominated solutions only). This is aimed at increasing the algorithm performance, both accelerating the convergence towards the non-dominated frontier and ensuring the solutions attained are well-spread over the frontier (an important issue in real-world problems [18]).

The EA encompasses the definition of a degree of robustness associated with each solution, which is embedded into the fitness assessment together with the non-dominance test. That is, in each non-dominance level, the evolutionary process favors more robust solutions that are then are more likely to contribute for the next generation.

The main steps of this algorithm are the following:

- The fitness of the individuals composing the main population is computed;
- From the main population (consisting of *POP* individuals) *POP-E* individuals are selected by using a tournament technique (*E* is the size of the elite set);
- A new population is formed by the *POP-E* offspring generated by crossover and mutation, and the *E* most robust individuals (elite) in the secondary population;
- The fitness of individuals is evaluated by a dominance test and by taking into account the degree of robustness, which defines an approximation to the Pareto front;
- The non-dominated solutions are computed and they are processed to update the secondary population using a sharing technique, if necessary.

The population consists of individuals represented by an array of NN integer values (NN being the number of network nodes where it is possible to install a new capacitor or change the capacity of a capacitor already installed). The index of the array corresponds to a network node and the value therein denotes the type of capacitor to install in that node (the capacitor type is indexed by an index ranging from 0 to J – 0 means no capacitor; that is, J different capacitor sizes can be installed).

The fitness value of a solution depends on its degree of robustness and the dominance test. For each solution, the fitness computation uses a "non-dominated sorting" technique as in "NSGA-II", [19], [26], and involves determining several solution fronts. For more details about this technique, see [9]-[11].

The sharing mechanism for updating the secondary population uses a niche scheme whose radius is a dynamic value and the degree of robustness of the solutions. This mechanism is applied after computing all non-dominated solutions candidate for the secondary population. These are all the non-dominated solutions in the set formed by the secondary population and the main population. This mechanism is only applied when the number of solutions candidate for the secondary population (*NCPS*) is greater than the size of this population (*NPS*). For more details about this technique, see [9]-[11].

The initial population consists of randomly generated feasible non-dominated solutions only. In problems with few feasible non-dominated solutions, the initial population may also contain some feasible dominated solutions (the ones needed to complete the population).

Uniform crossover has been used, with probability *pc*. In each generation a new mask is created.

The capacitor type is indexed by an index ranging from 0 to J. The mutation consists in modifying (with a probability *pm*) the current index value to one of other possible values.

5 Illustrative Results of a Real-World Case Study

This approach has been applied to an actual Portuguese radial distribution system, the main characteristics of which in terms of line length, resistance and inductance are summarized in Table 1. Other input data to the model are the sizes and installation costs of each capacitor type (Table 2) as well as the active and reactive power (load) at each node of the distribution network.

Table 1. Network characteristics

	Minimum	Maximum	Average	Standard Deviation
Line length (m)	256	4027	856	559.6
Resistance (Ω /Km)	0.213	1.5	0.745	0.393
Inductance (Ω/Km)	0.356	0.395	0.379	0.011

Table 2. Capacitor dimension and acquisition cost

	Maximum Capacity (kVAr)	Composition (kVAr)	Cost (€)
C_1	30	7.5+7.5 + 7.5 + 7.5	1 800
C_2	60	15 + 15 + 30	2 000
C_3	75	15 + 30 + 30	2 198

The performance of the distribution system has been assessed under several conditions, before and after the compensation solutions (considering the node's voltage magnitude within the bounds nominal voltage ± 10%). Whereas for medium (70% of the peak load) and light (30% of the peak load) load conditions all the nodes satisfy the lower and upper bounds on the node's voltage magnitude, for peak load conditions about 40% of the nodes do not respect those quality of service constraints. This means

that the system operation under peak conditions is impaired by this poor voltage profile. Fig. 2 shows the Pareto-front obtained before incorporating the concept of degree of robustness into the EA. The following parameter values have been used: POP = 40; NPS = 30; E = 0.1 NPS; pc = 0.9; pm = 0.1; and number of iterations = 2000, which are then also used when the degree of robustness is included.

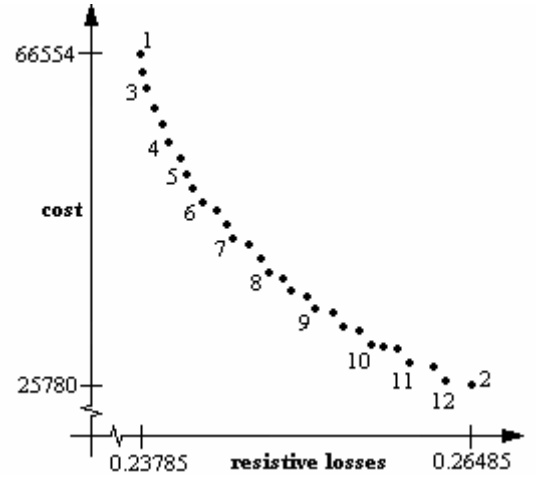


Fig. 2. Pareto-front obtained without considering the degree of robustness

Table 3 displays a set of selected non-dominated solutions representing different compensation schemes and different trade-offs between the objective functions along the non-dominated front. Solutions 1 and 2 are the non-dominated solutions that individually optimize resistive losses and cost objective functions, respectively. The information in Table 3 enables to conclude that there are no feasible solutions without making an investment of, at least, 25780 € (solution 2), otherwise the bounds on the node voltage magnitudes would not be satisfied as a consequence of the poor voltage profile of this network.

Table 3. Sample of non-dominated solutions (Fig. 2)

Sol	Losses	Cost	Cap. type 1	Cap. type 2	Cap. type 3	Total
1	**0.23785**	66554	0	8	23	31
2	0.26485	**25780**	1	1	10	12
3	0.23838	62354	1	5	23	29
4	0.24017	55762	0	7	19	26
5	0.24156	51762	0	5	19	24
6	0.24286	48158	0	1	21	22
7	0.24545	43762	0	1	19	20
8	0.24834	39564	0	0	18	18
9	0.25213	35168	0	0	16	16
10	0.25671	30772	0	0	14	14
11	0.25984	28376	0	1	12	13
12	0.26275	26178	0	1	11	12

Table 4 displays the structure of a sub-set of these non-dominated solutions, in terms of capacitor size and location. Each digit in the sequence represents the capacitor type (0 for no capacitor installed), and the position in the sequence represents the node number. For example, in solution 2 that optimizes the cost objective function the deployment structure indicates that a capacitor of type 3 is installed on nodes 22, 25 and 27, etc.

Table 4. Type and location of capacitors for some solutions

Sol	Capacitors type in each node
1	00000000000200000033003003030000000000003000002202000303303333330002230233002030003033300000000
2	00000000000000000000003003030000000000000000000000000000000000000300000000000120300030333300000
5	00000000000200000033003003030000000000000000000020003033033330300002300300020300020333000000000
9	00000000000000000000300300303000000000000000000000000000000303330030000000030003030003033300000

Fig. 3 displays the non-dominated solutions obtained with different p1 and p2 values for the non-dominated solutions: **(a)** p1 = p2 = 100%, **(b)** p1 = 95% and p2 = 100%, **(c)** p1 = 100% and p2 = 95%, and **(d)** p1 = p2 = 95%. For the dominated and infeasible solutions p1 = p2 = 100% in all examples. The values of the other parameters (associated with the robustness analysis study) are the following: h = 100; δ = 0.004 for resistance and δ = 0.002 for inductance of branches, and δ = 0.001 for active and δ = 0.003 for reactive power at nodes; $\eta = (\eta_1, \eta_2) = (0.004, 200)$ for non-dominated solutions, and $\eta = (\eta_1, \eta_2) = (0.002, 200)$ for dominated and infeasible solutions; q = 1 for the non-dominated solutions and q = 0 for dominated and infeasible solutions.

The thresholds p1 and p2 may be perceived as a measure of the exigency of robustness. If p1 = 100% then all $f^{s'}(x)$ in all neighboring scenarios s' tested are better than $f^{s}(x)$ or belong to the predefined η-neighborhood of $f^{s}(x)$. If p1 = 95% then the $f^{s'}(x)$ that are better than $f^{s}(x)$ or belong to the η-neighborhood of $f^{s}(x)$ are at least 95%. So, it is more probable that a solution x with degree of robustness k in the first case (p1 = 100%) has actually this degree of robustness than in the second case (p1 = 95%) when p1 is relaxed. A similar interpretation can be made for parameter p2 regarding solution feasibility.

For p1 = p2 = 100% (Fig. 3 (a)) most solutions have a degree of robustness 1 and a few solutions near the optimum of the cost of objective function have a degree of robustness 0.

As p1 or/and p2 decreases, thus decreasing the level of exigency of the robustness, the non-dominated solutions present a higher degree of robustness. The most robust solutions are generally located towards the best values for the resistive losses objective function. The relaxation of the p1 or/and p2 values enable to obtain better

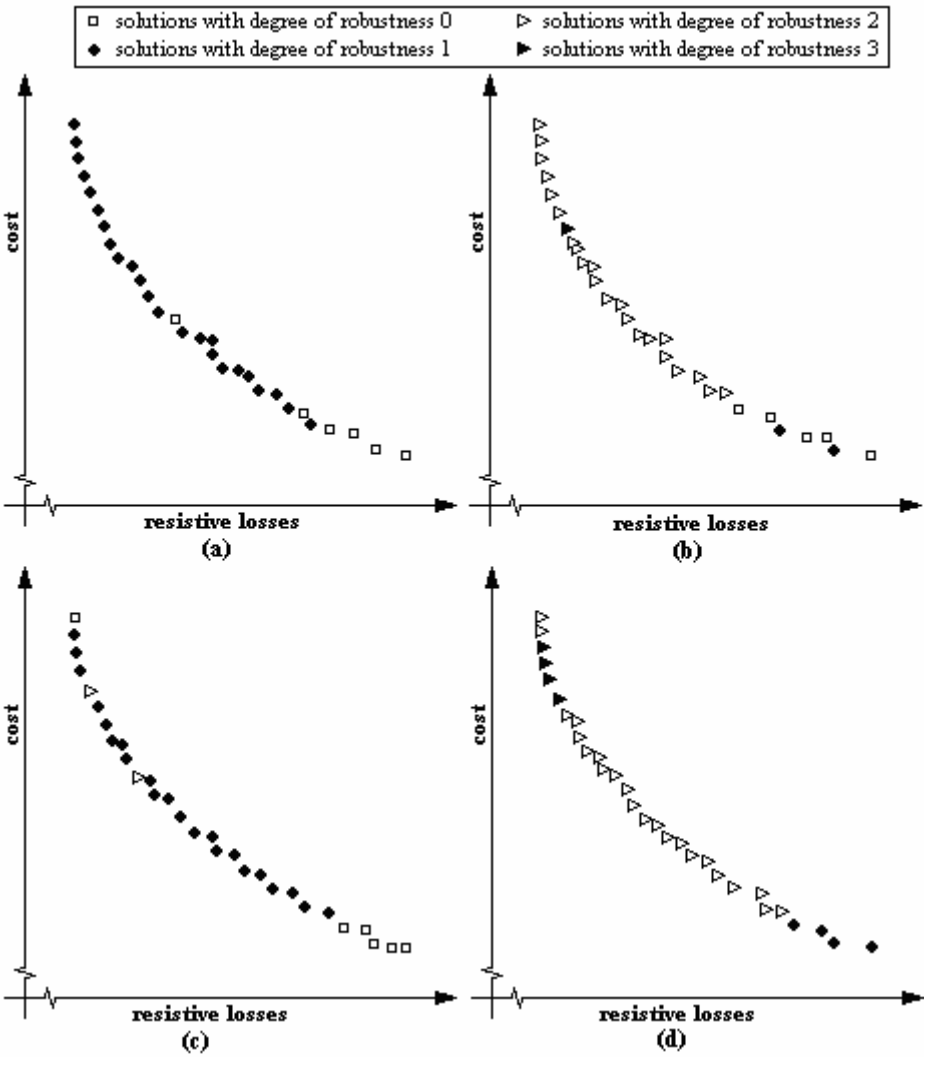


Fig. 3. Pareto-front obtained with **(a)** p1 = p2 = 100%, **(b)** p1 = 95% and p2 = 100%, **(c)** p1 = 100% and p2 = 95%, **(d)** p1 = p2 = 95%

discrimination regarding robustness for the solutions in the Pareto front. In this way, it becomes clearer where the most robust non-dominated solutions are located.

In general, it is expected that a DM strive for well-balanced solutions (that is, presenting trade-offs with satisfactory values for both objective functions) and displaying a high degree of robustness. The information on the degree of robustness can be used to complement the evaluation of the merit of non-dominated solutions based on the objective function values and the trade-offs that are at stake.

6 Conclusions

In this paper, an approach to robustness analysis in evolutionary multi-objective optimization, in which the values of the objective function coefficients and the constraint coefficients are subject to small perturbations, is applied to a combinatorial problem of locating and sizing capacitors for reactive power compensation in electric radial distribution networks. This problem has been modeled as a multi-objective programming problem, considering two objective functions of technical and economical nature - minimization of system losses and minimization of capacitor installation costs.

The concept of degree of robustness is incorporated into the EA, particularly in the computation of the fitness value assigned to the solutions. This approach enables to classify the solutions of the Pareto-front according to their degree of robustness.

Information on the robustness of solutions, and not just on their structure and objective function value trade-offs, is relevant for assisting a DM in assessing the merit of non-dominated solutions and selecting a satisfactory compromise solution that exhibits a higher degree of stability in face of perturbations.

References

1. Branke, J.: Creating Robust Solutions by means of an Evolutionary Algorithm. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 119–128. Springer, Heidelberg (1998)
2. Branke, J.: Efficient Evolutionary Algorithms for Searching Robust Solutions. In: Adaptive Computing in Design and Manufacture (ACDM), pp. 275–286. Springer, Heidelberg (2000)
3. Jin, Y., Sendhoff, B.: Trade-Off between Performance and Robustness: An Evolutionary Multiobjective Approach. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 237–251. Springer, Heidelberg (2003)
4. Tsutsui, S., Ghosh, A.: Genetic Algorithm with a Robust Solution Searching Scheme. IEEE Transactions on Evolutionary Computation 1(3), 201–219 (1997)
5. Deb, K., Gupta, H.: Introducing Robustness in Multiple-Objective Optimization. Evolutionary Computation 14(4), 463–494 (Winter 2006)
6. Hughes, E.J.: Evolutionary Multi-Objective Ranking with Uncertainty and Noise. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 329–343. Springer, Heidelberg (2001)
7. Teich, J.: Pareto-Front Exploration with Uncertain Objectives. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 314–328. Springer, Heidelberg (2001)
8. Li, M., Azarm, S., Aute, V.: A Multi-Objective Genetic Algorithm for Robust Design Optimization. In: Genetic and Evolutionary Computation Conference (GECCO 2005), Washington, DC, USA, pp. 771–778 (2005)
9. Barrico, C., Antunes, C.H.: Robustness Analysis in Multi-Objective Optimization Using a Degree of Robustness Concept. In: IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, Canada, pp. 1887–1892. IEEE Press, Los Alamitos (2006)
10. Barrico, C., Antunes, C.H.: A New Approach to Robustness Analysis in Multi-Objective Optimization. In: 7th International Conference on Multi-Objective Programming and Goal Programming (MOPGP 2006), Loire Valley, City of Tours, France (2006)

11. Barrico, C., Antunes, C.H.: An Evolutionary Approach for Assessing the Degree of Robustness of Solutions to Multi-Objective Models. In: Ong, Y.S., Yaochu, J., Shengxiang, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. Studies in Computational Intelligence, vol. 51, pp. 565–582. Springer, Heidelberg (2007)
12. Jin, Y., Branke, J.: Evolutionary Optimization in Uncertain Environments – A Survey. IEEE Transactions on Evolutionary Computation 9(3), 1–15 (2005)
13. Antunes, C.H., Barrico, C., Gomes, A., Pires, D., Martins, A.: On the Use of Evolutionary Algorithms for Reactive Power Compensation in Electrical Distribution Networks - Experiments on a Case Study. In: 6th Metaheuristics International Conference (MIC 2005), Viena, Austria, pp. 514–519 (2005)
14. Antunes, C.H., Pires, D., Barrico, C., Gomes, A., Martins, A.: A Multi-objective Evolutionary Algorithm for Reactive Power Compensation in Distribution Networks. In: Applied Energy. Elsevier, Amsterdam (2009) (accepted)
15. Pires, D.F., Martins, A.G., Antunes, C.H.: A multiobjective model for VAR planning in radial distribution networks based on Tabu Search. IEEE Transactions on Power Systems 20(2), 1089–1094 (2005)
16. Fonseca, C.M., Fleming, P.J.: An Overview of Evolutionary Algorithms in Multiobjective Optimization. Evolutionary Computation 3(1), 1–16 (1995)
17. Coello, C., Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems. Kluwer Academic Publishers, Dordrecht (2002)
18. Gomes, A., Antunes, C.H., Martins, A.: A multiple objective evolutionary approach for the design and selection of load control strategies. IEEE Transactions on Power Systems 19(2), 1173–1180 (2004)
19. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley and Sons, New York (2001)
20. Iba, K.: Reactive power optimization by genetic algorithm. IEEE Transactions on Power Systems 9(2), 685–692 (1994)
21. Lee, K.-Y., Bai, X., Park, Y.-M.: Optimization method for reactive power planning by using a modified simple genetic algorithm. IEEE Transactions on Power Systems 10(4), 1843–1850 (1995)
22. Kim, K.-H., You, S.-K.: Voltage profile improvement by capacitor placement and control in unbalanced distribution systems using GA. In: IEEE Power Engineering Society Summer Meeting, vol. 2, pp. 800–805. IEEE Press, Los Alamitos (1999)
23. Levitin, G., Kalyuhny, A., Shenkman, A., Chertkov, M.: Optimal capacitor allocation in distribution systems using a genetic algorithm and a fast energy loss computation technique. IEEE Transactions on Power Delivery 15(2), 623–628 (2000)
24. Delfanti, M., Granelli, G., Marannino, P., Montagna, M.: Optimal capacitor placement using deterministic and genetic algorithms. IEEE Transactions on Power Systems 15(3), 1041–1046 (2000)
25. Baran, B., Vallejos, J., Ramos, R., Fernandez, U.: Reactive power compensation using a multi-objective evolutionary algorithm. In: IEEE Porto Power Tech. Conference. IEEE Press, Los Alamitos (2001)
26. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions Evolutionary Computation 6(2), 182–197 (2002)

Staff Scheduling with Particle Swarm Optimisation and Evolution Strategies

Volker Nissen and Maik Günther

Technical University of Ilmenau, Chair of Information Systems in Services,
D-98684 Ilmenau, Germany
volker.nissen@tu-ilmenau.de, maik.guenther@gmx.de

Abstract. The current paper uses a scenario from logistics to show that modern heuristics, and in particular particle swarm optimization (PSO) can significantly add to the improvement of staff scheduling in practice. Rapid, sub-daily planning, which is the focus of our research offers considerable productivity reserves for companies but also creates complex challenges for the planning software.

Keywords: staff scheduling, sub-daily planning, particle swarm optimization, combinatorial optimization, evolution strategy.

1 Introduction to the Problem of Staff Scheduling

Staff scheduling involves the assignment of an appropriate employee to the appropriate workstation at the appropriate time while considering various constraints. This work describes a method for solving the problem of *subdaily* staff scheduling with individual workstations. According to current research employees spend up to 36% of their working time unproductively, depending on the branch [17]. Major reasons include a lack of planning and controlling. The problem can be faced with demand-oriented staff scheduling. Key planning goals are increased productivity, reduction of staff costs, prevention of overtime, motivation of employees with positive results for sales and service [19].

In practice, the application of a system for staff scheduling has not been very prevalent up to now. Most often planning takes place based on prior experience or with the aid of spreadsheets [1]. It is obvious that the afore-mentioned goals of demand-oriented staff scheduling cannot be realised with these planning tools. Even with popular staff planning software employees are regularly scheduled for one workstation per day. However, in many branches, such as trade and logistics, the one-employee-one-station concept does not correspond to the actual requirements and sacrifices potential resources. Therefore, sub-daily planning should be an integral component of demand-oriented staff scheduling.

In the following section, the application problem is stated more formally. Then, we discuss work related to our own research before developing approaches based on PSO and evolution strategies (ES) for sub-daily staff scheduling in section 4. Section 5 describes the practical planning scenario and experimental setup before the empirical results are presented and discussed in section 6. The paper concludes with a short summary and some indications for future work.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 228–239, 2009.

2 Formal Statement of the Problem

The problem starts out assuming a set of employees $\mathcal{E} = \{1, \ldots, E\}$, a set of workstations $\mathcal{W} = \{1, \ldots, W\}$ and a discrete timeframe $\mathcal{T}$ with the index $t = 0, \ldots, T-1$, where each period t of the range has a length l_t greater than zero. The demand d_{wt} of employees per workstation and period cannot be negative.

$$\begin{array}{ll} l_t > 0 & \forall t \in \mathcal{T} \\ d_{wt} \geq 0 & \forall w \in \mathcal{W} \text{ and } \forall t \in \mathcal{T} \end{array} \tag{1}$$

The availability of employees is known at the beginning of the sub-daily planning and is determined using the binary variable a_{et}.

$$a_{et} = \begin{cases} 1 & \text{if employee } e \text{ is available at period } t \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The assignment of an employee to a workstation is controlled using the binary variable x_{ewt}.

$$x_{ewt} = \begin{cases} 1 & \text{if employee } e \text{ is assigned to workstation } w \text{ at period } t \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

An employee can only be associated with a workstation w in the period t if he or she is actually present.

$$\sum_{w=1}^{W} x_{ewt} \leq a_{et} \qquad \forall e \in \mathcal{E} \text{ and } \forall t \in \mathcal{T} \tag{4}$$

Additionally, an employee can only be designated to one workstation at a time.

$$\sum_{w=1}^{W} x_{ewt} \leq 1 \qquad \forall e \in \mathcal{E} \text{ and } \forall t \in \mathcal{T} \tag{5}$$

Any workstation can require a set of qualifications Q_w, and employees have a set of qualifications Q_e at their disposal. If an employee is planned for a workstation but does not meet all necessary qualifications, error points P_q are generated for the duration of the assignment according to the error point size c_q.

$$P_q = \sum_{t=0}^{T-1} \sum_{w=1}^{W} \sum_{e=1}^{E} c_q l_t x_{ewt} \qquad \begin{array}{ll} c_q > 0 & \text{if employee } e \text{ is not qualified} \\ & \text{for workstation } w, \\ c_q = 0 & \text{else} \end{array} \tag{6}$$

If a discrepancy arises from the workstation staffing target d_{wt}, error points P_d are generated for the duration and size of the erroneous assignment according to the error point size. Different types of errors can be distinguished: c_{do} represents

overstaffing when the demand $d_{wt} > 0$, c_{dn} signals overstaffing when the demand $d_{wt} = 0$, c_{du} signals cases of understaffing.

$$P_d = \sum_{t=0}^{T-1} \sum_{w=1}^{W} (c_{dn} + c_{do} + c_{du}) l_t \left| \left(\sum_{e=1}^{E} x_{ewt} \right) - d_{wt} \right|, \text{ with:} \tag{7}$$

$c_{dn} >$ if workstation w is overstaffed at period t and $d_{wt} = 0$, else $c_{dn} = 0$

$c_{do} >$ if workstation w is overstaffed at period t and $d_{wt} > 0$, else $c_{do} = 0$

$c_{du} >$ if workstation w is overstaffed at period t and $d_{wt} = 0$, else $c_{du} = 0$

To avoid an excessive number r_e of sub-daily workstation (job) rotations for any employee c_r error points arise for such (job) rotations.

$$P_r = c_r \sum_{e=1}^{E} r_e \tag{8}$$

Therefore, the objective function to be minimised becomes:

$$minP = P_q + P_d + P_r. \tag{9}$$

3 Related Work

The basic principles of particle swarm optimisation (PSO) were developed by Kennedy and Eberhart among others [10] [11]. Swarm members are assumed to be massless, collision-free particles that search for optima with the aid of a fitness function within a solution space. In this process each single particle together with its position embodies a solution to the problem [23]. While looking for the optimum, a particle does not simply orient itself using its own experience but also using the experience of its neighbours [8]. This means that the particles exchange information, which can then positively influence the development of the population in the social system as a whole [15].

Modifications of standard real-valued PSO exist for binary variables, where the speed of a particle is used as the probability for the change of the binary value [11]. This approach, however, has several limitations and was changed from binary to decimal variables in [24]. Another PSO-variant was developed for sequence planning tasks [20]. In 2007 Poli analysed the IEEE Xplore database for the thematic grouping of PSO applications [16]. Of approximately 1100 publications only one work is focused specifically on timetabling [6] which is related to our own application problem. In [6], the authors adjust PSO to the combinatorial domain. No longer is the position of a particle determined by its speed but rather by using permutation operators. In [5] university timetabling was also approached with PSO.

In [7] Ernst et al. offer a summary of papers related to the issue of staff scheduling - about 700 papers between the years 1954 and 2004 have been included. They identify certain categories of problems, such as the category *flexible*

demand. This category is characterised by little available information on schedules and upcoming orders. A demand per time interval is given as well as a required qualification. Thus, the application problem discussed here can be classified in the group flexible demand schemes. It can additionally be classed under task assignment. *Task assignment* is used to generate assignments requiring certain qualifications and needing to be completed in a certain period of time, which are then distributed amongst the employees. The employees have already been assigned shifts.

As work related to our research Vanden Berghe [22] presents an interesting, though non-PSO heuristic to sub-daily planning. Here, demand is marked by sub-daily time periods , which allows the decoupling of staff demand from fixed shifts resulting in fewer idle times. However, scheduling ist not performed at the detailed level of individual workstations as in our research.

In [13] Schaerf and Meisels provide a universal definition of an employee timetabling problem. Both the concepts of shifts and of tasks are included, whereby a shift may include several tasks. Employees are assigned to the shifts and assume task for which they are qualified. Since the task is valid for the duration of a complete shift, no sub-daily changes of tasks (or rather workstations) are made. Blöchlinger [4] introduces timetabling blocks (TTBs) with individual lengths. In this model employees may be assigned to several sequential TTBs, by which subdaily time intervals could be represented within a shift. Blöchlinger's work also considers tasks; however, a task is always fixed to a TTB. Essentially, our problem of the logistics service provider represents a combination of [13] (assignment of staff to tasks) and [4] (sub-daily time intervals), but with the assignment periods (shifts) of the employees already being set.

Staff scheduling is a hard optimization problem. In [9] Garey and Johnson demonstrate that even simple versions of staff scheduling problems are NP-hard. Kragelund and Kabel [12] show the NP-hardness of the general employee timetabling problem. Moreover, Tien and Kamiyama prove in [21] that practical personnel scheduling problems are generally more complex than the TSP which is itself NP-hard. Thus, heuristic approaches appear justified for our application. Apparently, there exists no off-the-shelf solution approach to the kind of detailed sub-daily staff planning problem considered here. A PSO approach and evolution strategy for this application are outlined in the following section. We assume the reader is familiar with standard-PSO [11] [8] and standard-ES [2] [3].

4 PSO Approach and Evolution Strategy

4.1 Problem Representation

To apply PSO and the evolution strategy, the sub-daily staff scheduling problem needs to be conveniently represented. A two-dimensional matrix is applied. Each particle in the swarm (for PSO) has an own matrix that determines its position. Also, each individual in the ES-population uses a matrix to represent it's solution to the application problem. The rows of the matrix signify employees and the columns signify each time period of the length $l_t > 0$. To mark times in which an

Table 1. Assignment of workstations in a two-dimensional matrix

employee	period							
	0	1	2	3	4	5	6	...
1	1	1	1	1	1	1	1	
2	0	0	2	2	2	2	2	
3	0	0	1	1	2	2	2	
4	0	0	6	6	6	6	2	
5	3	3	3	2	2	0	0	
...								

employee is not present due to his work-time model, a dummy workstation is introduced (in table 1: workstation 0). For example, employee two is absent in the first two periods and then is assigned to workstation 2. Assignment changes can only be made on non-dummy workstations, so that no absent employee is included.

To lower the complexity the number of dimensions should be reduced. This can be realised via a suitable depiction of time. Within the planned day, time is viewed with a time-discrete model. An event point (at which a new time interval begins) occurs when the allocation requirement for one or more workstations or employee availability change. With this method, however, the periods are not equally long any more, so that their lengths need to be stored.

4.2 Outline of Combinatorial PSO for This Application

At the start of PSO the initialisation of the particle position does not take place randomly. Rather, valid assignments w.r.t. the hard constraints are made that use information from the company's current full-day staff schedule to set which employee works at which station. Therefore, valuable foreknowledge is not wasted. Based on this plan, improved solutions can now be determined that include plausible workstation changes.

In each iteration the new particle position is determined by traversing all dimensions and executing one of the following actions with predefined probability. The probability distribution was heuristically determined in prior tests:

- No change: The workstation already assigned remains. (prob. p1)
- Random workstation: A workstation is randomly determined and assigned. Only those assignments are made for which the employee is qualified. The probability function is uniformly distributed. (prob. p2)
- pBest workstation: The corresponding workstation is assigned to the particle dimension from pBest, the best position found so far by the particle. Through this, the individual PSO component is taken into account. (prob. p3)
- gBest workstation: The corresponding workstation is assigned to the particle dimension from gBest (or rather lBest if a gBest neighbourhood topology is not being used). gBest (lBest) represents the best position of all particles globally (in the local neighbourhood). The social behaviour of the swarm is controlled with these types of assignments. (prob. p4)

By considering the best position of all particles, the experience of the swarm is included in the calculation of the new position. Premature convergence on a sub-optimal position can be avoided by using the lBest topology, in which a particle is only linked to its neighbour. The extent to which the swarm acts individually or socially is determined by the probability with which the workstation is assigned from pBest, gBest or lBest. The behaviour of the PSO-heuristic is relatively insensitive to changes of p1, p3, and p4. The optimal value for p2 depends on the problem size. Pre-tests revealed that a value of 0.3% for p2 works best for the problem investigated here. The other probabilities were set at p1=9.7%, p3=30%, and p4=60% with a gBest topology.

The characteristics of PSO have not been changed with these modifications. There are merely changes in the way to determine a new particle position, so that the calculation of the velocity is not needed. The current form of position determination makes it unnecessary to deal with dimension overruns. All other peculiarities of PSO regarding social or global behaviour remain. Even all neighbourhood topologies established as part of continuous parameter optimisation in standard-PSO remain and can be used without restrictions. In our implementation, PSO terminates after 400,000 inspected solutions. In the future, other convergence-based termination criteria could be employed. The following pseudocode presents an overview of the implemented PSO.

```
1: initialise the swarm
2: determine pBest for each particle and gBest
3: loop
4: for i = 1 to number of particles
5:     calculate new position // use the 4 alternative actions
6:     if f(new position)<f(pBest) then pBest=new position // new pBest
7:     if f(pBest)<f(gBest) then gBest=pBest // new gBest
8: next i
9: until termination
```

4.3 Outline of Evolution Strategy for This Application

PSO-results are compared to several variants of the evolution strategy, originally developed by Rechenberg and Schwefel [2] [3]. In our application, it was an objective to preserve the basic characteristics of evolution strategies, even though they are most often used for continuous parameter optimization. However, evolution strategies have proved to be powerful heuristics in combinatorial optimization, see for instance [14].

The ES-population is initialized with valid solutions w.r.t the hard problem constraints. Again, information from the company's current full-day staff schedule is used. (μ, λ)-selection (comma-selection) as well as $(\mu + \lambda)$-selection (plus-selection) are used as well as different population sizes. The best solution found during an experimental run is always stored and updated in a "golden cage". It represents the final solution of the run. Following suggestions in the literature [2] [3], the ratio μ/λ is set to 1/5 - 1/7 during the practical experiments. The recombination of parents to create an offspring solution works as follows:

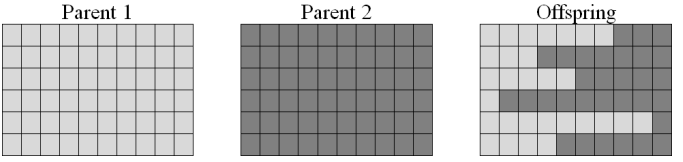


Fig. 1. Recombination operator employed

A crossover point is determined independently and at random for each employee (row) of a solution and the associated parts of the parents are exchanged (see fig. 1).

Mutation of an offspring is carried out by picking an employee at random and changing the workstation assignment for a time interval chosen at random. It must be ensured, though, that valid assignments are made w.r.t. the hard problem constraints. The number of employees selected for mutation follows a $(0, \sigma)$-normal distribution so that small changes are more frequent than large ones. Results are rounded and converted to positive integer numbers. Other approaches are possible here [18]. The mutation stepsize sigma is controlled self-adaptively using a log-normal distribution and intermediary recombination, following the standard scheme of evolution strategies [3].

In prior tests, other recombination and mutation schemes as well as selection pressures were tried, but they performed worse than the current approach. The ES terminates when 400,000 solutions have been inspected to allow for a fair comparison with PSO. The following pseudocode presents an overview of the implemented ES.

1: initialise the population with μ individuals
2: evaluate the μ individuals
3: loop
4: copy and recombine parents to generate λ offspring
5: mutate the λ offspring
6: evaluate the λ offspring
7: select $((\mu + \lambda)$ or $(\mu, \lambda))$ best individuals as the new generation
8: until termination

5 Test Problem and Experimental Setup

The present problem originates from a German logistics service provider. This company operates in a spatially limited area 7 days a week almost 24 hours a day. The tasks of employees concern loading and unloading, short distance transportation and other logistic services. The employees are quite flexible in terms of their working hours, which results in a variety of working-time models. There are strict regulations especially with regard to qualifications because the assignment of unqualified employees might lead to significant material damage and personnel injury. The employer regularly invests a lot of time and money in qualification measures so that many different employees can work at several

different workstations. The personnel demand is known for each workstation, resulting in high planning security. Currently, monthly staff scheduling is carried out manually within MS EXCEL™. Employees are assigned a working-time model and a fixed workstation each day. Several considerations are included, such as presence and absence, timesheet balances, qualifications and resting times etc.

The personnel demand for the workstations is subject to large variations during the day. However, employees are generally scheduled to work at the same workstation all day, causing large phases of over- and understaffing. This lowers the quality of service and the motivation of employees and leads to unnecessary personnel costs as well as downtime. At this time, sub-daily workstation rotation is only rarely used in the planning. Usually, department managers intervene directly on site and reassign the employees manually. Obviously, demand-oriented staff scheduling cannot be realised with this approach.

The planning problem covers seven days (20 hours each), divided into 15-minute intervals. It includes 65 employees and, thus, an uncompressed total of 36,400 dimensions for the optimization problem to be solved. The general availability of the employees is known for each interval from the previous full-day planning. Employee shift planning was done for 13 possible shifts plus a planned off-shift. Nine different workstations need to be filled, with seven having qualification requirements. The variety of qualifications was summarised in four qualification groups.

A staff schedule is only valid if any one employee is only assigned to one workstation at a time and if absent employees are not included in the plan. These hard constraints can be contrasted with soft constraints, which are penalised with error points. The exact determination of error point counts is in practice an iterative process and will not be covered in detail here. The error points used here are from an interview with the logistics service provider and reflect that companys requirements.

All test runs were conducted on a PC with an Intel 4 x 2.67 GHz processor and 4 GB of RAM. Thirty independent runs were conducted each time for each of the experiments to allow for statistical testing. The runtime of ca. 25 minutes for a single run is similar for all solution methods tested in the next section.

6 Results and Discussion

The full-day manual staff schedule without sub-daily workstation changes results in 411,330 error points after an evaluation that included the penalties arising from the afore-mentioned constraints.

The results of the methods are shown in table 2. All heuristics for sub-daily staff scheduling significantly outperform the manual full-day schedule in terms of total error points. This demonstrates the value of sub-daily scheduling as compared to today's standard staff scheduling approaches which not only waste resources but also demotivates personnel and deteriorates quality of service. Generally, the problems of understaffing and overstaffing for periods without demand are greatly reduced. On the other hand, all heuristics lead to more

Table 2. Comparison (error points) of the different sub-daily scheduling heuristics, based on 30 independent runs each. Best results are bold and underlined.

heuristic	error		number of job-changes	wrong qualifications in minutes	understaffing in minutes	overstaffing in minutes		number of fitness evaluations
	mean	min				$demand > 0$	$demand = 0$	
manual plan	411330	411330	0,0	1545	20130.0	14610.0	33795.0	-
PSO (20)	**52162**	**51967**	1666.8	0	7478.5	28488.0	7265.5	400000
PSO (40)	52222	52085	1730.2	0	7568.6	28112.1	7731.4	400000
PSO (100)	52591	52400	1778.5	19.1	8136.8	27874.1	8537.7	400000
PSO (200)	53727	53467	2220.3	0	7658.5	28017.0	7916.5	400000
ES (1,5)	55987	55545	1616.8	0	7994.5	26163.0	10106.5	400001
ES (1+5)	55893	55575	1604.2	0	7970.5	26181.0	10064.5	400001
ES (10,50)	56948	55744	1677.3	0	8093.0	25560.0	10808.0	400010
ES (10+50)	56484	55701	1664.8	0	8029.5	25819.5	10485.0	400010
ES (30,200)	63953	58587	1536.8	0	8999.5	21132.5	16142.0	400030
ES (30+200)	63634	58449	1531.7	0	8906.0	21165.5	16015.5	400030

overstaffing in periods with $demand > 0$ as compared to the initial plan. This approach, however, is sensible because employees can still support each other instead of being idle when $demand = 0$.

Interestingly, the PSO heuristic provides the best results with a rather small swarm size of 20 particles. ES reacted similarly, with the (1+5)- and (1,5)-strategy providing the best results on average. There is no clear advantage for the comma- or plus-selection scheme in the evolution strategies tested here. Moreover, recombination apparently does not guarantee an advantage for the ES on this type of problem. The (1+5)- and (1,5)-strategy with a parent population size of one make no use of recombination, but still perform better than the other ES-variants with larger populations and recombination applied. An increased population size without using recombination also created results worse than those stated in the table, though. Thus, neither a large population size nor the use of recombination necessarily lead to success with evolution strategies.

PSO(20) and ES(1+5) provided the best mean error results in their respective groups. With 30 independent runs for each heuristic it is possible to test the

Table 3. T-test results for pairwise comparison of heuristics

H_1	T	df	significance H_0 (l-tailed)	mean difference	95% confidence intervall of differences	
					lower	upper
PSO(20) < ES(1+5)	-86.19	39.02	< 0.001	-3731.07	-3818.62	-3643.51
PSO(20) < ES(1,5)	-100.26	42.26	< 0.001	-3825.33	-3902.31	-3748.35
PSO(200) < ES(1+5)	-45.85	49.15	< 0.001	-2166.60	-2261.55	-2071.65
PSO(200) < ES(1,5)	-53.07	53.45	< 0.001	-2260.87	-2346.30	-2175.44

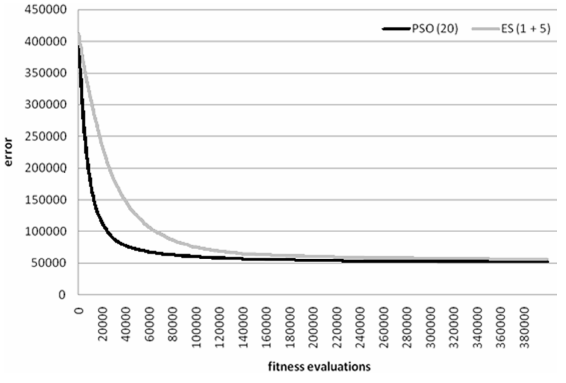


Fig. 2. Convergence chart for PSO(20) and ES(1+5)

statistical significance of the performance difference between both solution methods with a t-test (see table 3). A Levene-test revealed the heterogeniety of variances between both groups ($F = 21.23, p < 0.001$). The corresponding t-test with a 95% confidence interval confirms the better performance of PSO(20) with a very high statistical significance ($p < 0.001$ for H_0). A further test was conducted to compare the worst parameterisation of PSO tested here (PSO(200)) with the best parameterisation of ES, the ES(1+5). Even here PSO outperforms the ES on a highly significant level ($p < 0.001$ for H_0). The results are equally significant when the comparison is made between PSO and the ES(1,5) as can be seen from table (3). This success must be attributed to the operators of PSO since the coding of PSO and ES are identical. A second reason concerns the fewer strategy parameters in our PSO-approach which are more easily adapted to the application domain. Fig. 2 shows the convergence behaviour of best variants from PSO and ES in comparison. Not only does PSO generate the better final solution, but it also demonstrates a more rapid convergence towards good solutions. This is generally a desirable characteristic, particularly when the available time for an optimization is rather limited (not the case here).

Notwithstanding the fact that the PSO heuristic was able to provide better results for this problem, it also has one technical advantage over ES. The

PSO outlined in this paper only requires the varying of two parameters (swarm size and p2), which can both be very easily set. ES, on the other hand, offers more parameterisation possibilities (selection pressure, recombination scheme, plus or comma selection etc.), resulting in greater heuristic complexity from a user's perspective. However, while our ES-approach adheres quite closely to the standard-ES procedure, other parameterisations are certainly possible that could potentially improve over the performance here.

7 Conclusion and Future Work

Sub-daily staff scheduling is a meaningful practical problem area. Using an actual planning scenario, it was demonstrated that particle swarm optimisation produces far better results than traditional full day scheduling. Sub-daily scheduling significantly increases the value contributions of individual staff members. Because PSO in its traditional form is not suitable for the planning problem at hand, the method was adapted to the combinatorial domain without sacrificing the basic PSO-mechanism. PSO also outperformed different variants of the evolution strategy on this problem. The superior performance must be attributed to the operators and parameters of PSO since the coding of PSO and ES are identical. In future research, these promising results are to be expanded by creating further test problems with the aid of cooperating companies. Moreover, other heuristics from roughly comparable problems in the literature are currently being adapted to our domain and tested to further validate the results.

References

1. ATOSS Software AG, FH Heidelberg (eds.): Standort Deutschland 2006. Zukunftssicherung durch intelligentes Personalmanagement. München (2006)
2. Bäck, T. (ed.): Handbook of Evolutionary Computation. Institute of Physics Publishing, Bristol (2002)
3. Beyer, H.G., Schwefel, H.P.: Evolution strategies: a comprehensive introduction. Natural Computing 1, 3–52 (2002)
4. Blöchlinger, I.: Modeling Staff Scheduling Problems. A Tutorial. European Journal of Operational Research 158, 533–542 (2004)
5. Brodersen, O., Schumann, M.: Einsatz der Particle Swarm Optimization zur Optimierung universitärer Stundenpläne. Techn. Rep. 05/2007, Univ. of Göttingen (2007)
6. Chu, S.C., Chen, Y.T., Ho, J.H.: Timetable Scheduling Using Particle Swarm Optimization. In: Proceedings of the International Conference on Innovative Computing, Information and Control (ICICIC 2006), Beijing, vol. 3, pp. 324–327 (2006)
7. Ernst, A.T., Jiang, H., Krishnamoorthy, M., Owens, B., Sier, D.: An Annotated Bibliography of Personnel Scheduling and Rostering. Annals of OR 127, 21–144 (2002)
8. Fukuyama, Y.: Fundamentals of Particle Swarm Optimization Techniques. In: Lee, K.Y., El-Sharkawi, M.A. (eds.) Modern Heuristic Optimization Techniques with Applications to Power Systems, pp. 24–51. Wiley-IEEE Press, New York (2003)

9. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
10. Kennedy, J., Eberhart, R.C.: Particle Swarm Optimization. In: Proc. of the IEEE Int. Conf. on Neural Networks, pp. 1942–1948. IEEE, Piscataway (1995)
11. Kennedy, J., Eberhart, R.C., Shi, Y.: Swarm Intelligence. Kaufmann, San Francisco (2001)
12. Kragelund, L., Kabel, T.: Employee Timetabling. An Empirical Study, Master's Thesis, Department of Computer Science, University of Aarhus, Denmark (1998)
13. Meisels, A., Schaerf, A.: Modelling and Solving Employee Timetabling. Annals of Mathematics and Artificial Intelligence 39, 41–59 (2003)
14. Nissen, V., Gold, S.: Survivable Network Design with an Evolution Strategy. In: Yang, A., Shan, Y., Bui, L.T. (eds.) Success in Evolutionary Computation, Studies in Computational Intelligence, pp. 263–283. Springer, Berlin (2008)
15. Parsopoulos, K.E., Vrahatis, M.N.: Recent Approaches to Global Optimization Problems through Particle Swarm Optimization. Nat. Comp. 1, 235–306 (2002)
16. Poli, R.: An Analysis of Publications on Particle Swarm Optimization. Report CSM-469, Dep. of Computer Science, University of Essex, England (2007)
17. Proudfoot Consulting: Produktivitätsbericht 2007. Company Report (2007)
18. Rudolph, G.: An Evolutionary Algorithm for Integer Programming. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 139–148. Springer, Heidelberg (1994)
19. Scherf, B.: Wirtschaftliche Nutzenaspekte der Personaleinsatzplanung. In: Fank, M., Scherf, B. (eds.) Handbuch Personaleinsatzplanung, pp. 55–83. Datakontext, Frechen (2005)
20. Tasgetiren, M.F., Sevkli, M., Liang, Y.C., Gencyilmaz, G.: Particle Swarm Optimization Algorithm for Single Machine total Weighted Tardiness Problem. In: Proceedings of the CEC 2004, pp. 1412–1419. IEEE, Piscataway (2004)
21. Tien, J., Kamiyama, A.: On Manpower Scheduling Algorithms. SIAM Rev. 24(3), 275–287 (1982)
22. Vanden Berghe, G.: An Advanced Model and Novel Meta-heuristic Solution Methods to Personnel Scheduling in Healthcare. Thesis, University of Gent (2002)
23. Veeramachaneni, K.: Optimization Using Particle Swarm with Near Neighbor Interactions. In: Cantú-Paz, E., Foster, J.A., Deb, K., Davis, L., Roy, R., O'Reilly, U.-M., Beyer, H.-G., Kendall, G., Wilson, S.W., Harman, M., Wegener, J., Dasgupta, D., Potter, M.A., Schultz, A., Dowsland, K.A., Jonoska, N., Miller, J., Standish, R.K. (eds.) GECCO 2003. LNCS, vol. 2723, pp. 110–121. Springer, Heidelberg (2003)
24. Veeramachaneni, K., Osadciw, L., Kamath, G.: Probabilistically Driven Particle Swarms for Optimization of Multi-valued Discrete Problems: Design and Analysis. In: Proceedings of the IEEE SIS 2007, Honolulu, pp. 141–149 (2007)

University Course Timetabling with Genetic Algorithm: A Laboratory Excercises Case Study

Zlatko Bratković, Tomislav Herman, Vjera Omrčen, Marko Čupić, and Domagoj Jakobović

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia
{zlatko.bratkovic,tomislav.herman,vjera.omrcen,
marko.cupic,domagoj.jakobovic}@fer.hr

Abstract. This paper describes the application of a hybrid genetic algorithm to a real-world instance of the university course timetabling problem. We address the timetabling of laboratory exercises in a highly constrained environment, for which a formal definition is given. Solution representation technique appropriate to the problem is defined, along with associated genetic operators and a local search algorithm. The approach presented in the paper has been successfully used for timetabling at the authors' institution and it was capable of generating timetables for complex problem instances.

1 Introduction

The university timetabling problem and its variations are a part of the larger class of timetabling and scheduling problems. The aim in timetabling is to find an assignment of entities to a limited number of resources while satisfying all the constraints. Two forms of university timetabling problems may be recognized in today's literature: *examination timetabling* and *course timetabling* problems, where the differences between those types usually depend on the university involved. The problem can be further specialized as either *post enrollment based* or *curriculum based*. In post enrollment problems, the timetable must be constructed in such a way that all students can attend the events on which they are enrolled, whereas in curriculum problems the constraints are defined according to the university curricula and not based on enrollment data.

Due to inherent problem complexity and variability, most of the real-world university timetabling problems are NP-complete. This calls for the use of heuristic algorithms that do not guarantee an optimal solution, but are in many cases able to produce a solution that is "good enough" for practical purposes. It has been previously shown that metaheuristic-based techniques (such as evolutionary algorithms, tabu-search etc.) are especially well suited for solving these kinds of problems, and this work is an example of that approach.

The paper focuses on a *laboratory exercise timetabling problem* (LETP), which we define as a type of university course timetabling problem (UCTP). The motivation for this work emerged from a need for automated timetable generation

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 240–251, 2009.

at the authors' institution. The timetables could no longer be constructed using traditional methods due to the increased complexity caused by teaching curriculum reforms. The work described here is a part of the research of two different metaheuristics for timetable construction: *genetic algorithm* (GA) and *ant colony optimization* (ACO). In this paper we give a formal definition of the LETP problem and apply a hybrid genetic algorithm to solve real-world instances of the problem. The main contributions of the paper are the definition of solution representation and genetic operators that are tailored to the complex set of timetable constraints, as well as a local search algorithm for additional solution refinement. The result is a GA-based system, capable of producing usable timetables, that is highly adaptive to various idiosyncratic requirements that may be imposed by a particular institution.

A concise overview of some general trends in automated timetabling can be found in [1,2,3,4,5]. Many university course timetabling problems in literature have been intentionally simplified, since real-world examples hold numerous features that make algorithm implementation and performance tracking complicated. A general trend that can be noticed in recent years is that the research focuses on metaheuristic algorithms, instead of application-specific heuristics [2,5,3].

While there is ample research based on simplified artificial problem instances [6,7,8,9,10], we were unable to find an approach that would encompass the requirements imposed by post-enrollment laboratory exercises timetabling, particularly when applied to a large number of students and courses. Since the complexity of the problem significantly depends on the defined constraints [11,12], we present a variant of the problem with additional characteristics which are particularly suited for laboratory timetabling.

The remainder of this paper is organized as follows: Section 2 introduces the actual timetabling problem and in Section 3 we elaborate our approach. Section 4 presents the results while Section 5 concludes the paper and discusses future work.

2 University Course Timetabling Problem

2.1 Problem Statement

Timetable construction is an NP-complete combinatorial optimization problem [13] that consists of four finite sets: a set of meetings, a set of available resources (e.g., rooms, staff, students), a set of available time slots and a set of constraints. The problem is to assign resources and time slots to each given meeting, while maintaining constraints satisfied to the highest possible extent. University course timetabling problem (UCTP) is a timetabling problem where a set of courses and a set of attending courses for each of the students is defined, a course being a set of events that need to take place in the timetable. The main characteristic that discriminates the university course timetabling from other types of timetabling problems is the fact that students are generally allowed to choose courses they wish to enroll [14]. A set of constraints is usually divided into *hard constraints*, whose violation makes the timetable suggestion infeasible, and *soft constraints*, rules that improve the quality of timetables, but are allowed to be violated.

Since this description of UCTP usually does not cover all the requirements imposed by a particular institution, we define additional elements of the problem which allow its application to more specific timetabling instances. The presented model was used for organizing laboratory exercises at the authors' institution, but it can also be used to describe various instances of course timetabling.

2.2 Laboratory Exercise Timetabling Problem

We define the laboratory exercise timetabling problem (LETP) as a six-tuple:

$$LETP = (T, L, R, E, S, C)\,,$$

where T is a set of *time quanta* in which the scheduling is possible, L is a set of limited assets present at university, R is a set of rooms, E is a set of events that need scheduling, S is a set of attending students, and C is a set of constraints.

- A set of time quanta in which scheduling the exercises is possible is denoted T. We assume that the durations of all the exercises can be quantified as a multiple of a fixed time interval, a *time quantum*, denoted t_q. A *time slot* is defined as one or more consecutive time quanta in the timetable. When choosing the duration of the quantum, one needs to make a trade-off between finer granularity of scheduling in time and the larger size of the search space. In all of the examples below, we will assume that the quantum duration is 15 minutes and the exercises can be scheduled between 8:00 and 20:00.
- A set of all the limited assets (resources) available for laboratory exercises is denoted L. One can concieve assets that are required for many different laboratory exercises, but are available in only limited amounts. For example, a laboratory exercise may use a commercial software package, but university can be in possession of only a limited number of licenses. This limits the concurrency of laboratory exercises, as the assets are shared among various courses. For each resource $l \in L$, a quantity, denoted $quantity_l$, is defined as the number of workplaces that can use the resource concurrently.
- With each room, we associate a pair of properties: $(size_r, T_r)$, $size_r \in \mathbb{N}$, $T_r \subseteq T$, defined as follows:
  - The *workplace* is defined as an atomic room resource varying from room to room, such as seats in ordinary classrooms, computers in computer classrooms, etc. For each room $r \in R$, the number of workplaces, denoted $size_r \in \mathbb{N}$, is defined.
  - The rooms required for laboratory exercises could be used for other educational purposes such as exams or lectures. Therefore, for each room $r \in R$ a set of time quanta in which the room is not occupied, denoted $T_r \subseteq T$ is defined.
- Event $e \in E$ is defined as a single laboratory exercise of a course. A single event may be scheduled in one or more *event instances* in different time slots. For example, one instance of the event may be held at 8:00-9:00 on Monday

(for one group of students) and the other at 10:00-11:00 on Tuesday (for the rest of the students). Each event has a set of properties defined as follows:

- Each event e has a duration, denoted $dur_e \in \mathbb{N}$, defined as a multiple of time quanta. For example, the event "Artificial Intelligence exercise 1" may have a duration of $dur_{AI} = 4$ quanta, or 60 minutes.
- Event timespan, denoted $span_e \in \mathbb{N}$, can be defined to ensure that all instances of the event are scheduled within a specified time interval. It is defined as a difference between the end time of the last instance and the start time of the first instance of the event. For example, let us consider an exercise whose part is a brief quiz. To ensure fairness of the test for all students, the staff may demand that all instances of the exercise have to take place within one day (or, even more restrictive, within a certain number of consecutive time quanta).
- The events may take place in different types of rooms, varying from computer classrooms to specialized electronic or electrical engineering laboratories. Hence, for each event the acceptable room set $R_e \subseteq R$ is defined.
- For each event e, a nonempty subset of suitable time quanta, denoted $T_e \subseteq T$ can be defined. For example, the staff of a course may demand events to be scheduled only on Wednesday and Friday, due to the organisation issues.
- Certain events may require the use of one or more limited assets. The set of assets used by the event is denoted $L_e \subseteq L$.
- Usually, members of teaching staff are present at events in order to help the students carry out the exercise. The number of staff present may depend on the event and the room the event is held in. The staff can be viewed as a form of limited asset of an event. The number of staff available for event e is denoted $staff_e \in \mathbb{N}$. The value $usage_{e,r}$ is defined as the number of teaching staff used for event e being scheduled at room r. The usage is undefined for $r \notin R_e$.
- A maximum number of rooms used concurrently for an event, denoted $rooms_e \in \mathbb{N}$, can be defined.
- Some pairs of events may require a partial ordering relation between them. This requirement is apparent in courses with exercises that build on top of each other. For example, event "AI exercise 1" and event "AI exercise 2" need to be scheduled in the same week, but it must be ensured that the second event is scheduled after the first. Also, the students must have at least one whole day between these exercises to prepare properly. A relation, denoted $\succ_d$ can be defined for a pair of events, $\succ_d$: $E \times E$. The events are in relation $e_2 \succ_d e_1$ iff each instance of e_2 is scheduled at least d days after the last instance of event e_1.
- For each event, the number of students per workplace, denoted $spw_e \in \mathbb{N}$ is defined. For example, the staff of some courses may prefer that students in computer classrooms do their exercise on their own, and other prefer group-work, allowing two or more students per workplace.

– Set S is the set of students that are to be scheduled. Each student $s \in S$ has a set of properties defined as follows:

  - As the students are required to attend lectures and exams along with the exercises, it is not possible to assume that the student will always be available for the event. Thus, for each student a set of time quanta when the student is free $T_s \subseteq T$ is defined.
  - Depending on the student's selection of enrolled courses, the student is required to attend a nonempty set of events, denoted $E_s \subseteq E$.

– The requirements of the courses are represented in a set of constraints C. The constraints are divided into hard constraints C_h, which are essential for the courses, and soft constraints C_s, which may require some manual intervention if they are not met. Hard constraints C_h are defined as follows:

  - The room can be occupied by at most one event at any time.
  - The room must be free for use at the time scheduled.
  - Event can be placed only in a room $r \in R_e$ that is suitable for that event.
  - The room r, when used for event e, can accommodate no more than $size_r \cdot spw_e$ students.
  - Event e can be held only at the time defined as suitable for that event.
  - When the event e is placed in a schedule, it occupies the consecutive dur_e quanta belonging to the same day, dur_e being the duration of the event e.
  - Event e must be scheduled within the total $span_e$ for the event.
  - When the ordering relation $\succ_d$ exists between two events, it must be satisfied in the timetable.
  - Asset $l \in L$ can be used concurrently on at most $quantity_l$ workplaces. Thus, the number of students concurrently attending an event is limited by the number of assets the students consume.
  - When the event is placed concurrently in rooms, enough teaching staff must be available to attend the event.
  - Event e can be concurrently placed in at most $rooms_e$ rooms.
  - The students must attend all the events they are enrolled in.

  The set of soft constraints C_s contains two elements:

  - Students can attend an event only at the time when she or he is free from other educational activities.
  - Students can attend only one event at a time.

  Defining these constraints as soft may seem irrational, but the reasoning behind this is as follows: 'hard' constraints are simply those that are at all times satisfied for any individual (i.e., any solution) in GA population in our implementation, whereas for the 'soft' constraints this may not be the case. 'Soft' constraints are defined as such because it was not known in advance whether there even exists a solution that satisfies all the constraints (given

the complex requirements). In other words, our approach tries to find the best solution within the imposed constraints and possibly to give a feedback to the course organisers if some are still severely violated. In the remainder of the text, the term 'feasible solution' denotes the one that satisfies only the hard constraints as defined above.

3 Solving LETP with Genetic Algorithm

3.1 Solution Representation

The adequate solution representation for the LETP suggests itself from the definition of the problem: each solution *sol* (a timetable) contains all the events that have to be scheduled. Each event e is, in turn, scheduled in one or more *event instances*, where each instance is defined with the following: a time slot, a subset of feasible rooms, and a subset of students that attend event e. Events must be allocated in enough instances so that no students are left unscheduled. We consider an event to be the basic unit of heredity in a chromosome.

- *Event instance* $i_e = (ts_i, R_i, S_i)$, where:
  - ts_i denotes the time slot allocated to particular event instance i_e
  - R_i denotes the allocated room set $R_i \subseteq R_e$
  - S_i denotes allocated student subset $S_i \subseteq S_e$
- *Event instance set* $I(e) = \{i_{1e}, \cdots, i_{me}\}$
- *Solution* $sol = \{I(e_1), \cdots, I(e_n)\}$, $\forall e_i \in E$.

3.2 Creation of Initial Population

Each member of the initial population (a single solution) is created using the following procedure:

while set of events E not empty **do**
  select random event e and remove it from E;
  D_e = set of valid days for event e;
  S_e = set of students that attend event e;
  while (D_e not empty) AND (S_e not empty) **do**
    select random day d and remove it from D_e;
    $TS_{e,d}$ = set of valid timeslots for event e in day d;
    while ($TS_{e,d}$ not empty) AND (S_e not empty) **do**
      select random timeslot ts and remove it from $TS_{e,d}$;
      create event instance $i = (ts, R_i, S_i)$ with $R_i = \emptyset$ and $S_i = \emptyset$;
      for every suitable room $r \in R_e$ **do**
        if (r is available in ts) AND (r meets event requirements) **then**
          reserve room r and add it to R_i;
          assign $size_r \cdot spw_e$ students from S_e to r and add them to S_i;
          remove assigned students from S_e;

```
        end if
      end for
    end while
  end while
  if S_e not empty then
    creation unsuccessful;
  end if
end while
```

In the above procedure, 'valid days' and 'valid timeslots' denote suitable days and slots according to the time constraints, while 'room meets requirements' condition ensures all other requirements are met (such as the use of assets, staff availability, etc.). Thus, every member of the population satisfies the given hard constraints, whereas soft constraints may be violated.

It is obvious that the creation of a solution may not always succeed; the above algorithm only succeeds at some *success rate*, dependent of the given set of events and their requirements. However, even a small success rate allows the creation of the desired number of solutions, since the algorithm is only performed at the beginning of the evolution process. In our experiments the success rate ranged between 5% and 75%, so we were always able to build the population in this way. On the other hand, failure to do so could suggest the infeasibility of the given constraints.

3.3 Fitness Function

Fitness function evaluates the quality of a solution and it is proportional to the number of *conflicts* in a given solution. A conflict is a violation of soft constraints which occurs if a student is scheduled to more than one event in the same time slot. The number of conflicts is equal to the number of *time quanta* during which events overlap. The exact fitness measure is defined as:

$$fitness = \sum_{s \in S} \sum_{t_q \in T} \frac{N_{e,s,t_q} \cdot (N_{e,s,t_q} - 1)}{2}$$

where N_{e,s,t_q} represents the number of events the student is scheduled to in current time quantum. For instance, if a student is scheduled on two events that overlap in four time quanta, then the fitness value equals four. The best solution will have fitness value of zero, meaning that no conflicts exist in the solution.

The calculation of the fitness function may be time consuming since it is necessary to iterate through all the students and all the time slots in a solution. In our implementation, the fitness is evaluated *incrementally* after an individual has changed due to genetic operations. In other words, only the part of the solution that has undergone some changes is reevaluated, which significantly speeds up the fitness calculation.

3.4 Genetic Operators

In order to apply genetic algorithm to aforementioned problem, appropriate genetic operators need to be devised. Due to the fact that every solution needs to

satisfy all of the hard constraints, the result of each genetic operation needs to be a solution with the same property. Thus, we introduce especially crafted crossover and mutation operators that are closed over the space of feasible solutions. The crossover operator is defined as follows:

- Let us define $I_{sol}(e)$ as a set of event instances assigned to event $e \in E$ in one particular solution $sol \in P$, where P is current population.
- We define the relation $\otimes$ over $\mathcal{SOL}$ and $\mathcal{I}$ where $\mathcal{I}$ denotes a power set of event instances and $\mathcal{SOL}$ denotes the LETP search space (the set of all possible solutions). The characteristic function of the relation $\chi : \mathcal{SOL} \times \mathcal{I} \rightarrow \{\mathcal{T}, \mathcal{F}\}$ is defined to be true iff the insertion of a particular set of instances $I \in \mathcal{I}$ into $sol \in \mathcal{SOL}$ does not cause violation of hard constraints.
- Now, we consider two particular solutions $a \in P$ and $b \in P$ contained in current population P as parents. Crossover operator takes the parent solutions a and b and produces a solution *child*. Having the relation $\otimes$ defined as above, we describe the crossover operator using the following pseudo code:

Crossover (soluton a, solution b)
solution $child = \emptyset$;
randomly select subset of events $E_1 \subset E$;
$E_2 = E - E_1$;
for all events $e \in E_1$ **do**
  add set of instances $I_a(e)$ to *child*;
end for
for all events e in E_2 **do**
  if $child \otimes I_b(e)$ **then**
    add set of instances $I_b(e)$ to *child*;
  else
    randomly allocate $I(e)$;
    if $child \otimes I(e)$ **then**
      add $I(e)$ to *child*;
    else
      discard *child*;
    end if
  end if
end for

In the above procedure, the creation of the child may not always succeed because the allocation of new event instances cannot always be performed without the violation of hard constraints. In that case, the procedure is repeated until it succeeds or a predefined number of repetitions is performed.

The mutation operator selects a random event and removes the associated set of instances from the solution. The removed event instances are then generated randomly with regard to hard constraints. This operation may be repeated more than once as defined by the *mutationLevel* parameter (defined in subsection 3.6). The mutation operator is applied on child solution after the crossover operation (with a certain probability) and it uses the following pseudo code:

Mutation (solution a)
n = random value between (1, *mutationLevel*);
for $i = 1$ to n **do**
  randomly select event e and remove instance set $I_a(e)$ from solution a;
  randomly allocate new instance set $I(e)$ so that $a \otimes I(e)$ holds;
  add $I(e)$ to a;
end for

3.5 Local Search Algorithm

Since genetic operators are designed to satisfy only hard constraints, the number of conflicts in a solution (which are the consequence of soft constraints violation) may increase during the evolution. To counter that, we implemented a fast local search algorithm which improves the solution significantly.

The local search algorithm operates on students with conflicted schedules within a single solution. The algorithm randomly choses a single student among those and tries to find another instance (in another time slot) of the same event that causes no conflicts for the chosen student. If such instance is not found, the algorithm selects a random instance, but in both cases the student is allocated to another instance. This operation is repeated (for randomly chosen students) until a certain number of consecutive iterations without fitness improvement occurs, where the number of repetitions is predefined. Local search is applied to every individual produced by the crossover operator or modified by mutation.

The primary goal of local search is further reduction of soft constraint violations, since genetic operators are designed to optimize room allocation in different time slots. After the rooms are allocated, the assignment of students to event instances is a subproblem contained in LETP that local search is used to optimize.

3.6 GA Parameters and Adaptation

The presented implementation has an adaptive parameter called *mutationLevel*. Mutation level defines the maximum number of events that will be rescheduled when a solution undergoes mutation. Mutation level is updated if stagnation

Table 1. Genetic algorithm parameters

parameter	value
population size	30
selection algorithm	tournament selection
tournament size	3
individual mutation probability	55 %
initial/maximum mutation level	1 / 10
initial stagnation threshold	5
stop criteria	10000 generations or min fitness = 0

is detected. Stagnation occurs when there is no improvement in population in particular number of generations called *stagnationThreshold*. If stagnation is detected the following parameter updates are performed:

- $stagnationThreshold = stagnationThreshold \cdot 1.5$;
- $mutationLevel = \min(mutationLevel + 1, maxMutationLevel)$;

When improvement occurs, the adaptive parameters are reset. The parameters of the genetic algorithm are shown in Table 1.

4 Results

The described hybrid GA was successfully applied to laboratory exercises scheduling at the authors' institution during the last two semesters. In this paper we present the actual results which were adopted in the students' schedule and used by the departments organizing the exercises. In each semester, there are several periods during which the exercises may take place. The number and durations of those periods are determined by the curriculum and the lectures schedule, which is made prior to the laboratory scheduling. Each scheduling period, denoted a *laboratory cycle*, is in fact a separate and independent timetabling problem with associated dataset.

The timetabling problems at authors' institution had different durations and a greatly varying number of events and attending students (total number of students ranged from several hundred to more than two thousand). To estimate problem size and dificuilty, we introduce the *student events sum* $S_{s,e}$ as a measure of a single laboratory cycle complexity. $S_{s,e}$ is an aggregated number of events that each student must attend, defined as $S_{s,e} = \sum_{s \in S} |E_s|$.

Table 2. Algorithm performance on twelve different *laboratory cycles*

				fitness - best of run				comparison		
cycle	N_e	$S_{s,e}$	days	min	avg	max	st.dev.(σ)	rnd.	LS only	GA only
1	51	7081	9	24	228.6	286	77.2	14615	1937	6647
2	9	2104	5	0	0.0	0	0	4565	399	2667
3	11	2553	5	0	16.2	32	10.1	5839	395	2522
4	16	4868	5	0	28.4	146	44.9	9755	684	4346
5	6	1586	5	0	0.0	0	0	3771	379	2668
6	8	2471	5	0	0.0	0	0	4838	610	2320
7	15	3648	5	0	7.4	12	5.5	7830	623	3424
8	19	5430	4	82	89.9	106	9.6	13486	1505	6021
9	9	3843	5	0	59.0	126	54.95	7088	506	4166
10	8	1701	5	0	3.55	8	5.81	4254	41	3210
11	11	1783	5	0	124.8	132	2.5	3646	466	1974
12	21	5934	5	0	184.3	292	48.1	12477	10540	6476

The algorithm was used in twelve different timetabling problems as listed in the Table 2, where N_e denotes the number of events to be scheduled and *days* is the cycle duration (the statistics are derived from 10 runs on each problem). We managed to find a schedule with no conflicts for ten out of twelve problem instances. The remaining two instances were subsequently proven to be impossible to schedule without conflicts, by identifying a single event with infeasible requirements posed by course organizers, in combination with existing lectures' schedule. For instance, the best found fitness value of 24 in the first cycle resulted from six students that were double booked in four time quanta each. The actual problem datasets are available at http://morgoth.zemris.fer.hr/jagenda.

To illustrate the effectivenes of hybrid algorithm components, we also include in Table 2 *the best results* obtained with random generation of solutions, local search only (LS) and genetic algorithm without local search (GA).

5 Conclusions and Future Work

This paper describes the application of a hybrid genetic algorithm to a complex timetabling problem and shows that, with appropriate solution representation and genetic operators, it is possible to obtain solutions of a very good quality.

The problem is formulated as a laboratory exercises timetabling problem and its formal definition is given. Although highly constrained, we believe that the definition is quite general and may be applied to a wider class of problems, simply by omitting some of the requirements or further specializing the existing ones. For instance, the set L of limited assets may be removed or the time quantum duration may be changed if the problem at hand allows it. The scheduling of the events may be forced to a single instance by defining event timespan equal to the event duration, a whole semester timetable can be generated by repeating a single cycle, etc. With all those adaptations, the solution representation and evolutionary algorithm need not be changed at all, although for a greatly simplified variant of the problem, some other algorithm may consequently obtain the results faster.

The algorithm presumes that every solution in every stage of evolution process satisfies the defined set of hard constraints. To achieve that, we define a solution initialization procedure for building the initial population. The procedure may not always succeed in creation of a valid solution, so it must be repeated until a desired number of solutions are created. While the creation success rate may be low, it still does not present a problem since the initialization of the whole population is performed only once, at the beginning of the evolution. The drawback of this approach is that it is possible to have a set of requirements that would make it impossible to create a valid solution. However, in all our experiments we have not encountered that situation.

The genetic operators preserve the mentioned property of the solutions while trying to combine 'adequately' scheduled events. The efficiency of the operators when used without local search is generally not high, and in that case the convergence is relatively slow. The inclusion of the local search operator, that concentrates

on student allocation only, has proven beneficial to the optimization process as it improved the quality of the individuals and accelerated the convergence.

A possible improvement could be gained by devising and experimenting with different variants of crossover and mutation. Furthermore, a systematic experimentation is needed regarding the values of various parameters used in the algorithm, since they can have a significant impact on the performance. Nevertheless, even the relatively simple operators used in the algorithm succeeded in producing solutions of acceptable quality.

References

1. McCollum, B.: University timetabling: Bridging the gap between research and practice. In: Burke, E.K., Rudová, H. (eds.) PATAT 2007. LNCS, vol. 3867, pp. 15–35. Springer, Heidelberg (2007)
2. Qu, R., Burke, E., McCollum, B., Merlot, L., Lee, S.: A Survey of Search Methodologies and Automated Approaches for Examination Timetabling. Technical Report NOTTCS-TR-2006-4, School of CSiT, University of Nottingham (2006)
3. Burke, E., Petrovic, S.: Recent research directions in automated timetabling. European Journal of Operational Research 127(2), 266–280 (2002)
4. Schaerf, A.: A survey of automated timetabling. In: 115. Centrum voor Wiskunde en Informatica (CWI), p. 33 (1995) ISSN 0169-118X
5. Lewis, R.: A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum (2007)
6. Azimi, Z.: Hybrid heuristics for Examination Timetabling problem. Applied Mathematics and Computation 163(2), 705–733 (2005)
7. Azimi, Z.N.: Comparison of Methheuristic Algorithms for Examination Timetabling Problem. Applied Mathematics and Computation 16(1), 337–354 (2004)
8. Rossi-Doria, O., Sample, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., Stützle, T.: A Comparison of the Performance of Different Metaheuristics on the Timetabling Problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 329–351. Springer, Heidelberg (2003)
9. Socha, K., Sampels, M., Manfrin, M.: Ant Algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: Raidl, G.R., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E., Meyer, J.-A., Middendorf, M. (eds.) EvoIASP 2003, EvoWorkshops 2003, EvoSTIM 2003, EvoROB/EvoRobot 2003, EvoCOP 2003, EvoBIO 2003, and EvoMUSART 2003. LNCS, vol. 2611, pp. 334–345. Springer, Heidelberg (2003)
10. Socha, K., Knowles, J., Sampels, M.: A $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System for the University Timetabling Problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002)
11. Burke, E.K., Jackson, K., Weare, J.K., Automated, R.: university timetabling: The state of the art. The Computer Journal 40(9), 565–571 (1997)
12. van den Broek, J., Hurkens, C., Woeginger, G.: Timetabling problems at the TU Eindhoven. In: PATAT, pp. 123–138 (2006)
13. Cooper, T.B., Kingston, J.H.: The complexity of timetable construction problems. In: Proc. of the 1st Int. Conference on the Practice and Theory of Automated Timetabling (ICPTAT 1995), pp. 511–522 (1995)
14. Gross, J.L., Yellen, J.: Handbook of Graph Theory. CRC Press, Boca Raton (2004)

A Hybrid Algorithm for Computing Tours in a Spare Parts Warehouse

Matthias Prandtstetter, Günther R. Raidl, and Thomas Misar

Institute of Computer Graphics and Algorithms
Vienna University of Technology, Vienna, Austria
{prandtstetter,raidl}@ads.tuwien.ac.at

Abstract. We consider a real-world problem arising in a warehouse for spare parts. Items ordered by customers shall be collected and for this purpose our task is to determine efficient pickup tours within the warehouse. The algorithm we propose embeds a dynamic programming algorithm for computing individual optimal walks through the warehouse in a general *variable neighborhood search* (VNS) scheme. To enhance the performance of our approach we introduce a new self-adaptive *variable neighborhood descent* used as local improvement procedure within VNS. Experimental results indicate that our method provides valuable pickup plans, whereas the computation times are kept low and several constraints typically stated by spare parts suppliers are fulfilled.

1 Introduction

Nowadays, spare parts suppliers are confronted with several problems. On the one hand, they should be able to supply spare parts both on demand and as fast as possible. On the other hand, they have to keep their storage as small as possible for various economic reasons. Storage space itself is expensive, but more importantly by adding additional capacity to the stock, the complexity of administration increases substantially. Therefore, the demand for (semi-)automatic warehouse management systems arises. Beside keeping computerized inventory lists additional planning tasks can be transferred to the computer system. For example, lists containing all articles to be reordered can be automatically generated.

Obviously the main task to be performed within a spare parts warehouse is the issuing of items ordered by customers and to be sent to them as quickly as possible. For this purpose, several warehousemen traverse the storage and collect ordered articles which will then be brought to a packing station where all items are boxed and shipped for each customer. Of course, the possible savings related with minimizing collecting times of items are high and therefore effort should be put into a proper tour planning. Various constraints related to capacities of trolleys used for transporting collected articles, structural conditions of the warehouse and delivery times guaranteed to the customers have to be considered.

The rest of this paper is organized as follows: The next section gives a detailed problem definition. In Sec. 3 we present an overview of related work. A new hybrid approach based on *variable neighborhood search* and *dynamic programming* is presented in Sec. 4. Experimental results and conclusions complete the paper.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 25–36, 2009.

2 Problem Definition

The problem as considered within this paper can be defined as follows: We are given a warehouse with a storage layout similar to that presented in Fig. 1, i.e. several racks are aligned such that two types of aisles arise: *rack aisles* and *main aisles*, whereas we assume that there are two main aisles with an arbitrary number of rack aisles lying between them. While rack aisles provide access to the racks main aisles only act as an interconnection between the rack aisles and the packing station. We denote by $\mathcal{R} = \{1, \ldots, n_{\mathrm{R}}\}$ the set of racks located in the rack aisles.

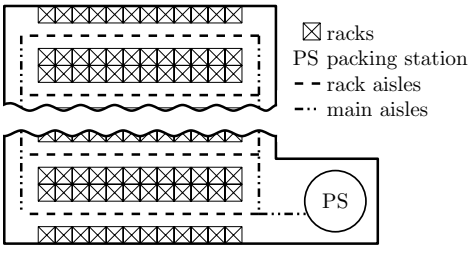


Fig. 1. An exemplary storage layout. Main aisles (vertical in this sketch) and rack aisles (horizontally aligned) are joining each other orthogonally.

In addition, a set of articles $\mathcal{A}$, with $\mathcal{A} = \{1, \ldots, n_{\mathrm{A}}\}$, $n_{\mathrm{A}} \geq 1$, is given. Each article $a \in \mathcal{A}$ is stored at a non-empty set $\mathfrak{R}_a$ of one or more racks, i.e. $\emptyset \neq \mathfrak{R}_a \subseteq \mathcal{R}$. The quantities of articles $a \in \mathcal{A}$ are given by $\mathfrak{q}_a : \mathfrak{R}_a \to \mathbb{N}$.

We assume that a homogeneous fleet of n_{T} trolleys used for carrying collected items exists, each having capacity $\mathfrak{c}$. A group of n_{W} warehousemen, $1 \leq n_{\mathrm{W}} \leq n_{\mathrm{T}}$, is operating these trolleys and issuing ordered articles.

A set of customer orders is given, whereas each order consists of a list of articles with demands to be shipped to a specific address. Further, a latest delivery time to be met is associated with each of these customer orders. Although the assignment of orders to customers is important for a production system we are only interested in the quantities of each article to be collected in the warehouse for this work, since extra workers are assigned to pack all items according to orders. Therefore, we define the set $\mathcal{O}$ of orders as the set of tuples $(a, \mathfrak{d}_a) \in \mathcal{O}$, with $|\mathcal{O}| = n_{\mathrm{O}}$, stating the total integer demand $\mathfrak{d}_a \geq 1$ of each article $a \in \mathcal{A}$. In addition, we assume that the capacities of all trolleys together, i.e. $n_{\mathrm{T}} \cdot \mathfrak{c}$, is greater than the amount of articles to be collected.

Let us denote by set $\mathcal{S}$ a finite set of selections, whereas a selection $S \in \mathcal{S}$ is a set of triples (a, δ, r) such that $r \in \mathfrak{R}_a$, $1 \leq \delta \leq \mathfrak{q}_a(l)$, and there exists an order $(a, \mathfrak{d}_a) \in \mathcal{O}$ with $\mathfrak{d}_a \geq \delta$. Further, we denote by $\mathcal{T}$ a set of tours whereas for each $S_i \in \mathcal{S}$ a tour $T_i \in \mathcal{T}$ exists. By tour T_i we understand a walk through the warehouse visiting all locations contained in S_i such that the corresponding items of S_i can be collected. The length of tour $T_i \in \mathcal{T}$ is denoted by $c(T_i)$.

A solution $x = (\mathcal{S}, \mathcal{T}, \Pi)$ to the given problem consists of a set $\mathcal{T}$ of tours corresponding to the selections in $\mathcal{S}$ as well as a mapping $\Pi : \mathcal{T} \to \{1, \ldots, n_{\mathrm{W}}\}$ of tours to workers such that

- $|\mathcal{S}| = |\mathcal{T}| \leq n_{\mathrm{T}}$,
- tour $T_i \in \mathcal{T}$ collects all articles $a \in S_i$, $S_i \in \mathcal{S}$,
- $\sum_{S \in \mathcal{S}} \sum_{(a,\delta,l) \in S} \delta = \mathfrak{d}_a$, for all $a \in \mathcal{A}$,
- worker $\Pi(T_i)$ processes tour $T_i \in \mathcal{T}$,
- all time constraints are met, i.e. for each customer order it has to be guaranteed that tour $T \in \mathcal{T}$ picking up the last not yet collected article must be finished before the specific delivery time.

We formulate the given problem as an optimization problem in which the total length of the tours, i.e. $\sum_{T \in \mathcal{T}} c(T)$, as well as the violations of the capacity constraints defined by the trolleys, i.e. $\sum_{S \in \mathcal{S}} \max\left\{\sum_{(a,\delta,l) \in S} \delta - \mathfrak{c}, 0\right\}$, should be minimized. For weighting the relative importance of violating capacity constraints compared to tour lengths, we introduce a weighting coefficient ω such that the objective function can be written as

$$\min \sum_{T \in \mathcal{T}} c(T) + \omega \cdot \sum_{S \in \mathcal{S}} \max\left\{\sum_{(a,\delta,l) \in S} \delta - \mathfrak{c}, 0\right\} \tag{1}$$

For this work, ω is set to the maximum possible length of one tour picking up one article plus one. Such a choice for ω implies that it is always better to use an additional tour for collecting an item than violating a capacity constraint.

3 Related Work

Obviously, the stated problem is related to warehouse management in general. An introduction to this topic as well as an overview over tour finding, storage management and other related tasks is given in [1].

It is obvious that the stated problem forms a special variant of the well known *vehicle routing problem* (VRP) [2]. In fact, the classical VRP is extended by additional domain specific constraints. In the classical VRP one wants to find a set of tours minimal with respect to their total length starting at a depot and visiting a predefined set of customers. Further, the problem studied here is related to the *split delivery VRP* [3], the *VRP with time windows* [4] and the *capacitated VRP* [5]. To our knowledge, there exists so far no previous work considering a combination of these variants of VRP in connection with the additionally defined constraints.

Beside this obvious relationship with VRPs, this problem is also related to the *generalized network design problems* [6] with respect to the possibility to collect one article from different locations within the warehouse. At a time only one node of such a cluster has to be visited.

4 A Hybrid Variable Neighborhood Search Approach

Based on the fact that the problem examined within this paper is strongly related to the VRP, we expect that exact approaches are limited to relatively small instances. In addition, short computation times are important, since the observation was made that new orders are committed continuously by customers which implies that the algorithm is restarted frequently. Since recently highly effective *variable neighborhood search* (VNS) [7] approaches have been reported for diverse variants of the VRP [8,9] we also based our approach on a similar concept. Within our hybrid VNS, *variable neighborhood descent* (VND) [7] is used as embedded local search procedure, and subproblems corresponding to the computation of individual tours for collecting particular items are solved by means of dynamic programming [10], exploiting the specific structure of the warehouse.

4.1 The Basic Principle

In this work, we assume that all orders stated by customers can be fulfilled with respect to the capacity constraints defined by the trolleys, i.e. the total capacity of the trolleys is not exceeded. Anyhow, in real-world settings the problem may arise that these constraints cannot be satisfied. In that case a straightforward preprocessing step is used, which partitions the set of orders such that for each partition the capacity constraints are satisfied.

The tour planning algorithm mainly consists of two parts: (1) the allocation of articles to at most n_{T} selections and (2) the computation of concrete routes through the warehouse for collecting all items assigned to the previously determined selections. Anyhow, both of these parts have to be executed intertwined, since the evaluation of the mapping of articles to tours is based on the lengths of these tours. Therefore, these two steps are repeated until no further improvement can be achieved. Finally, an assignment of the walks to n_{W} warehousemen is done, such that the latest finishing time is as early as possible. As soon as additional orders are committed by customers the whole algorithm is restarted from the beginning. This can be done efficiently by using a straightforward incremental update function.

4.2 Assignment of Articles to Tours

One crucial point of our algorithm is the assignment of articles to selections such that in a second step walks through the warehouse can be computed. Nevertheless, the capacity constraints stated by the trolleys as well as the maximum number of available trolleys n_{T} have to be regarded during this allocation step.

Construction Heuristic. For quickly initializing our algorithm we developed a construction method called *collision avoiding heuristic* (CAH). The main idea of CAH is to divide the storage into $m \geq 1$ physically non-overlapping zones whereupon each one is operated by one trolley, i.e. m selections are generated. For this work, we set m to n_{W}.

Since the capacities of the trolleys are not regarded within this initialization procedure the solution qualities produced by this heuristic are not outstanding. The required computation times are, however, very low. Therefore, CAH can be used for providing *ad hoc* solutions such that the workers start collecting the first scheduled item while the rest of the tours is improved in the meantime.

Improvement Heuristic. For improving solutions generated by CAH, we present a *variable neighborhood search* (VNS) approach using an adapted version of *variable neighborhood descent* (VND) as subordinate. The basic idea of VNS/VND is to systematically swap between different neighborhood structures until no further improvement can be achieved. In fact, the crucial task in designing such an approach is the proper definition of appropriate moves used for defining the neighborhood structures incorporated in VNS and VND, respectively. In our approach, the following seven different move types were implemented:

BreakTour(i) Selection $S_i \in \mathcal{S}$ is removed from $\mathcal{S}$ and all articles assigned to S_i are randomly distributed over all other selections $S_j \in \mathcal{S} \setminus S_i$.

MergeTour(i, j) Selections $S_i \in \mathcal{S}$ and $S_j \in \mathcal{S}$ are both removed from $\mathcal{S}$ and merged with each other into a new selection $S_{i'}$, which is then added to $\mathcal{S}$.

ShiftArticle(i, j, a) Any solution generated by this move differs from the underlying solution in one article a which is moved from selection S_i to selection S_j, with $a \in S_i$ and $S_i, S_j \in \mathcal{S}$.

ShiftArticleChangeRack(i, j, a, r) Analogously to the ShiftArticle move, this move shifts an article $a \in S_i$ to selection S_j, with $S_i, S_j \in \mathcal{S}$. In addition to this, a is now collected from rack $r \in \mathfrak{R}_a$ regardless of the position it was acquired before.

SplitTour(i) By applying this move, selection $S_i \in \mathcal{S}$ is split into two new selections $S_{i'}$ and $S_{i''}$ such that $|S_{i'}| = |S_{i''}|$ or $|S_{i'}| = |S_{i''}| + 1$. Selection S_i is removed from $\mathcal{S}$, whereas $S_{i'}$ and $S_{i''}$ are added.

SwapArticle(i, j, a_1, a_2) This move swaps two articles $a_1 \in S_i$ and $a_2 \in S_j$, with $S_i, S_j \in \mathcal{S}$.

SwapArticleChangeRack$(i, j, a_1, a_2, r_1, r_2)$ This move is similar to the SwapArticle move. After swapping articles $a_1 \in S_i$ and $a_2 \in S_j$ between selections $S_i \in \mathcal{S}$ and $S_j \in \mathcal{S}$, the rack of a_1 is changed to $r_1 \in \mathfrak{R}_{a_1}$ and that of a_2 is changed to $r_2 \in \mathfrak{R}_{a_2}$.

Based on these move types, the neighborhood structures for VNS and VND are defined, whereas the neighborhoods $N_1(x), \ldots, N_{k_{\max}}(x)$, with $1 \leq k_{\max} \leq |\mathcal{S}| - 1$, used within the shaking phase of VNS are purely based on BreakTour moves such that within $N_k(x)$, with $1 \leq k \leq k_{\max}$, k randomly chosen BreakTour moves are applied to x. The neighborhood structures $\mathcal{N}_1, \ldots, \mathcal{N}_6$ for VND are defined by a single application of one of the other six move types, such that $\mathcal{N}_1, \ldots, \mathcal{N}_6$ apply SplitTour, MergeTour, ShiftArticle, ShiftArticleChangeRack, SwapArticle, SwapArticleChangeRack, respectively.

In addition to the proper definition of neighborhood structures, a beneficial order used for systematically examining them is necessary and has a great influence on the performance of VND (cf. [11,12]). Preliminary tests showed that the

contributions of neighborhood structures $\mathcal{N}_1$ and $\mathcal{N}_2$ are relatively high during the beginning of VND but dramatically decrease after only a few iterations. This is due to the fact that splitting and merging of tours is only important as long as the capacity constraints are either violated or highly over-satisfied, i.e. there is significant capacity left in more than one trolley. Anyhow, in most of the iterations, i.e. in about 95% of the iterations, no improvement can be achieved by these neighborhoods. Therefore, some dynamic order mechanism guaranteeing that neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_2$ are primarily examined during the beginning phase of VND while being applied less frequently during the later iterations seems to be important.

In contrast to self-adaptive VND as proposed by Hu and Raidl [11], we do not punish or reward neighborhood structures based on their examination times, but reorder the neighborhoods according to their success rates, i.e. the ratios of improvements over examinations, only. The neighborhood order is updated each time an improvement on the current solution could be achieved.

In addition, we adapted VND such that not only improvements on the current solution are accepted but also moves can be applied which leave the current objective value unchanged. To avoid infinite loops, at most ten non-improving subsequent moves are allowed in our version of VND, whereas the i-th non-improving move is accepted with probability $1 - (i - 1) \cdot 0.1$, only. All counters regarding the acceptance of non-improving moves are reset as soon as an improvement could be achieved. As step function a next improvement strategy was implemented, whereas a random examination order was chosen to uniformly sample the current neighborhood.

4.3 Computing Individual Tours

Another crucial point of the proposed algorithm is the computation of concrete tours which will be used by the warehousemen for collecting a specific set of ordered items. Although the decision which article to collect next will finally be made by the workers themselves, the system provides them a suggestion. Further, the evaluation of the assignment of articles to selections is based on the shortest possible tours, and therefore an efficient tour computation is needed.

Please note that a tour as used within this work does not correspond to tours as used within works related to the traveling salesman problem or the VRP. In fact, the main difference lies therein that tours within a storage are allowed to visit each point of interest, i.e. among others the *packing station*, crossings of aisles and rack positions, more than once. This is simply induced by the circumstance that in most cases no direct connection between two points of interest exists. Consequently, paths between points of interest can be walked along more than once within one tour. Anyhow, an upper bound for the number of times the same passage is walked can be provided based on the following two observations.

Theorem 1. *Given is a tour T, which is of shortest length with respect to a set of points of interest, i.e. all of these points are visited by T. Further, we assume that there exist two adjacent points of interest v and w which are twice*

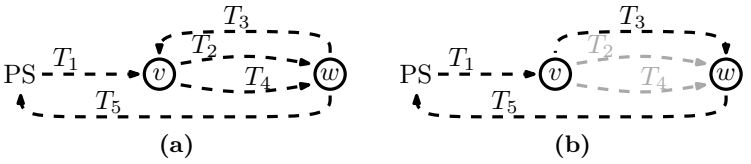


Fig. 2. How to construct a tour T' from a given tour T under the assumption that T visits two times location w immediately after location v

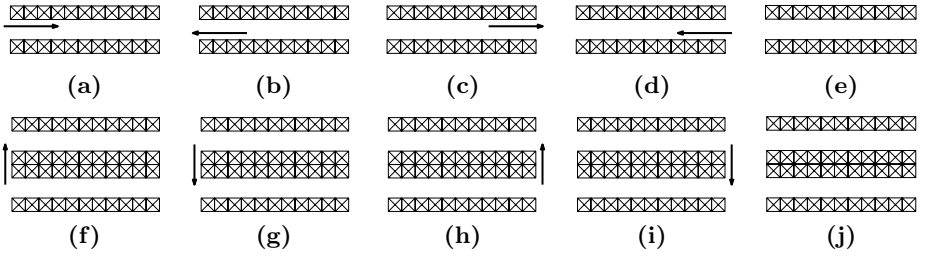


Fig. 3. In (a)–(e) the five basic aisle operations (AOs) are presented, whereas (f)–(j) show basic inter-aisle operations (IOs)

visited immediately consecutively in T. Then the passage between v and w is once traversed from v to w and once vice versa in T.

Proof. Let us assume that the passage between points v and w is traversed twice in the same direction. Then, we can split tour T into five subwalks T_1, T_2, T_3, T_4 and T_5 as shown in Fig. 2a, whereas PS denotes the packing station. A new tour T' can be built by passing segment T_1 from PS to v followed by traversing walk T_3 from v to w and finally walking along T_5 from w to PS, see Fig. 2b. Since v and w are adjacent, i.e. no other point of interest has to be visited when walking from v to w, T' visits the same points of interest as T. Furthermore, since subwalks T_2 and T_4 are not traversed within T', T' is shorter than T, which is a contradiction to the assumption that T is optimal.

Lemma 1. *Given is an optimal tour T with respect to a set of points of interest. Then any two adjacent points v and w are visited at most twice immediately consecutively by T.*

Proof. lemma directly follows from Theorem 1. Under the assumption that points v and w are visited more than two times immediately consecutively the passage between these two points has to be traversed at least twice in the same direction.

Based on the special structure induced by warehouse layouts similar to that shown in Fig. 1, we define *aisle operations* (AOs) and *inter-aisle operations* (IOs). While AOs are representations of the walks to be performed within rack aisles, IOs correspond to movements in main aisles. In Fig. 3 the sets of basic

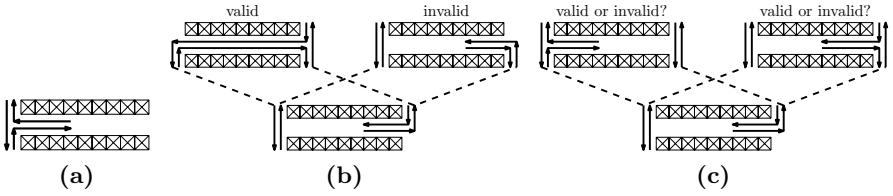


Fig. 4. Figure (a) shows a module representing that part of a tour entering and leaving the RA from and to the left side. This aisle operation is then suitably joined with the rest of the tour by appropriate inter-aisle operations. For some combinations of modules (b) it can be directly decided whether or not they are valid. In other cases (c) this decision has to be postponed.

AOs and IOs are shown. By appropriately combining these basic operations, so-called modules can be defined, which will then be used for representing parts of tours, for an example see Fig. 4a. Based on Theorem 1 it can be concluded that the number of different module types needed for representing a tour is limited.

Although it is now obvious that tours can be built by selecting an appropriate module for each aisle to visit, it can be observed that the resulting tours may contain subtours, which are not connected to the rest of the tour, see for example Fig. 4b. Unfortunately, as shown in Fig. 4c, the decision whether a combination of modules is valid cannot always be made as soon as the next module is selected. Let us denote by $\mathcal{N}_c(j)$ the set of those modules j' which might be connected with module j with respect to the IOs of j and j', i.e. all modules j' forming together with j possibly valid tour parts. Further, we denote by $\mathcal{N}_v(j)$ the set of those modules $j' \in \mathcal{N}_c(j)$ such that the usage of modules j and j' results in a definitely valid tour (part).

Therefore, we introduce two $(n+1) \times (\nu)$ matrices σ and τ, with n being the number of aisles containing items to be selected and ν indicating the maximum number of potentially used module types. An entry σ_{ij}, with $1 \leq i \leq n$ and $1 \leq j \leq \nu$, corresponds to the length of a valid tour T' which visits all rack locations in aisles 1 to i storing articles to be shipped to customers and performs in aisle i the operations corresponding to module j. Analogously, an entry τ_{ij} corresponds to the total length of tour parts which visit all racks in aisles 1 to i storing articles to be shipped and perform in aisle i the operations corresponding to module j. Anyhow, these tour parts need not to be connected with each other and therefore it has to be assured that they are going to be joined into one (big) tour by operations performed in any aisle $> i$. Now, let us assume that $c_i(j)$ denotes the length of the tour part(s) represented by module j when applied to aisle i and module μ represents the IOs necessary for reaching the first aisle from the packing station. Then, the entries of σ and τ can be computed by using the following recursive functions:

$$\sigma_{0\mu} = \tau_{0\mu} = 0 \tag{2}$$

$$\sigma_{0j} = \tau_{0j} = \infty \qquad \text{for } j \in \{1, \ldots, \nu\} \setminus \{\mu\} \tag{3}$$

$$\sigma_{ij} = c_i(j) + \min \left\{ \begin{array}{l} \{\sigma_{i-1j'} : j' \in \mathcal{N}_{\mathrm{v}}(j)\} \cup \\ \{\tau_{i-1j'} : j' \in \mathcal{N}_{\mathrm{v}}(j)\} \end{array} \right\} \quad \begin{array}{l} \text{for } i \in \{1, \ldots, n\} \\ \text{for } j \in \{1, \ldots, \nu\} \end{array} \tag{4}$$

$$\tau_{ij} = c_i(j) + \min \left\{ \begin{array}{l} \{\sigma_{i-1j'} : j' \in \mathcal{N}_{\mathrm{c}}(j)\} \cup \\ \{\tau_{i-1j'} : j' \in \mathcal{N}_{\mathrm{c}}(j)\} \end{array} \right\} \quad \begin{array}{l} \text{for } i \in \{1, \ldots, n\} \\ \text{for } j \in \{1, \ldots, \nu\} \end{array} \tag{5}$$

For decoding the optimal tour, one first needs to identify module J used for aisle n in an optimal tour, i.e. $J = \arg\min_{j \in \{1,\ldots,\nu\}} \{\sigma_{nj}\}$. Then, the computations based on Eq. (4) and (5) have to be performed backwards. Anyhow, it can be easily proven that $\sigma_{nJ} \neq \infty$ definitely holds. In case of ties any module can be chosen.

4.4 Assignment of Workers to Tours

In a final step, an assignment of workers to tours has to be computed such that the latest finishing time is as early as possible while regarding the guaranteed delivery times. For this purpose, we implemented a General VNS scheme using a greedy construction heuristic and random moves for the shaking phase. There are three different move types defining neighborhood structures for VND: The first one reassigns one tour from one worker to another worker. The second one swaps two tours between two workers. The third move type rearranges the schedule of one worker such that one tour is shifted towards the beginning of the current working day. The used neighborhood order corresponds to this order and is fixed. A first improvement strategy is used as step function in VND and random moves are applied for the shaking phase of VNS.

5 Experimental Results

For evaluating the performance of the proposed method several test runs were performed. Our algorithm was implemented in Java and all tests were run on a single core of a Dual Opteron with 2.6GHz and 4GB of RAM. All instances were randomly generated based on the characteristics of real-world data, i.e. storage layouts and typical customer orders, provided by our industry partner Dataphone GmbH.

For each of the 20 instances, we performed 40 independent runs, whereas for the half of those runs we allowed that workers may reverse within one aisle, while for the remaining ones we assured that once an aisle is entered, it is completely traversed by the warehousemen. To avoid excessive computation times, a time limit of 1200 seconds for VNS was applied. See Tab. 1 for a detailed listing of the obtained results. The column labled init presents the initial values obtained by the so called s-shaped heuristic [1], whereas we simply try to collect as much articles as possible during one tour regarding the capacities of the trolleys used. The objective values provided within the nex columns correspond to the weighted sum as presented in Eq. 1. The standard deviations (presented in parentheses) are relatively low with respect to the tour lengths. Taking a look at these average values it can be observed that the tour lengths can be reduced by about 20% on average when

Table 1. Average results over 20 runs for 20 instances. The initial values, the objective values (including standard deviations in parentheses), the number of tours (n_T), the average filling degree of the trolleys used (quota) and the average computation times in seconds are opposed for instances allowing to reverse in aisle and disallowing turning around. The last column presents the p-values of an unpaired Wilcoxon rank sum test, for evaluating whether the tours with turning around are 20% shorter than those without reversing.

		reversing disabled				reversing enabled				
inst.	init	objective	n_T	quota	time	objective	n_T	quota	time	p-Val
(01)	3750	3059.0 (132.6)	3.0	74.2	37.5	2281.0 (44.7)	3.0	74.2	35.5	<0.01
(02)	4200	3213.0 (156.5)	3.7	79.3	29.2	2348.0 (85.6)	3.5	85.0	59.1	<0.01
(03)	4260	3560.0 (109.0)	4.0	75.5	47.5	2541.5 (80.0)	4.0	75.5	78.7	<0.01
(04)	3210	2962.0 (35.8)	3.0	91.8	22.6	2298.5 (4.9)	3.0	91.8	30.6	<0.01
(05)	4020	3605.0 (105.6)	4.1	88.1	37.8	2466.5 (39.1)	4.0	90.3	55.5	<0.01
(06)	5060	4671.5 (106.7)	5.7	78.3	61.2	3576.0 (74.9)	5.4	82.0	114.6	<0.01
(07)	6050	5575.5 (79.3)	7.0	81.7	83.8	4052.5 (73.5)	6.8	82.9	192.7	<0.01
(08)	5650	5602.5 (155.8)	7.0	84.9	95.6	4159.5 (98.9)	7.0	85.5	212.6	<0.01
(09)	7000	6583.0 (246.9)	8.0	86.4	132.5	4887.5 (117.6)	8.0	86.4	334.1	<0.01
(10)	5070	4995.5 (252.1)	6.0	78.8	88.2	3589.5 (104.1)	5.8	82.2	128.8	<0.01
(11)	10740	9254.0 (268.4)	12.2	81.9	391.4	6158.5 (149.3)	11.1	90.4	845.4	<0.01
(12)	9350	8155.0 (175.9)	12.6	79.3	255.4	5952.0 (136.9)	11.7	85.4	689.0	<0.01
(13)	9970	8939.0 (256.2)	12.0	83.2	323.9	6102.0 (156.7)	11.3	88.0	715.7	<0.01
(14)	9520	9082.5 (246.5)	12.6	79.6	370.7	6165.5 (181.2)	11.7	85.8	864.3	<0.01
(15)	7690	7473.0 (270.4)	11.5	86.9	279.2	5860.5 (74.9)	11.2	89.2	673.0	0.02
(16)	11510	8878.0 (240.3)	12.2	81.9	716.4	6465.0 (165.9)	11.7	85.8	1200.0	<0.01
(17)	11460	8251.5 (216.7)	12.3	80.9	782.2	6261.5 (161.7)	11.7	85.4	1200.0	<0.01
(18)	11740	8520.0 (187.8)	12.6	79.3	748.5	6238.5 (159.9)	11.5	86.9	1200.0	<0.01
(19)	11480	8990.0 (216.8)	12.1	82.6	828.3	6349.0 (207.0)	11.7	85.8	1200.0	<0.01
(20)	12260	9644.5 (286.9)	12.6	79.6	819.0	6635.0 (164.4)	11.8	85.0	1200.0	<0.01

it is allowed to reverse within an aisle. To statistically confirm this observation, we performed an unpaired Wilcoxon rank sum test. The corresponding p-values are shown in the last column of Tab. 1. Regarding the number of tours as well as the filling degree of the trolleys, no significant difference between those runs allowing reversing and those forbidding it can be identified. Finally, taking a closer look at the computation times, it can be observed that for those instances including the option to reverse the running times are longer. This can be reasoned by the fact that the number of aisle operations is more restricted for those runs forbidding reversing. Since the available computation time was limited, the results obtained for instances 16–20 might be further improved when using looser time limits. Regarding the last step of the algorithm, i.e. the assignment of tours to workers, all violations of the time constraints could be resolved within a few iterations of the corresponding VNS procedure.

Regarding the performance of the proposed neighborhoods, i.e. the ratio of improvements over examinations, we observed that all of them except $\mathcal{N}_6$ contribute substantially to the final solution whereas neighborhood $\mathcal{N}_6$ did almost

Table 2. Average contributions of the individual neighborhood structures to the final solutions. The numbers represent the average ratios of improvements over examinations over 20 runs for 20 instances with 25, 50, 100, 200 different items to be collected (#it.).

		reversing disabled						reversing enabled					
inst.	#it.	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$	$\mathcal{N}_6$	$\mathcal{N}_1$	$\mathcal{N}_2$	$\mathcal{N}_3$	$\mathcal{N}_4$	$\mathcal{N}_5$	$\mathcal{N}_6$
(01)	25	3.6	10.6	60.2	34.4	46.9	0.2	5.9	15.9	70.2	61.5	18.2	0.0
(02)	25	4.1	12.0	61.0	32.2	33.8	0.0	5.6	19.7	73.0	46.2	16.3	1.1
(03)	25	3.6	11.8	60.5	40.6	49.9	0.1	4.4	17.1	73.4	55.2	27.9	0.0
(04)	25	4.5	11.1	52.1	27.4	34.6	0.0	7.4	18.2	70.9	42.8	6.9	0.4
(05)	25	6.6	15.2	65.9	29.3	45.2	0.8	9.1	19.9	74.2	58.0	32.3	0.0
(06)	50	10.4	17.6	69.6	3.7	18.7	0.0	10.7	23.3	81.8	53.5	7.6	0.0
(07)	50	12.2	20.7	74.3	6.0	42.0	0.0	12.9	25.7	85.6	59.6	48.2	0.0
(08)	50	13.5	19.5	72.3	1.7	55.9	0.0	19.1	28.6	87.9	29.9	40.2	0.0
(09)	50	13.9	20.3	74.4	41.2	46.3	0.0	17.8	27.8	85.8	46.8	37.4	0.0
(10)	50	9.8	17.4	70.8	9.2	22.3	0.0	12.6	23.1	81.8	66.9	18.5	0.0
(11)	100	16.0	27.1	76.3	7.9	24.6	0.0	21.6	29.4	88.5	44.0	37.0	0.0
(12)	100	18.9	29.9	77.8	0.9	24.7	0.0	18.4	28.8	88.3	47.2	39.2	0.0
(13)	100	16.5	26.2	75.9	5.9	26.7	0.0	23.5	29.3	87.3	60.4	44.9	0.0
(14)	100	14.3	27.6	78.2	1.1	19.5	0.0	19.4	27.2	88.0	50.5	33.8	0.0
(15)	100	18.2	23.2	73.4	1.1	12.8	0.0	20.1	26.7	85.3	51.1	30.8	0.0
(16)	200	18.0	30.3	75.3	2.9	8.1	0.0	25.7	31.5	88.8	46.4	30.5	0.0
(17)	200	17.9	28.8	75.7	0.0	7.9	0.0	27.0	31.8	88.8	39.3	49.2	0.0
(18)	200	19.2	28.6	76.0	3.5	15.0	0.0	28.1	32.2	88.8	53.8	38.6	0.0
(19)	200	16.1	28.1	74.5	1.8	17.3	0.0	26.2	29.8	88.4	35.9	35.1	0.0
(20)	200	16.4	28.2	75.4	1.1	6.7	0.0	27.5	34.7	90.2	46.6	32.8	0.0

never add an improvement (see Tab. 2). Nevertheless, for the small instances with 25 articles to be shipped, some improvements could be achieved even by $\mathcal{N}_6$, and therefore we included it here. Regarding the other neighborhood structures, $\mathcal{N}_3$, i.e. the neighborhoods based on the ShiftArticle move, performed best. It is interesting that neighborhood structure $\mathcal{N}_4$ did worse for those instances forbidding turning around. However, this can be explained by the fact that for these instances the degree of freedom is less than for the others which results therein that once an aisle i has to be entered all ordered articles stored within this aisle can be collected with less expenses from aisle i than from any other aisle. The same observation holds for neighborhood structure $\mathcal{N}_5$, although this effect is less prominent for the underlying move type.

6 Conclusions

In this paper, we proposed a new hybrid algorithm combining *variable neighborhood search* (VNS) with dynamic programming (DP) for solving a real-world scheduling and tour finding problem within a spare parts warehouse. For boosting the performance of the neighborhood structures used within *variable neighborhood descent* (VND), we used a new self-adaptive VND rearranging the

neighborhoods according to their success rates. Individual optimal tours within the spare parts warehouse are computed by means of dynamic programming, whereas the special structure of the storage is exploited.

Experimental results showed that this approach performs good for instances based on real-world characteristics. Further, we showed that the total tour lengths can be reduced by about 20% on average when reversing within aisles is allowed. Regarding the computation times, our approach is able to provide good results within 1200 seconds which correspond to acceptable time limits in real-world scenarios.

References

1. de Koster, R., Le-Duc, T., Roodbergen, K.J.: Design and control of warehouse order picking: A literature review. European Journal of Operational Research 182(2), 481–501 (2007)
2. Toth, P., Vigo, D.: The Vehicle Routing Problem. Monographs on Discrete Mathematics and Applications, vol. 9. SIAM, Philadelphia (2002)
3. Dror, M., Laporte, G., Trudeau, P.: Vehicle routing with split deliveries. Discrete Applied Mathematics 50(3), 239–254 (1994)
4. Solomon, M.M.: Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations Research 35(2), 254–265 (1987)
5. Ralphs, T.K., Kopman, L., Pulleyblank, W.R., Trotter, L.E.: On the capacitated vehicle routing problem. Mathematical Programming 94(2–3), 343–359 (2003)
6. Feremans, C., Labbe, M., Laporte, G.: Generalized network design problems. European Journal of Operational Research 148(1), 1–13 (2003)
7. Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover, Kochenberger (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer Academic Publisher, Dordrecht (2003)
8. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research (2007) (in press), doi:10.1016/j.ejor.2007.08.048
9. Ostertag, A., Dörner, K.F., Hartl, R.F.: A variable neighborhood search integrated in the POPMUSIC framework for solving large scale vehicle routing problems. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) HM 2008. LNCS, vol. 5296, pp. 29–42. Springer, Heidelberg (2008)
10. Bellman, R.E.: Dynamic Programming. Dover Publications Inc., Mineola (2003)
11. Hu, B., Raidl, G.R.: Variable neighborhood descent with self-adaptive neighborhood-ordering. In: Cotta, C., Fernandez, A.J., Gallardo, J.E. (eds.) Proceedings of the 7th EU/MEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics (2006)
12. Puchinger, J., Raidl, G.R.: Relaxation guided variable neighborhood search. In: Proceedings of the XVIII Mini EURO Conference on VNS (2005)

A New Binary Description of the Blocks Relocation Problem and Benefits in a Look Ahead Heuristic

Marco Caserta, Silvia Schwarze, and Stefan Voß

University of Hamburg, Institute of Information Systems,
Von-Melle-Park 5, 20146 Hamburg, Germany
{caserta,schwarze}@econ.uni-hamburg.de, stefan.voss@uni-hamburg.de

Abstract. We discuss the blocks relocation problem (BRP), a specific problem in storing and handling of uniform blocks like containers. The BRP arises as an important subproblem of major logistic processes, like container handling on ships or bays, or storing of palettes in a stacking area. Any solution method for the BRP has to work with the stacking area and needs to draw relevant information from there. The strength of related approaches may rely on the extensive search of neighborhood structures. For an efficient implementation, fast access to data of the current stacking area and an efficient transformation into neighboring states is needed. For this purpose, we develop a binary description of the stacking area that fulfills the aforementioned requirements. We implement the binary representation and use it within a look ahead heuristic. Comparing our results with those from literature, our method outperforms best known approaches in terms of solution quality and computational time.

1 Introduction

Efficient handling in warehouses and inventories is a major issue to decrease logistic costs and processing times. This includes fast access to stored items. We consider a particular case where N uniform blocks (e.g., containers, palettes) are piled up on each other in a two-dimensional area. In this setting, it is only possible to access blocks that are located on top of a stack. Each block is given a unique priority number $n = 1, \ldots, N$; thus the priorities establish a total order of the considered blocks. Blocks have to be retrieved according to their priority (smallest number first). Thus, whenever the block with the highest priority (the target block) is not accessible due to other blocks piled on top of it, we have to perform relocation activities to free the target block.

We assume that we have to clear the complete stacking area according to the given priorities. (At a container terminal this refers to completely clearing a bay and moving all the containers of the bay onto a vessel.) Furthermore, we assume that whenever a target block is accessible, it is going to be retrieved immediately. That is, retrievals are carried out while the relocation process is running. The objective of our approach is to minimize the number of relocations performed till

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 37–48, 2009.

the stacking area is cleared. For configuration of the stacking area, we assume that we have a given number $W > 0$ of stacks and a maximum number $H > 0$ of tiers (where this is the same for all stacks).

Whenever we have a pair of blocks located in the same stack, and the priority of the lower block is higher than the priority of the upper block, we say that this pair of blocks forms a *deadlock*. This definition is given in a more general version by [5] in the context of blocks-world planning. In the case of a deadlock, the upper block will cause an extra relocation. However, one relocation might resolve more than one deadlock. A related, but different notion is that of a *mis-overlay*, described, e.g., by [8]. To obtain the number of mis-overlays in a stacking area, one has to count the number of blocks that are placed above blocks with higher priority. This value is also called *confirmed relocation* [6].

The BRP has been addressed before with the following contributions being most successful in solving the problem. Kim and Hong [6] develop a branch-and-bound method and a heuristic based on expected values of future relocations. Furthermore, [3] propose two mathematical programming formulations and a simple heuristic rule. The corridor method is applied to the BRP by [4].

Related problem settings are found in the area of container handling. For instance, in stowage planning, loading, unloading, and sorting processes are described for container ships that visit a number of ports on a round-trip [2,1]. Moreover, in a second class of problems the sorting process is separated from the retrieval process. To this class belong the location of incoming containers in a stacking area [7,13] and pre-marshalling problems [9,8]. For a general overview on container handling see [11,10].

To be comparable with approaches presented in literature, we carry over the following assumption A1 from [6] and [4]. Note that by establishing *A1*, optimal solutions might be cut away, as illustrated by [3].

Assumption A1: Let k be the number of the current target blocks. Before retrieving k, only relocations of blocks located above k are permitted.

For solving the BRP, we propose a heuristic approach which explores the neighborhood of a given stacking area. Two stacking area states are neighbored if one can be transformed into the other through exactly one relocation. To enhance the search procedure, a score based on simple heuristic rules as well as a look ahead mechanism is used to estimate the quality of an intermediate solution and to focus on promising regions of the search space.

Building on a simple greedy algorithm such as, e.g., a construction heuristic the pilot method [12] is a metaheuristic not necessarily based on a local search in combination with an improvement procedure. It primarily looks ahead for each possible local choice (by computing a so-called "pilot" solution), memorizing the best result, and performing the respective move. One may apply this strategy by successively performing a greedy heuristic for all possible local steps (i.e., starting with all incomplete solutions resulting from adding some not yet included element at some position to the current incomplete solution). The look ahead mechanism of the pilot method is related to increased neighborhood depths as

it exploits the evaluation of neighbors at larger depths to guide the neighbor selection at depth one. For a survey on the pilot method including similar ideas being investigated under the acronym rollout method we refer to [12].

In most applications, it is reasonable to restrict the pilot process to some evaluation depth. That is, the method is performed up to an incomplete solution (e.g., partial assignment) based on this evaluation depth and then completed by continuing with a conventional heuristic.

Regarding the BRP even an evaluation depth of one seems reasonable as it prevents the search from immediate capture in simple basins of attraction. The strength as well as the drawback of this approach lies in extensive repetition of calculations and the extensive search of neighborhood structures. Thus, fast access to information concerning the current status of the stacking area and fast exploration of the neighborhood are crucial. To achieve an efficient implementation, defining an appropriate representation of the problem setting is essential.

The major contribution of this paper is the introduction of a new binary representation of a stacking area as it is given in the BRP. This encoding generates a $(N+W) \times (N+W)$-matrix providing fast access to all relevant information of a current stacking area. Moreover, the neighborhood of a current stacking area is described by a set of simple matrix manipulations. By developing a problem-independent encoding, we allow for use of generic approaches, using the full strength of highly developed toolboxes. We apply the binary description for the implementation of a randomized metaheuristic for the BRP. Our experimental results illustrate that we outperform current approaches provided in literature in terms of computational time and solution quality.

The outline of the paper is as follows. The new representation is described in Section 2. Algorithmic details can be found in Sections 3 and 4. Section 5 provides a comparison of the results given in literature and our results. Finally, we give a summary and some ideas for extending our research.

2 Binary Encoding of the BRP

To adapt heuristics and metaheuristics to a particular optimization problem, a good representation of the considered problem is needed. We are looking for a translation of the BRP into a problem-independent description (e.g., a binary one). Afterwards, this allows to apply especially metaheuristics without knowing details about the actual problem description. Together with developing such an encoding goes the definition of a valid neighborhood and transformation steps between solutions in a neighborhood. In this section, we propose such a binary formulation together with a neighborhood transformation for the BRP.

The idea is to transform a current stacking area state to a binary $(N+W) \times (N+W)$-matrix. Rows and columns $1, \ldots, N$ correspond to the N blocks stored in the initial stacking area. Moreover, rows and columns $N+1, \ldots, N+W$ correspond to the W stacks of an initial stacking area and may be interpreted as W "artificial" blocks that lie at tier 0 and are not going to be retrieved. Consider Figure 1 with a bay with three stacks and three tiers as an illustration of that

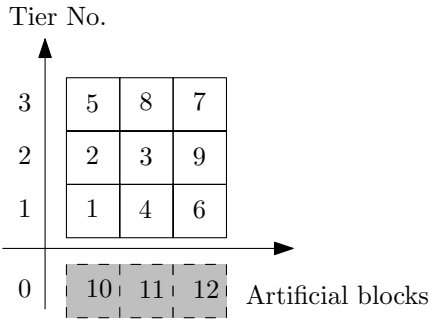


Fig. 1. Stacking area including artificial blocks

interpretation. Artificial block p is related to stack $p - N$. The function of the artificial blocks is to keep track of the actual stacks and to indicate empty stacks.

Let us indicate with $N' = N+W$ the total number of rows (as well as columns) of the binary matrix. Moreover, let us denote the binary matrix by A and define its elements for all blocks $i, j = 1, \ldots, N'$ as follows:

$$a_{ij} = \begin{cases} 1, & \text{if } i \text{ is located in the same stack, at a position below } j \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

For instance, the stacking area given in Figure 1 is transformed into the matrix:

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Given this matrix, row four indicates that above block four are located block three and block eight. On the other hand, column seven says that blocks six, nine and twelve are below block seven. However, block $p = 12$ is an artificial block and represents the stack $p - N = 12 - 9 = 3$. Thus, we obtain the additional information that block seven is located on stack three.

Apart from having a compact and elegant representation of the stacking area, the generated binary matrix allows to draw a number of relevant information immediately, e.g.,

- which blocks are on top of a stack and can be moved,
- which blocks are already cleared (i.e., moved from the bay or retrieved to the vessel),
- whether two blocks are in the same stack and which one is located above the other,
- how many deadlocks we have, and
- stack and tier number of a block.

In particular, A fulfills the following properties:

1. A *zero row* i (i.e., $a_{ij} = 0 \; \forall \; j$) indicates that i is the uppermost block in its stack and therefore it can be either retrieved or relocated. Moreover, i is free to stack another block onto it.
2. A *zero column* j (i.e., $a_{ij} = 0 \; \forall \; i$) indicates that "nothing is below j." There are only two cases, where zero columns appear:
   - A block k already retrieved to the vessel is indicated by a zero column.
   - Columns $N+1, \ldots, N'$ corresponding to the artificial blocks are zero columns.
3. The stacking area is cleared (all blocks are retrieved to the vessel) if A is a *zero matrix* (i.e., $a_{ij} = 0 \; \forall \; i, j$)
4. Given two non-artificial blocks $j, k \in \{1, \ldots, N\}$ that are located in the stacking area (i.e., columns j and k contain at least one non-zero element each). If column j *dominates* column k (i.e., $a_{ij} \geq a_{ik} \; \forall \; i$) then block j is located somewhere above block k (i.e., $a_{kj} = 1$).
5. Deadlocks are displayed by entries in the upper triangular of the matrix A that equal one. Thus, the number of deadlocks is given by

$$D(A) = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} a_{ij} \; .$$

   Note that columns $N+1, \ldots, N'$ are zero columns anyway and need not be taken into the sum. Furthermore, $D(A)$ does not provide the number of *forced relocations* as one relocation might resolve more than one deadlock.
6. The diagonal of A contains only zero elements.
7. Considering block k, let $s(k)$ and $t(k)$ denote stack and tier number, respectively. We have:

$$s(k) = -N + \sum_{i=N+1}^{N'} i \cdot a_{ik} \qquad \text{and} \qquad t(k) = \sum_{i=1}^{N'} a_{ik} \; .$$

8. Each stacking area can be transformed to a $(N') \times (N')$-binary matrix following formulation (1) but not vice versa.

Once we have a representation of the stacking area, we need to define retrieval and relocation activities. To maintain feasibility of a stacking area, particular conditions have to be satisfied before these operations can be carried out. We define the preconditions together with the transformations to matrix A next.

Relocation of block p directly on top of block k.

Precondition: Both blocks p and k have to be the highest block in their stack, i.e., $a_{pj} = 0$ and $a_{kj} = 0 \; \forall \; j = 1, \ldots, N'$ has to hold.
Optional: If the height of the stacking area is bounded by H, the following condition has to be satisfied in addition: There are at most $H - 2$ blocks below k, i.e., $\sum_{i=1}^{N} a_{ik} \leq H - 2$. Note that only rows $1, \ldots, N$ are considered in the sum, as artificial blocks do not count concerning height.

Transformation: Only column p is going to be changed - everything which has been below k is now also below p and, in addition, k itself is below p. We obtain the transformed matrix A' by

$$a'_{ij} = \begin{cases} a_{ik} & \text{for } j = p, i \neq k \\ 1 & \text{for } j = p, i = k \\ a_{ij} & \text{for } j \neq p, i = 1, \ldots, N' \end{cases} \tag{2}$$

It is easy to see that the computational complexity of the operation "relocation" is $\mathcal{O}(N')$.

Retrieve block p from the stacking area (to the vessel).

Precondition: We have three preconditions: *First*, block p has to be within the stacking area, i.e., there has to be a non-zero element in column p. *Second*, block $p - 1$ has to be already retrieved (zero column $p - 1$). *Third*, block p has to be the highest block in its stack (zero row p). Consequently, the following three requirements have to be satisfied:

(i) $\sum_{i=1}^{N'} a_{ip} > 0$,
(ii) $a_{ip-1} = 0 \; \forall \; i = 1, \ldots, N'$,
(iii) $a_{pj} = 0 \; \forall \; j = 1, \ldots, N'$.

Transformation: Only column p is changed, namely, we set each element to zero. We obtain the transformed matrix A' by

$$a'_{ij} = \begin{cases} 0 & \text{for } j = p, i = 1, \ldots, N' \\ a_{ij} & \text{for } j \neq p, i = 1, \ldots, N' \end{cases} \tag{3}$$

The computational complexity of the operation "retrieval" is $\mathcal{O}(N')$.

After having an encoding available as well as a set of rules coordinating relocation and retrieving operations, we conclude by defining the following.

Definition 1 (BRP in terms of binary encoding). *Given an initial matrix A, find a sequence of feasible relocation and retrieving activities that terminates with a zero matrix A' such that the number of relocations is minimized.*

3 A Simple Heuristic Approach

When implementing our approach for the BRP, we take advantage of some simple heuristic rules (proposed by [3]) that will guide our search procedure. Thus, before describing the details of the procedure in the subsequent section, we will have a brief look into the adaption of the heuristic rules to the proposed binary encoding in matrix A.

Given a stack s, denote the smallest block number within s by min_s. For empty stacks, set $min_s = N + 1$. In short, the heuristic rule is the following (see [3] for details):

- Relocate only blocks located above the target block (the block that has to be retrieved next)
- When a block r has to be relocated, choose the new stack by applying the following rules:
  1. (*Relocation without new deadlock*) Choose among those stacks, where $r < min_s$ is satisfied, a stack that minimizes min_s.
  2. (*Relocation causes new deadlock*) If there is no stack satisfying $r < min_s$ and no empty stack then choose a stack that maximizes min_s.

To implement the heuristic rules, the following information about the current state of the stacking area is required and can be taken easily from the matrix A:

1. *Identify stack number of block n:* Check column n for the unique positive entry in rows $N+1, \ldots, N'$. Stack number is row number minus N. $(\mathcal{O}(W))$

2. *Identify blocks located above block n:* Check row n for positive entries (only elements regarding columns $1, \ldots, N$). $(\mathcal{O}(N))$

3. *Identify set of empty stacks:* Find zero rows among rows $N + 1, \ldots, N'$. From each row number, subtract N to obtain the number of the empty stack. $(\mathcal{O}(WN))$

4. *Identify the highest block in stack s:* Identify blocks located above block n (see Item 2).
   From the set of associated rows, find the unique zero row. $(\mathcal{O}(N^2))$

5. *Identify min_s for a given stack s:* Check entries of row $N + s$. The column number of the first positive entry provides min_s. Stack s is empty if there is no positive entry. $(\mathcal{O}(W))$

The overall computational complexity of the heuristic approach is $\mathcal{O}(N^2WH)$.

4 The Algorithm

As presented in Section 2, matrix A provides a complete representation of a bay. In the following, let us indicate with A^k the matrix associated with the bay

after k iterations. Therefore, A^0 corresponds to the initial bay. Consequently, the solution to the BRP can be seen as a sequence of matrices $A^0, A^1, \ldots, A^K$, where A^k is obtained from A^{k-1} through the application of a valid move and A^K is a zero matrix.

In the following, let us indicate with t^k the target block, *i.e.*, the block with highest priority within A^k, and with s^k the stack upon which the target is currently lying. In addition, let us indicate with n_i the number of blocks located above block i at any point in time. It is worth noting that the values $n_i, i = 1, \ldots, N'$, are initially computed in $\mathcal{O}(N'^2)$ and subsequently updated at each iteration in $\mathcal{O}(N')$. Finally, let us indicate with j^k the uppermost element in the stack containing the target at iteration k, *i.e.*, stack s^k. It is easy to see that such block can be found in $\mathcal{O}(N)$ by using the information provided by n_i. Note that for clarity one might refer to these values by indicating their specific iteration k, i.e., by using values n_i^k.

The proposed algorithm consists of four basic steps, iteratively repeated until all blocks have been retrieved from the bay, *i.e.*, until we obtain a zero matrix A^K. The four-step algorithm can be summarized as follows (for the sake of readability, we omit index k):

Target Identification: Block t with lowest priority in the bay is identified, along with its stack s. Since all the information related to block t is contained in row t and column t of the current matrix A, we sequentially scan the matrix from row 1 to row N. Therefore, finding the target is carried out in at most $\mathcal{O}(N)$.

Choice Design: We first need to identify the uppermost element in s, *i.e.*, j. If $j = t$, we retrieve the target (as presented in Section 2, this can be done in $\mathcal{O}(N')$) and we go back to the *target identification* phase. Otherwise, we define a neighborhood of the current configuration $\mathcal{N}(A)$ made up by all the matrices that can be reached from A through the application of a valid move. Note that this refers to using the concept of the pilot method with evaluation depth one. Given the current target t located onto stack s, and the current block to be relocated j, we can create at most $W - 1$ different matrices by relocating block j onto any other stack, excluding s. The actual cardinality of neighborhood $\mathcal{N}(A)$ could be smaller than $W-1$ if, *e.g.*, height limits are imposed.

Let us indicate with S the set of available stacks, *i.e.*, stacks onto which it is possible to relocate block j. We define a valid move as a transformation function that operates on the current configuration A, in such a way that $m : A \rightarrow A'$, where A' is the configuration obtained after relocating block j from stack s to stack $s' \in S$, *i.e.*, $A' = m(A, j, s')$. Therefore, the current neighborhood can be defined as:

$$\mathcal{N}(A) = \{A' : A' = m(A, j, s'),\ s' \in S\}$$

Move Evaluation and Selection: We define a greedy score $g(A')$ for each feasible configuration $A' \in \mathcal{N}(A)$. Such score is computed by applying upon

A' the greedy heuristic described in Section 3. The value returned by the heuristic, *i.e.*, the number of further relocations required to clear up the current bay A', is taken as score. Once such scores have been computed, we apply a "roulette-wheel" mechanism to randomly select the next configuration A' and its associated move. Such mechanism allows for different solutions to be selected at each iteration, while still preserving a measure of attractiveness of each selection proportional to the greedy score.

Trajectory fathoming: Given the best upper bound, we apply a simple logical test to detect whether the current trajectory is dominated by a previously found feasible solution. In such case, the current trajectory can be fathomed and the algorithm is restarted. Otherwise, if the logical test fails, the next iteration of the algorithm is performed.

Algorithm 1. `Matrix-Algorithm()`

Require: initial configuration A^0
Ensure: number of relocations z^*, set of relocations
1. $z^* \leftarrow \infty$
2. compute $n_i,\ i = 1, \ldots N$ {`counter no. blocks above` i - $\mathcal{O}(N'^2)$}
3. **while** `stopping_criterion()` is not reached **do**
4. $k \leftarrow 0$ {`relocations counter`}
5. **for** $t = 1, \ldots, N$ **do**
6. **while** $n_t > 0$ **do**
7. identify the uppermost element j $\{\mathcal{O}(N)\}$
8. define $\mathcal{N}(A)$ $\{O(WN')\}$
9. **for all** $A' \in \mathcal{N}(A)$ **do**
10. compute $g(A')$ {`greedy score` – $\mathcal{O}(N^2WH)$}
11. **end for**
12. select a move and define $A^{k+1} = m(A_k, j, s')$ {`roulette-wheel`}
13. $k \leftarrow k + 1$
14. **for all** blocks i in stacks s and s' **do**
15. update n_i $\{\mathcal{O}(N')\}$
16. **end for**
17. apply logical test trajectory fathoming $\{\mathcal{O}(1)\}$
18. **end while**
19. retrieve target block t from A^k $\{\mathcal{O}(N')\}$
20. **end for**
21. **if** $k < z^*$ **then**
22. $z^* \leftarrow k$ and save best trajectory
23. **end if**
24. **end while**

A pseudo-code of the algorithm is presented in `Matrix-Algorithm()`. In the following, the four-phase algorithm is repeated until a complete trajectory is built, *i.e.*, until all blocks of the bay have been retrieved in the predefined order. In turn, the trajectory construction scheme is iteratively repeated until a stopping criterion is reached.

5 Computational Results

In this section we present computational results on randomly generated instances. All tests presented in this section have been carried out on a Pentium IV Linux Workstation with 512Mb of RAM. The algorithm has been coded in C++ and compiled with the GNU C++ compiler using the -O option.

We designed an experiment that uses the same instances presented in [4]. In this section, we compare the results of the proposed algorithm with those of [6] (heuristic based on expected value of future relocations) and of [4] (metaheuristic corridor method). We focus our attention on tests on "realistic" instances, whose size matches the typical size of a bay found at container terminals. The optimal solution of these instances is currently unknown. The random generation process of the instances takes as input two parameters, the number of stacks W and the number of tiers H, and randomly generates a rectangular bay configuration of size $N = H \times W$, where N indicates the total number of blocks in the bay. For each combination of W and H we generated 40 different instances.[1]

Table 1. Computational results. Each row shows average values over 40 runs.

Bay Size		*KH*		*CM*		Matrix-Algorithm	
H	*W*	No.	Time	No.	Time	No.	Time
6	6	37.3	0.1	32.4	7.94	31.8	0.38
6	10	75.1	0.1	49.5	15.72	47.6	0.65
10	6	141.6	0.1	102.0	30.13	82.9	0.93
10	10	178.6	0.1	128.3	65.42	121.3	1.57

In Table 1, we compare the results of the proposed scheme with those obtained running the codes of [6], *KH* for short, and the corridor method (*CM*) of [4] on the same set of instances. It is worth noting that all values reported in the table are *average* values, computed over 40 different instances of the same class. This helps in offsetting instance specific biases in the reported results.

In Table 1, the first two columns define the instance size, in terms of number of tiers H and number of stacks W. Columns three and four report the results, in terms of number of relocations and computational time, required by the heuristic of [6]. Similarly, columns five and six report results of [4] on the same instances, both in terms of relocation moves and computational time (in CPU seconds), while columns seven and eight summarize the results of our proposed algorithm.

The computational results illustrate two issues. First, it is observed that the running times of `Matrix-Algorithm()` are considerably shorter compared to those of the *CM*, the only other metaheuristic in Table 1. This speed-up is due to the fast access to information taken from the stacking area which is enabled by the presented binary encoding. Moreover, by implementing the search procedure

[1] The code and all the instances used during the experiment can be obtained from the authors upon request.

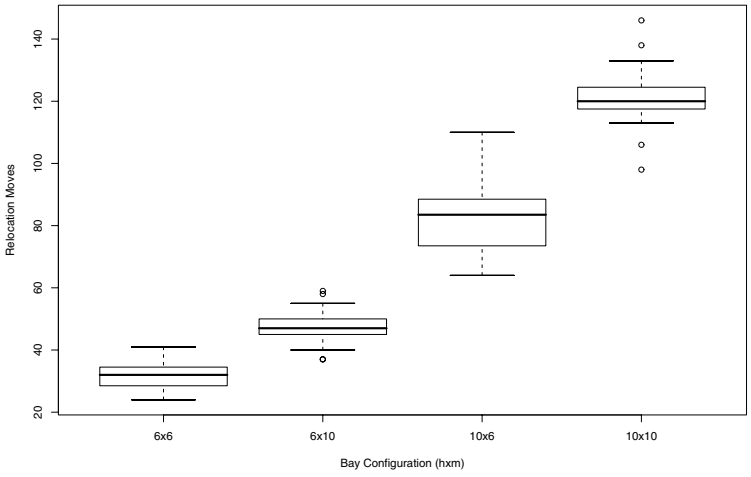


Fig. 2. Variability of results over 40 runs for each instance class

given in `Matrix-Algorithm()`, we are able to improve the solution quality, i.e., decrease the number of relocations, with respect to the *KH* and *CM* approaches.

In order to further illustrate the robustness of the proposed algorithm, in Figure 2 we graphically present the variability of the results on the instances. As shown in Figure 2, the algorithm is quite robust with respect to the initial configuration of the bay.

6 Conclusion

We have presented a new binary encoding for stacking areas where homogeneous blocks are stored on stacks. Such a stacking area has been described in the blocks relocation problem. This new representation allows fast access to information related to the current status of the stacking area as well as fast transformation into neighboring solutions. Thus, it is suited as a base for developing metaheuristic search strategies. We have implemented one approach by developing a randomly (roulette-wheel) guided look ahead mechanism based upon the idea of the pilot metaheuristic by taking advantage of simple heuristic rules to compute scores of intermediate solutions. Our experimental results illustrate that we are able to outperform current results from literature in terms of computational time and solution quality.

For future research it seems worthwhile to adapt the presented concepts to pre-marshalling problems. For instance, it seems straight forward to adapt the presented binary encoding to the pre-marshalling problem (see, e.g., [9]). Pre-marshalling is related to blocks relocation with the main difference that no blocks are retrieved during the relocation process. In other words, relocation and retrieval phases are separated in pre-marshalling. Thus, all deadlocks have to be

resolved before the retrieval process starts. We can exploit property number 5 of matrix A and obtain the following definition.

Definition 2 (Pre-marshalling problem in terms of binary encoding). *Given an initial matrix A, find a sequence of feasible relocation activities that terminates with a matrix A' that contains only zeros in the upper triangular such that the number of relocations is minimized.*

Moreover, it would be of interest to develop a mathematical programming formulation based on the binary encoding extending ideas from [3] and to investigate its suitability to solve the blocks relocation and the pre-marshalling problem and compare the results with those obtained in this paper.

References

1. Avriel, M., Penn, M., Shpirer, N.: Container ship stowage problem: complexity and connection to the coloring of circle graphs. Discrete Applied Mathematics 103, 271–279 (2000)
2. Avriel, M., Penn, M., Shpirer, N., Witteboon, S.: Stowage planning for container ships to reduce the number of shifts. Annals of Operations Research 76, 55–71 (1998)
3. Caserta, M., Schwarze, S., Voß, S.: A mathematical formulation for the blocks relocation problem. Working Paper, Institute of Information Systems, University of Hamburg (2008)
4. Caserta, M., Voß, S., Sniedovich, M.: An algorithm for the blocks relocation problem. Working Paper, Institute of Information Systems, University of Hamburg (2008)
5. Gupta, N., Nau, D.S.: On the complexity of blocks-world planning. Artifical Intelligence 56(2-3), 223–254 (1992)
6. Kim, K.H., Hong, G.P.: A heuristic rule for relocating blocks. Computers & Operations Research 33, 940–954 (2006)
7. Kim, K.H., Park, Y.M., Ryu, K.R.: Deriving decision rules to locate export containers in container yards. European Journal of Operational Research 124, 89–101 (2000)
8. Lee, Y., Chao, S.-L.: A neighborhood search heuristic for pre-marshalling export containers. European Journal of Operational Research 196, 468–475 (2009)
9. Lee, Y., Hsu, N.Y.: An optimization model for the container pre-marshalling problem. Computers & Operations Research 34, 3295–3313 (2007)
10. Stahlbock, R., Voß, S.: Operations research at container terminals: a literature update. OR Spectrum 30, 1–52 (2008)
11. Steenken, D., Voß, S., Stahlbock, R.: Container terminal operations and operations research- a classification and literature review. OR Spectrum 26, 3–49 (2004)
12. Voß, S., Fink, A., Duin, C.: Looking ahead with the pilot method. Annals of Operations Research 136, 285–302 (2005)
13. Yang, J.H., Kim, K.H.: A grouped storage method for minimizing relocations in block stacking systems. Journal of Intelligent Manufacturing 17, 453–463 (2006)

A Plasmid Based Transgenetic Algorithm for the Biobjective Minimum Spanning Tree Problem

Sílvia M. D. Monteiro, Elizabeth F. G. Goldbarg, and Marco C. Goldbarg

Department of Informatics and Applied Mathematics, Universidade Federal do Rio Grande do Norte, Campus Universitário Lagoa Nova, Natal, Brazil
silvinha_treze@yahoo.com.br, {beth,gold}@dimap.ufrn.br

Abstract. This paper addresses the application of a plasmid based transgenetic algorithm to the biobjective spanning tree problem, an NP-hard problem with several applications in network design. The proposed evolutionary algorithm is inspired on two major evolutionary forces: the horizontal gene transfer and the endosymbiosis. The computational experiments compare the proposed approach to another transgenetic algorithm and to a GRASP algorithm proposed recently for the investigated problem. The comparison of the algorithms is done with basis on the binary additive ε-indicator. The results show that the proposed algorithm consistently produces better solutions than the other methods.

Keywords: Biobjective minimum spanning tree, plasmid, transgenetic algorithm.

1 Introduction

A spanning tree of a connected undirected graph $G = (N, E)$ is an acyclic subgraph of G with n - 1 edges, where $n = |N|$. If G is a weighted graph, a minimum spanning tree, MST, of G is spanning tree for which the sum of the weights of its edges is minimum over all spanning trees of G. The MST is a well known combinatorial optimization problem with applications in distinct areas such as, networks design and clustering. Its theoretical importance comes from the fact that it may be utilized in approximation algorithms for other combinatorial optimization problems such as the Traveling Salesman [10] and the Steiner Tree [11]. The MST is solvable in polynomial time and a survey containing several algorithms for this problem is presented by Bazlamaçci and Hindi [3]. Although the problem is polynomial, constraints often render it NP-hard [7]. Examples include the degree-constrained minimum spanning tree problem, the maximum-leaf spanning tree problem, and the shortest-total-path-length spanning tree problem. Another difficult variant is the multi-criteria minimum spanning tree problem, *mc-MST*. Aggarwal et al. [1] showed that the 0-1 knapsack problem can be reduced by a polynomial function to the biobjective spanning tree, thus proving that the latter problem is NP-hard. As its counterpart with one objective the mc-MST arises in many real world applications, such as in network design where the edge weights can be associated, for instance, to reliability restrictions and installation costs.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 49–60, 2009.

Bio-inspired algorithms have shown to be powerful tools to deal with multi-objective problems. In this paper an evolutionary algorithm based on a natural vehicle of horizontal gene transfer is applied to the biobjective minimum spanning tree.

The inspiration for the proposed heuristics comes from two major evolutionary forces: the horizontal gene transfer and the endosymbiosis [9]. Horizontal or lateral gene transfer refers to the acquisition of foreign genes by organisms. It occurs extensively among prokaryotes (living forms whose cells do not have nucleus) and provides organisms with access to genes in addition to those that can be inherited [12]. The term "endosymbiosis" specifies the relationship between organisms which live one within another (symbiont within host) in a mutually beneficial relationship. The Serial Endosymbiotic Theory proposed by Margulis [15] states that a new organism can emerge from the fusion of two or more independent beings. Today, researchers recognize the horizontal transfer of functional genes between organisms as a determinant factor of the endosymbiotic origin of specialized cellular parts called organelles [16]. A known vehicle of horizontal gene transfer that is of special interest for this paper is the plasmid. Plasmids, in nature, are mobile genetic particles that can replicate independently of the chromosome. They are composed of DNA and can be thought as mini-chromosomes. By means of genetic engineering, scientists are able to construct plasmids with artificially created DNA. Artificial DNA is called recombinant DNA. The plasmids formed with recombinant DNA are often referred as recombinant plasmids.

In the Transgenetic Algorithms, where a co-evolutionary process is thought to occur between a host cell and its endosymbionts, three contexts of information are considered: the endosymbionts, the host and the transgenetic vectors. The endosymbionts, also called endosymbiont chromosomes or, simply, chromosomes, are the base of the search since they encode problem solutions. Unlike other evolutionary approaches, chromosomes do not share genetic material directly by means of crossover or recombination. Information about the problem being tackled by the algorithm (*a priori* information) and information about the heuristic search (*a posteriori* information) are stored in the host's context. Agents, called transgenetic vectors, are responsible for the exchanging of information between the host and the endosymbionts. These agents are inspired on natural mechanisms of horizontal gene transfer, such as the plasmids. The transgenetic vectors manipulate the chromosomes, modifying their codes and promoting the random variation that is necessary for the exploration and exploitation of the search space.

A computational experiment shows that the proposed algorithm obtains high quality results for the biobjective spanning tree problem. The results are compared with recent approaches presented for the investigated problem.

The paper is organized as follows. Section 2 presents the multi-objective minimum spanning tree problem. Transgenetic algorithms are discussed in section 3. The algorithm proposed for the investigated problem is described in section 4. Section 5 presents the computational experiments methodology and results. Finally, section 6 points out some conclusions.

2 Multi-objective Minimum Spanning Tree Problem

The general multi-objective minimization problem (with no restrictions) can be stated as:

$$\text{“minimize”}\ f(x) = (f(x_1), \ldots, f(x_k)), \text{ subjected to } x \in X \tag{1}$$

where x is a discrete value solution vector and X is a finite set of feasible solutions. Function $f(x)$ maps the set of feasible solutions X in $\Re^k$, $k > 1$ being the number of objectives. Once, usually, there is not only a single solution for the problem, the word minimize has to be understood in another context. Let $x,y \in X$, then x dominates y, written $x \succ y$, if and only if $\forall i,\ i=1,...k$, $f(x_i) \leq f(y_i)$ and $\exists i$, such that $f(x_i) < f(y_i)$. The set of optimal solutions $X^* \subseteq X$ is called Pareto optimal. A solution $x^* \in X^*$ if there is no $x \in X$ such that $x \succ x^*$. The nondominated solutions are said also to be efficient solutions. Thus, to solve a multi-criteria problem, one is required to find the set of efficient solutions. Solutions of this set can be divided in two classes: the supported and nonsupported efficient solutions. The supported efficient solutions can be obtained by solving the minimization problem with a weighted sum of the objectives. More formally [5],

$$\text{minimize} \sum_{i=1,\ldots,k} \alpha_i f(x_i) \tag{2}$$

$$\text{where} \sum_{i=1,\ldots,k} \alpha_i = 1,\ \alpha_i > 0,\ i = 1,\ldots,k \tag{3}$$

The nonsupported efficient solutions are those which are not optimal for any weighted sum of objectives. This set of solutions is a major challenge for researchers that deal with the *mc-MST*.

Given a graph $G = (N,E)$, a vector of non negative weights $w_{ij} = (w_{ij}^1,..., w_{ij}^k)$, $k > 1$, is assigned to each edge $(i,j) \in E$. Let S be the set of all possible spanning trees, $T = (N_T,E_T)$, of G and $W = (W^1,..., W^k)$, where

$$W^q = \sum_{(i,j) \in E_T} w_{ij}^q,\ q=1,..,k. \tag{4}$$

The problem seeks $S^* \subseteq S$, such that $T^* \in S^*$ if and only if $\nexists\, T \in S$, such that $T \succ T^*$. In this work the biobjective problem is considered, although the proposed algorithm can be adapted to consider $k > 2$ objectives.

Exact algorithms based on the branch-and-bound technique were proposed by Ramos et al. [18] and Sourd et al. [21] for the biobjective spanning tree. The largest instances solved by these algorithms are 20 [18] and 150 [21]. A k-best exact algorithm is presented by Steiner and Radzik [22] who apply their method to instances up to 80 nodes.

Given the problem difficulty and high applicability, a number of works have been dedicated to this problem [14], [18], [22], [23]. Among the heuristic approaches, evolutionary algorithms were introduced by Gen et al. [8], Zhou and Gen [23], Knowles [13] and Rocha et al. [19, 20], and a GRASP algorithm was presented by Arroyo et al. [2].

3 Transgenetic Algorithm

Transgenetic algorithms are evolutionary computing techniques based on living processes where cooperation is the main evolutionary strategy [9]. Those biological processes contain the movement of genetic material between organisms and endosymbiotic interactions. The Transgenetic Algorithms accomplish the search on the space of solutions of optimization problems by means of a computational context that is inspired in the information sharing between a host cell and a population of endosymbionts. Three contexts of information are considered in Transgenetic Algorithms:

- Solutions that are represented in chromosomes, also called endosymbiont chromosomes;
- Information about the problem being tackled and information about the evolutionary search performed by the algorithm, called host's information;
- Entities that are used to modify the candidate solutions, called transgenetic vectors.

The individuals of the population of candidate solutions are thought as endosymbionts within a host cell. They constitute the population of endosymbiont chromosomes.

The information about the problem and the evolutionary search is thought as the host's genetic information. The information is said to be *a priori*, if it exists independent of the search being performed by the algorithm. This type of information is, usually, inherent to the problem or the instance being tackled, such as upper or lower bounds, heuristic solutions and results of statistical analysis of the problem structure, among others. *A posteriori* information is obtained during the algorithm execution and is related to the search. For example, new best solutions or partial solutions may arise from the population of chromosomes. *A priori* and *a posteriori* information are stored in a repository, called host's repository. They are used by the transgenetic vectors to modify the solutions in the population. These vectors are inspired in the natural horizontal gene transfer mechanisms and are named accordingly. Their function is to modify the solutions represented in the endosymbiont chromosomes, generating the variation necessary to accomplish the search. The transgenetic vectors are the unique tools to perform intensification and diversification tasks in the Transgenetic Algorithms.

The transgenetic vectors implemented in the algorithm presented in this paper are the plasmids and the recombinant plasmids. They are composed of an information string and a method to manipulate the chromosomes. The information string of these vectors is a sequence of genes that represents a partial solution. Plasmids and recombinant plasmids differ in the way the information strings are obtained. Plasmids obtain their information string from one source of the host's repository. The information string of recombinant plasmids can be obtained from two or more sources of the host's repository, it can also be generated by a heuristic or exact method and, finally, the information string can be obtained by mixing information obtained in the host's repository with information generated with some method. In this paper, the information string of the recombinant plasmid is generated with a heuristics.

The other component of the transgenetic vectors is the method they utilize to manipulate the chromosomes. The method of the plasmids, recombinant or not, is composed of two procedures: attack (p_1) and transcription (p_2). Procedure p_1 implements a criterion that establishes whether a given chromosome is susceptible or not to

be manipulated by a given vector. Procedure p_2 defines how the information string is inserted into the chromosome. The number of genes in the plasmid's information string is said to be the length of the plasmid.

Algorithm TA presents a general framework of the Transgenetic Algorithms. An initial population of chromosomes is created and evaluated. The host's repository is initialized with *a priori* information and, if some interesting information exists in the population, then that information is also included in the host's repository. Steps 3 to 8 are repeated while a stop condition is not met. In step 4 the transgenetic vectors are generated and in step 5 a set of chromosomes is chosen to be manipulated. If the action of the transgenetic vectors on the chromosomes produces new information, then the host's repository is updated in step 7.

```
Algorithm TA
1 Generate and evaluate a population of chromosomes
2 Initialize the host's repository (HR)
3 Repeat
4    Generate transgenetic vectors
5    Select chromosomes for manipulation
6    Manipulate the selected chromosomes
7    Update HR
8 until a stopping criterion is satisfied
```

4 Transgenetic Algorithm for the mc-MST

Algorithm Plas-TA presents a pseudo-code of the transgenetic algorithm implemented for the biobjective spanning tree problem. At first, *#popSize* chromosomes are generated. The spanning trees are represented in the chromosomes with the correspondent set of edges that compose the tree [17]. The algorithm executes a fixed number of iterations, *#numGer*. Two methods are used to generate the chromosomes of the initial population. The first method is a version of the Kruskal's algorithm for the biobjective spanning tree with the inclusion of random choices of the edges that compose the tree [19]. This method is called *rmc-Kruskal*. At each constructive step, one edge is randomly chosen from a restricted list of candidates (RLC) as in a GRASP algorithm [6]. The RLC is composed of edges whose costs are, at most, *#tolPer* percent greater than the minimum cost of the edges out of the tree. In the experiments, the value adopted for *#tolPer* was 5%. The costs of the edges are calculated by equation 2. If two identical trees are generated by this algorithm, only one of them is maintained in the population. This method is executed *#popSize* times. If less than *#popSize* distinct individuals are generated, then the population is completed with chromosomes generated with the RandomWalk method [17].

```
Algorithm Plas-TA
1 generate_initial_population(P={C_1,...,C_#popSize})
2 create_archive(G_A,P)
3 for i ← 1 to #numGer
4    create_set_of_plasmids(#plasNum)
5    for j ← 1 to #popSize
6       t ← random(1,#plasNum)
```

```
7        C_result ← plasmid(C_j, t)
8        if (better (C_result,C_j))
9           C_j ← C_result
10       else ctr[j] ← ctr[j] + 1
11       if (ctr[j] = 2)
12          C_result ← recombPlasmid(C_j)
13          ctr[j] ← 0
14          if (better(C_result,C_j))
15             C_j ← C_result
16       end_if_line_11
17       update(G_A,C_j)
18    end_for_j
19 end_for_i
```

The algorithm maintains an archive of nondominated solutions, *G_A*, that is limited to 300 solutions and is initialized with the nondominated solutions of the initial population. The objective space of the solutions in *G_A* is represented by a multidimensional grid structure divided in cells [13]. A new solution is added to this archive if it is not dominated by any solution in *G_A*. Solutions in *G_A* that are dominated by the new solution are discarded. If the storage limit is violated, then a solution in the most populous cell of the grid is randomly selected and withdrawn.

The proposed algorithm uses one plasmid and two recombinant plasmids. The *G_A* is the source of information for the plasmids. At each iteration step one solution of a cell with few solutions is randomly chosen to be the source of information for the plasmids of that iteration. Then, *k* randomly chosen edges of the tree compose the information string of the plasmid. The information string of the first recombinant plasmid is obtained from solutions generated with the rmc-Kruskal. A spanning tree is created with this method and a fragment of it is randomly chosen to be the information string of the first recombinant plasmid. The length of the information string of the plasmid and the first type of recombinant plasmid is chosen at random in the interval $[0.30n,0.60n]$ and both transgenetic vectors use the same manipulation method. Given the edges of the information string, the procedure p_2 builds a tree with these edges and the edges of the original solution that do not induce a cycle. If necessary, random edges are added to the tree until a spanning tree is formed.

The information string of the second type recombinant plasmid is built with the mcPrim method [14]. When this transgenetic vector attacks an individual, a random scalarizing vector is applied to evaluate the total cost of the tree. Only a small fragment of the original tree, with length in the interval $[0.05n,0.1n]$, is maintained. This fragment is randomly chosen such that the smaller the cost of an edge, the bigger its probability to remain in the tree. The Prim´s method, based on the scalarized costs, chooses the edges to build the remaining of the tree. The edges added to the tree by the mcPrim constitute the genetic information string of the second type recombinant plasmid.

The same procedure p_1 is used to verify if a manipulation is accepted or not for the three transgenetic vectors of the proposed algorithm. It is implemented in procedure *better*( ). The input data of this procedure are C_j, j=1,…,#*popSize*¸ the original chromosome and C_{result}, the chromosome that results from the manipulation of C_j. Procedure *better* (C_{result}, C_j) states that C_{result} replaces C_j in the population, if C_{result} dominates C_j or if C_{result} is nondominated considering all solutions in *G_A*.

At each iteration step, a set of transgenetic vectors is generated in step 4. This set contains *#plasNum*/2 plasmids and *#plasNum*/2 recombinant plasmids of the first type. One of these transgenetic vectors is randomly selected in step 6 to manipulate all chromosomes of the current population.

The algorithm maintains a failure counter for each individual of the population, *ctr*. The failure counter is initialized with 0. The counter of each individual is incremented whenever a manipulation by a plasmid or a first type recombinant plasmid does not result in the replacement of the original chromosome by the manipulate chromosome. When the counter of a given individual reaches 2, the individual is manipulated by the second type recombinant plasmid and its counter is set to 0 again.

5 Computational Experiments

A first experiment compared the performance of the proposed algorithm with another effective transgenetic algorithm presented previously [20]. Two main reasons contributed for choosing this algorithm to be compared with the proposed one. First, both are transgenetic algorithms and, second, the TA proposed by Rocha et al. [20] presented better results than the algorithm AESSEA presented by Knowles [14] and the Memetic Algorithm presented by Rocha et al. [19].

Table 1. Instance parameters

Id	Conc		Corr	Anticorr
	ζ	η	β	β
50	0.03	0.125	0.7	-0.7
100_1	0.01	0.02	0.3	-0.3
100_2	0.02	0.1	0.7	-0.7
200_1	0.05	0.2	0.3	-0.3
200_2	0.08	0.1	0.7	-0.7
300_1	0.03	0.1	0.3	-0.3
300_2	0.05	0.125	0.7	-0.7
400_1	0.025	0.125	0.3	-0.3
400_2	0.04	0.2	0.7	-0.7
500_1	0.02	0.1	0.3	-0.3
500_2	0.03	0.15	0.7	-0.7
600_1	0.0016	0.1	0.125	-0.125
600_2	0.002	0.02	0.95	-0.95
700_1	0.0014	0.03	0.35	-0.35
700_2	0.001	0.008	0.7	-0.7
800_1	0.00125	0.035	0.45	-0.45
800_2	0.0015	0.03	0.05	-0.05
900_1	0.0011	0.009	0.15	-0.15
900_2	0.002	0.01	0.85	-0.85
1000_1	0.001	0.2	0.4	-0.4
1000_2	0.0005	0.1	0.9	-0.9

Both algorithms were implemented in C++ and were executed in a Pentium 4, 3.2GHz processor with 1Gb RAM, using KUbuntu 7.10 and gcc compiler. A set of 63 instances were generated with the method proposed by Knowles [13] as complete graphs with 2 objectives. These instances are divided in three groups of 21 instances. Each group is composed by instances belonging, respectively, to the classes concave (conc), correlated (corr) and anti-correlated (anticorr). Each class contains one instance with 50 vertices and 10 pairs of instances with n in the interval [100, 1000]. To create correlated and anti-correlated instances, it's necessary to specify a correlation factor β. Two parameters, ζ and η, are used to generate concave instances. Table 1 shows the parameters used to create the set of instances.

For each instance, 30 independent runs were performed for both algorithms. The parameters of the proposed algorithm, referred as Plas-TA, are: #*numGer* = 30, #*popSize* = 150, #*popIni* = 142 and #*plasNum* = 10.

The binary additive ε-indicator, $I_{\varepsilon+}$, proposed by Zitzler et al. [24], was used to compare the performances of the transgenetic algorithms. Given two approximation sets, A and B, a value $I_{\varepsilon+}(A,B) < 0$ indicates that every solution of B is strictly dominated by at least one solution of A. Values $I_{\varepsilon+}(A,B) \leq 0$ and $I_{\varepsilon+}(B,A) > 0$ indicate that every solution of B is weakly dominated by at least one solution of A. Values $I_{\varepsilon+}(A,B) > 0$ and $I_{\varepsilon+}(B,A) > 0$ indicate that neither A weakly dominates B nor B weakly dominates A. The Mann-Withney (U-test) statistical test is used to verify the statistical significance of the results [4]. The p-values that resulted from the U-test are shown in table 2 for each instance of each class. The transgenetic algorithm of Rocha et al. [20] is referred as TA in table 2.

Table 2. p-values resultant from the comparison of Plas-TA with TA

Id	Conc		Corr		Anticorr	
	Plas-TA	**TA**	**Plas-TA**	**TA**	**Plas-TA**	**TA**
50	**0**	1	**0**	1	**0**	1
100_1	**0**	1	**0**	1	**0**	1
100_2	**0**	1	**0**	1	**0**	1
200_1	**0**	1	**0**	1	**0**	1
200_2	**0**	1	**0**	1	**0**	1
300_1	**0**	1	**0**	1	**0**	1
300_2	**0**	1	**0**	1	**0**	1
400_1	**0**	1	**0**	1	0.26	0.74
400_2	**0**	1	**0**	1	**0.03**	0.97
500_1	**0**	1	**0**	1	**0**	1
500_2	**0**	1	**0**	1	**0**	1
600_1	**0**	1	**0**	1	**0**	1
600_2	**0.01**	0.99	**0**	1	**0**	1
700_1	0.61	0.39	**0**	1	**0**	1
700_2	**0**	1	**0**	1	**0**	1
800_1	0.18	0.82	**0**	1	**0**	1
800_2	0.86	0.14	**0**	1	**0**	1
900_1	**0**	1	**0**	1	**0**	1
900_2	**0**	1	**0**	1	**0**	1
1000_1	0.24	0.76	**0**	1	**0**	1
1000_2	0.25	0.75	**0**	1	**0**	1

Considering 0.05 as the significance level of the statistical test, table 2 shows that Plas-TA outperforms TA in 16 concave instances, 21 correlated instances and 20 anti-correlated instances. The proposed algorithm is not outperformed by TA in any instance.

Table 3 exhibits the execution times, in seconds, obtained by both algorithms. According to the data presented, Plas-TA exhibits significantly better processing times than TA.

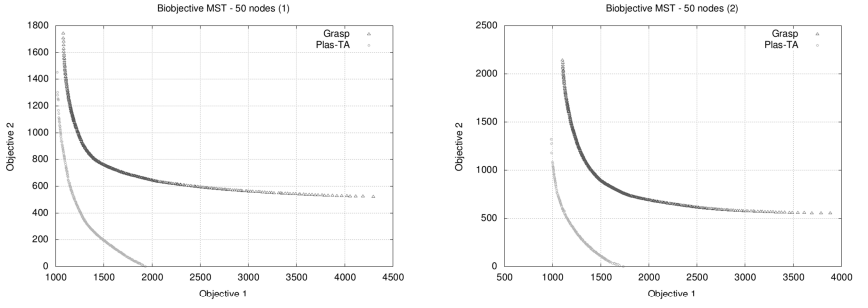
Table 3. Execution times

Id	Conc		Corr		Anticorr	
	Plas-TA	**TA**	**Plas-TA**	**TA**	**Plas-TA**	**TA**
50	0	4	0	3	0	12
100_1	1	6	1	7	2	22
100_2	1	7	1	6	2	28
200_1	5	19	6	25	6	60
200_2	5	15	6	23	6	76
300_1	13	39	13	48	14	93
300_2	13	34	13	47	14	125
400_1	25	64	25	74	26	134
400_2	25	67	24	75	25	169
500_1	40	98	41	109	41	178
500_2	40	94	40	98	41	213
600_1	59	137	59	138	59	190
600_2	59	134	54	133	58	251
700_1	84	185	84	184	84	291
700_2	84	181	81	180	82	297
800_1	109	236	108	235	107	354
800_2	110	235	110	230	110	264
900_1	141	293	140	298	140	362
900_2	140	289	133	290	135	399
1000_1	150	337	175	356	172	470
1000_2	172	354	165	342	169	456

A second computational experiment compared the performance of the proposed transgenetic algorithm with a GRASP algorithm, GRA, recently proposed in the literature [2]. The set of instances of this experiment was kindly provided by the GRASP's authors, as well as, the approximation sets obtained by their algorithm. The set contains 3 groups of 5 instances, with 20, 30 and 50 vertices. The edge costs of each group of instances are uniformly distributed in the intervals [30, 200], [20,100] and [10,50], respectively. The binary additive ε-indicator was used in this test. The results are organized in table 4, where Plas-TA/GRA corresponds to $I_{\varepsilon+}$(Plas-TA,GRA) and GRA/Plas-TA is equivalent to $I_{\varepsilon+}$(GRA,Plas-TA). According to this table, the solutions in the approximation sets obtained with the GRASP algorithm is strictly dominated by at least one solution of Plas-TA for all instances. Therefore, regarding this indicator, the solutions obtained by Plas-TA are better results than the ones obtained by the GRASP algorithm. These results are illustrated by the objective space of the approximation sets generated by both algorithms for four instances that are presented in figures 1 and 2.

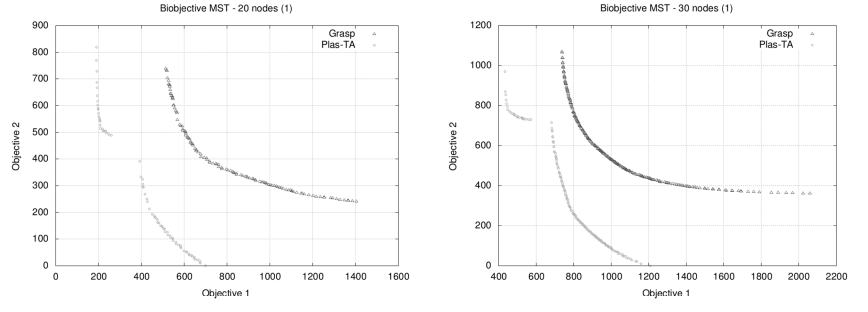
Table 4. Values obtained for the binary additive ε

Id	20 vertices		30 vertices		50 vertices	
	Plas-TA /GRA	GRA/ Plas-TA	Plas-TA /GRA	GRA/ Plas-TA	Plas-TA /GRA	GRA/ Plas-TA
1	-149	324	-99	379	-63	591
2	-133	364	-113	380	-118	632
3	-138	373	-99	369	-56	589
4	-117	386	-150	369	-88	604
5	-82	289	-140	395	-47	567



(a) (b)

Fig. 1. Objective space of the approximation sets obtained by Plas-TA and GRA for instances (a) 50.1 and (b) 50.2 with n=50



(a) (b)

Fig. 2. Objective space of the approximation sets obtained by Plas-TA and GRA for (a) instance 20.1 with n=20 and (b) instance 30.1 with n=30

6 Conclusions

This work presented a plasmid-based Transgenetic Algorithm for the biobjective spanning tree problem. The approximation sets generated with the proposed algorithm

are compared to the approximation sets generated by an efficient transgenetic algorithm proposed by Rocha et al. [20] and to a GRASP algorithm proposed by Arroyo et al. [2]. The comparisons are based on the binary additive epsilon quality indicator. Statistical tests show that the proposed algorithm is highly effective when compared with the other algorithms, outperforming both of them regarding the quality of the generated approximation sets. The proposed algorithm also presents better execution times than the TA.

Acknowledgments. We want to thank Arroyo, Vieira and Viana, who kindly provided us the instance set they used to test their GRASP algorithm and the approximation sets obtained. We are also thankful to ANP, Brazilian National Oil Agency, PRH-22 project, who partially supported this research.

References

1. Aggarwal, V., Aneja, Y., Nair, K.: Minimal spanning tree subject to a side constraint. Computers & Operations Research 9, 287–296 (1982)
2. Arroyo, J.E.C., Vieira, P.S., Vianna, D.S.: A GRASP Algorithm for the Multi-criteria Minimum Spanning Tree Problem. Annals of Operations Research 159, 125–133 (2008)
3. Bazlamaçci, C.F., Hindi, K.S.: Minimum-weight Spanning Tree Algorithms A Survey and Empirical Study. Computers and Operations Research 28, 767–785 (2001)
4. Conover, W.J.: Practical Nonparametric Statistics, 3rd edn. John Wiley & Sons, Chichester (2001)
5. Ehrgott, M., Gandibleux, X.: A Survey and Annotated Bibliography of Multiobjective Combinatorial Optimization. OR Spektrum 22, 425–460 (2000)
6. Feo, T.A., Resende, M.G.C.: Greedy Randomized Adaptive Search Procedures. Journal of Global Optimization 6, 109–133 (1995)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-completeness. Freeman, New York (1979)
8. Gen, M., Ida, K., Kim, J.R.: A Spanning Tree-Based Genetic Algorithm for Bicriteria Topological Network Design. In: Proceedings of 1998 IEEE International Conference on Evolutionary Computing, pp. 15–20 (1998)
9. Goldbarg, M.C., Bagi, L.B., Goldbarg, E.F.G.: Transgenetic algorithm for the traveling purchaser problem. European Journal of Operational Research (2008) (accepted)
10. Gutin, G., Punnen, A.P.: Traveling Salesman Problem and Its Variations. Kluwer Academic Publishers, Dordrecht (2002)
11. Hakami, S.L.: Steiner's Problem in Graphs and Its Implications. Networks 1, 113–133 (1971)
12. Jain, R., Rivera, M.C., Moore, J.E., Lake, J.A.: Horizontal Gene Transfer Accelerates Genome Innovation and Evolution. Molecular Biology and Evolution 20(10), 1598–1602 (2003)
13. Knowles, J.D.: Local-Search and Hybrid Evolutionary Algorithms for Pareto Optimization. Ph.D Thesis. Department of Computer Science, University of Reading, Reading, UK (2002)
14. Knowles, J.D., Corne, D.W.: A Comparison of Encodings and Algorithms for Multiobjective Spanning Tree Problems. In: Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001), pp. 544–551 (2001)

15. Margulis, L.: Symbiosis in Cell Evolution: Microbial Communities in the Archean and Proterozoic Eons. W.H. Freeman, New York (2002)
16. Pierce, S.K., Massey, S.E., Hanten, J.J., Curtis, N.E.: Horizontal Transfer of Functional Nuclear Genes Between Multicellular Organisms. The Biological Bulletin 204, 237–240 (2003)
17. Raidl, G.R.: An Efficient Evolutionary Algorithm for the Degree-constrained Minimum Spanning Tree Problem. In: Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000), pp. 104–111. IEEE Press, Los Alamitos (2000)
18. Ramos, R.M., Alonso, S., Sicília, J., González, C.: The Problem of the Optimal Biobjective Spanning Tree. European Journal of Operational Research 111, 617–628 (1998)
19. Rocha, D.A.M., Goldbarg, E.F.G., Goldbarg, M.C.: A Memetic Algorithm for the Biobjective Minimum Spanning Tree Problem. In: Gottlieb, J., Raidl, G.R. (eds.) EvoCOP 2006. LNCS, vol. 3906, pp. 183–194. Springer, Heidelberg (2006)
20. Rocha, D.A.M., Goldbarg, E.F.G., Goldbarg, M.C.: A New Evolutionary Algorithm for the Biobjective Minimum Spanning Tree. In: ISDA 2007 Seventh International Conference on Intelligent Systems Design and Applications, 2007. Proceedings of ISDA 2007, Rio de Janeiro, vol. 1, pp. 735–740. IEEE Computer Society, Danvers (2007)
21. Sourd, F., Spanjaard, O., Perny, P.: Multiobjective Branch and Bound. Application to the Biobjective Spanning Tree Problem. In: Proceedings of the 7th International Conference on Multi-Objective Programming and Goal Programming (2006)
22. Steiner, S., Radzik, T.: Solving the Biobjective Minimum Spanning Tree Problem using a k-best Algorithm. Technical Report TR-03-06, Department of Computer Science, King's College, London (2003)
23. Zhou, G., Gen, M.: Genetic Algorithm Approach on Multi-Criteria Minimum Spanning Tree Problem. European Journal of Operational Research 114, 141–152 (1999)
24. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., Fonseca, V.G.: Performance Assessment of Multiobjective Optimizers: An Analysis and Review. IEEE Transactions on Evolutionary Computation 7(2), 117–132 (2003)

A Tabu Search Algorithm with Direct Representation for Strip Packing

Jean-Philippe Hamiez*, Julien Robet, and Jin-Kao Hao

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

Abstract. This paper introduces a new tabu search algorithm for a two-dimensional (2D) Strip Packing Problem (2D-SPP). It integrates several key features: A *direct* representation of the problem, a *satisfaction-based* solving scheme, two different *complementary* neighborhoods, a *diversification* mechanism and a *particular* tabu structure. The representation allows inexpensive basic operations. The solving scheme considers the 2D-SPP as a succession of satisfaction problems. The goal of the combination of two neighborhoods is (to try) to reduce the height of the packing while avoiding solutions with (hard to fill) tall and thin wasted spaces. Diversification relies on a set of historically "interesting" packings. The tabu structure avoids visiting similar packings. To assess the proposed approach, experimental results are shown on a set of well-known benchmark instances and compared with previously reported tabu search algorithms as well as the best performing algorithms.

Keywords: Tabu search, strip packing, direct representation, multi-neighborhoods.

1 Introduction

Packing (and cutting) problems are optimization problems which are NP-hard in the general case. "Small" objects of various shapes (regular or not) and dimensions have to be packed *without overlap*, with rotation and "guillotine" cuts[1] allowed or not, into other larger objects. These larger objects are usually called "containers" for the 3D cases (all dimensions fixed or infinite height) and "bins" (all dimensions fixed) or "strips" (only width fixed, infinite height) in 2D. Objectives are, for instance, to minimize the number of containers and / or to maximize the material used (hence to minimize the wasted area). The most studied category of such problems seems to be in the 2D space.

This paper is dedicated to the 2D (non-guillotine and without rotation) Strip Packing Problem (2D-SPP) which can be informally stated as follows. Given a set of rectangular objects, pack them into a strip of an infinite height and fixed width while minimizing the height of the packing. 2D-SPP is a NP-hard combinatorial

* Contact author.

[1] The guillotine constraint imposes a sequence of *edge-to-edge* cuts.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 61–72, 2009.

optimization problem with a number of practical applications such as cardboard packing, glass and metal cutting or publicity scheduling for instance [1,2,3,4,5].

Given the NP-hard nature of 2D-SPP, many (meta)heuristic procedures have been tried: Greedy Randomized Adaptive Search Procedure (GRASP) [6], Intensification / Diversification Walk (IDW) [7], Simulated Annealing [8,9,10,11], Tabu Search (TS) [8,12,13], Genetic Algorithm [9,10,11,13,14,15,16,17,18,19], hybrid (meta)heuristic [11,13,20], HyperHeuristic [21]. Exact algorithms have also been considered but they are usually limited to "small instances" (up to 200 objects) [22,23,24,25]. Among these procedures, the approximate GRASP and IDW approaches [6,7] are probably the best performing ones.

In this paper, we introduce a new TS algorithm dedicated to the 2D-SPP (TSD for "Tabu Search with Direct representation"). Compared with previous algorithms for the 2D-SPP, our TSD has several notable features. First, it uses a *direct representation* of the problem (location of the objects on the strip) while many previous attempts manipulate permutations of the objects. Second, TSD treats the optimization problem (minimizing the height of the packing) as successive *satisfaction* problems: Starting from a packing s_0 (obtained with a greedy method e.g.) of height $H(s_0)$, TSD tries to solve the 2D-SPP with decreasing values of $H(s_0)$. Finally, our algorithm includes two different *complementary neighborhoods*, a *diversification* mechanism and handles a *particular tabu structure*. Preliminary computational results suggest that TSD may be of great interest to efficiently solve the 2D-SPP.

In the next section, the 2D-SPP is formally stated. Section 3 is devoted to the detailed presentation of our dedicated TS algorithm for the 2D-SPP. Experimental results are shown in Sect. 4 on a set of well-known benchmarks and compared with previous TS attempts and the best performing state-of-the-art algorithms. We finally discuss possible extensions in Sect. 5 before concluding.

2 Problem Formulation

Let P (for "Plane") be a 2D vertical space with fixed width W and infinite height. The bottom-left corner of P stands for the $(0,0)$ point of an xy-plane where the x-axis (respectively y-axis) is the direction of the width (resp. height) of P. The set of $n > 1$ objects (Rectangles) to be positioned in P is $R = \{r_1, \ldots, r_n\}$ where the $w$eight (resp. $h$eight) of each r_i $(1 \leq i \leq n)$ is $0 < w_i^r \leq W$ (resp. $h_i^r > 0$).

According to these notations, the 2D-SPP is then to determine the (x_i^r, y_i^r) coordinates of the bottom-left corner of all $r_i \in R$ (i.e. the location of each r_i in P) so as to minimize the $y_i^r + h_i^r$ value of the highest object in P, see (1). This can be formally stated as follows:

$$\text{Minimize: } H = \max_{1 \leq i \leq n} (y_i^r + h_i^r) \tag{1}$$

$$\text{subject to: } 0 \leq x_i^r \leq W - w_i^r \wedge y_i^r \geq 0 \tag{2}$$

$$\wedge \left(x_i^r \geq x_j^r + w_j^r \vee x_i^r + w_i^r \leq x_j^r \right. \tag{3}$$

$$\left. \vee\ y_i^r \geq y_j^r + h_j^r \vee y_i^r + h_i^r \leq y_j^r \right) \quad . \tag{4}$$

where (2) forces each r_i to be inside P and (3–4) specify that any two r_i and r_j objects ($i \neq j$) must not overlap horizontally and vertically, respectively.

3 TSD: A Tabu Search with Direct Representation

TS is an advanced local search method using general mechanisms and rules as guidelines for smart search [26]. In Sect. 3.1–3.6, we first describe the components of our TSD algorithm for the 2D-SPP where all p variables (with subscripts) are *p*arameters whose values will be given in the experimentation part (Sect. 4.1). The general procedure is finally summarized in Sect. 3.7.

3.1 Search Space: A Direct Representation

Many approaches for the 2D-SPP consider a search *S*pace S composed of the set of all permutations of the objects, see [13,14] for instance. More precisely, for a given n-set of objects to be packed, a permutation of $[1 \dots n]$ is used to introduce an order for all the objects which is followed by a given placement heuristic (or "decoder"). In other words, given a particular permutation π and placement heuristic ϕ, one can pack all the objects using ϕ and according to the order indicated by π. Based on this permutation representation, several greedy placement heuristics have been investigated for the 2D-SPP. BLF (Bottom Left Fill) is such a heuristic [27]. Basically, BLF places each object at the left-most and lowest possible free area. It is capable of filling enclosed wasted areas. Notice that, according to the way BLF is implemented, its worst time complexity goes from $\mathrm{O}(n^3)$ [28] to $\mathrm{O}(n^2)$ [29] for a permutation of n objects.

TSD does not code packings with permutations but adopts a *direct* representation where a "solution" $s \in S$ (optimal or not) is a $\{L, E\}$ set:

- L, for "*L*ocation" (of the rectangular objects to be positioned in P), is an n-vector. It indicates the coordinates (x_i^r, y_i^r) of the bottom-left corner of each rectangle $r_i \in R$ in P.
- E is a set of *rectangular* "*E*mpty" spaces in P. Each $e_i \in E$ is characterized by the coordinates (x_i^e, y_i^e) of its bottom-left corner, a width $0 < w_i^e \leq W$ and a height $0 < h_i^e \leq H(s)$ with $0 \leq x_i^e \leq W - w_i^e$ and $0 \leq y_i^e \leq H(s) - h_i^e$. Each $e_i \in E$ is a *maximal* rectangle, i.e. $\forall (e_i, e_j) \in E \times E / i \neq j, x_i^e < x_j^e \vee x_i^e + w_i^e > x_j^e + w_j^e \vee y_i^e < y_j^e \vee y_i^e + h_i^e > y_j^e + h_j^e$ (e_i is not included into e_j). Note that the notion of "maximal rectangular empty space" seems to have been independently introduced in [30] (where it is called "maximal area") and [8] ("maximal hole"). In particular, it was proved in [30] that $|E|$ is at most in $\mathrm{O}(n^2)$.

3.2 Initial Solution

In local search algorithms, the initial solution s_0 specifies where the search begins in S. TSD uses the BLF procedure [27] to construct s_0, where the π permutation

orders the $r_i \in R$ first by decreasing width, and, second, by decreasing height if necessary (randomly last). We employed this decoder / order since previous experiments suggested that the `BLF` placement algorithm usually outperforms other ϕ decoders, see [28,31] for instance.

3.3 Fitness Function

To evaluate a solution $s \in S$, `TSD` uses the following $f$itness (or "evaluation") function f to be minimized:

$$f(s) = \begin{cases} 0 \text{ if } \lceil R \rceil = \emptyset \\ \sum_{r_i \in \lceil R \rceil} w_i^r * (y_i^r + h_i^r - H^* + p_H) \text{ otherwise } . \end{cases} \quad (5)$$

where $\lceil R \rceil \subseteq R$ is the set of rectangles $r_i / y_i^r + h_i^r > H^* - p_H$ (integer $0 < p_H < H^*$, for decrement of the Height) and H^* is the best height found, initially the height $H(s_0)$ of the starting solution s_0 introduced in the previous section.

Roughly speaking, the value $f(s)$ is the area of rectangles exceeding $H^* - p_H$ in P with $f(s) = 0$ meaning $H(s) < H^*$, see Fig. 1 for an example. In other words, f measures the quality of s with respect to the current satisfaction problem considered, defined by H^* and p_H: Is there a solution $s \in S / H(s) \leq H^* - p_H$? f is used to compare any $(s, s') \in S \times S$: s is better than s' if $f(s) < f(s')$. `TSD` maintains a set S^* of best solutions according to (5) with $|S^*| \leq p_*$ and $S^* = \{s_0\}$ at the beginning of the search. S^* is used for the diversification process described in Sect. 3.6.

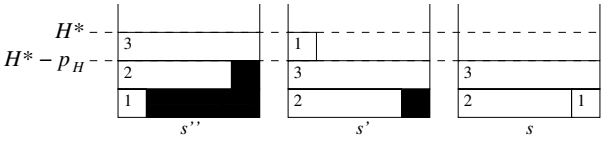


Fig. 1. Let r_1 be the unit square. $f(s'') = W$ since $\lceil R \rceil = \{r_3\}$ for s''. Similarly, $f(s') = 1$ and $f(s) = 0$.

3.4 Neighborhoods and Their Exploitation

A Neighborhood $N : S \rightarrow S$ is an application used to explore S (and to guide the search process) such as $\forall s \in S, s' \in N(s)$ if s and s' only differ by a particular operation called a "move" (noted μ). `TSD` integrates two different *complementary* neighborhoods called N_1 (performed with probability p_N) and N_2 (probability $1 - p_N$), both of them are based on the principle of ejection chains. The goal of this combination is to reduce the height of the packing while avoiding solutions with (hard to fill) tall and thin wasted spaces.

Each time a move is performed from s to s' (at iteration m), S^* and H^* are updated if necessary, and only whenever $s' \notin S^*$, with the following rules where $\lceil s^* \rceil$ (resp. $\lfloor s^* \rfloor$) is the worst (resp. a best, found at iteration m^*) element in S^* according to (5) (consider the $\lceil s^* \rceil$ introduced the most recently):

- $f(s') < f(\lfloor s^* \rfloor) \Rightarrow \delta \leftarrow m - m^*$, $p_D \leftarrow m + p_I * \delta$, $m^* \leftarrow m$. δ (number of moves required to improve $\lfloor s^* \rfloor$), p_I (for "*I*ncrement") and p_D are used for the *D*iversification process detailed in Sect. 3.6
- $f(s') = 0 \Rightarrow H^* \leftarrow H(s')$
- $|S^*| < p_* \Rightarrow S^* \leftarrow S^* \cup \{s'\}$
- $|S^*| = p_* \wedge f(s') < f(\lceil s^* \rceil) \Rightarrow S^* \leftarrow S^* \setminus \lceil s^* \rceil \cup \{s'\}$

N_1 and N_2 are based on the ejection chain principle and share a common characteristic: They move (at least) one rectangle r_i to another location. This new location for r_i may generate overlaps with a set $R_{r_i} \subset R$ of other rectangles: $R_{r_i} = \left\{r_j \in R / j \neq i \wedge x_i^r < x_j^r + w_j^r \wedge x_i^r + w_i^r > x_j^r \wedge y_i^r < y_j^r + h_j^r \wedge y_i^r + h_i^r > y_j^r\right\}$. To repair the overlaps between r_i and all $r_j \in R_{r_i}$ (i.e. to insure $s' \in S$), all $r_j \in R_{r_i}$ are removed from P, sorted like in Sect. 3.2 and, then, relocated with `BLF`.

Finally, notice that changing the location of r_i and the deletion or repositioning of all $r_j \in R_{r_i}$ (possibly) imply updates of E. This is done using the efficient "incremental" procedures introduced in [8,20].

N_1: Reduce the Height of the Packing. This is done by moving a $r_i \in \lceil R \rceil$ **below** its current location (x_i^r, y_i^r), at the bottom-left corner either of an empty space $e_j \in E$ (defining a sub-neighborhood N_1^E) or of another $r_j \in R$ (defining N_1^R).

Start with N_1^E. From the current s, **all** $r_i \in \lceil R \rceil$ (considered from the highest and left-most to the lowest and right-most) are first tried to be relocated to the $\left(x_j^e, y_j^e\right)$ coordinates of **all** $e_j \in E / y_j^e < y_i^r \wedge x_j^e + w_i^r \leq W$. This generates $|\lceil R \rceil|$ sets $N_1^E(s,i)$ of neighbors: $N_1^E(s) = \cup_{r_i \in \lceil R \rceil} N_1^E(s,i)$. Let $\lfloor N_1^E(s) \rfloor \subseteq N_1^E(s)$ be the set of the *best evaluated* neighbors of s according to N_1^E and (5).

If $f(s') = 0 \, \forall s' \in \lfloor N_1^E(s) \rfloor$, select randomly one $s' \in \lfloor N_1^E(s) \rfloor$ minimizing (1) to become the new "current" solution for the next iteration ($s \leftarrow s'$). Otherwise, if $f(s') < f(s) \, \forall s' \in \lfloor N_1^E(s) \rfloor$ make $s \leftarrow s'$ (select s' randomly if $\lfloor N_1^E(s) \rfloor$ contains more than one such element).

Possibly Apply N_1^R. If N_1^E cannot improve s according to (5), i.e. if $f(s') \geq f(s) \, \forall s' \in \lfloor N_1^E(s) \rfloor$, try to do so with N_1^R. In this case, $N_1^R(s)$ is explored in a **random** order and the **first** improving move (if available) is performed.

From the current s, a random $r_i \in \lceil R \rceil$ is first selected. Then, all $r_j \in R / j \neq i \wedge y_j^r < y_i^r \wedge x_j^r + w_i^r \leq W$ are considered in a random order. If transferring r_i at $\left(x_j^r, y_j^r\right)$ leads to an improved solution s', the exploration of $N_1^R(s)$ halts and the search continues from s' ($s \leftarrow s'$). Otherwise, another such r_j is selected. If all r_j have been tried, the exploration of $N_1^R(s)$ continues with another r_i.

In the Worst Case, Make a Best Non-Improving Move. Let $\lfloor N_1(s) \rfloor$ be the set of the *best evaluated* neighbors of s according to N_1^E, N_1^R and (5). If N_1 cannot improve s, i.e. $f(s') \geq f(s) \, \forall s' \in \lfloor N_1(s) \rfloor$, a random **non-improving** neighbor $s' \in \lfloor N_1(s) \rfloor$ is selected for the next iteration: $s \leftarrow s'$.

***N_2*: Avoid Solutions with Tall and Thin Wasted Spaces.** This second neighborhood relies on the following empirical observation: Some empty spaces (usually tall and thin) have a low area / perimeter ratio. Intuitively, since they are often located on the borders of the strip, they cannot be used with N_1^E. Indeed, preliminary computational experiments, using only N_1, has shown that some of them (those with a maximum perimeter) were persistent and hard to fill. N_2 has thus been designed to (try to) avoid these situations, i.e. to concentrate on these particular empty spaces to limit their number.

Let $\lceil E \rceil \subseteq E$ be the set of empty spaces with maximum perimeter: $\lceil E \rceil = \left\{ e_{j'} \in E / w^e_{j'} + h^e_{j'} \geq w^e_{j''} + h^e_{j''} \forall e_{j''} \in E \right\}$. Choose the empty space $e_j \in \lceil E \rceil$ located the highest and the left-most in P, i.e. $y^e_j \geq y^e_{j'} \wedge x^e_j \leq x^e_{j'} \forall e_{j'} \in \lceil E \rceil$. e_j is said to be "not adjacent" to $r_k \in R$ (shortly noted "$e_j \cap r_k = \emptyset$") if $x^r_k > x^e_j + w^e_j \vee x^r_k + w^r_k < x^e_j \vee y^r_k > y^e_j + h^e_j \vee y^r_k + h^r_k < y^e_j \vee \left(\left(x^r_k = x^e_j + w^e_j \vee x^r_k + w^r_k = x^e_j \right) \wedge \left(y^r_k + h^r_k = y^e_j \vee y^r_k = y^e_j + h^e_j \right) \right)$. Let $\lceil R_{e_j} \rceil \subset R$ be the set of rectangles with a greater area than that of e_j and that are not adjacent to e_j: $\lceil R_{e_j} \rceil = \left\{ r_k \in R / w^r_k * h^r_k > w^e_j * h^e_j \wedge e_j \cap r_k = \emptyset \right\}$.

From the current s, **all** $r_i \in \lceil R_{e_j} \rceil$ (considered from the highest and left-most to the lowest and right-most) are first tried to be relocated to the four corner of e_j. To be more precise, the bottom-left corner of each $r_i \in \lceil R_{e_j} \rceil$ is positioned at (x^e_j, y^e_j) if $x^e_j + w^r_i \leq W$, $(x^e_j + w^e_j - w^r_i, y^e_j)$ if $x^e_j + w^e_j - w^r_i \geq 0$, $(x^e_j, y^e_j + h^e_j - h^r_i)$ if $y^e_j + h^e_j - h^r_i \geq 0 \wedge x^e_j + w^r_i \leq W$ and $(x^e_j + w^e_j - w^r_i, y^e_j + h^e_j - h^r_i)$ if $y^e_j + h^e_j - h^r_i \geq 0 \wedge x^e_j + w^e_j - w^r_i \geq 0$. This generates $| \lceil R_{e_j} \rceil |$ sets $N_2(s,i)$ of neighbors with $1 \leq |N_2(s,i)| \leq 4$: $N_2(s)$ is the union of these sets. Let $\lfloor N_2(s) \rfloor$ be the set of the *best evaluated* neighbors of s according to N_2 and (5): $\lfloor N_2(s) \rfloor = \{ s' \in N_2(s) / \forall s'' \in N_2(s), f(s') \leq f(s'') \}$. Similarly to N_1, if $f(s') = 0 \ \forall s' \in \lfloor N_2(s) \rfloor$, select randomly one $s' \in \lfloor N_2(s) \rfloor$ minimizing (1). Otherwise, choose $s' \in \lfloor N_2(s) \rfloor$ at random for the next iteration.

3.5 Tabu List

One fundamental component of TS is a special short-term memory that maintains a selective history of the search. It is composed of previously encountered solutions or, more generally, pertinent attributes of such solutions. The aims of a *T*abu *L*ist (shortly "*TL*") are to avoid cycling and to go beyond local optima.

At current iteration m, since a `TSD` move μ_m from s to a neighbor $s' \in N(s)$ consists in relocating (at least) one $r_i \in R$ from (x^r_i, y^r_i) to another location (x'^r_i, y'^r_i), i.e. $x'^r_i \neq x^r_i \vee y'^r_i \neq y^r_i$, it seems quite natural to forbid r_i to return to (x^r_i, y^r_i) from s'. This "reverse" move (noted μ_m^{-1}), that can be characterized by $\mu_m^{-1} = (i, x^r_i, y^r_i)$, will then be stored in TL (shortly $TL \leftarrow TL \cup \left\{ \mu_m^{-1} \right\}$) for a duration TT (called the "*T*abu *T*enure") to indicate that μ_m^{-1} is forbidden, at least up to iteration $m + TT$. In `TSD`, TT is a random integer number from $[p_{\min TT}, \ldots, p_{\max TT}]$.

This strategy has one main drawback. Assume that μ_m has been performed and that the next move μ_{m+1} relocates a $r_j \in R$ to (x^r_i, y^r_i) such that $j \neq i$ and μ_{m+1} is not tabu ($\mu_{m+1} \notin TL$). If r_j and r_i are of the same dimensions,

i.e. $w_i^r = w_j^r \wedge h_i^r = h_j^r$, μ_{m+1} may return the search to s (already visited) or to solutions (already visited or not) too close to s.

To avoid these situations, TSD does not record (i, x_i^r, y_i^r) but $(w_i^r, h_i^r, x_i^r, y_i^r)$. Note that the following simple "aspiration criterion" (that removes a tabu status) is available in TSD: A tabu move is always accepted if it leads to a solution s' that is better than the best solution $\lfloor s^* \rfloor$ ever found ($f(s') < f(\lfloor s^* \rfloor)$). Furthermore, if **all** potential moves are tabu, we select the first one with the lowest tabu duration that would be performed if all tabu status were (temporarily) removed.

3.6 Diversification

The TSD algorithm maintains a set S^* of high quality solutions obtained from the beginning of the search (see Sect. 3.3 and 3.4). These elite solutions are used as candidates for diversification.

When the current search cannot be improved for a number of iterations, TSD picks one solution $s^* \in S^*$ at random. This solution is then slightly perturbed by moving a random $r_i \in R$ to $(x, H^* - p_H - h_i^r)$, where x is chosen randomly from $[0 \dots W - w_i^r]$. Rectangles overlapping with r_i are relocated like in Sect. 3.4. This perturbed solution becomes finally the new current solution ($s \leftarrow s^*$) with the tabu list reset to empty (except the move used for perturbation).

3.7 TSD: An Overview

The TSD algorithm begins with an initial solution (Sect. 3.2). Then it proceeds iteratively to visit a series of (locally best) solutions following the neighborhoods (Sect. 3.4). At each iteration, the current solution s is replaced by a neighboring solution s' even if s' does not improve s.

While it is not mentioned here for simplicity, note that TSD can also end (see Step 2 below) before reaching the maximum time *L*imit p_L. This may occur each time S^* is updated whenever the optimum height H_{OPT} (or an upper bound) is known and $H(\lfloor s^* \rfloor) \leq H_{OPT}$.

1. *Initialization.* $m \leftarrow 0$ (current number of iterations), build s using BLF. $H^* \leftarrow H(s)$, $S^* \leftarrow \{s\}$, $m^* \leftarrow 0$, $TL \leftarrow \emptyset$.
2. *Stop condition.* If elapsed time has reached p_L Then: Return H^* and $\lfloor s^* \rfloor$.
3. *Diversification.* If $m > p_D$ Then:
   - Choose at random $s \in S^*$, $r_i \in R$ and $x \in [0 \dots W - w_i^r]$.
   - Update s by relocating r_i to $(x, H^* - p_H - h_i^r)$, defining a move μ.
   - $TL \leftarrow \{\mu^{-1}\}$, $m^* \leftarrow m$, $p_D \leftarrow m + p_I * \delta$. Possibly update S^* or H^*.
4. *Exploration of the neighborhood.* Let N be N_1 or N_2 according to p_N. Update s according to $N(s)$, defining a move μ. $m \leftarrow m + 1$. $TL \leftarrow TL \cup \{\mu^{-1}\}$. Possibly update S^*, H^* or p_D. Go to step 2.

4 Experimentations

We used a set of 21 well-known benchmark instances [28] available from http://mo.math.nat.tu-bs.de/packlib/xml/ht-eimhh-01-xml.shtml to compare

Table 1. Main characteristics of the test problems from [28]

Category	Instances	W	n	H_{OPT}
C1	C1P1, C1P2, C1P3	20	16, 17, 16	20
C2	C2P1, C2P2, C2P3	40	25	15
C3	C3P1, C3P2, C3P3	60	28, 29, 28	30
C4	C4P1, C4P2, C4P3	60	49	60
C5	C5P1, C5P2, C5P3	60	73	90
C6	C6P1, C6P2, C6P3	80	97	120
C7	C7P1, C7P2, C7P3	160	196, 197, 196	240

TSD with previously reported TS algorithms as well as the best performing approaches. The main characteristics of these instances are given in Tab. 1. Note that these benchmarks have a known optimal height H_{OPT}.

4.1 Experimentation Conditions

The comparison is based on the percentage gap γ of a solution s from the optimum or its best bound (H_{OPT}): $\gamma(s) = 100 * (H(s) - H_{OPT}) / H(s)$. Similarly to the best-known approaches considered in Tab. 2, mean gap $\overline{\gamma}$ (resp. best gap γ^*) is averaged over a number of 10 runs (resp. over best runs only), each run being *L*imited to p_L seconds.

The TSD parameters are: $p_H = 1$ (to define the current satisfaction problem to solve), $p_* = 30$ (maximum size of S^*), $p_N = 0.65$ (probability to explore neighborhood N_1), $p_{\min TT} = 5$ and $p_{\max TT} = 15$ (minimum and maximum *T*abu *T*enure), $p_D = 10$ and $p_I = 3$ (for diversification), $p_L \in [60 \dots 2\,700]$ (time *L*imit, in seconds). TSD is coded in C++ and all computational results were obtained running TSD on a 2 Ghz Dual Core PC.

4.2 Computational Results

TSD is compared in Tab. 2 with the previously reported TS algorithms denoted as "TS1" [13] and "TS2" [12] and the best performing approaches: GRASP [6] and IDW [7]. Note that TS was also tried in [8], achieving "good performance", but no numerical results were reported. While the stopping criterion per run for GRASP, IDW and TS1 is also a time *L*imit p_L (60, 100 and 360 seconds resp.), TS2 used a maximum number of iterations (1 500).

In Table 2, "-" marks mean unknown information, "Mean C*i*" are averaged values on category C*i*, the last three lines reporting averaged values for the largest (and hardest) instances and all the 21 instances, and (minimum) number of instances optimally solved. No $\overline{\gamma}$ or γ^* is mentioned for IDW, TS2 and TS1 since this information is not given in [7,12,13].

According to Tab. 2, TS1 is the worst performing (TS) approach for the benchmark tried. Indeed, $\gamma^* = 0$ only for C2P1 and C2P3 while other methods *always* solved at least 6 instances.

On the "smallest" (easiest) instances C1–C3, TS2 is the best method since it always solved **all** the 9 instances. However, TSD (and GRASP) compares well with TS2 since only one instance was not solved optimally ($H^* = 31$ for C3P2).

The largest instances (C4–C7) are quite challenging since no approach reached H_{OPT} to our knowledge on these instances (except IDW perhaps but this is not clearly mentioned in [7]). Note that, while IDW achieved here the smallest $\overline{\gamma}$ value, TSD is slightly better than GRASP on C6 ($1.37 < 1.56$) and C7 ($1.23 < 1.36$), see also line "Mean C4–C7" where $1.33 < 1.41$.

Table 2. Mean and best percentage gap ($\overline{\gamma}$ and γ^* resp.) on instances from [28]

Instances	TSD		TS1 [13]	TS2 [12]	GRASP [6]		IDW [7]
	$\overline{\gamma}$	γ^*	γ^*	γ^*	$\overline{\gamma}$	γ^*	$\overline{\gamma}$
C1P1	0	0	9.09	0	0	0	-
C1P2	4.76	0	9.09	0	0	0	-
C1P3	0	0	4.76	0	0	0	-
Mean C1	1.59	0	7.65	0	0	0	0
C2P1	0	0	0	0	0	0	-
C2P2	0	0	6.25	0	0	0	-
C2P3	0	0	0	0	0	0	-
Mean C2	0	0	2.08	0	0	0	0
C3P1	0	0	3.23	0	0	0	-
C3P2	3.23	3.23	9.09	0	3.23	3.23	-
C3P3	0	0	9.09	0	0	0	-
Mean C3	1.08	1.08	7.14	0	1.08	1.08	2.15
C4P1	1.64	1.64	6.25	-	1.64	1.64	-
C4P2	1.64	1.64	4.76	-	1.64	1.64	-
C4P3	1.64	1.64	3.23	-	1.64	1.64	-
Mean C4	1.64	1.64	4.75	-	1.64	1.64	1.09
C5P1	1.1	1.1	5.26	-	1.1	1.1	-
C5P2	1.1	1.1	3.23	-	1.1	1.1	-
C5P3	1.1	1.1	6.25	-	1.1	1.1	-
Mean C5	1.1	1.1	4.91	-	1.1	1.1	0.73
C6P1	1.64	0.83	4.76	-	1.56	0.83	-
C6P2	0.83	0.83	3.23	-	1.56	0.83	-
C6P3	1.64	0.83	3.23	-	1.56	0.83	-
Mean C6	1.37	0.83	3.74	-	1.56	0.83	0.83
C7P1	1.23	1.23	-	-	1.64	1.64	-
C7P2	1.23	1.23	-	-	1.19	0.83	-
C7P3	1.23	1.23	-	-	1.23	1.23	-
Mean C7	1.23	1.23	-	-	1.36	1.23	0.41
Mean C4–C7	1.33	1.2	-	-	1.41	1.2	0.76
Mean C1–C7	1.14	0.84	-	-	0.96	0.84	0.4
$\#H_{OPT}/21$	8		≥ 2	9	8		9

5 Possible Extensions

The TSD approach reported in this paper is in fact the first version of an ongoing study. In this section, we discuss possible extensions which are worthy of further study and would help to improve the performance of TSD: Diversification, combined utilization of neighborhoods and evaluation function. All these points merit certainly more investigations and constitute our ongoing work.

Diversification: The diversification technique described in Sect. 3.6 is based on a random perturbation strategy. This strategy can be reinforced by a more elaborated strategy using useful information extracted from high quality solutions. For instance, it would be possible to identify a set of *critical* objects

that prevent the search from converging toward a good packing and then focus on these objects in order to realize a *guided* diversification.

Combined utilization of neighborhoods: Two neighborhoods are proposed in Sect. 3.4. They are used in a particular manner, applying N_1 with probability p_N or N_2 with probability $1 - p_N$. These two neighborhoods can also be employed in other combined ways, for instance, by the union of N_1 and N_2 ($N_1 \cup N_2$) or sequentially (token-ring, $N_1 \rightarrow N_2 \rightarrow N_1 \ldots$).

Evaluation function: The current evaluation function (see Sect. 3.3) is unable to distinguish two solutions with the same height. However, such a situation occurs often during a search process. To overcome this difficulty, it would be useful to introduce an additional criterion into the evaluation function. For instance, the free surface under H^* may be such a potential criterion. As such, we can say a solution s' is better than another solution s if s' has a larger total free area under H^* than s does even if both solution have the same height. Indeed, it would be easier to improve s' than s.

6 Conclusions

In this paper, we presented `TSD`, a `T`abu `S`earch algorithm for the 2D Strip Packing Problem. `TSD` uses a solution strategy which traits the initial *optimization* problem as a succession of *satisfaction* problems: Starting from a packing s_0 of height H, `TSD` tries to solve the 2D-SPP with decreasing values of H.

`TSD` uses a direct representation of the search space which permits inexpensive basic operations. Two *complementary* neighborhoods using ejection chains are explored in a combined way by `TSD`. The goal of this combination is to reduce H^* (hence to solve the current satisfaction problem) while avoiding solutions with (hard to fill) tall and thin wasted spaces. A *specific* fitness function f is designed to guide the search. A *diversification* mechanism, relying on a set of historically best packings, helps to direct the search to promising and unexplored regions. The tabu structure includes knowledge of the problem to avoid visiting similar packings.

Preliminary computational results were reported on a set of 21 well-known benchmark instances, showing competitive results in comparison with two recent best performing algorithms. The results on the largest and hardest instances are particularly promising. Several issues were identified for further improvements.

Acknowledgments. We would like to thank the reviewers of the paper for their useful comments. This work was partially supported by two grants from the French "Pays de la Loire" region (MILES and RadaPop projects).

References

1. Wäscher, G., Haußner, H., Schumann, H.: An improved typology of cutting and packing problems. European Journal of Operational Research 183(3), 1109–1130 (2007)
2. Dowsland, K., Dowsland, W.: Packing problems. European Journal of Operational Research 56(1), 2–14 (1992)

3. Sweeney, P., Ridenour Paternoster, E.: Cutting and packing problems: A categorized, application-orientated research bibliography. Journal of the Operational Research Society 43(7), 691–706 (1992)
4. Fowler, R., Paterson, M., Tanimoto, S.: Optimal packing and covering in the plane are NP-complete. Information Processing Letters 12(3), 133–137 (1981)
5. Garey, M., Johnson, D.: Computers and Intractability: A Guide to the Theory of NP-Completness. W.H. Freeman and Company, San Francisco (1979)
6. Alvarez-Valdes, R., Parreño, F., Tamarit, J.: Reactive GRASP for the strip-packing problem. Computers & Operations Research 35(4), 1065–1083 (2008)
7. Neveu, B., Trombettoni, G.: Strip packing based on local search and a randomized best-fit. In: Fifth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: First Workshop on Bin Packing and Placement Constraints (CPAIOR: BPPC), Paris, France, May 22 (2008)
8. Neveu, B., Trombettoni, G., Araya, I.: Incremental move for strip-packing. In: Avouris, N., Bourbakis, N., Hatzilygeroudis, I. (eds.) Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), vol. 2, pp. 489–496. IEEE Computer Society, Los Alamitos (2007)
9. Soke, A., Bingul, Z.: Hybrid genetic algorithm and simulated annealing for two-dimensional non-guillotine rectangular packing problems. Engineering Applications of Artificial Intelligence 19(5), 557–567 (2006)
10. Zhang, D., Kang, Y., Deng, A.: A new heuristic recursive algorithm for the strip rectangular packing problem. Computers & Operations Research 33(8), 2209–2217 (2006)
11. Zhang, D., Liu, Y., Chen, S., Xie, X.: A meta-heuristic algorithm for the strip rectangular packing problem. In: Wang, L., Chen, K., S. Ong, Y. (eds.) ICNC 2005. LNCS, vol. 3612, pp. 1235–1241. Springer, Heidelberg (2005)
12. Alvarez-Valdes, R., Parreño, F., Tamarit, J.: A tabu search algorithm for a two-dimensional non-guillotine cutting problem. European Journal of Operational Research 183(3), 1167–1182 (2007)
13. Iori, M., Martello, S., Monaci, M.: Metaheuristic algorithms for the strip packing problem. In: Pardalos, P.M., Korotkikh, V. (eds.) Optimization and Industry: New Frontiers. Applied Optimization, vol. 78, pp. 159–179. Springer, Heidelberg (2003)
14. Gómez-Villouta, G., Hamiez, J.P., Hao, J.K.: A dedicated genetic algorithm for two-dimensional non-guillotine strip packing. In: Proceedings of the 6th Mexican International Conference on Artificial Intelligence, Special Session, MICAI, Aguascalientes, Mexico, pp. 264–274. IEEE Computer Society, Los Alamitos (2007)
15. Bortfeldt, A.: A genetic algorithm for the two-dimensional strip packing problem with rectangular pieces. European Journal of Operational Research 172(3), 814–837 (2006)
16. Mukhacheva, E., Mukhacheva, A.: The rectangular packing problem: Local optimum search methods based on block structures. Automation and Remote Control 65(2), 248–257 (2004)
17. Yeung, L., Tang, W.: Strip-packing using hybrid genetic approach. Engineering Applications of Artificial Intelligence 17(2), 169–177 (2004)
18. Leung, T., Chan, C., Troutt, M.: Application of a mixed simulated annealing-genetic algorithm heuristic for the two-dimensional orthogonal packing problem. European Journal of Operational Research 145(3), 530–542 (2003)
19. Gomez, A., de la Fuente, D.: Solving the packing and strip-packing problems with genetic algorithms. In: Mira, J., Sánchez-Andrés, J. (eds.) IWANN 1999. LNCS, vol. 1606, pp. 709–718. Springer, Heidelberg (1999)

20. Neveu, B., Trombettoni, G., Araya, I., Riff, M.C.: A strip packing solving method using an incremental move based on maximal holes. International Journal on Artificial Intelligence Tools 17(5), 881–901 (2008)
21. Araya, I., Neveu, B., Riff, M.C.: An efficient hyperheuristic for strip-packing problems. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) Adaptive and Multilevel Metaheuristics. Studies in Computational Intelligence, vol. 136, pp. 61–76. Springer, Heidelberg (2008)
22. Kenmochi, M., Imamichi, T., Nonobe, K., Yagiura, M., Nagamochi, H.: Exact algorithms for the 2-dimensional strip packing problem with and without rotations. European Journal of Operational Research (2008) (to appear)
23. Bekrar, A., Kacem, I., Chu, C.: A comparative study of exact algorithms for the two dimensional strip packing problem. Journal of Industrial and Systems Engineering 1(2), 151–170 (2007)
24. Lesh, N., Marks, J., McMahon, A., Mitzenmacher, M.: Exhaustive approaches to 2D rectangular perfect packings. Information Processing Letters 90(1), 7–14 (2004)
25. Martello, S., Monaci, M., Vigo, D.: An exact approach to the strip packing problem. INFORMS Journal on Computing 15(3), 310–319 (2003)
26. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Dordrecht (1997)
27. Baker, B., Coffman Jr., E., Rivest, R.: Orthogonal packings in two dimensions. SIAM Journal on Computing 9(4), 846–855 (1980)
28. Hopper, E., Turton, B.: An empirical investigation of meta-heuristic and heuristic algorithms for a 2D packing problem. European Journal of Operational Research 128(1), 34–57 (2001)
29. Chazelle, B.: The bottom-left bin-packing heuristic: An efficient implementation. IEEE Transactions on Computers 32(8), 697–707 (1983)
30. El Hayek, J.: Le problème de bin-packing en deux-dimensions, le cas non-orienté : résolution approchée et bornes inférieures. Ph.D thesis, Université de Technologie de Compiègne, France (2006) (in French)
31. Imahori, S., Yagiura, M., Nagamochi, H.: Practical algorithms for two-dimensional packing. In: Gonzalez, T. (ed.) Handbook of Approximation Algorithms and Metaheuristics. Chapman & Hall/CRC Computer & Information Science Series, ch. 36, vol. 13. CRC Press, Boca Raton (2007)

An ACO Approach to Planning

Marco Baioletti, Alfredo Milani, Valentina Poggioni, and Fabio Rossi

Dipartimento di Matematica e Informatica
University of Perugia, Italy
{baioletti,milani,poggioni,rossi}@dipmat.unipg.it

Abstract. In this paper we describe a first attempt to solve planning problems through an Ant Colony Optimization approach. We have implemented an ACO algorithm, called ACOPlan, which is able to optimize the solutions of propositional planning problems, with respect to the plans length. Since planning is a hard computational problem, metaheuristics are suitable to find good solutions in a reasonable computation time. Preliminary experiments are very encouraging, because ACOPlan sometimes finds better solutions than state of art planning systems. Moreover, this algorithm seems to be easily extensible to other planning models.

1 Introduction

Automated planning [1] is a very important research field in Artificial Intelligence. Planning has been extensively and deeply studied and now many applications of automated planning exist, ranging from robotics to manufacturing and to interplanetary missions [1].

A planning problem can be concisely described as the problem of finding a plan, i.e. a sequence of actions, which starts from a known initial state and leads to a final state that satisfies a given goal condition. Each action can be possibly executed only in some states and in this case, after its execution, the current state is altered by changing the value of some state variables.

Planning problems are usually defined in terms of a decision problem, in which the aim is to find any solution. Sometimes just finding a solution is satisfactory. But there are many problems which have several solutions, and just stopping the search phase after having found a first plan may not be a good idea, because this plan may be sensitively more expensive than other possible ones. The cost of a plan is an important feature to take into account, because high cost plans can be useless, almost like non executable plans. Therefore, in these situations the purpose of the search is finding optimal or near–optimal plans in terms of a given *objective function*.

There are many planning models, which extend the classical one, in which the main aim is to optimize a *metric* function that measures the quality of the plans found. Perhaps the problems in which the role of an objective function to be optimized is more natural are those belonging to the numerical extension of planning (see for instance [2]). In this model, the problems are not only described

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 73–84, 2009.

in terms of propositional variables (as in the classical model), but they can also contain numerical variables in preconditions, effects, initial state, goals and, of course, objective function. For instance, a cost could be assigned to each action in terms of both needed fuel and covered distance, and the goal condition could require to minimize both of them. Besides the apparent importance of optimization in planning, this topic has not been extensively studied in extended models, such as planning with numerical resources. In fact there exist only a few optimal planners and there are a relatively small number of other planners which usually produce good solutions, without guaranteeing the optimality.

However, even in the case of pure propositional planning, the number of actions is an interesting objective function too and finding optimal solutions can be a very hard problem. For instance, Gupta and Nau [3] have proved that optimal planning, for the well known blocks world problem, is NP–complete.

In any case, optimal planning is a (hard) combinatorial optimization problem, for which there exist only a few standard algorithmical techniques. These methods are often not so efficient and they are only able to solve small instances of interesting planning problems.

In order to solve this difficulty, the main idea of this paper is to adapt the well known Ant Colony Optimization metaheuristic to optimal planning. Clearly, being ACO a stochastic algorithm, there is no guarantee that an optimal solution is found, but we hope that, as shown in many other applications of ACO, this method produces either excellent or optimal solutions. In fact, ACO has been successfully used in many combinatorial optimization problems [4], being competitive with the best ad–hoc algorithms for these problems.

Hence, we have implemented, as a first step, an ACO–based propositional planner which tries to optimize plans in terms of their lengths, in order to test if ACO can be effectively used in planning for optimization. This planner performs a forward search in the state space in which the ants are guided by the pheromone values and by a heuristic function typically used in planning. Since we will show that this implementation is able to find very good solutions, it is likely that ACO can be successfully implemented in other models of planning, as we will discuss in the conclusions.

As far as we know, this is the first application of a metaheuristic to optimal automated planning. Genetic programming has been used in planning in [5], but this technique has not been shown to be promising and it seems to have been soon abandoned.

One of the most difficult problem we have coped with, is how to assign a meaningful score to plans which are not solutions, because, in this case, the plan length is not suitable to evaluate the quality of a plan. It is obvious that these situations appear often during the first generations and it is very important to force the ants to produce legal plans as much as possible. Therefore, we decided to use a further scoring function which takes into account the distance between the states reached by the sequence and the goals. Discarding illegal solutions or just counting the number of goals reached would be too naive techniques, which will loose many important information about the planning domain.

The paper is structured as follows. In section 2 a short introduction to AI Planning is provided, while in Section 3 we show how we applied ACO to AI Planning. In section 4 some preliminary experimental results are shown. In Section 5 some related works are discussed. Section 6 concludes the paper by describing some possible improvements and extensions of this work.

2 A Brief Introduction to Automated Planning

The standard reference model for planning is the Propositional STRIPS model [1], called also "Classical Planning". In this model the world states are described in terms of a finite set $\mathcal{F}$ of propositional variables: a state s is represented with a subset of $\mathcal{F}$, containing all the variables which are true in s. A problem is a triple $(\mathcal{I}, \mathcal{G}, \mathcal{A})$, in which $\mathcal{I}$ is the initial state, $\mathcal{G}$ denotes the goal states, and $\mathcal{A}$ is a finite set of actions.

An action $a \in \mathcal{A}$ is described by a triple $(pre(a), add(a), del(a))$. $pre(a) \subseteq \mathcal{F}$ is the set of preconditions: a is executable in a state s if and only if $pre(a) \subseteq s$. $add(a)$, $del(a) \subseteq \mathcal{F}$ are respectively the sets of positive and negative effects: if an action a is executable in a state s, the state resulting after its execution is $Res(s, a) = s \cup add(a) \setminus del(a)$. Otherwise $Res(s, a)$ is undefined.

A linear sequence of actions $(a_1, a_2, \ldots, a_n)$ is a plan for a problem $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ if a_1 is executable in $\mathcal{I}$, a_2 is executable in $s_1 = Res(\mathcal{I}, a_1)$, a_3 is executable in $s_2 = Res(s_1, a_2)$, and so on. A plan $(a_1, a_2, \ldots, a_n)$ is a solution for $(\mathcal{I}, \mathcal{G}, \mathcal{A})$ if $\mathcal{G} \subseteq s_n$.

Planning problems are usually stated as a pure search problem, in which the purpose is finding, if any, a solution plan. This computational problem is known to be PSPACE–complete.

There exist many algorithmical solutions for planning. Among all the proposed algorithms, three approaches are important.

A first approach is to design ad–hoc algorithms. Until the mid 90ies, the dominating technique has been Partial Order Planning. A more recent way is the graph–based approach, also used in [6]. A second approach is to translate a planning problem into a different combinatorial problem, like propositional satisfiability [7], integer programming or constraint satisfaction problems, and then to use fast solvers designed for these problems. A third possibility is to formulate planning problems as heuristic search problems, as done in HSP [8] and in FF [9].

3 ACO and Planning

One of most important methods proposed to solve planning problems is the heuristic search in the state space. The search moves from the initial state through state space by using well known search methods (A^*, *Hill climbing*, etc.) guided by a heuristic function, and it stops when a state containing the goals is reached. These methods are usually deterministic.

In our approach, an ACO algorithm performs a metaheuristic search in the state space, in order *to optimize* some *quantitative* characteristics of a planning problem, as the length of solutions found, the consumption of resources, and so on. In other words, we are regarding to a planning problem as an *optimization* problem, not only as a *search* problem.

3.1 Ant Colony Optimization

Ant Colony Optimization (ACO) is a metaheuristic designed to tackle Combinatorial Optimization problems and introduced since early 90s by Dorigo et al. [10,4].

ACO is a *constructive* meta–heuristic, i.e. each ant builds a sequence of elements, called components, in an incremental stochastic way. Each solution component is assigned a *pheromone* value, i.e. a numerical real quantity. The vector of all pheromone values T is the *pheromone model*. At each construction step, the next component to add to the current partial solution is chosen according to both a probability distribution on the feasible components (i.e. components that satisfy problem constraints), induced by the pheromone values, and a heuristic function that estimates how promising each component is. When all ants have completed the solution construction process, the pheromone values of the components which take part in the best found solutions are increased. In this way the probability of each component to be drawn in the next iterations changes. As time goes by, ants learn which components take part in the best solutions, because their pheromone values tend to increase.

Many ACO variants differ just on the pheromone update method adopted: the solutions set involved, the rule for updating, etc. The solutions considered for the update are often *best–so–far* (best solution found in all the iterations already executed) or *iteration–best* (best solution found in the current iteration) or both. In certain variants, limits for maximum and minimum pheromone values are used [11]. Another important ACO feature is the so called *forgetting* mechanism: at each iteration, before increasing pheromone values of component taking part in the best solutions, all the values are decreased by an *evaporation* rate ρ, in order to prevent a premature convergence to suboptimal solutions.For more details on ACO see [12,4].

3.2 Planner Ants

In our approach, ants build up plans starting from the initial state $s_0 = \mathcal{I}$ and by executing actions step by step. Each ant chooses the next action a to execute in the current state s_j by drawing from the set $\mathcal{A}(s_j)$ of all actions which are executable in s_j. After the execution of an action a the current state is updated as $s_{j+1} = Res(s_j, a)$.

We might apparently assume a single action as a *solution component*. But the same action a may be executed in different states leading to different successor states. In other words, the execution of an action a can have a different utility, depending on the state in which it is executed.

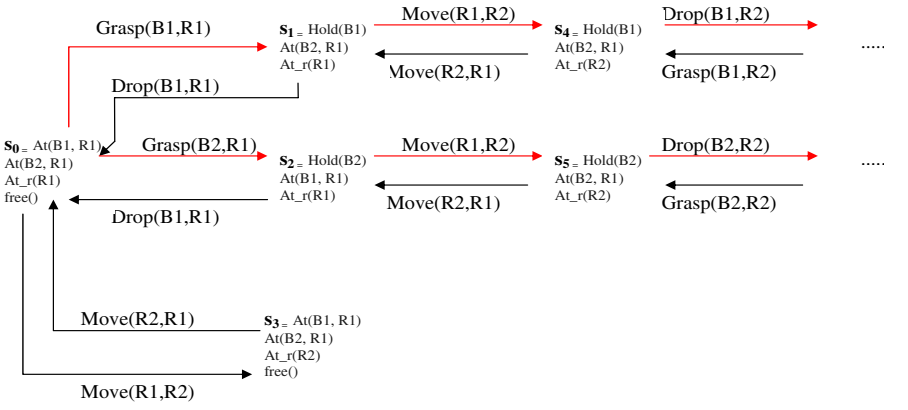


Fig. 1. Initial part of state space graph for the Gripper domain

For example, let us consider the domain *Gripper* in which there are two balls (*B1, B2*) and two rooms (*R1,R2*). In the initial state s_0, both balls are in $R1$. The goal requires that both $B1$ and $B2$ are in $R2$. The available actions are $Move$ from a room to another one, $Grasp$ a ball, and $Drop$ a ball. The initial part of the related state space graph is shown in Fig. 1.

Action *move(R1, R2)* may be a useful action or not depending on the state in which is executed. For example executing this action in s_0 is not useful to reach the goals, instead executing this action in s_1 is useful. So, if we assume a single action as a *solution component*, increasing the pheromone value of *move(R1, R2)* increases its probability to be drawn in the next iterations in both s_1 (where it is desirable) and in s_0 (where it is not desirable). This consideration leads to assume a couple state/action $c_i^j = (s_j, a_i)$ as a *solution component*. In this way, increasing the pheromone value of (s_1, *move(R1, R2)*) increases the probability of drawing this action when it is only useful. The whole set of couples state/action constitutes the pheromone model that we have called *state–action*.

Another possibility is to assume a couple step/action $c_i^j = (t_j, a_i)$ as a *solution component*, where t_j is the time step at which a_i is executed. However, knowing that an action a is executed at step t_j instead of step $t_j + 1$ seems intuitively less interesting than knowing in which state an action is executed. The whole set of couples step/action constitutes the pheromone model that we have called *level–action*.

Referring to pheromone model *state–action*, the rule to calculate the *transition probabilities* is the classical one:

$$p(c_i^j) = \frac{[\tau_i^j]^\alpha \cdot [\eta(s_j, a_i)]^\beta}{\sum_{a_k \in \mathcal{A}(s_j)} [\tau_k^j]^\alpha \cdot [\eta(s_j, a_k)]^\beta}, \quad \forall a_i \in \mathcal{A}(s_j) \tag{1}$$

where τ_i^j is the pheromone value assigned to the component $c_i^j = (s_j, a_i)$, $\eta(s_j, a_i)$ is a heuristic function that evaluates how much is promising to execute a_i in the

state s_j, $\mathcal{A}(s_j)$ is the set of actions executable in s_j and α, β are parameters to determine the relative importance of pheromone value and heuristic estimation.

3.3 Heuristic Estimation η

We decided to use as function η the heuristic *Fast-Forward (FF)* [9]. *FF* estimates the distance from a state to the goals, i.e. the number of needed actions to reach the goals. *FF* exploits the basic idea of relaxing the original planning problem by ignoring deleting effects of all actions, which is used in other heuristic systems too (for instance *HSP* [8]). To evaluate a given state, *FF* builds a relaxed planning graph (by ignoring delete effects) which is extended until all goals are reached. Then, it attempts to extract a relaxed plan in a *GraphPlan* style, i.e. performing a backward search from the goals to the initial state. The number of needed actions is the estimated distance. Moreover, *FF* provides some pruning techniques too, in order to exclude some space state branches from search. We adopted the so called *Helpful Actions* (actions that seem more promising than other ones) method in order to increment the η value for these actions.

So, at each construction step, our algorithm calculates η values of the states resulting by the execution of each feasible action in the current state by using the following rule:

$$\eta(s_c, a_i) = \begin{cases} \dfrac{1}{h(Res(s_c, a_i))(1-k)} & \text{if } a_i \text{ is a Helpful Action} \\ \dfrac{1}{h(Res(s_c, a_i))} & \text{otherwise} \end{cases} \qquad \forall a_i \in \mathcal{A}(s_c) \tag{2}$$

where s_c is the current state, $h(Res(s_c, a_i))$ is the heuristic value of $Res(s_c, a_i)$, $k \in [0, 1]$ is a reduction rate (in our tests we usually set $k \in [0.15, 0.5]$) to increase the transition probabilities of *Helpful Actions*.

3.4 Plan Evaluation

At end of each iteration, a quality evaluation of all plans built by the ants is needed, in order to perform a pheromone update. An intuitive (trivial) criterion is to consider the number of goals reached, but this is useless when no plans reach any goal.

The basic idea to evaluate the quality of a plan p is keeping track of the minimum heuristic value h_{min} (i.e. the minimum distance from goals) that p has reached during the execution of their actions: the smaller h_{min}, the higher the plan quality. Moreover, if two plans have the same h_{min}, then the plan which reaches first this value is better. For instance, let us consider the situation in which three ants build three different plans: *Plan A*, which reaches $h_{min} = 2$ at the construction step 2, *Plan B*, which reaches $h_{min} = 3$ at the construction step 2, and *Plan C*, which reaches $h_{min} = 2$ at the construction step 1. We assume that the quality of both *Plan C* and *Plan A* is better than the quality of *B*, because they get lower minimum heuristic values. Moreover, we assume

that the quality of C is better than A, because C gets the value 2 at a previous construction step.

In this way we are able to compare two different plans, in order to decide which is better. A *quantitative* measure $Q(p)$ for the quality of a plan p can be easily defined as

$$Q(p) = \left(\frac{1}{1+h_{min}}\right)^{\gamma} \left(\frac{1}{t_{min}}\right)^{\delta} \tag{3}$$

where t_{min} is the step at which p reaches h_{min}, and both γ and δ are parameters to tune the importance of two terms.

To consider also the number of reached goals, we can add a term to the equation above:

$$Q(p) = \left(\frac{1}{1+h_{min}}\right)^{\gamma} \left(\frac{1}{t_{min}}\right)^{\delta} \left(1+\frac{g_{found}}{|\mathcal{G}|}\right)^{\theta} \tag{4}$$

where g_{found} is the number of goals reached, $|\mathcal{G}|$ is the total number of goals and θ is a parameter to adjust the importance.

3.5 Pheromone Update

In our framework we perform a pheromone update considering *best–so–far* and *iteration–best* solutions. Referring to the *state–action* pheromone model the rule is the classical one used in the HyperCube Framework for ACO [13]:

$$\tau_i^j = (1-\rho)\tau_i^j + \rho \sum_{p \in P_{upd} \mid c_i^j \in p} \frac{Q(p)}{\sum\limits_{p' \in P_{upd}} Q(p')} \tag{5}$$

where τ_i^j is the pheromone value of c_i^j, ρ is the pheromone evaporation rate, P_{upd} is the set of solutions (plans) which are involved in the update, Q is the quality plan evaluation function.

After some preliminary experiments, we decided that, for each $p \in P_{upd}$, only the pheromone values relative to the first t_{min} actions are only updated, the other ones are neglected. The main reason is that what a plan does after having reached its "best" state can be ignored, because it probably moves in a wrong direction, i.e. away from the goals.

The pseudo code of the resulting algorithm, called *A*COPlan, is shown in figure 1.

4 Experimental Results

ACOplan has been tested over some domains taken from last International Planning Competitions (IPC). In general these domains are used as standard benchmarks to compare planner performances. We run a set of systematics tests over the domains *Rovers* and *Driverlog*. They have been chosen among the set of

Algorithm 1. The algorithm ACOPlan

```
1: s_best ← ∅
2: InitPheromone(T, c)
3: while termination condition not met do
4:     s_iter ← ∅
5:     for m ← 1 to number of ants do
6:         s_p ← ∅
7:         state ←initial state of the problem
8:         for i ← 1 to max number of construction step do
9:             A_i ← feasible actions on state
10:            H_i ← ∅
11:            HA_i ← GetHelpfulActions(state, A_i)
12:            for all a_i^j in A_i do
13:                h_i^j ← heuristic value of a_i^l
14:                H_i ← H_i ∪ h_i^j
15:            end for
16:            a_k ← ChooseAnAction(T, H_i, A_i, HA_i)
17:            extend s_p adding a_k
18:            update state
19:        end for
20:        if f(s_p) > f(s_iter) then
21:            s_iter ← s_p
22:        end if
23:    end for
24:    if f(s_iter) > f(s_best) then
25:        s_best ← s_iter
26:    end if
27:    UpdatePheromone(T, s_best, s_iter, ρ)
28: end while
```

benchmark domains because they offer a good variety and the corresponding results allow us interesting comments[1] .

We chose to compare ACOplan with LPG, HSP and FF.

LPG is a very performant planner and, when running with *-quality* option, it gives solution plans with, in general, a number of actions very close to the optimum (sometimes it can find solutions with the optimum number of actions). It is a non deterministic planner, so the results collected here are the mean values obtained over 100 runs.

HSP can run with several options. In particular, with the options *-d backward*, *-h h2max* and *-w 1*, it produces optimal plan in the number of actions. Nevertheless, in this setting, either it often fails to report a solution in a reasonable CPU time or it runs out of memory; for this reason, we have chosen to run it with default options also, in order to solve a larger set of problems and collect more results.

[1] A complete repository and detailed descriptions of these domains can be found in the ICAPS website www.icaps-conference.org

Table 1. Results for *Driver* domain collecting solution lengths and CPU time

	ACOplan		**HSP**		**FF**		**HSP -opt**		**LPG**	
Problem	length	time	length	time	length	time	length	time	length	time
Driverlog_1	7	0.01	7	0.01	8	0.01	7	0.01	7	0.26
Driverlog_2	19	3.87	24	0.01	22	0.01	19	3905.92	19	0.33
Driverlog_3	12	0.06	14	0.01	12	0.01	12	0.14	12	0.12
Driverlog_4	16	24.32	21	0.01	16	0.01	16	5854.79	16	0.84
Driverlog_5	19	2.79	22	0.01	22	0.01	–	–	19	2.87
Driverlog_6	11	47.35	13	0.01	13	0.01	11	0.72	12	1.21
Driverlog_7	13	0.4	15	0.01	17	0.01	13	5796.76	13	0.32
Driverlog_8	22	29.50	26	0.01	23	0.01	–	–	24	0.12
Driverlog_9	22	12.87	28	0.03	31	0.01	–	–	22	0.08
Driverlog_10	17	96.56	24	0.03	20	0.01	–	–	18	1.67
Driverlog_11	19	899.56	24	0.07	25	0.01	–	–	21	3.19
Driverlog_12	36	432.68	44	0.07	52	0.22	–	–	41	4.95
Driverlog_13	26	119.61	33	0.16	36	0.09	–	–	29	5.33
Driverlog_14	34	5440.28	44	1.66	38	0.12	–	–	35	18.40
Driverlog_15	44	4920.89	48	1.65	48	0.03	–	–	39	7.12

FF has no option to choose and it runs in default version.

Also ACOplan has many parameters that have to be chosen. After some preliminary tests we decided to use this setting: 10 ants, 5000 iterations, $\alpha = 2$, $\beta = 5$, $\rho = 0.15$, $c = 1$, $k = 0.5$, pheromone model state–action. Being a non deterministic system, like LPG, the results collected here are the mean values obtained over 100 runs.

In Table 1 and Table 2 results of tests over *Driverlog* and *Rovers* domains are shown. For HSP it has been not possible to collect results in the domain *Rovers* because this planner had some troubles to solve it. In the first column the problem names are listed; in the next columns the length of solution plans and the execution times to reach the best solutions are reported for each planner. For LPG and ACOPlan these columns report average lengths (rounded to the closest integer) and times. The column entitled *HSP -opt* contains the results for HSP called with options guaranteeing the optimality. The symbol – in table entries means that the corresponding problem has not been solved in 2 hours of CPU times or because of memory fault.

Results in Table 1 for the *Driver* domain show how the quality of solutions synthesized by ACOplan is practically always better than the ones extracted by FF and HSP and is often better than the ones extracted by LPG. Only in one case LPG extracts a better solution. For instance, with respect to FF, on the average, the percentage improvement is 15%, with a top of 31%. Moreover, the available data for the optimal version of HSP show how the length of the solution extracted by ACOplan is actually the optimum length.

Results in Table 2 for the *Rovers* domain show similar results where the percentage improvement is 8% with respect to FF and 10% with respect to LPG with a top 15% in both cases.

Table 2. Results for *Rovers* domain collecting solution lengths and CPU time

	ACOplan		**FF**		**LPG**	
Problem	length	time	length	time	length	time
Rover_1	10	0.04	10	0.01	10	0.26
Rover_2	8	0.04	8	0.01	8	0.01
Rover_3	11	0.26	13	0.01	12	0.01
Rover_4	8	0.08	8	0.01	8	0.26
Rover_5	22	0.25	22	0.01	22	0.51
Rover_6	36	26.35	38	0.01	40	0.26
Rover_7	18	0.34	18	0.01	21	0.02
Rover_8	26	0.98	28	0.01	29	0.51
Rover_9	31	160.66	33	0.18	34	0.01
Rover_10	35	38.28	37	0.04	37	0.26
Rover_11	32	207.72	37	0.04	35	0.54
Rover_12	19	1.15	19	0.02	22	0.12
Rover_13	43	156.52	46	0.67	45	1.03
Rover_14	28	80.61	28	0.02	30	2.32
Rover_15	41	979.70	42	0.04	46	1.54
Rover_16	41	250.89	46	0.12	45	2.32
Rover_17	47	5126.78	49	0.16	50	3.75
Rover_18	41	4758.65	42	0.87	48	4.87
Rover_19	65	3427.45	74	0.65	76	5.02
Rover_20	–	–	96	101	0.43	32.76

Nevertheless we have obtained good results from an optimality point of view, the same cannot be said about efficiency. Anyway this is not surprising because we have still a quite simple implementation; on the contrary the number of solved problems with respect to the optimal HSP is encouraging and a dramatic improvement of performances is predictable.

5 Related Works

There are several relevant planners which can be directly related to this work. First of all, we have to refer to LPG [6] because it is based on a stochastic algorithm and it is considered by the planning scientific community a state–of–art planner. It is important to note that stochastic approaches to planning did not have the due attention by the planning community (with respect to the standard deterministic approaches) even if it has been proved they can give very good performances also in the case of optimality problems. Moreover, we have to refer to heuristic planners like HSP [8] and FF [9] for two different reasons: *(i)* the heuristic of our planner is directly inspired from the FF's one, *(ii)* the HSP planner can run in an optimal version and its results could be used also to compare the solution plans given by the planners. Finally, some words have to be spent about the optimality concept in planning. The notion of optimal plan is first introduced as makespan (both as number of actions and as number of

steps) and then it has been refined to consider any metric which can also include resources, time and particular preferences or time trajectory constraints. To the best of our knowledge the optimal version of HSP is the best optimal planner with respect the number of actions of linear plans, so we have included it in our experimental tests.

6 Conclusions and Future Works

In this paper we have described a first application of the Ant Colony Optimization meta–heuristic to Optimal Propositional Planning. The preliminary empirical tests have shown encouraging results and that this approach is a viable method for optimization in classical planning. For these reasons we are thinking to improve and extend this work in several directions.

First of all, we have planned to modify the implementation of the ACO system, in particular the use of heuristic functions requiring a smaller computation time, because it is possible that a less informative and less expensive heuristic function can be used without having a sensitive loss of performance.

Then, another idea is to change the direction of the search in the state space: using "regressing" ants, which start from the goal and try to reach the initial state. Backward search methods has been successfully used in planning.

Finally, we are considering to apply ACO techniques also to other types of planning. The extension of classical planning which appears to be appealing for ACO is planning with numerical fluents: in this framework an objective function can be easily defined. It is almost straightforward to extend our ACO system (with "forward" ants) in order to handle the numerical part of a planning problem, even if it could be problematic to use the complete state in the solution components. Also the extension to handle preferences seems to be straightforward, being necessary only a modification in the computation of $Q(p)$.

Acknowledgements

We acknowledge the usage of computers at IRIDIA (http://code.ulb.ac.be/iridia.home.php) Institute at *Université Libre de Bruxelles*, which Fabio Rossi was visiting from June to August 2008, for some of the computations done here. Moreover, a special thank to Dr. Thomas Stützle for his useful suggestions.

References

1. Nau, D., Ghallab, M., Traverso, P.: Automated Planning: Theory and Practice. Morgan Kaufmann, San Francisco (2004)
2. Hoffmann, J.: The metricff planning system: Translating ignoring delete lists to numerical state variables. Journal of Artificial Intelligence Research. Special issue on the 3rd International Planning Competition 20, 291–341 (2003)
3. Gupta, N., Nau, D.S.: On the complexity of blocks-world planning. Artificial Intelligence 56, 223–254 (1992)

4. Dorigo, M., Stuetzle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
5. Muslea, I.: Sinergy: A linear planner based on genetic programming. In: Steel, S. (ed.) ECP 1997. LNCS, vol. 1348, pp. 312–324. Springer, Heidelberg (1997)
6. Gerevini, A., Serina, I.: LPG: a planner based on local search for planning graphs. In: Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS 2002). AAAI Press, Toulouse (2002)
7. Kautz, H., Selman, B.: Unifying sat-based and graph-based planning. In: Proceedings of IJCAI 1999, Stockholm (1999)
8. Bonet, B., Geffner, H.: Planning as heuristic search. Artificial Intelligence 129(1-2) (2001)
9. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research 14, 253–302 (2001)
10. Dorigo, M., Maniezzo, V., Colorni, A.: Ant System: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics 26(1), 29–41 (1996)
11. Stuetzle, T., Hoos, H.H.: Max min ant system. Future Generation Computer Systems (16), 889–914 (2000)
12. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. Theor. Comput. Sci. 344(2-3), 243–278 (2005)
13. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics – Part B 34(2), 1161–1172 (2004)

An Artificial Immune System for the Multi-Mode Resource-Constrained Project Scheduling Problem

Vincent Van Peteghem and Mario Vanhoucke

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

Abstract. In this paper, an Artificial Immune System (AIS) for the multi-mode resource-constrained project scheduling problem (MRCPSP), in which multiple execution modes are available for each of the activities of the project, is presented. The AIS algorithm makes use of mechanisms which are inspired on the vertebrate immune system performed on an initial population set. This population set is generated with a controlled search method, based on experimental results which revealed a link between predefined profit values of a mode assignment and its makespan. The impact of the algorithmic parameters and the initial population generation method is observed and detailed comparative computational results for the MRCPSP are presented.

1 Introduction

Resource-constrained project scheduling has been a well-known and extensively studied research topic for the past decades. The optimization problem minimizes the makespan of the project, subject to precedence relations between the activities and limited renewable resource availabilities. When introducing different modes for each activity, with for every mode a different duration and different renewable and non-renewable resource requirements, the problem is generalised to the Multi-Mode Resource-Constrained Project Scheduling Problem (MRCPSP).

In the MRCPSP, a set N of n activities is given, where each activity $i \in N = \{1, ..., n\}$, can be performed in different execution modes, $m_i \in M_i = \{1, ..., |M_i|\}$. The duration of activity i, when executed in mode m_i, is d_{im_i}. Each mode m_i also requires $r^{\rho}_{im_ik}$ units for each of the resources in the set R^{ρ} of renewable resource types. For each renewable resource $k \in R^{\rho} = \{1, ..., |R^{\rho}|\}$, the availability a^{ρ}_k is constant throughout the project horizon. Activity i, executed in mode m_i, will also use $r^{\nu}_{im_il}$ nonrenewable resource units of the total available nonrenewable resource a^{ν}_l, with $l \in R^{\nu}$ and R^{ν} the set of nonrenewable resources. Also a set A of pairs of activities between which a precedence relationship exists, is given. The aim of the MRCPSP is to select exactly one mode for each activity in order to schedule the project with a minimal makespan, subject to the resource and precedence constraints. Formally, the MRCPSP can be stated as follows:

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 85–96, 2009.

$$\text{minimize } f_n \tag{1}$$

$$\text{subject to } f_i + d_{jm_j} \leq f_j \qquad \forall (i,j) \in A \tag{2}$$

$$\sum_{i \in S(t)} r^{\rho}_{im_i k} \leq a^{\rho}_k \qquad \forall k \in R^{\rho}, \forall m_i \in M_i, \forall t \tag{3}$$

$$\sum_{i=1}^{n} r^{\nu}_{im_i l} \leq a^{\nu}_l \qquad \forall l \in R^{\nu}, \forall m_i \in M_i \tag{4}$$

$$m_i \in M_i \qquad \forall i \in N \tag{5}$$

$$f_0 = 0 \tag{6}$$

$$f_i \in \text{int}^+ \qquad \forall i \in N \tag{7}$$

where $S(t)$ denotes the set of activities in progress in period $]t-1,t]$ and f_i the finish time of the i^{th} activity.

Several exact, heuristic and meta-heuristic procedures to solve the MRCPSP have been proposed in recent years. The first to present an enumeration scheme for solving the problem was [42], while an enumeration scheme-based procedure was presented by [35]. Branch-and-bound procedures were used by [18], [40] and [41], while a branch-and-cut algorithm was proposed by [45]. Also different single or multi-pass heuristics were presented by [8], [14], [23], [27], [30] and [33]. Genetic algorithms were proposed by [2], [19], [31] and [34] and the simulated annealing approach was used by [9], [22] and [39]. A tabu search procedure was presented by [32] and the methodolgy of particle swarm optimization was introduced by [21] and [44]. Recently, a hybridized scatter search procedure to solve the MRCPSP was proposed by [36].

In this study, we present an Artificial Immune System (AIS), a new search algorithm inspired by the mechanisms of a vertebrate immune system. In the next section the principles of this immune system are described while the proposed solution algorithm and the various AIS parameters are described in section 3. In section 4 the results of the computational experiments are reported as well as the comparison with other heuristics. Finally, the conclusions of the present work are summarized in section 5.

2 Vertebrate Immune System

An Artifical Immune System (AIS) is a computational algorithm proposed by [13] and inspired by theories and components of the vertebrate immune system. The vertebrate immune system is able to identify and kill disease-causing elements, called *antigens*, by the use of immune cells, of which the B-cells are the most common ones. These immune cells have receptor molecules on their surfaces (also called *antibodies*), whose aim is to recognize and bind to pattern-specific antigens.

Since the antibodies on the B-cells are able to kill these specific type of antigens (antibodies and antigens which shapes are complementary will rivit together), the B-cells will be stimulated to proliferate and to mature into non-dividing antibody

secreting cells (plasma cells), according to the principles of *clonal selection*. The degree of proliferation is directly proportional to the recognizing degree of the antigen and the proliferation is succeeded by cell divisions and results in a population of clones which are copies from each other [13].

To better recognize the antigens, a whole mutation and selection process, which is called the *affinity maturation*, is applied on the cloned cells. A first mechanism is the *hypermutation*, a process in which random changes take place in the variable region of the antibody molecules. The degree of hypermutation is inversely proportional to the affinity of the antibody to the antigen: the higher the affinity, the lower the mutation rate and vice versa. However, a large proportion of these mutated antibodies will be of inferior quality and will be non-functional in the immune system. Those cells are eliminated from the population and replaced by newly developed receptors in a process which is called *receptor editing*.

3 AIS Algorithm for the MRCPSP

The efficient mechanisms of an immune system make artificial immune systems useful for scheduling problems. AIS is used for solving job-shop [10][17], flow-shop [15] and resource-constrained project scheduling problems [5]. In this section, a problem-solving technique for the MRCPSP based on the principles of the vertabrate immune system is presented. The different generic steps in our AIS algorithm are presented in Figure 1 and will be discussed along the following subsections.

3.1 Representation

The MRCPSP can be divided into two subproblems: a first subproblem can be refered to as the *Mode Assignment Problem (MAP)*, whose aim is to generate

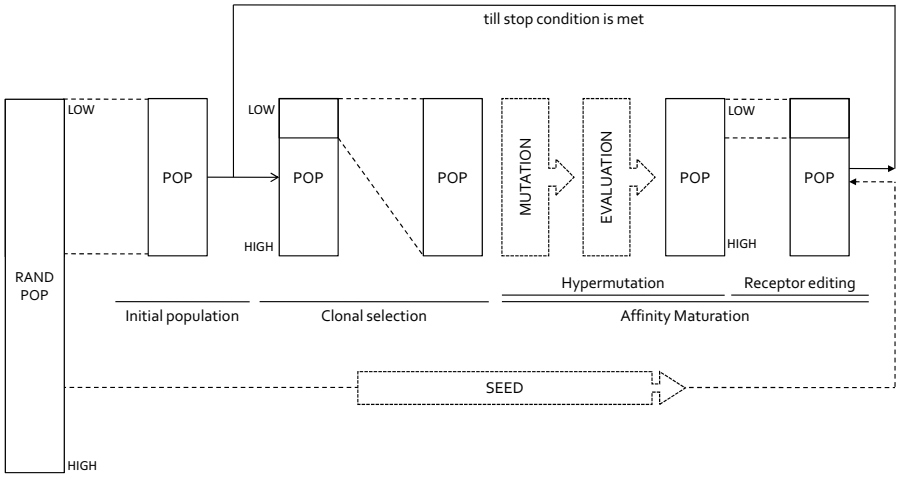


Fig. 1. Artificial Immune System: procedure

a feasible mode assignment list. This mode assignment list assigns a mode to each activity and appoints a duration and a resource consumption to each resource type. The list also determines the feasibility of the mode assignment. A mode assignment which uses more nonrenewable resources than available is called *infeasible*, otherwise the mode assignement is called *feasible*. If there is more than one nonrenewable resource, the problem of finding a feasible solution is NP-complete [27].

In a second subproblem, the order in which the activities needs to be scheduled must be determined. Given the duration and the resource consumptions of the different activities, the aim of the *Resource-constrained Project Scheduling Problem (RCPSP)* is to minimize the makespan of the project. The order in which the activities are scheduled is stored in an activity sequence list. Several representations are available in literature to denote the sequence in which the activities are scheduled [28]. In this paper we make use of the activity list representation, where a precedence-feasible solution is represented by an ordered list of activities. Each activity in the list appears in a position after all its predecessors.

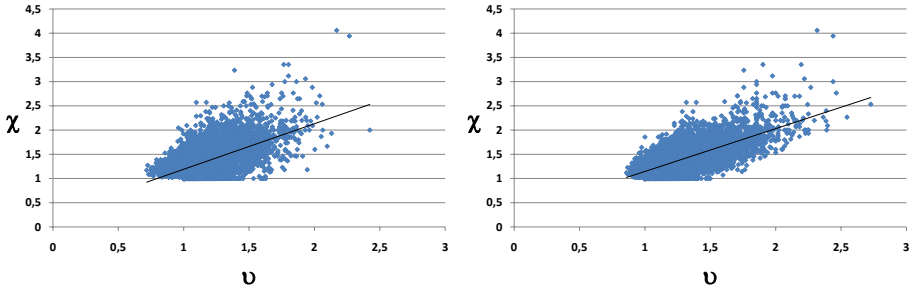
In the MRCPSP, a solution vector V is represented by two lists, a mode assignment list and an activity list respectively.

3.2 Initial Population

Several authors use random techniques to initiate the initial population. They argue that random initial starting solutions are more diverse and use less computational effort than heuristic procedures to produce initial solutions [4]. However, in this paper we use a more controlled generation of the initial population. In a first stage, the mode assignment list is generated, while in a second stage, the activity list is constructed based on this mode assignment list.

Mode assignment list generation. The MAP is very similar to the Multi-Choice Multi-Dimensional Knapsack Problem (MMKP), an extension of the well-known and extensively studied Knapsack Problem. In the MMKP, several classes have several items and each item has a non-negative *profit value* and requires a predefined number of resources. The aim of the MMKP is to pick exactly one item from each class in order to maximize the total profit value of the pick, subject to the resource constraints. Several exact and heuristic solution approaches for this problem have already been proposed [1][20][37].

To translate the MMKP to the MAP, a *profit value* needs to be assigned to each mode/activity combination. If a relationship between the makespan of a project that results from a specific mode assignment and its profit value can be found, the search space of the problem can be reduced to the most promising search regions. Experimental tests were performed on 90 problem instances from the test dataset as defined in section 4.1. For every problem instance 100 feasible and unique mode assignment lists were generated randomly. Based on the information of each mode assignment, a schedule was build using the bi-population genetic algorithm of [12]. Computational experience revealed that there is a



(a) Total work content as profit value (b) Sum of durations as profit value

Fig. 2. Relationship between makespan and profit value

direct proportional relationship between the makespan and the following two profit values:

- **Total Work Content** (TWC), defined as $\sum_{i=1}^{n} \sum_{k=1}^{|R^\rho|} r_{im_ik}^{\rho} d_{im_i}$
- **Sum of Durations** (SUM), defined as $\sum_{i=1}^{n} d_{im_i}$

In Figure 2, a graph is plotted with the results of these tests. To standardize the values on the vertical and horizontal axis, the following calculations were made:

- On the vertical axis, χ is shown. For each problem instance and for each generated mode assignment list, χ is calculated as the ratio of the makespan which results from a particular mode assignment to the best found makespan. The larger χ, the more the makespan deviates from the best found makespan of that problem instance. The χ-value of the mode assignment list which results in the best makespan is 1 and obviously, the value of χ can never be lower than 1.
- On the horizontal axis, the value υ is shown. υ is calculated as the ratio of the profit value of the mode assignment to the profit value of the mode assignment which results in the schedule with the best found makespan. υ can be lower than 1 since the profit value of a mode assignment with a higher makespan can be lower than the profit value of the mode assignment of the best makespan.

In both graphs a positive relationship between the profit value and the makespan can be observed. A Pearson correlation test revealed that the correlations for the TWC and SUM are 0.61 and 0.74, respectively.

Since there is a clear relationship between the profit value and the makespan, mode assignment lists with low profit values are preferred to mode assignment lists with high profit values. Therefore, to start with a good, but also diverse initial population, we create a start population, called $RANDPOP$, with a large number of randomly generated feasible mode assignment lists (in this paper: 4 times the number of elements in our population). For each mode assignment list

the corresponding profit value is calculated and the *POP* mode assignment lists with the lowest overall profit values are selected for use in the initial population. Once the mode assignment lists are generated, a duration and resource consumption can be assigned to each activity for each population element. Based on this information, the activity list can be generated.

Activity list generation. In the second stage, the activity list is generated. Several heuristics available in literature make use of well-performing priority rules to generate the sequence in which the activities should be scheduled. Some of the best performing priority rules are the *Latest Finish Time (LFT)*, the *Latest Start Time (LST)*, the *Minimum Slack (SLK)* and the *Maximum Remaining Work (RWK)*. More information about these priority rules can be found in [26] and [6].

For each mode assignment list, an activity list is generated using one of the proposed priority rules. Since different mode assignment lists are generated, also a diverse set of activity lists will be generated. If two or more equal mode assignment lists occur in the initial population, different priority rules are applied on the mode assignment lists to avoid equal solution vectors in the initial population.

After the generation of the activity list and the mode assignment list, a schedule is build for each of the *POP* population elements in the initial population.

3.3 Clonal Selection Process

A population of *POP* solution vectors is generated and for each solution vector the makespan is determined. Only the best $P_{clonal}\%$ solution vectors will be available for proliferation. For these solution vectors, the corresponding affinity value is determined as follows:

$$Affinity(V) = \frac{1}{2(makespan(V) - bestmakespan + 1)} \tag{8}$$

where *makespan(V)* refers to the makespan of solution vector V and *bestmakespan* refers to the best makespan found so far. The number of clones of a solution vector V in the population is given by the affinity of the solution vector V over the sum of affinities of all population solution vectors multiplied by the number of population elements. Since the cloning of the antibodies is done directly proportional to their affinity value, it can be noticed that solution vectors with higher makespans will appear less frequent than solution vectors with low makespans. The size of the antibody population is fixed and infeasible solution vectors can not be proliferated.

3.4 Affinity Maturation

After the proliferation, the affinity maturation is performed. This process is applied in two phases: first, a hypermutation procedure is applied on each solution vector of the population and afterwards the receptor editing mechanism is used.

Hypermutation. Since a solution vector contains both an activity list and a mode assignment list, a mutation process is applied on both lists. However, the process for both lists is different.

For the mutation on the *activity list*, a hypermutation rate η which defines the degree of modification in the activity list is calculated. The hypermutation rate for solution vector V can be formulated as follows:

$$\eta(V) = 100e^{-0.05*(makespan(V)-bestmakespan)} \tag{9}$$

The number of mutations is calculated as:

$$NumbMut(V) = 1 + \frac{(100 - \eta(V))n}{100} \tag{10}$$

The lower the makespan, the lower the hypermutation rate of the activity list will be and the lower the number of mutations. By applying this mutation process, the algorithm can explore the neighbourhood of the solution. That neighbourhood will expand when the hypermutation rate increases. A mutation is defined as follows: in the current activity list, an activity is chosen and is moved randomly to a new position. In order to respect the precendence constraints, the new position of the activity is lying between the position of its latest predecessor and the position of its earliest successor.

The mutation process for the *mode assignment list* is based on a frequency matrix of the mode assignment lists in the population. To assign a mode to an activity, a random population element is chosen and the mode for that population element activity is assigned to the cloned element activity. Modes that occur more in the population of good solutions will therefore occur more. This procedure shows similarities with the Harmony Search procedure [16]. In case the mode assignment list becomes infeasible, a *feasibility procedure* is applied such that the mode list becomes feasible by changing mode assignments and such that the modes with the lowest profit values (as defined in the previous section) are chosen first. After the mutation process, every new feasible solution vector is evaluated.

Receptor Editing. After the cloning and mutation process, a new population of antibodies is generated. Only the best $P_{editing}\%$ antibodies are preserved in the population. The other elements are eliminated and to preserve POP elements in the population, the population is seeded with high quality mode assignment lists from the start population $RANDPOP$. In that way, schedules with a low makespan stay incorporated in the antibody population and antibodies evolving to inferior search regions are deleted. The newly generated population is the start population for a new generation process. This process continues until the stop condition is met.

4 Computational Results

In order to prove the efficiency of our algorithm, we have tested our AIS solution procedure on a test set which is defined in section 4.1. In this section,

the impact of the different algorithmic parameters, such as P_{clonal}, $P_{editing}$ and the population size POP, is tested and the efficiency of our initial population generation method is proven. In section 4.2, we compare the proposed solution method with other metaheuristic procedures available in literature based on the well-known PSPLIB benchmark dataset [25].

4.1 Parameter Setting

For the generation of the instances of the test set, we have used the RanGen project scheduling instances generator developed by [43] and extended the projects to a multi-mode version. Each instance contains 20 activities, with three modes, two renewable and two non-renewable resources. The instances have been generated with the following settings: the order strength is set at 0.25, 0.50 or 0.75, the (renewable and nonrenewable) resource factor at 0.50 or 1 and the (renewable and nonrenewable) resource strength at 0.25, 0.50 or 0.75. Using 5 instances for each problem class, we obtain a problem set with 540 network instances. The project parameters are explained in [43], amongst others.

Extensive testing revealed that the optimal values for the different algoritmic parameters are P_{clonal}=25%, $P_{editing}$=20% and POP=450. Table 1 shows the average deviation from the minimal critical path *after the initial population generation* of POP population elements and *after 5000 schedules*. The 2 different profit values (TWC and SUM) and the 4 different priority rules (LFT, LST, SLK and RWK) are compared to the result of the solutions in which a random generated initial population is used.

Table 1. Average % deviation from minimal critical path

	After initial generation			**After 5000 schedules**		
	Profit values			*Profit values*		
Priority rules	Random	TWC	SUM	Random	TWC	SUM
Random	48.23%	42.60%	42.43%	22.42%	20.81%	20.52%
LFT	43.24%	38.32%	37.85%	22.14%	20.86%	20.55%
LST	42.35%	37.62%	**37.17%**	22.44%	20.97%	20.74%
SLK	44.86%	39.56%	39.13%	22.29%	21.06%	20.73%
RWK	42.52%	37.95%	37.51%	22.22%	20.95%	**20.50%**

A paired-samples T-test revealed a very significant ($p<0.01$) influence on the quality of the schedules using a controlled initial population generation method instead of a random generation method: the difference between the average deviation from the critical path for the random generated solutions (48.23%) and the average deviation for the controlled generated elements (LST/SUM - 37.17%) is 11.06%, which corresponds with an average decrease of 2.2 working days on an average makespan of 28 days (*information not available in Table 1*).

Regardless of which priority rule is used, the combination in which SUM is used as profit value always outperforms the other profit value TWC. This result is in accordance to the results of [6], who proposed several priority rules for the

mode assignment problem and who concluded that choosing the shortest feasible execution-mode is the most appropriate rule to minimize project duration.

The best solution after 5000 schedules is found for the combination RWK/SUM (20.50%). There is a significant difference ($p<0.01$) with the solution in which a random generated initial population is used. In the further course of this paper, the *Maximum Remaining Work (RWK)* will be used as priority rule and the *Sum of Durations (SUM)* will be used as profit value in our AIS algorithm.

4.2 Comparison

In this section, the algorithm is compared to the best performing heuristics and metaheuristics in published literature. As a benchmark, we use the well-known PSPLIB dataset [25] to compare with other existing procedures from the literature. The dataset for the multi-mode RCPSP contains project instances with 10, 12, 14, 16, 18, 20 and 30 activities, each with 2 renewable and 2 nonrenewable resources. For the instances with 30 activities only a set of best known heuristic solutions is available, for the other instances the optimal solutions are available. For each problem size, 640 instances were generated.

Table 2. Comparison with other heuristics - average % optimality gap - 5000 schedules

	J10	J12	J14	J16	J18	J20	J30
Jozefowska et al. (2001)	1.16	1.73	2.60	4.07	5.52	6.74	11.67
Alcaraz et al. (2003)	0.24	0.73	1.00	1.12	1.43	1.91	n.a.
Ranjbar et al. (2008)	0.18	0.65	0.89	0.95	1.21	1.64	n.a.
Jarboui et al. (2008)	0.03	0.09	0.36	0.44	0.89	1.10	2.35
This work	**0.02**	**0.07**	**0.20**	**0.39**	**0.52**	**0.70**	**1.55**

Table 3. Deviation measures - 5000 schedules

	J10	J12	J14	J16	J18	J20	J30
Max. deviation (%)	4.16	7.14	7.14	9.52	7,41	9.52	11.90
Optimal (%)	99.44	98.35	95.10	90.36	86.23	81.59	65.22

A comparison of the different algorithms on the J10 to J30 datasets of PSPLIB is made for the authors mentioned in Table 2. For the J10 to J20 datasets the average deviation from the optimal solution is shown, for the J30 dataset the average deviation from the best known solutions is presented. To make a fair comparison between the different heuristics, the evaluation is stopped after 5000 generated schedules. As [29] stated, one schedule corresponds to (at most) one start time assignment per activity. The number of generated schedules can be calculated as the sum of times each activity of the project has obtained a feasible start time divided by the number of activities of the project. As can be seen in Table 2, the results for our algorithm outperform the other heuristics. Table 3 shows the maximal deviation from the optimal solution for each instance set, as well as the percentage of instances for which an optimal solution was found.

5 Conclusions

In this paper, an artifical immune system is presented. The vertebrate immune system mechanisms which inspire AIS as solution methodology were used to solve the multi-mode resource-constrained project scheduling problem. To generate a good and diverse initial population, a controlled search procedure is used, which is based on an observed link between predefined profit values and the makespan of the mode assignment and which leads the search process more quickly to more interesting search regions. The proposed AIS algorithm proves its effectiveness as problem solving technique for the MRCPSP by generating competitive results for the different PSPLIB datasets. The introduction of this algorithm in other project scheduling problems such as DTRTP can lead to promising results.

References

1. Akbar, M.M., Rahman, M.S., Kaykobad, M., Manning, E.G., Shoja, G.C.: Solving the Multidimensional Multiple-Choice Knapsack Problem by constructing convex hulls. Computers and Operations Research 33, 1259–1273 (2006)
2. Alcaraz, J., Maroto, C., Ruiz, R.: Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. Journal of the Operational Research Society 54, 614–626 (2003)
3. Alvarez-Valdes, R., Tamarit, J.M.: Heuristic algorithms for resource-constrained project scheduling: A review and an empirical analysis. In: Slowinski, R., Weglarz, J. (eds.) Advances in Project Scheduling, pp. 113–134. Elsevier, Amsterdam (1989)
4. Anderson, E.J., Ferris, M.C.: Genetic Algorithm for Combinatorial Optimisation: The Assembly Line Balancing Problem. ORSA Journal on Computing 6, 161–173 (1994)
5. Argawal, R., Tiwari, M.K., Mukherjee, S.K.: Artificial Immune System Based Approach for Solving Resource Constraint Project Scheduling Problem. International Journal of Advanced Manufacturing Technology 34, 584–593 (2007)
6. Boctor, F.: Heuristics for scheduling projects with resource restrictions and several resource-duration modes. International Journal of Production Research 31, 2547–2558 (1993)
7. Boctor, F.: An adaption of the simulated annealing for solving resource-constrained project scheduling problems. International Journal of Production Research 34, 2335–2351 (1996)
8. Boctor, F.: A new and effcient heuristic for scheduling projects with resource restrictions and multiple execution modes. European Journal of Operational Research 90, 349–361 (1996)
9. Bouleimen, K., Lecocq, H.: A new effcient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. European Journal of Operational Research 149, 268–281 (2003)
10. Coello Coello, C.A., Rivera, D.C., Cortés, N.C.: Use of an Artificial Immune System for Job Shop Scheduling. In: Timmis, J., Bentley, P.J., Hart, E. (eds.) ICARIS 2003. LNCS, vol. 2787, pp. 1–10. Springer, Heidelberg (2003)
11. Davis, E.W., Patterson, J.H.: A comparison of heurstic and optimum solutions in resource-constrained project scheduling. Management Science 21, 944–955 (1975)

12. Debels, D., Vanhoucke, M.: A bi-population based genetic algorithm for the resource-constrained project scheduling problem. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganá, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3483, pp. 378–387. Springer, Heidelberg (2005)
13. De Castro, L.N., Timmis, J.I.: Artificial immune systems: a novel paradigm for pattern recognition. In: Alonso, L., Corchado, J., Fyfe, C. (eds.) Artificial Neural Networks in Pattern Recognition, pp. 67–84. University of Paisley (2002)
14. Drexl, A., Grünewald, J.: Nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 25, 74–81 (1993)
15. Engin, O., Döyen, A.: A New Approach to Solve Hybrid Flow Shop Scheduling Problems by Artificial Immune System. Future Generation Computer Systems 20, 1083–1095 (2004)
16. Geem, Z.W., Kim, J.H., Loganathan, G.V.: A New Heuristic Optimization Algorithm: Harmony Search. Simulation 76, 60–68 (2001)
17. Hart, E., Ross, P., Nelson, J.: Producing robust schedules via an artificial immune system. In: Proceedings of the ICEC 1998, pp. 464–469 (1998)
18. Hartmann, S., Drexl, A.: Project scheduling with multiple modes: a comparison of exact algorithms. Networks 32, 283–297 (1998)
19. Hartmann, S.: Project scheduling with multiple modes: a genetic algorithm. Annals of Operations Research 102, 111–135 (2001)
20. Hifi, M., Michrafy, M., Sbihi, A.: Heuristic algorithms for the multiple-choice multidimensional knapsack problem. Journal of Operational Research Society 55, 1323–1332 (2004)
21. Jarboui, B., Damak, N., Siarry, P., Rebai, A.: A combinatorial particle swarm optimization for solving multi-mode resource-constrained project scheduling problems. Applied Mathematics and Computation 195, 299–308 (2008)
22. Jozefowska, J., Mika, M., Rozycki, R., Waligora, G., Weglarz, J.: Simulated annealing for multi-mode resource-constrained project scheduling. Annals of Operations Research 102, 137–155 (2001)
23. Knotts, G., Dror, M., Hartman, B.: Agent-based project scheduling. IIE Transactions 32, 387–401 (2000)
24. Kolisch, R., Sprecher, A., Drexl, A.: Characterization and generation of a general class of resource-constrained project scheduling problems. Management Science 41, 1693–1703 (1995)
25. Kolisch, R., Sprecher, A.: PSPLIB - a project scheduling problem library. European Journal of Operational Research 96, 205–216 (1996)
26. Kolisch, R.: Serial and parallel resourceconstrained project scheduling methods revisited: Theory and computation. European Journal of Operational Research 90, 320–333 (1996)
27. Kolisch, R., Drexl, A.: Local search for nonpreemptive multi-mode resource-constrained project scheduling. IIE Transactions 29, 987–999 (1997)
28. Kolisch, R., Hartmann, S.: Project Scheduling - Recent Models, Algorithms and Applications. Kluwer Academic Publishers, Boston (1999)
29. Kolisch, R., Hartmann, S.: Experimental investigation of heuristics for resource-constrained project scheduling: An update. European Journal of Operational Research 174, 23–37 (2006)
30. Lova, A., Tormos, P., Barber, F.: Multi-mode resource-constrained project scheduling: Scheduling schemes, priority rules and mode selection rules. Inteligencia Artificial 30, 69–86 (2006)

31. Mori, M., Tseng, C.: A genetic algorithm for multi-mode resource-constrained project scheduling problem. European Journal of Operational Research 100, 134–141 (1997)
32. Nonobe, K., Ibaraki, T.: Formulation and tabu search algorithm for the resource-constrained project scheduling problem (RCPSP). Technical report, Kyoto University (2001)
33. Özdamar, L., Ulusoy, G.: A local constraint based analysis approach to project scheduling under general resource constraints. European Journal of Operational Research 79, 287–298 (1994)
34. Özdamar, L.: A genetic algorithm approach to a general category project scheduling problem. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 29, 44–59 (1999)
35. Patterson, J., Slowinski, R., Talbot, F., Weglarz, J.: Advances in Project Scheduling. Elsevier, Amsterdam (1989)
36. Ranjbar, M., De Reyck, B., Kianfar, F.: A hybrid scatter-search for the discrete time/resource trade-off problem in project scheduling. European Journal of Operational Research 193, 35–48 (2009)
37. Sbihi, A.: A best first search exact algorithm for the Multiple-choice Multidimensional Knapsack Problem. Journal of Combinatorial Optimization 13, 337–351 (2007)
38. Shahriar, A.Z.M., Akbar, M.M., Rahman, M.S., Newton, M.A.H.: A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. Journal of supercomputing 43, 257–280 (2008)
39. Slowinski, R., Soniewicki, B., Weglarz, J.: DSS for multiobjective project scheduling. European Journal of Operational Research 79, 220–229 (1994)
40. Sprecher, A., Hartmann, S., Drexl, A.: An exact algorithm for project scheduling with multiple modes. OR Spektrum 19, 195–203 (1997)
41. Sprecher, A., Drexl, A.: Solving multi-mode resource-constrained project scheduling problems by a simple, general and powerful sequencing algorithm. European Journal of Operational Research 107, 431–450 (1998)
42. Talbot, F.: Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. Management Science 28(10), 1197–1210 (1982)
43. Vanhoucke, M., Coelho, J., Debels, D., Maenhout, B., Tavares, L.: An evaluation of the adequacy of project network generators with systematically sampled networks. European Journal of Operational Research 187, 511–524 (2008)
44. Zhang, H., Tam, C., Li, H.: Multimode project scheduling based on particle swarm optimization. Computer-Aided Civil and Infrastructure Engineering 21, 93–103 (2006)
45. Zhu, G., Bard, J., Tu, G.: A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem. Journal on Computing 18, 377–390 (2006)

Beam-ACO Based on Stochastic Sampling for Makespan Optimization Concerning the TSP with Time Windows*

Manuel López-Ibáñez[1], Christian Blum[1], Dhananjay Thiruvady[2,3], Andreas T. Ernst[3], and Bernd Meyer[2]

[1] ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain
{m.lopez-ibanez,cblum}@lsi.upc.edu
[2] Calyton School of Information Technology, Monash University, Australia
{dhananjay.thiruvady,bernd.meyer}@infotech.monash.edu.au
[3] CSIRO Mathematics and Information Sciences, Australia
andreas.ernst@csiro.au

Abstract. The travelling salesman problem with time windows is a difficult optimization problem that appears, for example, in logistics. Among the possible objective functions we chose the optimization of the makespan. For solving this problem we propose a so-called Beam-ACO algorithm, which is a hybrid method that combines ant colony optimization with beam search. In general, Beam-ACO algorithms heavily rely on accurate and computationally inexpensive bounding information for differentiating between partial solutions. In this work we use stochastic sampling as an alternative to bounding information. Our results clearly demonstrate that the proposed algorithm is currently a state-of-the-art method for the tackled problem.

1 Introduction

The travelling salesman problem with time windows (TSPTW) [1] seeks to find an efficient route to visit a number of customers, starting and ending at a depot, with the added difficulty that each customer may only be visited within a certain time window. In practice, the TSPTW is an important problem in logistics. The TSPTW is proven to be NP-hard, and even finding a feasible solution is an NP-complete problem [2]. The problem is closely related to a number of important problems. For example, the well-known travelling salesman problem (TSP) is a special case of the TSPTW. The TSPTW itself can be seen as a special case with a single vehicle of the vehicle routing problem with time windows (VRPTW). The literature mentions two different objective functions for this problem. In this work we chose the optimization of the makespan as the objective. The ant

* This work was supported by grant TIN2007-66523 (FORMALISM) of the Spanish government. Moreover, Christian Blum acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Innovation.

C. Cotta and P. Cowling (Eds.): EvoCOP 2009, LNCS 5482, pp. 97–108, 2009.

colony optimization approaches from [3,4] are among the current state-of-the-art algorithms for the TSPTW when optimizing the makespan.

Ant colony optimization (ACO) is a metaheuristic that is based on the probabilistic construction of solutions [5]. At each algorithm iteration, a number of solutions are constructed independently of each other. A recently proposed ACO hybrid, known as Beam-ACO [6,7], employs at each iteration a probabilistic beam search procedure that constructs a number of solutions interdependently and in parallel. At each step, beam search keeps a certain number of the best partial solutions available for further extension [8]. These partial solutions are selected with respect to bounding information. Hence, accurate and inexpensive bounding information is a crucial component of beam search. A problem arises when the bounding information is either misleading or when this information is computationally expensive, which is the case for the TSPTW. López-Ibáñez and Blum [9] presented a first study with the aim to show that *stochastic sampling* [10,11] is a useful alternative to bounding information. Hereby, each given partial solution is completed a certain number of times in a stochastic way. The information that is obtained in this way is used to differentiate between different partial solutions.

In this work, apart from comparing our results to the best known ones from the literature, we also study the effect that different components of Beam-ACO with stochastic sampling have on the performance of the algorithm. In particular, we will evaluate the influence of the pheromone information and the effects of different degrees of stochastic sampling. The remainder of this work is organized as follows. In Section 2 we give a technical description of the TSPTW. Section 3 introduces the Beam-ACO algorithm for the TSPTW. In Section 4 we describe the experimental evaluation, and in Section 5 we offer conclusions and an outlook to future work.

2 The TSP with Time Windows

The TSPTW is formally defined as follows. Given an undirected complete graph $G = (N, A)$—where $N = \{0, 1, \ldots, n\}$ is a set of nodes representing the depot (node 0) and n customers, and $A = N \times N$ is the set of edges connecting the nodes—a solution to the problem is a tour visiting each node once, starting and ending at the depot. Hence, a tour is represented as $P = (p_0 = 0, p_1, \ldots, p_n, p_{n+1} = 0)$, where the sub-sequence $(p_1, \ldots, p_k, \ldots, p_n)$ is a permutation of the nodes in $N \backslash \{0\}$ and p_k denotes the index of the customer at the k^{th} position of the tour. Two additional elements, $p_0 = 0$ and $p_{n+1} = 0$, represent the starting depot and the final depot.

For every edge $a_{ij} \in A$ between two nodes i and j, there is an associated cost $c(a_{ij})$. This cost typically represents the travel time between customers i and j, plus a service time at customer i.

Furthermore, there is a time window $[e_i, l_i]$ associated to each node $i \in N$, which specifies that customer i cannot be serviced before e_i or visited later than l_i. In most formulations of the problem, waiting times are permitted, that is, a

node i can be reached before the start of its time window e_i, but cannot be left before e_i. Therefore, given a particular tour P, the departure time from customer p_k is calculated as $D_{p_k} = \max(A_{p_k}, e_{p_k})$, where $A_{p_k} = D_{p_{k-1}} + c(a_{p_{k-1},p_k})$ is the arrival time at the customer p_k in the tour.

Two different but related objectives for this problem are found in the literature. One is the minimization of the cost of the edges traversed along the tour. The other alternative is to minimize $A_{p_{n+1}}$, that is, the arrival time at the depot. The first objective is analogous to the objective of the TSP, while the second is similar to the concept of a *makespan* in scheduling problems. In this paper, we focus on the latter. Hence, we formally define the TSPTW as:

$$\min F(P) = A_{p_{n+1}} \ , \tag{1}$$

subject to $\Omega(P) = \sum_{k=0}^{n+1} \omega(p_k) = 0$, where $\omega(p_k) = 1$ if $A_{p_k} > l_{p_k}$, and 0 otherwise. Note that $A_{p_{n+1}}$ is recursively computed as $A_{p_{k+1}} = \max(A_{p_k}, e_{p_k}) + c(a_{p_k,p_{k+1}})$. In the above definition, $\Omega(P)$ denotes the number of time window constraints that are violated by tour P, which must be zero for feasible solutions.

3 The Beam-ACO Algorithm

In the following we first explain the solution construction, which is the crucial part of our Beam-ACO algorithm. The application of the (Beam-)ACO framework to any problem implies the definition of a pheromone model $\mathcal{T}$ and a solution construction mechanism. In fact, we need a solution construction mechanism for the probabilistic beam search as well as for stochastic sampling.

Stochastic Sampling. Given a partial solution, an ant a chooses at each construction step one customer j among the set $\mathcal{N}(P_a)$ of customers not included yet in the current partial tour P_a. Once all customers have been added to the tour, it is completed by adding node 0. The decision of which customer to choose at each step is done with the help of pheromone information and heuristic information. As for the pheromone information, $\forall a_{ij} \in A$, $\exists \tau_{ij} \in \mathcal{T}$, $0 \leq \tau_{ij} \leq 1$, where τ_{ij} represents the desirability of visiting customer j after customer i in the tour. The greater the pheromone value τ_{ij}, the greater is the desirability of choosing j as the next customer to visit in the current tour.

The decision of which customer to choose is made by firstly generating a random number q uniformly distributed within $[0, 1]$ and comparing this value with a parameter q_0 called the determinism rate. If $q \leq q_0$, j is chosen deterministically as the value with the highest product of pheromone and heuristic information, that is, $j = \arg\max_{k \in \mathcal{N}(P_a)}\{\tau_{ik} \cdot \eta_{ik}\}$, where i is the last customer added to the tour P_a, and η_{ij} is the heuristic information that represents an estimation of the benefit of visiting customer j after customer i. Otherwise, j is stochastically chosen from the following distribution of probabilities:

$$\mathbf{p}_i(j) = \frac{\tau_{ij} \cdot \eta_{ij}}{\sum_{k \in \mathcal{N}(P_a)} \tau_{ik} \cdot \eta_{ik}} \quad \text{if } j \in \mathcal{N}(P_a) \tag{2}$$

There are several heuristics that could be used for the TSPTW. When deciding which customer should be visited next, not only a small travel cost between customers (c_{ij}) is desirable, but also those customers whose time window finishes sooner should be given priority to avoid constraint violations. In addition, visiting those customers whose time window starts earlier may prevent waiting times. Hence, we use a heuristic information that combines the travel cost between customers, the latest service time (l_j) and the earliest service time (e_j). The values are first normalized to $[0, 1]$, with the maximum value corresponding to 0 and the minimum to 1, and then combined:

$$\eta_{ij} = \lambda^c \frac{c^{\max} - c_{ij}}{c^{\max} - c^{\min}} + \lambda^l \frac{l^{\max} - l_j}{l^{\max} - l^{\min}} + \lambda^e \frac{e^{\max} - e_j}{e^{\max} - e^{\min}} \tag{3}$$

where $\lambda^c + \lambda^l + \lambda^e = 1$ are weights that allow to balance the importance of each heuristic. In earlier experiments, we found out that no single combination of weights would perform optimally across all instances in our benchmark set. Therefore, we decided to define the weights randomly for each application of probabilistic beam search.

The solution construction mechanism described above may result in the construction of infeasible solutions. Therefore, it is necessary to define a way of comparing between different—possibly infeasible—solutions. This will be done lexicographically ($<_{lex}$) by first minimising the number of constraint violations (Ω) and, in the case of an equal number of constraint violations, by comparing the tour cost (F). More formally, we compare two different solutions P and P' as follows:

$$P <_{\text{lex}} P' \iff \Omega(P) < \Omega(P') \vee (\Omega(P) = \Omega(P') \wedge F(P) < F(P')) \tag{4}$$

Probabilistic Beam Search. The probabilistic beam search that we developed for the TSPTW is described in Algorithm 1. The algorithm requires three input parameters: $k_{\text{bw}} \in \mathbb{Z}^+$ is the so-called *beam width*, $\mu \in \mathbb{R}^+ \geq 1$ is a parameter that determines the number of children that can be chosen at each step, and N^{s} is the number of *stochastic samples* taken for evaluating a partial solution. Moreover, B_t denotes a set of partial tours called the *beam*. Hereby, index t denotes the current iteration of the beam search. At any time it holds that $|B_t| \leq k_{\text{bw}}$, that is, the beam is smaller than or equal to the beam width. A problem-dependent greedy function $\nu()$ is utilized to assign a weight to partial solutions.

At the start of the algorithm the beam only contains one partial tour starting at the depot, that is, $B_0 := \{(0)\}$. Let $C := \mathcal{C}(B_t)$ denote the set of all possible extensions of the partial tours in B_t. A partial tour P may be extended by adding a customer j not yet visited by that tour. Such a candidate extension of a partial tour is henceforth denoted by $\langle P, j \rangle$. At each iteration, at most $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ candidate extensions are selected from C by means of the procedure $\mathsf{ChooseFrom}(C)$ to form the new beam B_{t+1}. At the end of each step, the new beam B_{t+1} is reduced by means of the procedure Reduce in case it contains more than k_{bw} partial solutions. When the current iteration is equal to the number of customers ($t = n$), all elements in B_n are completed by adding the depot, and finally the best solution is returned.

Algorithm 1. Probabilistic Beam search (PBS) for the TSPTW

```
1: B_0 := {(0)}
2: randomly define the weights λ^c, λ^l, and λ^e
3: for t := 0 to n do
4:    C := C(B_t)
5:    for k = 1, ..., min{⌊μ · k_bw⌋, |C|} do
6:       ⟨P, j⟩ := ChooseFrom(C)
7:       C := C \ ⟨P, j⟩
8:       B_{t+1} := B_{t+1} ∪ ⟨P, j⟩
9:    end for
10:   B_{t+1} := Reduce(B_{t+1}, k_bw)
11: end for
12: output: arg min_lex {T | T ∈ B_n}
```

The procedure ChooseFrom(C) chooses a candidate extension $\langle P, j\rangle$ from C, either deterministically or probabilistically according to the determinism rate q_0. More precisely, for each call to ChooseFrom(C), a random number q is generated and if $q \leq q_0$ then the decision is taken deterministically by choosing the candidate extension that maximises the product of the pheromone information $\mathcal{T}$ and the greedy function $\nu()$: $\langle P, j\rangle = \arg\max_{\langle P',k\rangle \in C} \tau(\langle P', k\rangle) \cdot \nu(\langle P', k\rangle)^{-1}$, where $\tau(\langle P', k\rangle)$ corresponds to the pheromone value $\tau_{ik} \in \mathcal{T}$, supposing that i is the last customer visited in tour P'.

Otherwise, if $q > q_0$, the decision is taken stochastically according to the following probabilities:

$$\mathbf{p}(\langle P, j\rangle) = \frac{\tau(\langle P, j\rangle) \cdot \nu(\langle P, j\rangle)^{-1}}{\sum_{\langle P',k\rangle \in C} \tau(\langle P', k\rangle) \cdot \nu(\langle P', k\rangle)^{-1}} \tag{5}$$

The greedy function $\nu(\langle P, j\rangle)$ assigns a heuristic value to each candidate extension $\langle P, j\rangle$. In principle, for this purpose we could use the heuristic η given by Eq. (3), that is, $\nu(\langle P, j\rangle) = \eta(\langle P, j\rangle)$. As in the case of the pheromone information, the notation $\eta(\langle P, j\rangle)$ refers to the value of η_{ik} as defined in Eq. (3), supposing that i was the last customer visited in tour P. However, when comparing two extensions $\langle P, j\rangle \in C$ and $\langle P', k\rangle \in C$, the value of η might be misleading in case $P \neq P'$. We solved this problem by defining the greedy function $\nu()$ as the sum of the ranks of the heuristic information values that correspond to the construction of the extension. For an example see Fig. 1. The edge labels of the search tree are tuples that contain the (fictious) values of the heuristic information (η) in the first place, and the corresponding rank in the second place. For example, the extension 2 of the partial solution (1), denoted by $\langle(1), 2\rangle$ has greedy value $\nu(\langle(1), 2\rangle) = 1 + 2 = 3$.

Finally, the application of procedure Reduce(B_t) removes the worst $\max\{|B_t| - k_{\text{bw}}, 0\}$ partial solutions from B_t. As mentioned before, we use *stochastic sampling* for evaluating partial solutions. More specifically, for each partial solution, a number N^{s} of complete solutions is sampled as explained in the paragraph above

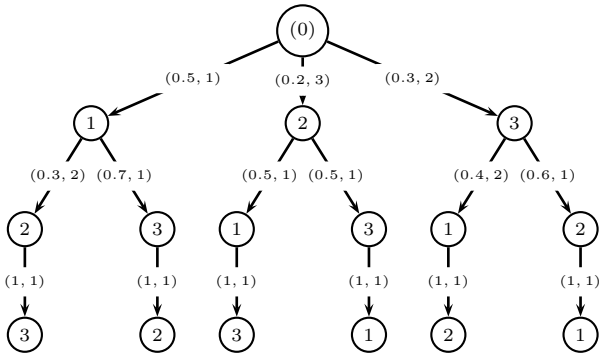


Fig. 1. Search tree corresponding to a problem instance with three customers. Edge labels are tuples that contain the heuristic information (η) in the first place, and the corresponding rank in the second place.

on stochastic sampling. The value of the best of these samples (with respect to Eq. 4) is used for evaluating the corresponding partial solution. Only the k_{bw} best partial solutions (with respect to their corresponding best samples) are kept in B_t and the others are discarded.

3.1 Beam-ACO Framework

The probabilistic beam search outlined in the previous section is used to construct solutions within an ACO algorithm that is implemented in the hyper-cube framework [12]. A high level description of the algorithm is given in Algorithm 2. The data structures used, in addition to counters and to the pheromone values, are: (*1*) the *best-so-far* solution P^{bf}, that is, the best solution generated since the start of the algorithm; (*2*) the *restart-best* solution P^{rb}, that is, the best solution generated since the last restart of the algorithm; (*3*) the *convergence factor* (*cf*), $0 \leq cf \leq 1$, which is a measure of how far the algorithm is from convergence; and (*4*) the Boolean variable *bs_update*, which becomes true when the algorithm reaches convergence.

Roughly, the algorithm works as follows. Initially, all variables are initialized. In particular, the pheromone values are set to their initial value 0.5. Then, a main loop is repeated until a termination criteria, such as a CPU time limit, is met. Each algorithm iteration consists of the following steps. First, a probabilistic beam search algorithm is executed, which returns solution P^{ib}. Then, after updating the best-so-far solution, a new value for the convergence factor *cf* is computed. Depending on this value, as well as on the value of the Boolean variable *bs_update*, a decision on whether to restart the algorithm or not is made. If the algorithm is restarted, all the pheromone values are reset to their initial value (0.5). The algorithm is iterated until the CPU time limit is reached. Once terminated, the algorithm returns the best solution found which corresponds to P^{bf}. In the following we describe the two remaining procedures of Algorithm 2 in more detail.

Algorithm 2. ACO algorithm for the TSPTW

input: $N^{\mathrm{s}}, k_{\mathrm{bw}} \in \mathbb{Z}^+$, $\mu \in \mathbb{R}^+$, $q_0 \in [0,1] \subset \mathbb{R}$
$P^{\mathrm{bf}} :=$ NULL, $P^{\mathrm{rb}} :=$ NULL, $cf := 0$, *bs_update* := FALSE
$\tau_{ij} := 0.5 \quad \forall \tau_{ij} \in \mathcal{T}$
while CPU time limit not reached **do**
  $P^{\mathrm{ib}} :=$ PBS(k_{bw},μ,N^{s}) /* see Algorithm 1 */
  if $P^{\mathrm{ib}} <_{\mathrm{lex}} P^{\mathrm{rb}}$ **then** $P^{\mathrm{rb}} := P^{\mathrm{ib}}$
  if $P^{\mathrm{ib}} <_{\mathrm{lex}} P^{\mathrm{bf}}$ **then** $P^{\mathrm{bf}} := P^{\mathrm{ib}}$
  $cf :=$ ComputeConvergenceFactor($\mathcal{T}$)
  if *bs_update* = TRUE **and** $cf > 0.99$ **then**
    $\tau_{ij} := 0.5 \quad \forall \tau_{ij} \in \mathcal{T}$
    $P^{\mathrm{rb}} :=$ NULL, *bs_update* := FALSE
  else
    if $cf > 0.99$ **then** *bs_update* := TRUE **end if**
    ApplyPheromoneUpdate(*cf*, *bs_update*, $\mathcal{T}$, P^{ib}, P^{rb}, P^{bf})
  end if
end while
output: P^{bf}

Procedure ComputeConvergenceFactor($\mathcal{T}$) computes the convergence factor *cf*, which is a function of the current pheromone values, as follows:

$$cf = 2 \left(\frac{\sum_{\tau_{ij} \in \mathcal{T}} \max\{\tau^{\max} - \tau_{ij}, \tau_{ij} - \tau^{\min}\}}{|\mathcal{T}| \cdot (\tau^{\max} - \tau^{\min})} - 0.5 \right) \tag{6}$$

where $\tau^{\max}$ and $\tau^{\min}$ are, respectively, the maximum and minimum pheromone values allowed. Hence, $cf = 0$ when the algorithm is initialized (or reset), that is, when all pheromone values are set to 0.5. In contrast, when the algorithm has converged, then $cf = 1$. In all other cases, *cf* has a value within $(0, 1)$.

The next step of the algorithm updates the pheromone information by means of the procedure ApplyPheromoneUpdate(*cf*, *bs_update*, $\mathcal{T}$, P^{ib}, P^{rb}, P^{bf}). In general, three solutions are used for updating the pheromone values. These are the *iteration-best* solution P^{ib}, the *restart-best* solution P^{rb}, and the *best-so-far* solution P^{bf}. The influence of each solution on the pheromone update depends on the state of convergence of the algorithm as measured by the convergence factor *cf*. Hence, each pheromone value $\tau_{ij} \in \mathcal{T}$ is updated as follows:

$$\tau_{ij} = \tau_{ij} + \rho \cdot (\xi_{ij} - \tau_{ij}) \ , \tag{7}$$

with $\xi_{ij} = \kappa^{\mathrm{ib}} \cdot P^{\mathrm{ib}}_{ij} + \kappa^{\mathrm{rb}} \cdot P^{\mathrm{rb}}_{ij} + \kappa^{\mathrm{bf}} \cdot P^{\mathrm{bf}}_{ij}$, where ρ is a parameter that determines the learning rate, P^{*}_{ij} is 1 if customer j is visited after customer i in solution P^{*} and 0 otherwise, κ^{ib} is the weight (i.e., the influence) of solution P^{ib}, κ^{rb} is the weight of solution P^{rb}, κ^{bf} is the weight of solution P^{bf}, and $\kappa^{\mathrm{ib}} + \kappa^{\mathrm{rb}} + \kappa^{\mathrm{bf}} = 1$. For our application we used a standard update schedule as shown in Table 1 and a value of $\rho = 0.1$.

After the pheromone update rule in Eq. (7) is applied, pheromone values that exceed $\tau^{\max} = 0.999$ are set back to $\tau^{\max}$ (similarly for $\tau^{\min} = 0.001$). This is done in order to avoid a complete convergence of the algorithm, which is a situation that should be avoided. This completes the description of our Beam-ACO approach for the TSPTW.

Table 1. Setting of κ^{ib}, κ^{rb} and κ^{bf} depending on the convergence factor *cf* and the Boolean control variable *bs_update*

bs_update	FALSE				TRUE
cf	$[0, 0.4)$	$[0.4, 0.6)$	$[0.6, 0.8)$	$[0.8, 1]$	—
κ^{ib}	1	2/3	1/3	0	0
κ^{rb}	0	1/3	2/3	1	0
κ^{bf}	0	0	0	0	1

4 Experimental Evaluation

We implemented Beam-ACO in C++ and used 30 instances originally provided by Potvin and Bengio [13] for testing. These instances are known to contain a mix of randomly-located and clustered customers. First, we performed a set of initial experiments in order to find appropriate values for various parameters of Beam-ACO. On the basis of these experiments we chose $k_{\text{bw}} = 10$, $\mu = 1.5$, $N^{\text{s}} = 5$, $q_0 = 0.9$, and a time limit of 60 CPU seconds per run and per instance. Each experiment was repeated 25 times with different random seeds. All experiments were run on a AMD Opteron 8218 processor, with 2.6 GHz CPU and 1 MB of cache size running GNU/Linux 2.6.24.

Comparison to the State-of-the-art. In Table 2 we compare the results of Beam-ACO with the results of two ACO algorithms proposed in the literature: ACS-TSPTW [3] and ACS-Time [4], where ACS-TSPTW is a variation of ACS-Time. These algorithms can be regarded as the current state-of-the-art for the TSPTW with makespan optimization. The structure of Table 2 is as follows. $\tilde{F}$ is the mean makespan obtained by each algorithm, and T_{CPU} is the mean computation time in seconds. The results of ACS-TSPTW and ACS-Time were obtained using an AMD Athlon CPU with 1.46 GHz, which should be about two times slower than our machine. For the results of Beam-ACO, we provide the corresponding standard deviations ("sd"). Finally, the column "Old Best-known" gives the previously best known result for each instance, while "New Best-known" gives the best-known makespan taking into account the results obtained by Beam-ACO.

Beam-ACO obtained a feasible solution in all 25 runs, while it is not clear how many feasible solutions were obtained by ACS-TSPTW and ACS-Time. In case no feasible solution was obtained in any of the 5 runs of ACS-TSPTW or ACS-Time, the corresponding entry is blank. In fact, ACS-TSPTW fails to find any feasible solution in three cases, whereas ACS-Time fails to find any feasible solution in five cases. Beam-ACO achieves a better performance than the other two algorithms in 22 out of 30 cases. In further seven cases our algorithm equals the best of the results obtained by the other two algorithms. Only for one instance (rc.202.3) Beam-ACO was not able to obtain the best result known. In many

Table 2. Comparison of results obtained by ACS-TSPTW, ACS-Time and Beam-ACO

		ACS-TSPTW		ACS-Time		Beam-ACO				Best-known	
Problem	n	$\tilde{F}$	T_{CPU}	$\tilde{F}$	T_{CPU}	$\tilde{F}$	sd	T_{CPU}	sd	Old	New
rc201.1	20	592.06	100.94	592.06	96.77	592.06	0.00	0.01	0.00	592.06	592.06
rc201.2	26	877.49	246.26	866.56	262.66	860.17	0.00	0.39	0.44	861.91	860.17
rc201.3	32	867.61	464.30	854.11	466.13	853.71	0.00	0.09	0.15	853.71	853.71
rc201.4	26	900.52	151.05	—	—	889.18	0.00	0.16	0.32	900.38	889.18
rc202.1	33	880.74	241.77	880.74	241.72	850.48	0.00	0.95	1.15	871.11	850.48
rc202.2	14	338.52	46.77	382.47	47.17	338.52	0.00	0.00	0.00	338.52	338.52
rc202.3	29	892.18	190.24	—	—	894.10	0.00	0.26	0.50	847.31	847.31
rc202.4	28	—	—	874.55	170.75	853.71	0.00	1.18	1.19	856.37	853.71
rc203.1	19	673.07	78.83	600.66	80.44	488.42	0.00	0.01	0.01	572.63	488.42
rc203.2	33	926.75	255.77	911.34	278.96	853.71	0.00	4.05	3.37	897.88	853.71
rc203.3	37	—	—	—	—	921.54	0.51	20.47	13.58	—	921.44
rc203.4	15	493.85	53.08	429.96	52.11	338.52	0.00	0.01	0.01	415.26	338.52
rc204.1	46	949.68	438.25	—	—	925.12	1.14	24.79	15.62	949.22	920.11
rc204.2	33	863.65	240.55	770.08	238.42	691.58	3.21	22.46	15.74	753.52	690.06
rc204.3	24	642.06	127.27	533.25	128.80	456.19	1.58	19.15	18.71	488.36	455.03
rc205.1	14	422.24	46.90	421.57	46.67	417.81	0.00	0.03	0.05	417.81	417.81
rc205.2	27	820.19	181.06	820.19	195.11	820.19	0.00	3.44	2.68	820.19	820.19
rc205.3	35	950.05	274.55	951.22	273.09	950.05	0.00	0.13	0.20	950.05	950.05
rc205.4	28	870.43	186.56	849.32	180.18	837.71	0.00	1.70	1.09	838.75	837.71
rc206.1	4	117.85	13.27	117.85	13.41	117.85	0.00	0.00	0.00	117.85	117.85
rc206.2	37	914.99	306.13	906.98	304.27	879.17	6.07	14.22	16.85	905.47	870.49
rc206.3	25	650.59	140.72	650.59	140.51	650.59	0.00	0.01	0.01	650.59	650.59
rc206.4	38	943.31	320.19	—	—	920.18	5.17	29.44	18.14	943.31	911.98
rc207.1	34	860.98	258.44	889.33	258.28	820.23	4.19	26.88	17.07	851.06	809.86
rc207.2	31	—	—	792.38	—	720.78	1.07	24.21	16.76	761.78	717.22
rc207.3	33	955.70	241.70	844.98	233.68	757.80	8.49	34.73	13.39	836.05	747.47
rc207.4	6	133.14	22.45	133.14	22.80	133.14	0.00	0.00	0.00	133.14	133.14
rc208.1	38	934.80	334.72	901.61	331.58	820.88	8.84	41.96	10.42	877.20	810.70
rc208.2	29	722.24	185.73	608.84	185.33	581.32	0.00	8.50	5.89	591.43	581.32
rc208.3	36	795.03	291.98	739.54	295.94	691.66	1.30	26.67	15.14	715.27	686.80

instances, Beam-ACO found the (presumably) optimal solution in all 25 runs, as illustrated by a zero standard deviation.

With respect to computation time, Beam-ACO is between 5 and 100 times faster than the other two algorithms. This difference of computation time between Beam-ACO and the other algorithms cannot be explained solely by differences in processor speed.

Summarizing, the results let us conclude that Beam-ACO is a new state-of-the-art algorithm for the TSPTW with makespan optimization.

Analysis of Beam-ACO. With the aim of obtaining a better understanding of the behaviour of Beam-ACO we conducted a series of additional experiments. First, we wanted to study the influence and the importance of the pheromone information, which is used during the construction process of probabilistic beam search

as well as for stochastic sampling. For that purpose we repeated the experiments with a version of Beam-ACO in which the pheromone update was switched off. This has the effect of removing the learning mechanism from Beam-ACO. In the presentation of the results this version is denoted by **noph**. In a second set of experiments we wanted to study the importance of stochastic sampling. Remember that, at each step of the probabilistic beam search, first a number of maximally $\lfloor \mu \cdot k_{\text{bw}} \rfloor$ extensions are chosen. Then, based on the results of stochastic sampling, procedure Reduce removes extensions until only the best k_{bw} extensions with respect to stochastic sampling are left. In order to learn if this reduction step is important, we repeated all the experiments with a version of Beam-ACO where $\mu = 1$ and $k_{\text{bw}} = 15$ (in order to compensate for the smaller μ value). Note that when $\mu = 1$, procedure Reduce is never invoked and stochastic sampling is never performed. In the presentation of the results this version of Beam-ACO is denoted by **no_ss**. Finally, we wanted to study how good stochastic sampling is in terms of an estimate, by applying it only after a certain number of iterations of each probabilistic beam search, that is, once the partial solutions in the beam of a probabilistic beam search contain a certain percentage of customers. More specifically, for the first $(n - (r^{\text{s}} \cdot n)/100)$ iterations of probabilistic beam search, stochastic sampling is not used. Instead, Reduce simply selects k_{bw} partial solutions at random. In contrast, for the remaining $(r^{\text{s}} \cdot n)/100$ iterations of probabilistic beam search, procedure Reduce uses the estimate provided by stochastic sampling for the elimination of partial solutions. Henceforth, we refer to parameter r^{s} as the *rate of stochastic sampling*. The value of this parameter is given as a percentage, where 0% means that no stochastic sampling is ever performed, while 100% refers to the Beam-ACO approach that always uses stochastic sampling. In our experiments we tested the following rates of stochastic sampling: $r^{\text{s}} = \{0\%, 25\%, 50\%, 75\%, 85\%, 100\%\}$. In the presentation of the results the corresponding algorithm versions are simply denoted by the value of parameter r^{s}.

Figure 2 shows the results of the different experiments described above for five problem instances that are rather difficult to solve. The barplots (in grey) compare the results with respect to the mean ranks obtained by each algorithm version over 25 runs. Note that standard deviations are shown as error bars. The ranks are calculated by sorting all solutions lexicographically. On the other hand, the boxplots (in white) show the distribution of computation time (in seconds) required by each algorithm version.

The following conclusions can be drawn. First, when no pheromone information is used (see algorithm **noph**), the performance of the algorithm drops significantly. Interestingly, the performance of Beam-ACO without pheromone update is always worse than the performance of Beam-ACO using at least $r^{\text{s}} = 50\%$ of stochastic sampling. Second, the use of stochastic sampling seems essential to achieve satisfactory results. When no stochastic sampling is used (see algorithm **no_ss**), the results achieved are worse than the ones obtained by Beam-ACO with stochastic sampling, and the algorithm requires significantly more computation time.

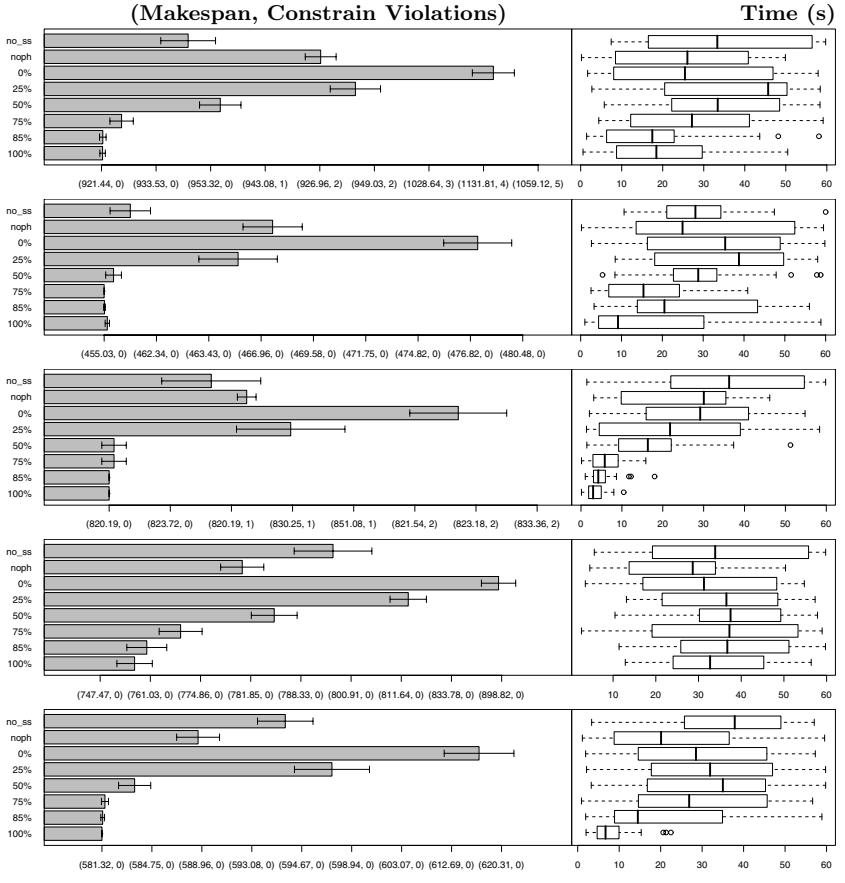


Fig. 2. Results concerning the analysis of Beam-ACO. From top to bottom the graphics concern instances rc.203.3, rc.204.3, rc.205.2, rc.207.3, and rc.208.2.

Finally, the results of the algorithm variants using different rates of stochastic sampling show a clear pattern. The performance of the algorithm increases with increasing rate of stochastic sampling. Disabling stochastic sampling completely ($r^{\mathrm{s}} = 0\%$), strongly affects the performance of Beam-ACO in a negative way. Starting from rates of stochastic sampling of at least 75%, the performance of the algorithm is already very close to—and sometimes the same as—the performance of Beam-ACO when always using stochastic sampling. However, even in those cases where a rate of stochastic sampling of 85% performs as well as the variant using $r^{\mathrm{s}} = 100\%$, the latter requires less computation time; see, for example, instances rc.204.3 and rc.208.2. This is particularly interesting because the variant using $r^{\mathrm{s}} = 100\%$ is the most expensive one in terms of computational effort. Therefore, this result suggests that stochastic sampling helps the algorithm to converge faster to the best solutions.

5 Conclusions

In this paper, we have proposed a Beam-ACO approach for the TSPTW with makespan optimization. Beam-ACO is a hybrid between ant colony optimization and beam search that, in general, relies heavily on bounding information that is accurate and computationally inexpensive. We studied a version of Beam-ACO in which the bounding information is replaced by stochastic sampling. Experiments were performed on a set of standard benchmark instances for the TSPTW, comparing the Beam-ACO approach to the best known methods from the literature. The results showed that Beam-ACO is a state-of-the-art algorithm for the TSPTW with makespan optimization. In a second set of experiments we analysed the influence of pheromone information and stochastic sampling on the performance of Beam-ACO. The results showed that both algorithmic components are essential for achieving high quality results. In the future we plan to improve the performance of our Beam-ACO approach further, for example, by the inclusion of local search.

References

1. Ohlmann, J.W., Thomas, B.W.: A compressed-annealing heuristic for the traveling salesman problem with time windows. INFORMS J. Comput. 19(1), 80–90 (2007)
2. Savelsbergh, M.W.P.: Local search in routing problems with time windows. Annals of Operations Research 4(1), 285–305 (1985)
3. Cheng, C.B., Mao, C.P.: A modified ant colony system for solving the travelling salesman problem with time windows. Mathematical and Computer Modelling 46, 1225–1235 (2007)
4. Gambardella, L., Taillard, E.D., Agazzi, G.: MACS-VRPTW: A multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 63–76. McGraw Hill, London (1999)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
6. Blum, C.: Beam-ACO–hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comp. & Op. Res. 32, 1565–1591 (2005)
7. Blum, C.: Beam-ACO for simple assembly line balancing. INFORMS J. Comput. 20(4), 618–627 (2008)
8. Ow, P.S., Morton, T.E.: Filtered beam search in scheduling. Int. J. Prod. Res. 26, 297–307 (1988)
9. López-Ibáñez, M., Blum, C.: Beam-ACO based on stochastic sampling: A case study on the TSP with time windows. In: Battiti, R., et al. (eds.) Proceedings of LION3. LNCS. Springer, Berlin (2009)
10. Juillé, H., Pollack, J.B.: A sampling-based heuristic for tree search applied to grammar induction. In: Proceedings of AAAI 1998, pp. 776–783. MIT press, Cambridge (1998)
11. Ruml, W.: Incomplete tree search using adaptive probing. In: Proceedings of IJCAI 2001, pp. 235–241. IEEE press, Los Alamitos (2001)
12. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE T. Syst. Man Cyb. – Part B 34(2), 1161–1172 (2004)
13. Potvin, J.Y., Bengio, S.: The vehicle routing problem with time windows part II: Genetic search. INFORMS J. Comput. 8, 165–172 (1996)