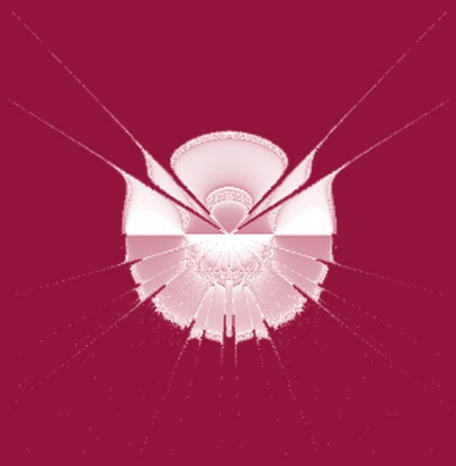
Peter Cowling
Peter Merz (Eds.)

Evolutionary Computation in Combinatorial Optimization

LNCS 6022

10th European Conference, EvoCOP 2010
Istanbul, Turkey, April 2010
Proceedings



Springer

Lecture Notes in Computer Science 6022

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

Peter Cowling Peter Merz (Eds.)

Evolutionary Computation in Combinatorial Optimization

10th European Conference, EvoCOP 2010
Istanbul, Turkey, April 7-9, 2010
Proceedings

Springer

Volume Editors

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

Peter Merz
FH-Hannover – University of Applied Sciences and Arts
Department of Business Administration and Computer Science
Ricklinger Stadtweg 120, 30459 Hannover, Germany
E-mail: peter.merz@fh-hannover.de

Cover illustration:
"Pelegrina Galathea" by Stayko Chalakov (2009) Aston University, UK

Library of Congress Control Number: 2010922335

CR Subject Classification (1998): F.1, C.2, H.4, I.5, I.4, F.2

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

ISSN 0302-9743
ISBN-10 3-642-12138-1 Springer Berlin Heidelberg New York
ISBN-13 978-3-642-12138-8 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper 06/3180

Preface

Metaheuristics continue to demonstrate their effectiveness for an ever-broadening range of 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 2010, the 10th European Conference on Evolutionary Computation in Combinatorial Optimization. It was held in Istanbul, Turkey, the 2010 European city of culture, during April 7–9, 2010, jointly with EuroGP 2010, the 13th European Conference on Genetic Programming, EvoBIO 2010, the 8th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, EvoPHD 2010, the 5th European Graduate Student Workshop on Evolutionary Computation, and EvoApplications 2010 (formerly EvoWorkshops), which consisted of the following 12 individual events: 7th European event on the Application of Nature-Inspired Techniques for Telecommunication Networks and other Parallel and Distributed Systems (EvoCOMNET), First European event on Evolutionary Algorithms and Complex Systems (EvoCOMPLEX), Second European Event on Nature-Inspired Methods for Environmental Issues (EvoENVIRONMENT), 4th European Event on Evolutionary and Natural Computation in Finance and Economics (EvoFIN), Second European Event on Bio-inspired Algorithms in Games (EvoGAMES), 12th European Event on Evolutionary Computation in Image Analysis and Signal Processing (EvoIASP), First European Event on Nature-Inspired Methods for Intelligent Systems (EvoINTELLIGENCE), 8th European Event on Evolutionary and Biologically Inspired Music, Sound, Art and Design (EvoMUSART), Third European Event on Bio-inspired algorithms for continuous parameter optimisation (EvoNUM), 7th European Event on Evolutionary

Algorithms in Stochastic and Dynamic Environments (EvoSTOC), and 4th European Event on Evolutionary Computation in Transportation and Logistics (EvoTRANSLOG). 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, 5482). Below we report statistics for each conference:

EvoCOP	Submitted	Accepted	Acceptance ratio
2001	31	23	74.2%
2002	32	18	56.3%
2003	39	19	48.7%
2004	86	23	26.7%
2005	66	24	36.4%
2006	77	24	31.2%
2007	81	21	25.9%
2008	69	24	34.8%
2009	53	21	39.6%
2010	69	24	34.8%

The rigorous, double-blind reviewing process of EvoCOP 2010 resulted in a strong selection among the submitted papers; the acceptance rate was 34.8%. 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: we are very grateful for their thorough work. EvoCOP 2010 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.

To celebrate the tenth anniversary of EvoCOP, we were very pleased to welcome, as plenary speakers, the founders of the EvoCOP series, Günther Raidl from the Vienna University of Technology, Austria and Jens Gottlieb from SAP, Walldorf, Germany. We would also like to express our sincere gratitude to two further internationally renowned invited speakers, who gave keynote talks at the conference: Kevin Warwick from the University of Reading, UK and Luca Cavalli-Sforza from the Stanford School of Medicine, USA.

The success of the conference resulted from the input of many people to whom we would like to express our appreciation. A. Şima (Etaner) Uyar was Local Chair, assisted by Sanem Sarıel-Talay, Şule Gündüz-Öğüdücü, Ayşegül Yayımlı, Gülşen Cebiroğlu-Eryiğit, H. Turgut Uyar and others from the Computer Engineering Department of Istanbul Technical University. The local organizers did an extraordinary job for which we are very grateful. We thank Marc Schoenauer from INRIA in France for his support with the MyReview conference

management system. We thank Stephen Dignum of the University of Essex, UK, assisted by Cecilia Di Chio of the University of Strathclyde, UK for an excellent website and publicity material. Thanks are also due to Jennifer Willies and the Centre for Emergent Computing at Napier University in Edinburgh, Scotland for administrative support and event coordination. We gratefully acknowledge sponsorship from Istanbul Technical University, Microsoft, and the Scientific and Technological Research Council of Turkey. Last, but not least, we would like to thank Carlos Cotta, 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 2010

Peter Cowling
Peter Merz

Organization

EvoCOP 2010 was organized jointly with EuroGP 2010, EvoBIO 2010, EvoPHD 2010, and EvoApplications 2010.

Organizing Committee

Chairs	Peter Cowling, University of Bradford, UK
	Peter Merz, University of Applied Sciences and Arts, Hannover, Germany
Local Chair	Şima Etaner-Uyar, Istanbul Technical University, Turkey
Publicity Chair	Stephen Dignum, University of Essex, UK

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
Peter Merz	University of Applied Sciences and Arts, Hannover, Germany
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
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

Keshav Dahal	University of Bradford, UK
Karl Doerner	Universität Wien, Austria
Benjamin Doerr	Max-Planck-Institut für Informatik, Germany
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
Graham Kendall	University of Nottingham, UK
Joshua Knowles	University of Manchester, UK
Mario Köppen	Kyushu Institute of Technology, Japan
Jozef Kratica	University of Belgrade, Serbia
Rhyd Lewis	Cardiff University, UK
Arne Løkketangen	Molde College, Norway
José Antonio Lozano	University of the Basque Country, Spain
Dirk C. Mattfeld	Technische Universität Braunschweig, Germany
Barry McCollum	Queen's University Belfast, UK
Juan Julián Merelo	University of Granada, Spain
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
Yuichi Nagata	Tokyo Institute of Technology, Japan
Volker Nissen	Technical University of Ilmenau, Germany
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
Franz Rothlauf	Johannes Gutenberg Universität, Mainz, Germany

Michael Sampels	Université Libre de Bruxelles, Belgium
Marc Schoenauer	INRIA, France
Jim Smith	University of the West of England, UK
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
Stefan Voss	University of Hamburg, Germany
Jean-Paul Watson	Sandia National Laboratories, USA
Fatos Xhafa	Universitat Politcnica de Catalunya, Spain
Takeshi Yamada	NTT Communication Science Laboratories, Kyoto, Japan

Table of Contents

Dual Sequence Simulated Annealing with Round-Robin Approach for University Course Timetabling

Salwani Abdullah[1], Khalid Shaker[1], Barry McCollum[2], and Paul McMullan[2]

[1] Center for Artificial Intelligence Technology,
Universiti Kebangsaan Malaysia, 43600 Bangi, Selangor, Malaysia
{salwani,khalid}@ftsm.ukm.my
[2] Department of Computer Science, Queen's University Belfast,
Belfast BT7 1NN United Kingdom
{b.mccollum,p.p.mcmullan}@qub.ac.uk

Abstract. The university course timetabling problem involves assigning a given number of events into a limited number of timeslots and rooms under a given set of constraints; the objective is to satisfy the hard constraints (essential requirements) and minimize the violation of soft constraints (desirable requirements). In this study we employed a Dual-sequence Simulated Annealing (DSA) algorithm as an improvement algorithm. The Round Robin (RR) algorithm is used to control the selection of neighbourhood structures within DSA. The performance of our approach is tested over eleven benchmark datasets. Experimental results show that our approach is able to generate competitive results when compared with other state-of-the-art techniques.

Keywords: Course timetabling, Dual Sequence Simulated Annealing, Round Robin algorithm.

1 Introduction

Timetabling problems can be defined as the assignment of resources for tasks under predefined constraints so that it minimises the violation of constraints. Typical timetabling areas are educational timetabling, sports timetabling, employee timetabling, and so on. Educational timetabling can be divided into school timetabling, exam timetabling, and course timetabling. In the university course timetabling problem, events (subjects, courses) have to be assigned into a number of periods and rooms while satisfying various constraints.

Over the past forty years, researchers have proposed various timetabling approaches that fall under Operational Research and Artificial Intelligence. Some of the approaches used are constraint-based methods, population-based approaches (e.g., genetic algorithms, ant colony optimization and memetic algorithms), meta-heuristic methods (e.g. tabu search and simulated annealing), variable neighbourhood search, hyper-heuristic and hybridization approaches, etc. Socha *et al.* [8] applied an ant based approach to the eleven datasets which are investigated here. Rossi-Doria *et al.* [17] presented a comparison of a number of metaheuristic methods on the same datasets. Burke *et al.* [1] introduced a tabu-based hyperheuristic and applied it to

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 1–10, 2010.

university course timetabling in addition to nurse rostering. A tabu search within a graph based hyper-heuristic (aiming to raise the level of generality) has been employed by Burke *et al*. [16] and tested on different problem domains, i.e. examination and course timetabling benchmark datasets. Abdullah *et al*. [4] developed a variable neighbourhood search approach which used a fixed tabu list to penalise particular neighbourhood structures. The authors continue their work by implementing a randomized iterative improvement approach using a composite of eleven neighbourhood structures (Abdullah *et al*. [5]). Abdullah *et al*. [6] presented a hybrid approach combining a mutation operator with their previous randomized iterative improvement algorithm [5]. The extended great deluge has been applied by McMullan [7] and tested to the same datasets which were originally introduced by Socha *et al*. [8]. A different version of the great deluge algorithm, called non-linear great deluge which generates non-linear decay rates for three different categories of datasets has been tested by Landa-Silva and Obit [9]. The combination of genetic algorithm and local search has been employed by Abdullah and Turabieh [11] and is able to produce promising results on the same test instances. Turabieh *et al*. [18] employed a hybridization of an electromagnetic-like mechanism with forced decay rate great deluge on the same problem instances and have obtained promising results. A comprehensive review on timetabling can be found in [12, 13, 14, 15].

This paper is organized as follows: the next section introduces the university course timetable problem with a set of hard and soft constraints. In section 3 we represent the proposed algorithm. The simulation results are represented in section 4, and finally conclusion and future work are represented in section 5.

2 Problem Descriptions

In university course timetabling, a set of courses are scheduled into a given number of rooms and timeslots. This usually takes place within a week and the resultant timetable replicated for as many weeks as the courses run. Also, students and teachers are assigned to courses so that the teaching delivery activities can take place. The course timetabling problem is subject to a variety of hard and soft constraints. Hard constraints need to be satisfied in order to produce a *feasible* solution. In this paper, we test our approach on the problem instances introduced by Socha *et al*. [8] who present the following hard constraints:

- *No student can be assigned to more than one course at the same time.*
- *The room should satisfy the features required by the course.*
- *The number of students attending the course should be less than or equal to the capacity of the room.*
- *Not more than one course is allowed to be assigned to a timeslot in each room.*

Socha *et al*. [8] also present the following soft constraints that are equally penalized:

- *A student has a course scheduled in the last timeslot of the day.*
- *A student has more than 2 consecutive courses.*
- *A student has a single course on a day.*

The problem has

- A set of N courses, $e = \{e_1,\ldots,e_N\}$.
- 45 timeslots.
- A set of R rooms.
- A set of F room features.
- A set of M students.

The objective of this problem is concerned with satisfying the hard constraints while minimizing as much as possible the violation of the soft constraints.

3 The Algorithm

The algorithm consists of two processes; the first process is concerned with producing an initial solution. The second process is to optimize the soft constraint cost of the initial solution generated in the first process, with three neighbourhood structures employed for this purpose.

3.1 Neighbourhood Structures

The different neighbourhood structures are outlined as follows:

N_1: Choose a single course at random and move to a feasible timeslot that can generate the lowest penalty cost.
N_2: Select two courses at random from the same room (the room is randomly selected) and swap timeslots.
N_3: Move the highest penalty course to a random feasible timeslot.

3.2 Constructive Heuristic

DSA starts with generating an initial solution for a new sequence. Least saturation degree is used to generate initial solutions which start with an empty timetable [7]. Those events with less rooms available and more likely as difficult to be scheduled will be attempted first, without taking into consideration the violation of any soft constraints, until the hard constraints are met. This process is carried out in the first phase. If a feasible solution is found, the algorithm stops. Otherwise, phase 2 is executed. In the second phase, neighbourhood moves (N_1 and/or N_2) are applied to attempt to move from an infeasible to feasible solution. N_1 is applied for a certain number of iterations. If a feasible solution is met, then the algorithm stops. Otherwise the algorithm continues by applying a N_2 neighbourhood structure for a certain number of iterations. In this work, across all instances tested, the solutions were made feasible before the improvement algorithm is applied.

3.3 Improvement Algorithm: DSA

During the optimization process a set of the neighbourhood structures outlined in subsection 3.1 are applied. The hard constraints are never violated during the timetabling process. The pseudo code for the algorithm implemented in this paper is given in Fig. 1.

```
Initialization Phase
Set iterations counter, Iter;
Set maximum number of iteration, Iter_max;
Set number of local consecutive non-improving solution, Lq;
Set maximum number of local consecutive non-improving solution, Lq_max;
Set number of global consecutive non-improving solution, Gq;
Set maximum number of global consecutive non-improving solution, Gq_max;
Set initial solution, Sol
Set best solution, BestSol← Sol
Set initial temperature T0
Set final temperature Tf;
Set decreasing rate α = (log (T0) - log (Tf)/Iter_max);

Improvement Phase
Generate feasible initial solution (Sol);
Iter ← 0;
Gq ← 0;
Lq← 0;
temp ← T0;
Do while (Iter <Iter_max or the penalty cost is zero)
// First scheme
      if (Gq > Gq_max){
          Save the best solution, BestSol obtained so far;
          Generate a feasible initial solution, Sol, for a new sequence;
          Reset Gq← 0;
       Lq← 0;
       Re-anneal (temp←T0);
        }
      Define a neighbourhood of Sol based on RR algorithm to generate Sol*;
      if (f(Sol*) < f(BestSol)
              BestSol← Sol*;
              Sol← Sol*;
             Reset  Gq← 0,
   else
//Second scheme
      Increase counter non-improvement solution by 1; Lq++;Gq++;
      if (Lq > Lq_max)
          Lq← 0;
          Generate a random number, RandNum in [0, 1];
          Calculate the acceptance propability of Sol*, Paccept(Sol*)
          if (RandNum < Paccept(Sol*)) // Paccept(Sol*) is a function to calculate the acceptance
                                       // probability of Sol*
                    Sol← Sol*;
   temp ←  temp/(1+ α);
   if (temp <Tf)
       Re-anneal (set temp←T0);
   Iter ++;
end do;
```

Fig. 1. The pseudo code of DSA

At the initialization phase, the algorithm sets initial values for all parameters used, such as initial temperature, final temperature, decreasing rate, maximum number of iterations, maximum number of local and global consecutive non-improving solutions, the quality of initial and best solutions, etc. In this work, the initial temperature T_0 is equal to 1000, the final temperature T_f is equal to 0.5 (i.e. the same parameters as those employed in [2]) and the number of iterations, *Iter_max* is set to be 200,000 (as in [Turabieh's paper, Abdullah's paper]). Based on our preliminary experiments, the counter for the local and global consecutive non-improving solutions (denoted as *Lq_max* and *Gq_max*) is set to 20 and 50 respectively.

The algorithm starts with a feasible initial solution generated by a constructive heuristic as discussed in section 3.2. At the improvement phase, the algorithm identifies two schemes. During improvement, within the first scheme the search is controlled by a counter denoted as *Gq* where it records the number of consecutive non-improving solutions. When *Gq* exceeds the limit of *Gq_max* (*Gq* > *Gq_max*) the approach will generate a new initial solution, Sol ensuring it is not a neighbour to the previous solution, and the best solution (*BestSol*) found so far is saved. All parameters are re-initialized. The temperature *temp* is re-annealed (set to T_0) and *Gq* is set to 0. Note that the algorithm saves the last best solution obtained so far before generating a new initial solution. A neighbour is defined using a round-robin algorithm to generate a new solution *Sol**. The quality of the new solution *f*(*Sol**) is calculated and compared with the quality of the best solution, *f*(*BestSol*). If there is an improvement, where *f*(*Sol**) < *f*(*BestSol*), the new solution *Sol** is accepted and *BestSol* is set to the new solution *Sol**.

Within the second scheme, where the new solution *Sol** is worse than the best solution *BestSol*, we introduce a counter *Lq* (that represents the number of consecutive non-improvement solutions) as a controller to escape from local optima. When *Lq* > *Lq_max*, a worse solution is accepted based on a certain probability. In this work, the new solution, *Sol** is accepted if the generated random number *RandNum* is less than the probability, which is computed as $exp\ (-\frac{f(Sol*)-f(Sol)}{f(Sol*)temp})$.

Much worse solutions are likely to be accepted if the value of *f*(*Sol**) is too large. This then can make it difficult for the search to converge. At every iteration, *temp* is decreased by α, defined as $\alpha = (\log (T_0) - \log (T_f)/Iter\_max)$. If the temperature *temp* falls below the final temperature T_f, the re-annealing process will take place i.e. *temp* is set to the initial temperature value, T_0. The algorithm stops when the maximum number of iterations *Iter_max* is reached or the penalty cost is zero.

3.4 Round Robin Algorithm, RR

The RR algorithm is employed to control the selection of the neighbourhood structures, which are ordered in sequence. In this work the neighbourhood structures are ordered as N_1, N_2 and N_3 (see subsection 3.1). A time slice or quantum is assigned for each neighbourhood structure in equal portions, in a circular order. The neighbourhood structure is dispatched in a FIFO manner at a given quantum denoted as *qtime* (which is set to 5 seconds). Note that in this paper, all parameters used are based on a number of preliminary experiments. The pseudo code for the RR algorithm is presented in Fig. 2.

```
Set quantum time, qtime;
Set a sequence of neighbourhood structures in a queue which is ordered as N_i where
i ∈ {1,…,K} and K = 3;
Set initial value to counter_qtime;
do while (qtime not met )
    Select a neighbourhood structure N_i in the queue where i ∈ {1,…,K} where K = 3;
A:  Apply N_i on current solution,Sol to generate new solution Sol*;
    if there is an improvement on the quality of the solution then
        repeat label A
    else
        insert N_i into the queue;
        counter_qtime = q_time;
end do
```

Fig. 2. The pseudo code for RR algorithm

After the completion of the time slice of a current neighbourhood structure, the preemption is given to the next neighbourhood waiting in a queue. The pre-empted neighbourhood is then placed at the back of the queue. When the neighbourhood structure N_i is unable to generate a better solution during the given quantum time, the neighbourhood structure will be added into the queue. In the next iteration (in Fig. 3), the first neighbourhood structure in the queue will be used to generate a new solution.

4 Experimental Results

The algorithm was implemented on a Pentium 4 Intel 2.33 GHz PC Machine using Matlab on a Windows XP Operating System. The algorithm was run for 200,000 evaluations with 10 test-runs on each dataset.

We have evaluated our results on the instances taken from Socha et al. [8] which are available at http://iridia.ulb.ac.be/~msampels/tt.data/. They are divided into three categories: small, medium and large. We deal with 11 instances: 5 small, 5 medium and 1 large. The parameter values defining the categories are given in Table 1.

Table 1. The parameter values for the course timetabling problem categories

Category	*Small*	*Medium*	*large*
Number of courses	100	400	400
Number of rooms	5	10	10
Number of features	5	5	10
Number of students	80	200	400

Fig. 3 shows the frequency charts of the neighbourhood structures that have been controlled by the RR algorithm for all datasets. The *x*-axis represents the data-sets while the number on the top of each column represents the frequency of the

neighbourhood structures being employed throughout the search that are able to generate better solutions. It can be seen from Figure 3 that the neighbourhood structure "N_2" is the most popular neighbourhood structure used in the algorithm, able to generate better solutions for all datasets, followed by the neighbourhood structure "N_3" for *small* datasets and "N_1" for *medium* and *large* datasets. We can see that the algorithm might require more than one particular neighbourhood structure for different datasets when exploring the search space. For some of the data sets, the frequencies reported in figure 3 between differing neighbourhood structures are close in value. This shows that problems with different size or complexity might require a combination of the different neighbourhood structures, allowing the search algorithm to explore the search space using varied strategies from one region to another region. The RR algorithm is shown as a useful mechanism in controlling and utilizing the employment of these neighbourhood structures.

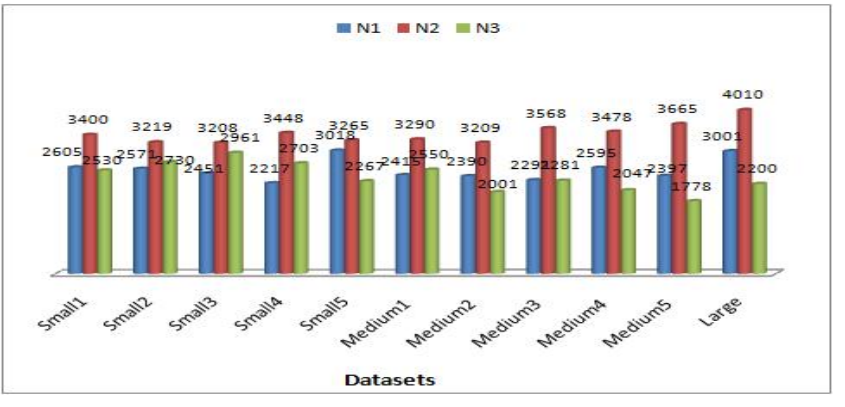


Fig. 3. Frequency of the neighbourhood structures used for all datasets

The best results out of 10 runs obtained are presented. Table 2 shows the comparison of the approach in this paper with other available approaches in the literature. These include genetic algorithm and local search by Abdullah and Turabieh [11], randomised iterative improvement algorithm by Abdullah *et al.* [5], graph hyper heuristic by Burke *et al.* [16], variable neighbourhood search with tabu by Abdullah *et al.* [4], hybrid evolutionary approach by Abdullah *et al.* [6], extended great deluge by McMullan [7], non linear great deluge by Landa-Silva and Obit [9], and electromagnetism-like mechanism approach by Turabieh *et al.* [18]. Note that the best results are presented in bold. It can be seen that our approach is able to produce competitive results and the best known result on *Medium2*.

Fig. 4 (a), (b) and (c) show the box plots of the penalty cost when solving *small*, *medium* and *large* instances, respectively. The results for the large dataset are less dispersed compared to *medium* and *small* (worse dispersed case in these experiments). We can see that the median is closer to the best than to the worst (max) in *small* and *medium* datasets; however the worst is closer to the median in large datasets. From analyzing these results, we believe that the size of the search space may not be dependent on the problem size due to the fact that the dispersion of solution points are

Table 2. Results comparison

Dataset	Our method		M1	M2	M3	M4	M5	M6		M7	M8
	Min	**Ave.**						**Min**	**Ave.**		
Small1	**0**	0.8	**0**	**0**	6	**0**	**0**	**0**	0.8	3	**0**
Small2	**0**	2	**0**	**0**	7	**0**	**0**	**0**	2	4	**0**
Small3	**0**	1.4	**0**	**0**	3	**0**	**0**	**0**	**1.3**	6	**0**
Small4	**0**	1	**0**	**0**	3	**0**	**0**	**0**	1	6	**0**
Small5	**0**	0.6	**0**	**0**	4	**0**	**0**	**0**	**0.2**	**0**	**0**
Medium1	93	132.2	175	242	372	317	221	**80**	**101.4**	140	175
Medium2	**98**	114.6	197	161	419	313	147	105	116.9	130	197
Medium3	149	162.0	216	265	359	357	246	**139**	**162.1**	189	216
Medium4	103	111.2	149	181	348	247	165	**88**	**108.8**	112	149
Medium5	98	113.1	190	151	171	292	130	**88**	**119.7**	141	190
Large	680	738.6	912	-	1068	-	**529**	730	834.1	876	912

Note:
M1: Genetic algorithm and local search by Abdullah and Turabieh [11].
M2: Randomised iterative improvement algorithm by Abdullah *et al*. [5].
M3: Graph hyper heuristic by Burke *et al*. [16].
M4: Variable neighbourhood search with tabu by Abdullah *et al*. [4].
M5: Hybrid evolutionary approach by Abdullah *et al*. [6].
M6: Extended great deluge by McMullan [7].
M7: Non linear great deluge by Linda-Silva and Obit [9].
M8: Electormagnetic-like mechanism approach by Turabieh et al. [18]

It can be seen that in general, our approach is better than other approaches reported in Table 2 (apart from M6) and is able to generate one best-published solution for *Medium2*.

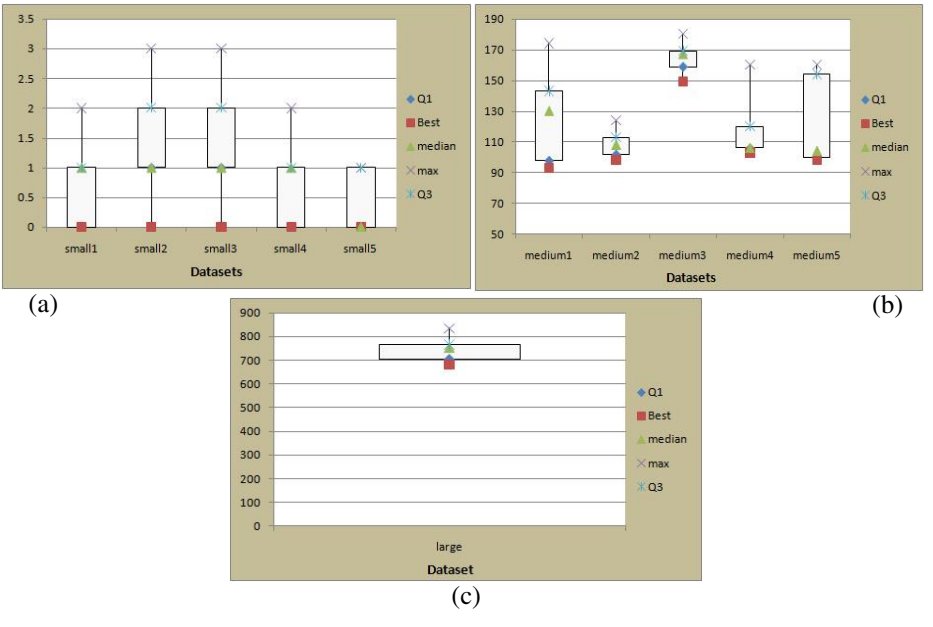


Fig. 4. (a), (b) and (c). Box plots of the penalty costs for small, medium and large datasets.

significantly different from one to another, even though the problems are from the same group of datasets with the same parameter values. This shows that the algorithm behaves differently on different datasets, possibly due to the different complexities of the datasets and the nature of the solution space. We believe that the algorithm might be able to obtain better results on all the datasets by introducing a mechanism that can adaptively and intelligently employ different neighbourhood structures for different situations, based on the quality of the solution in hand. This is subject to future work.

5 Conclusion and Future Work

This paper presents a dual sequence simulated annealing applied to the course timetabling problem. The round robin algorithm is employed within the dual sequence simulated annealing to control the selection of the neighbourhood structures given a slice time or quantum. In order to test the performance of our approach, experiments are carried out based on course timetabling problems and compared with state-of-the-art methods from the literature. Preliminary comparisons indicate that the dual sequence simulated algorithm is competitive with other approaches in the literature and able to produce one best known solution on *Medium2* dataset. In future work, efforts will be made to establish, compare and report on timings in relation to previously reported literature. We believe that the proposed approach can be adapted with new problems, thus the ITC2007 datasets will be the subject of future work.

References

1. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu search hyperheuristic for timetabling and rostering. Journal of Heuristics 9(6), 451–470 (2003)
2. Sekhar Pedamallu, C., Özdamar, L.: Comparison of Simulated Annealing. In: Interval Partitioning and Hybrid Algorithms in Constrained Global Optimization. Natural Computing Series. Springer, Heidelberg (2008)
3. Sekhar Pedamallu, C., Özdamar, L.: Investigating a hybrid simulated annealing and local search algorithm for constrained optimization. In: EJOR (2006) (in press), http://dx.doi.org/0,1016/j.ejor.2006.06.050
4. Abdullah, S., Burke, E.K., McCollum, B.: An Investigation of Variable Neighbourhood Search for Course Timetabling. In: The Proceedings of the 2nd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2005), New York, USA, July 18 -21, pp. 413–427 (2005)
5. Abdullah, S., Burke, E.K., McCollum, B.: Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for University Course Timetabling. In: Metaheuristics Progress in Complex Systems Optimization, pp. 153–169. Springer, Heidelberg (2007)
6. Abdullah, S., Burke, E.K., McCollum, B.: A Hybrid Evolutionary Approach to the University Course Timetabling Problem. In: IEEE Congres on Evolutionary Computation, pp. 1764–1768 (2007) ISBN: 1-4244-1340-0
7. McMullan, P.: An Extended Implementation of the Great Deluge Algorithm for Course Timetabling. In: Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A. (eds.) ICCS 2007. LNCS, vol. 4487, pp. 538–545. Springer, Heidelberg (2007)

8. Socha, K., Knowles, J., Samples, M.: A max-min ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G.A., Sampels, M. (eds.) Ant Algorithms 2002. LNCS, vol. 2463, pp. 1–13. Springer, Heidelberg (2002)
9. Landa-Silva, D., Obit, J.H.: Great Deluge with Nonlinear Decay Rate for Solving Course Timetabling Problems. In: Proceedings of the 2008 IEEE Conference on Intelligent Systems (IS 2008), pp. 8.11–8.18. IEEE Press, Los Alamitos (2008)
10. Hedar, A.-R., Fukushima, M.: Derivative-free filter simulated annealing method for constrained continuous global optimization. Journal of Global Optimization (2006) (to appear)
11. Abdullah, S., Turabieh, H.: Generating university course timetable using genetic algorithms and local search. In: The Third 2008 International Conference on Convergence and Hybrid Information Technology ICCIT, vol. I, pp. 254–260 (2008)
12. Burke, E.K., Petrovic, S.: Recent research directions. Iin automated timetabling. European Journal of Operation Research 140(2), 266–280 (2002)
13. Lewis, R., Paechter, B., McCollum, B.: Post enrolment based course timetabling: A description of the problem model used for track two of the second international timetabling competition Technical Report, Cardiff University (2007)
14. Lewis, R.: A survey of metaheuristic based techniques for university timetabling problems. OR Spectrum 30(1), 167–190 (2008)
15. Qu, R., Burke, E.K., McCollum, B., Merlot, L.T.G.: A survey of search methodologies and automated system development for examination timetabling. Journal of Scheduling 12(1), 55–89 (2009)
16. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A Graph-Based Hyper Heuristic for Educational Timetabling Problems. European Journal of Operational Research 176(1), 177–192 (2007)
17. Rossi-Doria, O., Samples, M., Birattari, M., Chiarandini, M., Dorigo, M., Gambardella, L.M., Knowles, J., Manfrin, M., Mastrolilli, M., Paechter, B., Paquete, L., Stützle, T.: A comparison of the performance of different meta-heuristics on the timetabling problem. In: Burke, E.K., De Causmaecker, P. (eds.) PATAT 2002. LNCS, vol. 2740, pp. 329–354. Springer, Heidelberg (2003)
18. Turabieh, H., Abdullah, S., McCollum, B.: Electromagnetism-like Mechanism with Force Decay rate Great Deluge for the Course Timetabling problem. In: Wen, P., Li, Y., Polkowski, L., Yao, Y., Tsumoto, S., Wang, G. (eds.) RSKT 2009. LNCS (LNAI), vol. 5589, pp. 497–504. Springer, Heidelberg (2009)

Heuristic and Exact Methods for the Discrete $(r \mid p)$-Centroid Problem*

Ekaterina Alekseeva, Nina Kochetova,
Yury Kochetov, and Alexandr Plyasunov

Sobolev Institute of Mathematics,
Novosibirsk State University, Russia
{ekaterina2,nkochet,jkochet,apljas}@math.nsc.ru

Abstract. In the discrete $(r \mid p)$-centroid problem two decision makers, a leader and a follower, compete to attract clients from a given market. The leader opens p facilities, anticipating that the follower will react to the decision by opening his own r facilities. The decision makers try to maximize their own profits. This Stackelberg game is Σ_2^P-hard. So, we develop a hybrid memetic algorithm for it. A probabilistic tabu search heuristic is applied for improving the offspring. To obtain an upper bound, we reformulate the problem as a mixed integer program with an exponential number of constraints and variables. Selecting some of them, we get the desired upper bound. To find optimal solutions, we iteratively modify the subset of the constraints and variables. This approach is tested on the benchmarks from the library *Discrete Location Problems*. The optimal solutions are found for $r = p = 5$, 100 clients, and 100 facilities.

1 Introduction

In this paper we study a competitive facility location model with two noncooperative decision makers: the leader and the follower. They compete to attract clients from a given market and wish to maximize their own profits. First, the leader opens p facilities. Later on, the follower opens r facilities. In fact, we have a noncooperative Stackelberg game. Following Hakimi [7], we call it the *discrete* $(r \mid p)$*-centroid problem*.

It is known that the problem is Σ_2^P-hard [9]. So, we deal with the more hard problem than any problem in the class NP. Polynomially solvable cases and complexity results can be found in [7], [9]. In order to find near optimal solutions, we present the game as a 0–1 linear bi-level problem and develop a hybrid memetic algorithm (HMA). The probabilistic tabu search heuristic (PTS) is used to improve each element of the population. To compute the leader profit, we solve the follower problem by commercial software.

To get an upper bound for this maximization problem, we rewrite the bi-level problem as a single level mixed integer linear program with an exponential

* This work was partly supported by the RFBR grant 08-07-00037, ADTP grant 2.1.1/3235.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 11–22, 2010.

number of constraints and variables. A similar approach is suggested in [10] for partial enumeration. If we extract a small family of constraints and variables, we get an upper bound. The PTS heuristic is used for generating the family. For exact approach we apply the idea of column generation. Computational experiments for Euclidean test instances from the benchmark library *Discrete Location Problems* (http://math.nsc.ru/AP/benchmarks/Competitive/p_med_comp_eng.html) indicate that the new HMA lower bound dominates the previous ones and the exact method allows to find the global optimum for $p = r = 5$, 100 facilities, and 100 clients.

The paper is organized as follows. In Section 2, we present the mathematical model. In Section 3, the lower bounds are discussed. We describe four lower bounds; three of them are quite simple, while the last one is based on the metaheuristic approach for the bi-level mathematical formulation. In Section 4, the upper bound based on the new reformulation of the problem as a mixed integer linear program with an exponential number of constraints and variables is presented. An exact column generation method is studied in Section 5. Computational results and conclusions are discussed in Sections 6 and 7.

2 Problem Formulation

We are given a set $I = \{1, \ldots, m\}$ of facilities and a set $J = \{1, \ldots, n\}$ of clients. A matrix (g_{ij}) defines the distances between clients and facilities. If client j is serviced from a facility, he gives a profit $w_j > 0$. The leader and the follower open facilities. First, the leader opens p facilities. Later on, the follower opens r facilities. Each client chooses the closest open facility. We need to find p facilities for the leader to maximize his profit. Let us present this game as a linear 0–1 bi-level programming problem. We define the decision variables [2]:

$$x_i = \begin{cases} 1 & \text{if facility } i \text{ is opened by the leader,} \\ 0, & \text{otherwise,} \end{cases}$$

$$y_i = \begin{cases} 1 & \text{if facility } i \text{ is opened by the follower,} \\ 0, & \text{otherwise,} \end{cases}$$

$$z_j = \begin{cases} 1 & \text{if client } j \text{ is serviced by the leader,} \\ 0 & \text{if client } j \text{ is serviced by the follower.} \end{cases}$$

For a given solution x, we can define the set of facilities which allow to capture client j by the follower: $I_j(x) = \{i \in I \,|\, g_{ij} < \min_{l \in I}(g_{lj} \,|\, x_l = 1)\}$, $j \in J$. Note that we consider *conservative* clients. If a client has the same distances to the closest leader and the closest follower facilities, he prefers the leader facility. So, the follower never opens a facility at a site where the leader has a facility [7]. Now the model can be written as a linear 0–1 bi-level programming problem [8]:

$$\max_x \sum_{j \in J} w_j z_j^*(x) \tag{1}$$

s.t.

$$\sum_{i \in I} x_i = p, \tag{2}$$

$$x_i \in \{0,1\}, \quad i \in I, \tag{3}$$

where $z_j^*(x)$ is a component of the optimal solution of the follower problem:

$$\max_{y,z} \sum_{j \in J} w_j (1 - z_j) \tag{4}$$

s.t.

$$\sum_{i \in I} y_i = r, \tag{5}$$

$$1 - z_j \leq \sum_{i \in I_j(x)} y_i, \quad j \in J, \tag{6}$$

$$x_i + y_i \leq 1, \quad i \in I, \tag{7}$$

$$y_i, z_j \in \{0,1\}, \quad i \in I, j \in J. \tag{8}$$

The objective function (1) defines the total profit of the leader. Equation (2) guarantees that the leader opens exactly p facilities. The objective function (4) defines the total profit of the follower. Equation (5) guarantees that the follower opens exactly r facilities. Constraints (6) determine the values of z by the decision variables y of the follower. Constraints (7) allow to open a facility by at most one decision maker. As we have mentioned, the constraints (7) are redundant. Nevertheless, we use them to reduce the feasible domain of the follower problem. Note that the optimal value for the problem does not change if we replace 0–1 variables z_j by continuous variables $0 \leq z_j \leq 1$. So, we deal with the mixed integer linear programming problem for the follower.

Matrix (g_{ij}) is used to define the set $I_j(x)$. In fact, we can use not only the distances, but any kind of preferences for the clients. For example, a client may prefer a facility with the minimal traveling and waiting time rather than with the minimal distance. In [12] the facility location models with general client preferences are studied. With almost no change, we can use the preferences in the definition of set $I_j(x)$. However, we preserve the distances for simplicity.

3 Lower Bounds

An arbitrary feasible solution of the problem (1)–(8) produces a lower bound. For a given solution x, we have to solve the problem (4)–(8) to get a feasible solution. It is an NP–hard problem [7]. We use the commercial CPLEX software for it. So, the rest of this section is devoted to describe various strategies for the selection of solution x.

The first and the simplest strategy is to ignore the follower [1]. The leader opens facilities to minimize the total distance between clients and his facilities. He wishes to service all clients and solves the classical p-median problem.

We use an optimal solution of this problem as solution x for the lower bound. This strategy is not so bad despite ignoring the follower. As we can see in our computational experiments, the leader loses more than half of the market, but we can improve the lower bound by a few percents only.

The second strategy is more sophisticated. The leader anticipates that the follower will react to his decision. So, $(p+r)$ facilities will be opened. According to the second strategy [1], the leader solves the $(p+r)$-median problem and opens the p most profitable facilities. As we will see below, this strategy is not perfect.

The third strategy is alternate. It is suggested for continuous locations in [5]. This heuristic is iterative. For a given solution of one decision maker, we find the optimal solution for another one. In discrete case this strategy produces a cycle. The best solution in the cycle is the result of the approach. If we use the previous strategies to create a starting solution, we can improve the profit of the leader. Surely, it is a more time consuming procedure.

The most powerful strategy is to solve the problem (1)–(8). We develop a hybrid memetic algorithm where a tabu search approach is used to improve the elements of the population [2]. Now we present the general framework of the method.

Hybrid memetic algorithm

1 Generate an initial population of the leader solutions.
2 Repeat until the stopping condition is met:
  2.1 Select two solutions x^1, x^2 from the population.
  2.2 Create a solution x by a recombination of x^1, x^2.
  2.3 Apply the random modification to x.
  2.4 Improve the solution by PTS heuristic.
  2.5 Update the population.
3 Return the best found solution.

We use the total number of iterations 2.1–2.5 as the stopping condition.

3.1 Initial Population

To create a high quality initial population at Step 1 of the framework, we apply the standard local improvement algorithm with random starting points. The well–known Swap neighborhood for the p-median problem is used for the improvements. Remember that we have to solve the problem (4)–(8) in order to compute the objective function value for each element of the neighborhood. The Swap neighborhood contains $p(m-p)$ elements. It is a time–consuming procedure. To reduce the running time, we use the first improvement pivoting rule, the randomization of neighborhood, and solve the linear programming relaxations to estimate the neighboring solutions.

The efficiency and robustness of the memetic algorithm depend on the population. We need different local optima. So, a new local optimum obtained is included into the population if the Hamming distance from this solution to each solution in the population is at least a given threshold. In our computational experiments, we use the threshold $\lceil 0,6p \rceil$.

3.2 Main Operators and Parameters

The selection, recombination, random modification (mutation), and replacement operators are used in the framework. The well–known tournament selection procedure [11] is applied to pick two solutions x^1, x^2. We select k solutions from the population at random and choose the best one as a parent. In our experiments, we put $k = 5$.

The recombination or crossover operator is a variant of the well-known uniform crossover [11]. The new solution x will contain all open facilities which are common for the parents. The rest of the open facilities are chosen at random with probability 0,5 from solutions x^1, x^2.

To involve a certain diversification, we use random modification of the offspring. The common bit–flip mutations are not appropriate for the problem. We may get an unfeasible solution. Instead, we produce some random modification according to the Swap neighborhood.

To update the population at Step 2.5, we use the steady–state–no–duplicates techniques. We check that no duplicate solutions are added to the population. Moreover, we calculate the Hamming distance between the new solution and the population and update the population if the distance is at least the threshold $\lceil 0, 6p \rceil$.

3.3 Tabu Search

In the well–known memetic algorithms, the standard local improvement procedure is applied to each element of the population. The algorithm finds local optima, and this feature promotes for finding global optimum or near optimal solutions. In our computational experiments for the case $w_j = 1, j \in J$, we discover a lot of local optima and plateaus. The standard local improvement procedure is not efficient for this case [1]. Therefore, we develop the PTS heuristic [6] and apply it instead of the local improvement. In [4] a tabu search algorithm is studied for the problem but a greedy procedure is applied for the follower problem. In this case we have no optimal solutions. The tabu search may produce solutions for the leader where the greedy approach has significant deviations from the optimum in the problem (4)–(8). Hence, this idea can be useful only for the instances with small p and r or particular cases of the problem. In the general case we have to apply the branch and bound method for finding optimal solution of the follower problem.

To reduce the running time at each step, we use a randomized neighborhood $N_q(x),\ q > 0$. It is the random part of the Swap neighborhood, where each element is included into the set $N_q(x)$ with probability q independently from other elements.

In order to evaluate elements of the neighborhood, we need to solve the follower problem. As mentioned above, it is an NP-hard problem. To reduce the running time, we replace the problem by its linear programming relaxation. Hence, we have a polynomial time procedure for finding the best element in the neighborhood. Below, we present the general framework of the PTS algorithm.

Probabilistic Tabu Search

1 Get offspring x from HMA and put $Tabu = \emptyset$.
2 Repeat until the stopping condition is met:
  2.1 Generate the neighborhood $N_q(x)$.
  2.2 If $N_q(x) \backslash Tabu \neq \emptyset$, then find the best element x' in the set $N_q(x) \backslash Tabu$; else $x' := x$.
  2.3 Put $x := x'$, update $Tabu$.
3 Return the best found solution.

The set $Tabu$ for the current solution contains some solutions from the Swap neighborhood. The pairs of swapping facilities are stored during a certain number of iterations, and the corresponding solutions are included into the set $Tabu$. We use the PTS algorithm for finding the high quality offspring at Step 2.4 of the HMA framework. Moreover, we collect the high quality solutions for the follower. As we will see in Section 4, any family of follower solutions allows us to compute an upper bound for the maximal profit (1) of the leader.

4 Upper Bounds

Let $\mathcal{F}$ be a family of the follower solutions. For $y \in \mathcal{F}$, $j \in J$, we introduce a set

$$I_j(y) = \big\{ i \in I \mid g_{ij} \leq \min_{l \in I} (g_{lj} | y_l = 1) \big\}.$$

The set $I_j(y)$ shows the facilities for the leader which allow him to keep the client j if the follower uses solution y. Now we rewrite the bi-level problem as a single level problem with an exponential number of constraints and variables. Let us introduce new variables:

$$z_{jy} = \begin{cases} 1 & \text{if client } j \text{ is serviced by the leader and} \\ & \text{the follower uses a solution } y, \\ 0 & \text{if client } j \text{ is serviced by the follower and} \\ & \text{the follower uses a solution } y, \end{cases} \qquad j \in J, y \in \mathcal{F},$$

$W \geq 0$ is the total profit of the leader.

If the family $\mathcal{F}$ contains all possible solutions for the follower, then the problem (1)–(8) is equivalent to the following linear 0–1 program:

$$\max W \tag{9}$$

s.t.

$$\sum_{j \in J} w_j z_{jy} \geq W, \quad y \in \mathcal{F}, \tag{10}$$

$$z_{jy} \leq \sum_{i \in I_j(y)} x_i, \quad j \in J, y \in \mathcal{F}, \tag{11}$$

$$\sum_{i \in I} x_i = p, \tag{12}$$

$$x_i, z_{jy} \in \{0, 1\}, i \in I, j \in J, y \in \mathcal{F}. \tag{13}$$

The objective function indicates the goal of the leader. The constraints (10) guarantee the best answer of the follower. The constraints (11) determine the market share for each follower solution. If the leader has no facilities in the set $I_j(y)$, then client j is serviced by a follower facility.

Note that the optimal value of the problem does not increase if we replace 0–1 variables z_{jy} by continuous variables $0 \leq z_{jy} \leq 1$. So, we get the mixed integer linear programming problem with m Boolean variables x_i, an exponential number of continuous variables z_{jy}, and constraints (10), (11).

In order to get an upper bound, we select a small subset of the *strong* solutions for the follower. Denote by $W(\mathcal{F})$ the optimal value of the problem (9)–(13) for the family $\mathcal{F}$. The most difficult task is finding an appropriate family. We produce it by the PTS algorithm at Step 2.4 of the HMA framework. So, we use metaheuristics to get lower and upper bounds.

5 An Exact Method

Let $x(\mathcal{F})$ be the optimal solution of the problem (9)–(13). The corresponding optimal solution of the follower problem (4)–(8) denoted by $y(\mathcal{F}) = y^*(x(\mathcal{F}))$, $z(\mathcal{F}) = z^*(x(\mathcal{F}))$. This solution defines a lower bound $LB(\mathcal{F}) = \sum_{j \in J} w_j z_j^*(\mathcal{F})$. If $LB(\mathcal{F}) = W(\mathcal{F})$, we have the optimal solution for the bi-level problem (1)–(8). Otherwise, we enlarge the family by adding $y(\mathcal{F})$ and repeat the calculations. This iterative method can be described as follows.

Iterative exact method

1 Choose an initial family $\mathcal{F}$.
2 Find the solution $x(\mathcal{F})$ and upper bound $W(\mathcal{F})$.
3 Solve the follower problem and find $y(\mathcal{F}), LB(\mathcal{F})$.
4 If $W(\mathcal{F}) = LB(\mathcal{F})$ then return the best found solution and STOP.
5 Include the solution $y(\mathcal{F})$ into the family $\mathcal{F}$ and go to Step 2.

Let us verify that the method is exact and finite indeed. Assume that we solve the follower problem at Step 3 and find $y(\mathcal{F})$, but $y(\mathcal{F}) \in \mathcal{F}$. From (10) we have $LB(\mathcal{F}) = \sum_{j \in J} w_j z_j^*(\mathcal{F}) = \sum_{j \in J} w_j z_{jy(\mathcal{F})} \geq W(\mathcal{F})$. So, $LB(\mathcal{F}) = W(\mathcal{F})$ and $x(\mathcal{F})$ is the optimal solution of the bi-level problem. The method is finite because $|\mathcal{F}| \leq \binom{m}{r}$.

Step 2 is the most time consuming. We have to solve a large scale optimization problem. If we use the branch and bound method, we get $W(\mathcal{F})$ and $x(\mathcal{F})$. But the method spends a lot of time proving optimality. Actually, we need the

solution only. Therefore, we may reduce the running time if replace the problem (9)–(13) by the following feasibility problem. By W^* denote the optimum for the bi-level problem (1)–(8) and consider the following system:

$$\sum_{j \in J} w_j z_{jy} > W^*, \quad y \in \mathcal{F}, \tag{14}$$

$$z_{jy} \leq \sum_{i \in I_j(y)} x_i, \quad y \in \mathcal{F}, j \in J, \tag{15}$$

$$\sum_{i \in I} x_i = p, \tag{16}$$

$$x_i \in \{0, 1\}, 0 \leq z_{jy} \leq 1, \quad i \in I, j \in J, y \in \mathcal{F}. \tag{17}$$

If we have a feasible solution $x(\mathcal{F})$ for it, we include $y(\mathcal{F})$ into family $\mathcal{F}$ and repeat the calculations. Otherwise, we can stop the search with the appropriate family. The feasibility problem is easier. We do not need to prove optimality. We may apply the feasibility pump [3], the branch and bound method with convenient objective function, or metaheuristics again.

Of course, we do not know the optimal value W^*. So, we use the best found value $W' \leq W^*$ by the HMA and update it during the search. Now the framework of the modified exact method is the following.

Modified exact method

1 Apply HMA to create an initial family $\mathcal{F}$ and W'.
2 Find feasible solution $x(\mathcal{F})$ for the system (14)–(17).
If it is infeasible then return the best found solution and STOP.
3 Solve the follower problem and find $y(\mathcal{F}), LB(\mathcal{F})$.
4 If $W' < LB(\mathcal{F})$ then $W' := LB(\mathcal{F})$.
5 Include $y(\mathcal{F})$ into family $\mathcal{F}$ and go to Step 2.

Further improvements of the method can deal with decreasing the family from time to time or generating several feasible solutions at Step 2.

6 Computational Results

The developed memetic algorithm and the exact method were coded in GAMS (General Algebraic Modeling System) and tested on the instances from the benchmark library *Discrete Location Problems*. For all instances, clients and facilities are in the same sites, $I = J$. The elements of matrix (g_{ij}) are Euclidean distances between points i, j in the two dimensional Euclidean plane. The points are chosen at random uniformly in the square 7000×7000. All experiments are carried out at the PC Pentium Intel Core 2, 1.87 GHz, RAM 2 Gb, running under the Windows XP Professional operating system.

In the first series of experiments we compare the lower and upper bounds. Table 1 shows computational results for the instances with $n = m = 100$, $p = r = 10$, $w_j = 1, j \in J$. The first column indicates the names of the instances. Columns $LB(p)$, $LB(p+r)$, AH, HMA present the lower bounds for the strategies based on the classical p-median problem, the $(p+r)$-median problem [1], the alternating heuristic [5], and the hybrid memetic algorithm, respectively. Note that the HMA lower bound dominates the other ones. Nevertheless, the difference between HMA and $LB(p)$ bounds is small. For the instance 511, these bounds coincide. We may conclude that $LB(p)$ is a good approximation for the leader behavior. Moreover, the running time for the $LB(p)$ is a few seconds only. For the HMA bound, we need about 7 hours of running time if we terminate calculations after 100 iterations with the population of size 25 and 100 iterations of the PTS algorithm for the local improvement to each individual. The alternating heuristic produces good approximations as well. It is more time consuming than $LB(p)$ strategy but can get a better lower bound. In brackets, we show the length of the cycle for the heuristic. Columns UB_0 and W show

Table 1. Lower and upper bounds

Instance	$LB(p)$	$LB(p+r)$	AH	HMA	UB_0	W
111	41	31	49 (2974)	50	74	54
211	41	36	45 (287)	49	70	57
311	46	41	44 (444)	48	73	55
411	41	39	45 (813)	49	72	58
511	48	40	47 (138)	48	70	55
611	42	39	46 (3607)	47	67	57
711	49	37	48 (675)	51	73	55
811	42	37	44 (255)	48	74	55
911	47	35	46 (2486)	49	68	54
1011	46	33	47 (4963)	49	70	54

two upper bounds. To compute UB_0, we need to rank the facilities for the follower [8]. We suppose that the follower uses this ranking instead of the optimal strategy. In this case the bi-level problem can be presented as a mixed integer linear program. An optimal value of the program is the upper bound UB_0. In our experiments, we use the ranking obtained by the Lagrangian relaxations [1]. In order to compute W, we need family $\mathcal{F}$. As mentioned above, we collect optimal solutions of the follower by the PTS algorithm at Step 2.4 of the HMA framework. The cardinality of the family is 400. Table 1 shows that the upper bound W dominates UB_0 substantially. Nevertheless, we cannot prove the optimality of the HMA solutions by substantially increasing the families. We believe that the HMA solutions are optimal. So, we need more intelligent search strategies for creating the families.

The second series of experiments is devoted to the modified exact method. Our goal is to investigate the influence of parameters p and r on the optimal families

Table 2. Optimal solutions

	$w_j = 1$			$w_j \in (0, 200)$		
Instance	*Opt*	*Iter*	*Time*	*Opt*	*Iter*	*Time*
111	47	123	120	4139 (47%)	98	65
211	48	69	60	4822 (45%)	127	37
311	45	231	3600	4215 (45%)	262	5460
411	47	111	150	4678 (47%)	128	900
511	47	106	120	4594 (44%)	190	720
611	47	102	90	4483 (47%)	121	660
711	47	115	180	5153 (46%)	167	2550
811	48	67	42	4404 (46%)	190	720
911	47	108	160	4700 (45%)	247	2520
1011	47	124	165	4923 (48%)	83	30

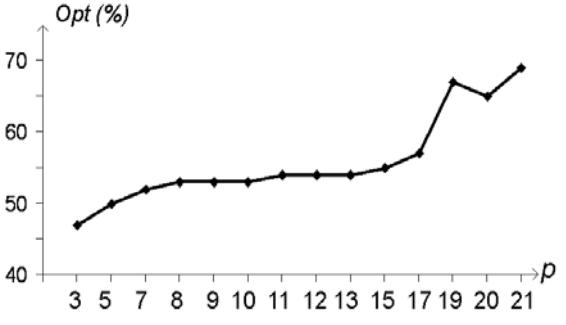


Fig. 1. The segment of the leader in percents, $p = r$

or the number of iterations, and study the market share. Table 2 presents computational results for Euclidean instances from the same library. For all instances, we have $n = m = 100$, $p = r = 5$ and two classes of weights: $w_j = 1$ and $w_j \in (0, 200), j \in J$. Note that up to now there are no reported results for $m > 70$. Columns *Opt* show optimal values for the instances and the leader profits in percents (market share). Columns *Iter* present the total number of iterations of the modified exact method. Columns *Time* indicate the running time, in minutes, for the method excluding the time for the HMA heuristic. We can see that the follower gets more than 50% of the market. He has some advantages for small p and r. For large values of p and r the leader has some advantages. Figure 1 indicates the segment of the leader, in percents, for $p = r$, $n = m = 50$, $w_j \in (0, 200), j \in J$.

The segment increases when p and r grow. The decision makers obtain a half of the market for $p = r = 7$. So, the leader should open many facilities to control the most of the market. Of course, his segment decreases when r grows and vice versa.

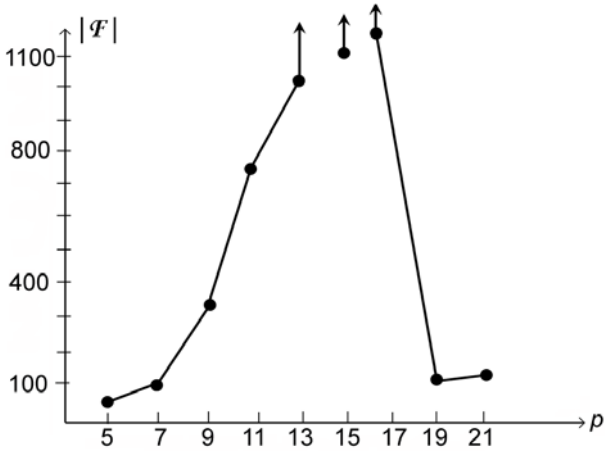


Fig. 2. The number of iterations for the modified exact method, $p = r$

In the third series of experiments we study the families of the follower solutions. Figure 2 shows the cardinalities of the families (the total number of iterations) for $n = m = 50$, $p = r$, $w_j \in (0, 200), j \in J$. The problem is easy when p and r are small or large. The total number of iterations is about one hundred. The problem becomes hard when $11 \leq p \leq 17$. We need more than a thousand follower solutions in the family. We guess that the case $p = r = [m/3]$ is the most difficult for the method even if we have the optimal value W^*. Huge family is a reason why we cannot prove optimality of the HMA solutions for $p = r = 10$.

7 Conclusions

We consider the well-known discrete $(r \mid p)$-centroid problem and develop a hybrid memetic algorithm for finding near optimal solutions and the exact column generation method. The problem is Σ_2^P–hard. We use commercial software to compute the objective function values for the feasible solutions of the leader. These solutions are used in the evolutionary algorithm as population members. A probabilistic tabu search heuristic is applied for improving the offspring at each step of the evolution. In order to get an upper bound, we reformulate the bi-level problem as a single level mixed integer programming problem with an exponential number of constraints and variables. Metaheuristics are used to collect an appropriate subset of the constraints and variables. To find the global optimum, we develop the modified iterative method. The feasibility subproblem is solved at each iteration. It seems interesting to study metaheuristics for the subproblem later on.

References

1. Alekseeva, E., Kochetova, N.: Upper and lower bounds for the competitive p-median problem. In: Proceedings of XIV Baikal International School–Seminar Optimization Methods and Their Applications, Irkutsk, vol. 1, pp. 563–569 (2008) (in Russian)
2. Alekseeva, E., Kochetova, N., Kochetov, Y., Plyasunov, A.: A hybrid memetic algorithm for the competitive p-median problem. In: Preprints of the 13th IFAC Symposium on Information Control Problems in Manufacturing, Moscow, pp. 1516–1520 (2009)
3. Bertacco, L., Fischetti, M., Lodi, A.: A feasibility pump heuristic for general mixed–integer problems. Discrete Optimization 4(1), 63–76 (2007)
4. Benati, S., Laporte, G.: Tabu search algorithms for the $(r|X_p)$–medianoid and $(r|p)$–centroid problems. Location Science 2(4), 193–204 (1994)
5. Bhadury, J., Eiselt, H., Jaramillo, J.: An alternating heuristic for medianoid and centroid problems in the plane. Computers and Operations Research 30, 553–565 (2003)
6. Glover, F., Laguna, M.: Tabu Search. Kluwer Acad. Publ., Boston (1997)
7. Hakimi, S.L.: Locations with spatial interactions: competitive locations and games. In: Mirchandani, P.B., Francis, R.L. (eds.) Discrete Location Theory, pp. 439–478. Wiley & Sons, Chichester (1990)
8. Kochetov, Y.A., Kononov, A.V., Plyasunov, A.V.: Competitive facility location models. Computational Mathematics and Mathematical Physics 49, 994–1009 (2009)
9. Noltermeier, H., Spoerhose, J., Wirth, H.C.: Multiple voting location and single voting location on trees. European J. Oper. Res. 181, 654–667 (2007)
10. Rodriguez, C.M.C., Perez, J.A.M.: Multiple voting location problems. European J. of Oper. Res. 191(2), 437–453 (2008)
11. Sastry, K., Goldberg, D., Kendall, G.: Genetic algorithms. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques, pp. 97–126. Springer, Heidelberg (2005)
12. Vasil'ev, I.L., Klimentova, K.B., Kochetov, Y.A.: New lower bounds for the facility location problem with clients' preferences. Computational Mathematics and Mathematical Physics 49, 1010–1020 (2009)

On the Benefit of Sub-optimality within the Divide-and-Evolve Scheme

Jacques Bibai[1,2], Pierre Savéant[2],
Marc Schoenauer[1], and Vincent Vidal[3]

[1] Projet TAO, INRIA Saclay & LRI-CNRS, Univ. Paris Sud, Orsay, France
firstname.lastname@inria.fr
[2] Thales Research & Technology, Palaiseau, France
firstname.lastname@thalesgroup.com
[3] ONERA – DCSD, Toulouse, France
Vincent.Vidal@onera.fr

Abstract. *Divide-and-Evolve* (DaE) is an original "memeticization" of Evolutionary Computation and Artificial Intelligence Planning. DaE optimizes either the number of actions, or the total cost of actions, or the total makespan, by generating ordered sequences of intermediate goals via artificial evolution, and calling an external planner to solve each subproblem in turn. DaE can theoretically use any embedded planner. However, since the introduction of this approach only one embedded planner had been used: the temporal optimal planner CPT. In this paper, we propose a new version of DaE, using time-based Atom Choice and embarking the sub-optimal planner YAHSP in order to test the robustness of the approach and to evaluate the impact of using a sub-optimal planner rather than an optimal one, depending on the type of planning problem.

1 Introduction

An Artificial Intelligence (AI) planning problem is specified by the description of an initial state, a goal state, 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, whose execution from the initial state transforms it into a state that includes the goal state. The quality criterion of a plan depends on the type of available actions: number of actions in the simplest case; total cost for actions with cost; total makespan for durative actions which, in addition, may temporally overlap.

Domain independent planning is a fundamental and dynamic field of AI. Planning problems have been tackled with a large number of methods and algorithms: heuristic search (LAMA [15], FF [11], Fast Downward [10], YAHSP [19]), local search (LPG [7,8]), and constraint programming (CPT [20,21]). Among all these directions of planning research, and following some considerable successes of evolutionary algorithms in other areas of AI, Genetic Planning was introduced with the purpose to translate those success to planning problems. Introduced in

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 23–34, 2010.

[12], several approaches to Genetic Planning have been proposed [18,14,22,23,4]. However, due to the limited performance of the resulting planning systems, the relevance of the possibility of application of EAs to planning was not deemed significant in comparison to the traditional methods.

Recently, the authors introduced *Divide-and-Evolve* (DaE) [16,17], an original hybridization of Evolutionary Algorithms (EAs) with Artificial Intelligence Planning. The baseline of DaE is to generate a sequence of partial states, thus replacing the initial problem by a sequence of hopefully simpler subproblems, and calling an external traditional planner to solve each subproblem in turn. The final solution is then built by concatenation of from all subproblem solutions. DaE has entered the last International Planning Competition (IPC) [2]: whereas hindered by the tight time limit (30mn), the quality of the solution plans it obtained on all the instances that it could solve in the temporal track was generally better than that of its competitors.

Until today, although DaE can theoretically use any embedded planner, all experiments based on DaE have been performed using the optimal planner CPT as the embedded planner. The goal of this paper is to compare the performances of the DaE approach when using either CPT or a sub-optimal planner, such as YAHSP. The robustness of the hybrid algorithms and the quality of the solution plans they can find will be experimentally compared on all kinds of AI planning problems. Furthermore, it has been empirically demonstrated [1] that the results of DaE can be improved by a careful choice of the atoms that are used to build the partial states: the new version of DaE that is proposed here makes use of such time-based atom choice.

The paper is organized as follows: Section 2 briefly introduces planning problems; Section 3 recalls the *Divide-and-Evolve* approach, representation, fitness function and variation operators; Section 4 presents the results. The last section discusses results of DaE using either CPT or YAHSP as embedded planner, and sketches some directions for future work.

2 Planning Problems

Domain-independent planners rely on the Planning Domain Definition Language (PDDL) [13], inherited from the STRIPS model [5], to standardise and represent a planning problem. The language has been extended for representing temporality and action concurrency in PDDL2.1 [6].

The description of a planning problem splits into 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 state changes, 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 might be a partial state, i.e., it can be true in many

different (complete) 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, and possibly a duration or a cost. Preconditions are logical constraints which apply domain predicates to the arguments and trigger the effects when they are satisfied. Effects enable state transitions by adding or removing atoms.

A solution to a planning problem is a consistent schedule of grounded actions whose execution in the initial state leads to a state that contains one goal state, i.e., where all atoms of the problem goal are true.

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

In order to solve a planning problem $\mathcal{P}_D(I, G)$, the basic idea of DAE is to find a sequence of states $S_1, \ldots, S_n$, and to use some embedded 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 generation and optimization of the sequence of states (S_i) is driven by an evolutionary algorithm, and this Section will describe its main components: the problem-specific representation, the fitness computation, and the variation operators.

3.1 Representation

As described in Section 2, a state is a list of atoms built over the set of predicates and the set of object instances. However, searching the space of complete states would result in a rapid explosion of the size of the search space. Moreover, goals of planning problem might be defined as partial states. It thus seems more practical to search only sequences of partial states, and to limit the choice of possible atoms used within such partial states. However, this raises the issue of the **choice of the atoms** to be used to represent individuals, among all possible atoms.

Previous experiments on different domains of temporal planning problems from the IPC benchmark series [1] demonstrated the need for a very careful choice of the atoms that are used to build the partial states. This lead the authors to propose a new method to build the partial states, based on the earliest time from which an atom can become true. Such time can be estimated by some admissible heuristic function (e.g h^1, h^2...) [9]. These start times are then used in order to restrict the candidate atoms for each partial state: A partial state is built at a given time by randomly choosing among several atoms that are possibly true at this time. The sequence of states is hence built by preserving the estimated chronology between atoms (**time consistency**). Heuristic function h^1 has been used for all experiments presented here.

Nevertheless, even when restricted to specific choices of atoms, the random sampling can lead to inconsistent partial states, because some sets of atoms can

be *mutually exclusive*[1] (`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. In practice, it is difficult to decide if several atoms are `mutex`. Nevertheless, binary `mutexes` can be approximated (i.e. not all pairs of mutually exclusive atoms can be discovered) with a variation of the h^2 heuristic function [9] in order to build quasi pairwise-mutex-free states (i.e., states where no pair of atoms are mutex).

An individual in the new version of DAE is hence represented as a variable length ordered h^1-time-consistent list of partial states, and each state is a variable length list of atoms that are not pairwise h^2-`mutex`.

3.2 Fitness, and Embedded Planners

The fitness of a list of partial states $S_1, \ldots, S_n$ is computed by repeatedly calling an external 'embedded' 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, and up to now, only CPT has been used within the DAE approach. CPT [20,21] is an exact planning system which combines a branching scheme based on Partial Order Causal Link (POCL) Planning with powerful and sound pruning rules implemented as constraints. But is optimality mandatory in order for DAE to obtain good quality results? In order to address this issue, a sub-optimal planner, YAHSP will be used here too. YAHSP [19] is a lookahead strategy planning system for sub-optimal STRIPS planning which uses the actions in the relaxed plan to compute reachable states in order to speed up the search process.

For any given k, if the chosen embedded planner succeeds in solving $\mathcal{P}_D(S_k, S_{k+1})$, the final complete state is computed by executing the solution plan from S_k, and becomes the initial state of the next problem. If all problems, $\mathcal{P}_D(S_k, S_{k+1})$ are solved by the chosen embedded planner, the individual is called *feasible*, and the concatenation of all solution 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 further optimised by rescheduling some of its actions, in a step called *compression* (see [17] for detailed discussion). The quality of the compressed plan defines the fitness of a feasible individual.

However, as soon as the chosen embedded planner 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 the chosen embedded planner, as its initial state is in fact partially unknown. Hence no quality in term of number of action, cost or makespan can be given to this individual. All such individuals receive a fitness that is higher than that of any feasible individual. Furthermore, in order to nevertheless give some selection pressure toward feasible individuals, such fitness takes into account the proportion of subproblems solved.

Finally, because the initial population contains randomly generated individuals, some of them might contain some subproblems that are in fact more difficult

[1] Several atoms are mutually exclusive when there exists no plan that, when applied to the initial state, yields a state containing them all.

than the original global problems. It was thus necessary to limit the embedded planner by imposing some complexity bound in order to discard too difficult subproblems. However, though it is hoped that all subproblems will ultimately be easy to solve, such limitation should not be too strong in order to nevertheless leave some degree of freedom to the search for solutions.

Here, CPT (resp. YAHSP) has been limited by a **maximal number of backtracks** (resp. a **maximal number of nodes**) that it is allowed to use to solve any of the subproblems. Those bounds are determined anew for each run by a two-step process: first, the initial population is evaluated using a very high bound (e.g. 100000 backtracks or nodes); the bounds for the rest of the run are then chosen as the median of the actual number of backtracks (resp. nodes) that have been used to find the solutions during these initial evaluations.

3.3 Initialization and Variation Operators

The initialization phase and the variation operators of the DAE algorithm respectively build the initial sequences of states and randomly modify some sequences during its evolutionary run.

The **initialization** of an individual is the following: first, the number of states is uniformly drawn between 1 and the number of estimated start times (see Section 3.1); For every chosen time, the number of atoms per state is uniformly chosen between 1 and the number of atoms of the corresponding restriction. Atoms are then chosen one by one, uniformly in the allowed set of atoms, and added to the individual if not `mutex` with any other atom that is already there.

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

Because an individual is a variable length list of states, and a state is a variable length list of atoms, the **mutation** operator can act at both levels: at the individual level by adding (**addState**) or removing (**delState**) a state; or at the state level by adding (**addAtom**) or removing (**delAtom**) some atoms in the given state.

Note that the initialization process and these variation operators maintain the chronology between atoms in a sequence of states and the local consistency of a state, i.e. avoiding pairwise mutexes.

3.4 Evolution Engine and Parameter Settings

A general issue in Evolutionary Computation (EC) lies in the number of parameters that the programmer has to tune (from population size to selection operators to rates of applications of variation operators), and the lack of theoretical guidance to help him. Experimental statistical procedures have been proposed

(e.g [24],[25]), that build on standard Design of Experiments methods and use the specificities of the EC domain to reduce the amount of computations.

In order to tune DAE, [3] proposed a two steps learning approach which involves choosing the probability and weights of each of the variation operators being used with racing [24], and then choosing which predicates will be used to describe the intermediate goals with statistical analysis. In this paper we use the first step of [3] approach in several domains of IPC benchmarks and chose to keep the best common parameters configuration for all experiments of this paper.

The **evolution engine** has been chosen [1] to be a (10+70)-ES: 10 parents generate 70 offspring using variation operators, and the best of those 80 individuals are 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 seen during 50 generations, with a maximum of 1000 generations altogether. The probabilities of individual-level application of crossover and mutation (p_{cross} and p_{mut}) are (0.2, 0.8) and the relative weights of the 4 mutation operators ($w_{addState}$, $w_{delState}$, $w_{addAtom}$, $w_{delAtom}$) are (3,1,1,1).

4 Experimental Results

Divide-and-Evolve has been implemented within the Evolving Objects framework[2], an open source, ANSI C++ STL-based evolutionary computation library. In order to illustrate the behaviour of DAE with each embedded planner, and to compare those implementations of DAE in all kind of planning problems, the following IPC benchmark domains have been used: `airport`, `satellite` and `logistics` domains for simple planning problem, `openstacks`, `scanalyser` and `woodworking` domains of the sixth IPC sequential satisficing track involving actions with costs, and `crewplanning`, `elevator` and `satellite time windows compiled` domains for temporal planning problem (actions with duration). Each domain has several instances of increasing complexity, resulting in a total of 470 problems.

Performance Measures:
All algorithms are given at most 2 hours of CPU time for each run on each problem instance. Their **efficiency** is then measured by the number of instances solved on each domain.

The quality of the plans are evaluated using IPC rules. For a given instance i, let Q_i^* be the best plan quality found among the competitor planners. The quality ratio for each planner is defined by Q_i^*/Q_i (in $[0, 1]$). The **quality score** of a planner for domain $\mathcal{D}$ is the sum over all instances of $\mathcal{D}$ of the quality ratios of this planner. The planner with the highest quality score is designated as the best performer on the domain. Note that if a planner cannot find a plan for a given instance after 2 hours, its quality ratio is set to 0 for this instance.

[2] http://eodev.sourceforge.net/

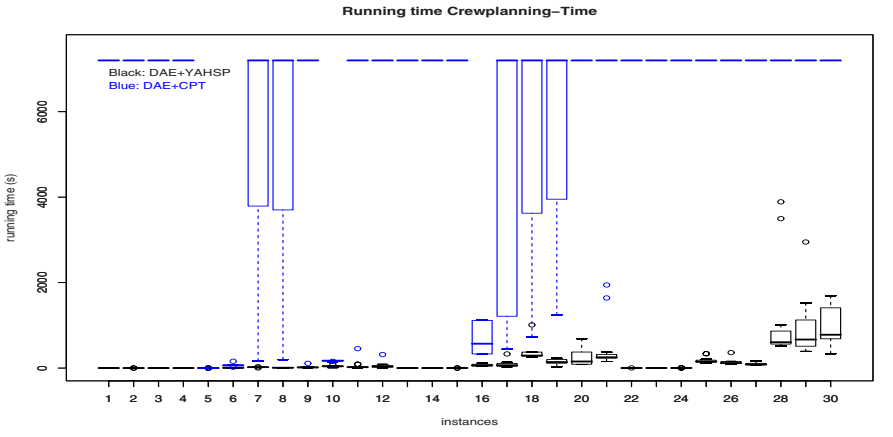


Fig. 1. Standard boxplots (see caption of Fig. 3 for boxplot interpretation) for the runtime distributions of DAE$_{\text{CPT}}$ (blue, always on top) and DAE$_{\text{YAHSP}}$ (black, at bottom) on `crewplanning-Time` domain. The total runtime was bounded by 2 hours, hence the upper-limit of almost all boxplots for CPT.

However, because DAE$_{\text{YAHSP}}$ and DAE$_{\text{CPT}}$ are stochastic algorithms, 11 runs are performed on each instance in order to assess their robustness. Their **efficiency** per domain is defined as the total number of instances that have been solved at least once. The **average efficiency** for a given domain $\mathcal{D}$ is defined as $\frac{\sum_{i;n_i>0} n_i}{\sum_{i;n_i>0} 1}$, where n_i is the number of successful runs (i.e., that found a plan) for instance i of $\mathcal{D}$. It lies in $[0, 11]$. The **average quality** for domain $\mathcal{D}$ is defined as the sum over all solved instances i of $\mathcal{D}$ of $\frac{1}{n_i} \sum_{\{j \text{ solved } i\}} \frac{Q_i^*}{q_j}$ where q_j is the quality of the plan found by run j – the closer from the efficiency, the better.

Results:
First column (resp. second column) of Table 1 shows for all algorithms the best efficiency $S_{planner}$ (resp. quality $Q_{planner}$) together with, in parentheses, the average efficiency (resp. average quality) for both DAE variants. Last column is the ratio $Q_{planner}/S_{planner}$. The mean values of those figures across test domains are also provided, by domain category, and over all domains.

Figure 3 shows, for all algorithms, the plan quality of all instances across 3 different domains (one of each category). Each column corresponds to an instance (number on the X axis). For the original planners YAHSP and CPT, symbols ('△' and '¤' respectively) indicate the plan quality found. For both DAE variants, standard boxplots sketch the distribution of the qualities of the 11 plans. Figure 1 displays, for both DAE variants, the standard boxplots for the distribution of the 11 running times, and figure 2 shows one typical example of the fitness behaviour along evolution on `elevator-Time` problem 2.

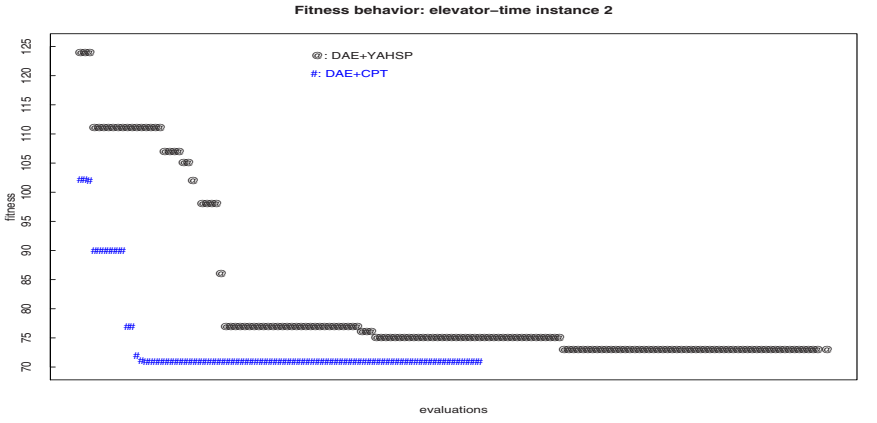


Fig. 2. Fitness behaviour of DAE$_{\text{CPT}}$ (#) and DAE$_{\text{YAHSP}}$ (@) on the easy `elevator-Time` problem number 2

Discussion:
First, DAE$_{\text{YAHSP}}$ solves significantly more problems (79.36% of all problems) than YAHSP alone (71.49% of all problems), and much more than DAE$_{\text{CPT}}$ (21.70%) and CPT alone (10% only). Then, DAE$_{\text{YAHSP}}$ has the best quality score (see last line of Table 1) for all kinds of planning problem. Furthermore, DAE$_{\text{YAHSP}}$ consistently finds (see Table 1) either the optimal value, or a value more than 94% of the optimal value (as found by CPT) (Figure 3), and always finds a better plan quality than YAHSP alone (Figure 3 and Table 1). The running times of DAE$_{\text{YAHSP}}$, for instance on the `crewplanning` domain (Figure 1), are always smaller than those of DAE$_{\text{CPT}}$ (in fact, 2 hours was not enough for DAE$_{\text{CPT}}$). Thus, the variance of plan quality of DAE$_{\text{YAHSP}}$ (Figure 3) is generally smaller than that of DAE$_{\text{CPT}}$.

However, although the DAE$_{\text{YAHSP}}$ planner has the best sub-optimal ratio over all tested domains (last line of Table 1), DAE$_{\text{CPT}}$ has the best ratio on temporal domains (line 13 of Table 1). This is due to the quality of the compression step with the CPT constraint representation, where causal links and partial orders are inferred and exploited – which is not the case when YAHSP solved the subproblems. Nevertheless, there is no absolute best method here: Even in the case where one DAE variant obtains the best ratio value on a given type of problems, there is always at least one domain of this type where the other variant performs better on all instances it could solve (see table 1). See for instance, the `crewplanning` domain for temporal planning problems, and `airport` and `woodworking` domains for the other types of planning problem.

In all tested domains, DAE$_{\text{CPT}}$ (respectively DAE$_{\text{YAHSP}}$) could solve more problems than CPT (respectively YAHSP) alone. Furthermore, the average efficiency of both variants is very high (close to the maximum value 11), DAE$_{\text{YAHSP}}$ being slightly more robust than DAE$_{\text{CPT}}$: when an instance is solvable, almost all runs succeed. Regarding the quality robustness, the average quality of both variants is generally more than 90% of the quality score.

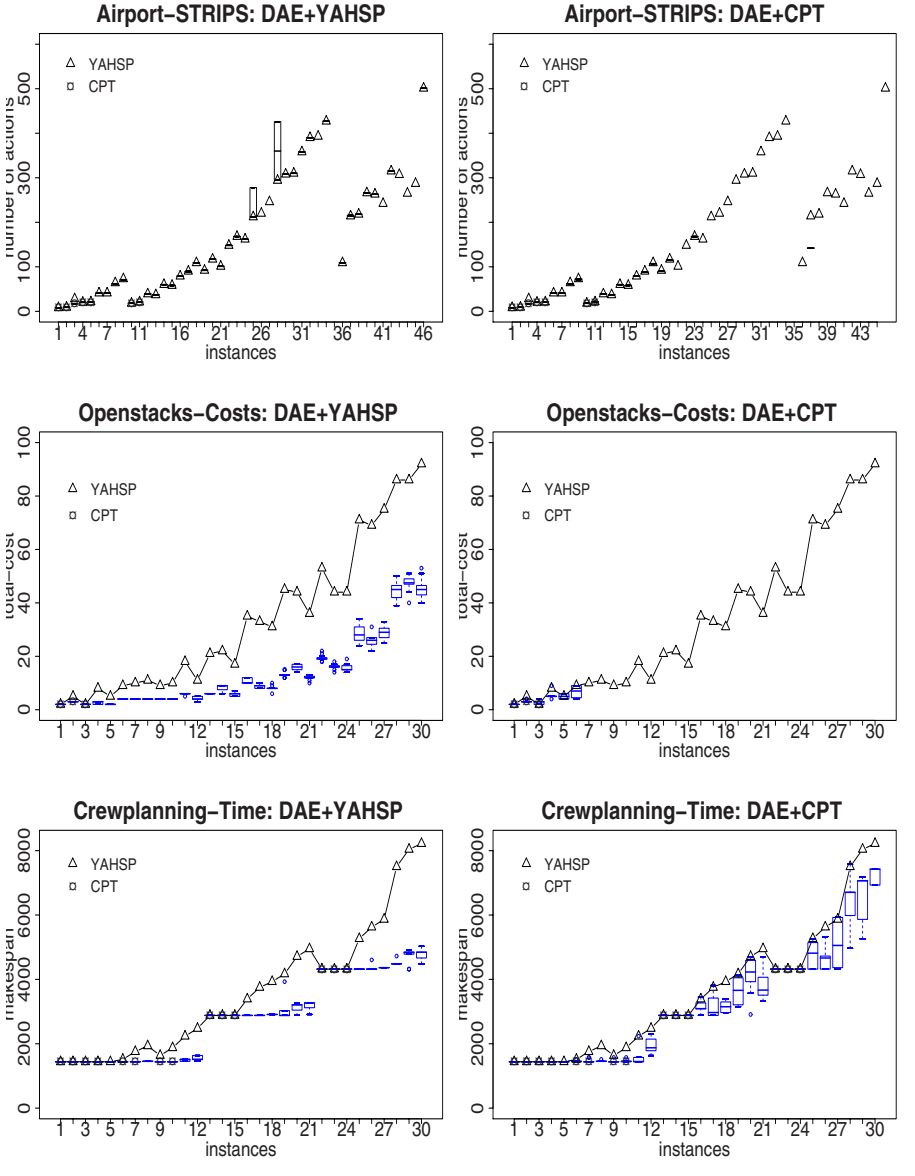


Fig. 3. Best plan quality found by CPT (¤), YAHSP value (△) and the corresponding DAE variant (for each instance, one standard boxplot sketches the distribution of the 11 runs) on `aiport-STRIPS`, `openstacks-Costs` and `crewplanning-Time` domains. The boxplots follow standard usage: the central box 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.

Table 1. Quality and scaling of the optimal planner CPT, sub-optimal planners YAHSP, DAE$_{CPT}$ and DAE$_{YAHSP}$ across the test domains. In column `Domain`(x), x denotes the total number of problem instances. Column 2-5 display the efficiency, i.e. number of instances solved (and also, for the DAE variants, the average number of successful runs – the closest to 11 the better). Column 7-10 show the quality score (and in parentheses, for DAE variants, the average efficiency, the closest to the quality score the better). See text for the exact definitions. The values in bold are the best values obtained on each type of planning problem (STRIPS, Cost, and Temporal). Column 11-15 displays the ratios $\frac{Quality\ Score}{Efficiency}$ on each domain (with means of those ratios across the domain types). The underlined values in those columns are the best values obtained on each domain by the sub-optimal planners YAHSP, DAE$_{CPT}$ and DAE$_{YAHSP}$.

Domain	Efficiency				Quality				Quality/ Total of solved problems			
	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$	YAHSP	CPT	DAE$_{YAHSP}$	DAE$_{CPT}$
Airport-STRIPS(50)	46	7	43 / 8.2	22 / 7.2	44.34	7	42.62 (42.1)	22 (22)	96.39%	100%	99.13%	100%
Satellite-STRIPS(36)	24	2	31 / 10.4	4 / 11	14.08	2	31 (30.8)	3.80 (3)	58.69%	100%	100%	95.24%
Logistics-STRIPS(198)	125	9	142 / 9.3	11 / 9.9	92.07	9	141.94 (133.5)	10.93 (10.9)	73.66%	100%	99.96%	99.45%
Total problems (284)	*195*	*18*	***216***	*37*	*150.50*	*18*	***215.56***	*36.74*	*76.25%*	***100%***	***99.70%***	*98.23%*
Openstacks-Cost(30)	30	3	30 / 10.9	6 / 6.8	11.60	3	30 (27.2)	5 (4.1)	38.70%	100%	100%	83.33%
Scanalyser-Cost(30)	27	3	28 / 10.5	6 / 7.7	16.48	3	27.81 (26)	5.92 (5.8)	61.04%	100%	99.35%	98.81%
Woodworking-Cost(30)	23	1	23 / 10.4	5 / 9	20.10	1	22.98 (22.3)	5 (3)	87.42%	100%	99.95%	100%
Total problems (90)	*80*	*7*	***81***	*17*	*48.19*	*7*	*80.80*	*15.92*	*62.39%*	***100%***	***99.77%***	*94.05%*
Crewplanning-Time(30)	30	14	30 / 10.9	30 / 9.5	24.65	14	29.97 (29.5)	29.01 (27.4)	82.19%	100%	99.93%	96.73%
Elevator-Time(30)	21	1	30 / 10.8	4 / 7	11.55	1	28.99 (23.6)	3.86 (3.6)	55.03%	100%	96.64%	96.74%
Satellite-Time(36)	10	7	16 / 9.2	14 / 8.1	7.47	7	15.15 (14.5)	14 (13.7)	74.78%	100%	94.72%	100%
Total problems (96)	*61*	*22*	***76***	*48*	*43.69*	*22*	***74.12***	*46.88*	*70.67%*	***100%***	*97.10%*	***97.82%***
Total (470)	336	47	**373**	102	242.39	47	**370.49**	99.56	69.77%	**100%**	**98.85%**	96.70%

5 Conclusion

Divide-and-Evolve is an original "memeticization" of Evolutionary Computation and AI Planning. However, since the introduction of this approach, only one embedded planner (the optimal CPT [20,21]) had been used within DAE. The results presented in this paper, relying on the implementation of the time-based Atom Choice introduced in [1], demonstrate that

- it is indeed possible to embed in DAE another planner (here, the sub-optimal YAHSP [19]);
- when using the suboptimal YAHSP, DAE can greatly improve the plan quality over that reached by YAHSP alone in all kind of planning problems;
- when embedding YAHSP, DAE is able to solve more problems within a 2 hours limit than when using the optimal planner CPT;
- the quality of the plans found by the $\text{DAE}_{\text{YAHSP}}$ version is very high, higher than that of DAE_{CPT} in simple and cost domains, and almost as good even in the case of temporal domains, where the compression step is much more efficient with the constraint-based CPT solver.

But there is still room for large improvements for DAE. First, it should be possible to define constraints in order to discard subproblems that are more difficult than the original global problem. This open issue could be addressed using for instance the empirical formulae of [1] that were designed to reduce the running time of DAE_{CPT} on temporal planning problems. Second, building on those results, it should be possible to combine several planners, taking advantage of the specificities of each of them by letting the Evolutionary Algorithm choose, for each subproblem, which planner to use. Such directions are the subject of on-going research.

References

1. Bibai, J., Schoenauer, M., Savéant, P.: Divide-And-Evolve Facing State-of-the-Art Temporal Planners during the 6^{th} International Planning Competition. In: Cotta, C., Cowling, P. (eds.) EvoCOP 2009. LNCS, vol. 5482, pp. 133–144. Springer, Heidelberg (2009)
2. Bibai, J., Savéant, P., Schoenauer, M., Vidal, V.: DAE: Planning as Artificial Evolution (Deterministic part). In: At International Planning Competition, IPC (2008), http://ipc.icaps-conference.org/
3. Bibai, J., Savéant, P., Schoenauer, M., Vidal, V.: Learning Divide-and-Evolve Parameter Configurations with Racing. In: Coles, A., et al. (eds.) ICAPS 2009, Workshop on Planning and Learning. AAAI Press, Menlo Park (2009)
4. Brié, A.H., Morignot, P.: Genetic Planning Using Variable Length Chromosomes. In: Proc. ICAPS (2005)
5. Fikes, R., Nilsson, N.: STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. Artificial Intelligence 1, 27–120 (1971)
6. Fox, M., Long, D.: PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. JAIR 20, 61–124 (2003)

7. Gerevini, A., Saetti, A., Serina, I.: On Managing Temporal Information for Handling Durative Actions in LPG. In: Cappelli, A., Turini, F. (eds.) AI*IA 2003. LNCS, vol. 2829, pp. 91–104. Springer, Heidelberg (2003)
8. Gerevini, A., Saetti, A., Serina, I.: Planning through Stochastic Local Search and Temporal Action Graphs in LPG. JAIR 20, 239–290 (2003)
9. Haslum, P., Geffner, H.: Admissible Heuristics for Optimal Planning. In: Proc. AIPS 2000, pp. 70–82 (2000)
10. Helmert, M.: The Fast Downward Planning System. JAIR 26(1), 191–246 (2006)
11. Hoffmann, J., Nebel, B.: The FF Planning System: Fast Plan Generation Through Heuristic Search. JAIR 14, 253–302 (2001)
12. Koza, J.R.: Genetic Programming: on the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)
13. McDermott, D.: PDDL – The Planning Domain Definition language (1998), http://ftp.cs.yale.edu/pub/mcdermott
14. 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)
15. Richter, S., Helmert, M., Westphal, M.: Landmarks Revisited. In: Proc. AAAI 2008, pp. 975–982. AAAI Press, Menlo Park (2008)
16. 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)
17. Schoenauer, M., Savéant, P., Vidal, V.: Divide-and-Evolve: a Sequential Hybridization Strategy using Evolutionary Algorithms. In: Michalewicz, Z., Siarry, P. (eds.) Advances in Metaheuristics for Hard Optimization, pp. 179–198. Springer, Heidelberg (2007)
18. Spector, L.: Genetic Programming and AI Planning Systems. In: Proc. AAAI 1994, pp. 1329–1334. AAAI/MIT Press (1994)
19. Vidal, V.: A Lookahead Strategy for Heuristic Search Planning. In: 14^{th} International Conference on Automated Planning & Scheduling - ICAPS, pp. 150–160 (2004)
20. Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. In: Proc. AAAI, pp. 570–577 (2004)
21. Vidal, V., Geffner, H.: Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming. Artificial Intelligence 170(3), 298–335 (2006)
22. Westerberg, C.H., Levine, J.: "GenPlan": Combining Genetic Programming and Planning. In: Garagnani, M. (ed.) 19^{th} Workshop PLANSIG 2000, The Open University (2000)
23. Westerberg, C.H., Levine, J.: Investigations of Different Seeding Strategies in a Genetic Planner. In: Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H. (eds.) EvoIASP 2001, EvoWorkshops 2001, EvoFlight 2001, EvoSTIM 2001, EvoCOP 2001, and EvoLearn 2001. LNCS, vol. 2037, pp. 505–514. Springer, Heidelberg (2001)
24. Yuan, B., Gallagher, M.: Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 172–181. Springer, Heidelberg (2004)
25. Yuan, B., Gallagher, M.: Combining Meta-EAs and Racing for Difficult EA Parameter Tuning Tasks. In: Parameter Setting in Evolutionary Algorithms, pp. 121–142. Springer, Heidelberg (2007)

A Real-Integer-Discrete-Coded Differential Evolution Algorithm: A Preliminary Study

Dilip Datta[1,2] and José Rui Figueira[2,3]

[1] Department of Mechanical Engineering, National Institute of Technology, Silchar - 788 010, India
datta_dilip@rediffmail.com
[2] CEG-IST, Center for Management Studies, Instituto Superior Técnico, TagusPark, Av. Cavaco Silva, 2780–990 Porto Salvo, Portugal
figueira@ist.utl.pt
[3] Associate Researcher at LAMSADE, Université Paris-Dauphine, Place De Lattre de Tassigny, 75 775 Paris Cedex 16, France

Abstract. The successful application of differential evolution (DE) algorithms to various real-valued problems encourages to develop some integer-coded versions of DE for working directly with integer and discrete variables of a problem. However, in most of those works, actually a real-valued solution is just converted into a desired integer-valued solution by applying some decoding mechanisms. Only a limited number of works are found, in which attempts are made for developing an actual integer-coded DE. In this article, a novel version of DE is proposed which can work directly with real, integer and discrete variables of a problem without any conversion. Applying to two non-linear real-integer-discrete-valued engineering design problems, the proposed DE is found successful in obtaining the known best solutions of the problems.

1 Introduction

The well-known genetic algorithm (GA) was proposed by Holland in 1975 [11] for functional optimization, which was made popular for engineering optimization by Goldberg in 1989 [8]. The classical GA is binary-coded, in which a real-valued design variable is to be encoded into 0-1 form, and then to be decoded back to its original form after the GA operations. In contrast, the differential evolution (DE) algorithm was proposed by Storn and Price [19] in 1995, which works directly with real-valued variables of a problem. However, after its successful application to a wide range of problems, it is now realized that DE is lacking to work with integer and discrete variables. To do so, such variables are to be encoded to real-form and then to be decoded back to their original form after the optimization process, which are analogous to encoding and decoding jobs of the classical GA for working with real variables. Therefore, researchers are now trying to develop some integer-coded versions DE, which would be able to work directly with integer and discrete variables, a similar job already done in GA for working directly with real variables. However, it is observed that, instead of an integer-coded DE, most of those works actually concentrate on developing some efficient

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 35–46, 2010.

encoding and decoding mechanisms for dealing with integer-valued variables in the real-coded DE. Only a limited number of works are found [9], in which attempts are made for developing an actual integer-coded DE.

In this article, a new binary-coded DE is proposed for directly dealing with integer and discrete variables of a problem without any conversion. It is then combined with a real-coded DE for forming a real-integer-discrete-coded DE, which can work with any type of variables. In order to improve the performance of the proposed DE, some changes are also made to the acceptance operator of the traditional DE. Moreover, a new diversity generating mechanism is also proposed for taking away the search from any local optimum. Applying to two non-linear real-integer-discrete-valued mechanical design problems, the proposed DE is found successful in obtaining the known best solutions of the problems.

2 Related Works

Pampará et al. [17] apply a trigonometric function to map a real-valued solution of DE into a 0-1 valued solution, which is already tested by Kanlikilicer et al. [12] as insufficient to work with many problems. Engelbrecht and Pampará [7] propose two approaches for solving 0-1 variable problems by real-coded DE, where the first approach uses a sigmoid function to convert a real-valued solution into a 0-1 valued solution, while the second approach maps a normalized real-valued solution into a 0-1 valued solution with 50% probability for each variable to become either 0 or 1. In the integer-coded versions of DE, proposed by Balamurugan and Subramanian [1], and Lampinen and Zelinka [13,14], a real value of DE is simply rounded to the nearest integer value.

Although the above versions of DE are called binary/integer-coded DE, these are actually real-coded DE only, whose solutions are just converted into binary/integer-valued solutions, which are similar with the work of Datta and Figueira [4], in which a decoding mechanism is used to convert a real-valued solution into a desired integer-valued solution. An attempt for developing an actual binary-coded DE is found in the work of Gong and Tuson [9], in which only 0-1 parent variables are considered for generating 0-1 child variables. However, they could not escape from the affect of the real-valued perturbance factor of DE, and hence, the value of a child variable equals either 0 or the real-valued perturbance factor, which is then randomly transformed to either 0 or 1.

3 The Real-Coded Differential Evolution Algorithm

Storn and Price [19,20] proposed a number of variants of DE, which are denoted as $DE/x/y/z$, where x denotes the mutated vector (also called the base vector), y is the number of difference vectors, and z is the crossover scheme. In this work, only the variant "DE/rand/1/bin" is considered, the meaning of which is that a random vector and one difference vectors will be applied for generating a mutant, and a trial will be obtained through a binary crossover scheme. In this DE, a population of N vectors (solutions), each of dimension D, is evolved through

the repeated actions of three major operators, namely mutation, crossover, and acceptance operators. In the mutation operation, four distinct vectors are considered at a time, out of which one is called the target vector, one is the base vector, and the other two are random vectors. Then, a mutant $\bar{x}_j$ against the variable x_j of the target vector is generated as follows:

$$\bar{x}_j^{(i,t)} = x_j^{(\alpha,t)} + F\left(x_j^{(\beta,t)} - x_j^{(\gamma,t)}\right);\ i = 1,\ldots,N;\ j = 1,\ldots,D\ ; \tag{1}$$

where t is the generation counter, and i, α, β and γ denote the target vector, base vector, and two random vectors, respectively. F is a scaling factor in (0,1), which controls the amount of perturbance to α by the difference of β and γ. In the crossover phase, a trial $\bar{\bar{x}}_j$ against x_j of the target vector is obtained as:

$$\bar{\bar{x}}_j^{(i,t)} = \begin{cases} \bar{x}_j^{(i,t)} & \text{if } r_j^{(t)} \leqslant p_c\,, \text{ or } j = \text{a random integer in } \{1,\ldots,D\}\ ; \\ x_j^{(i,t)} & \text{otherwise}\,; \end{cases} \tag{2}$$

where $r_j^{(t)}$ is a random number in (0,1), and p_c is the predefined crossover probability. Finally, for a minimization problem, the target vector is updated through the following elite preserving based acceptance operator:

$$\boldsymbol{x}^{(i,t+1)} = \begin{cases} \bar{\bar{\boldsymbol{x}}}^{(i,t)}\,, & \text{if } f\left(\bar{\bar{\boldsymbol{x}}}^{(i,t)}\right) < f\left(\boldsymbol{x}^{(i,t)}\right); \\ \boldsymbol{x}^{(i,t)}\,, & \text{otherwise}\,; \end{cases} \tag{3}$$

where $f\left(\bar{\bar{\boldsymbol{x}}}^{(i,t)}\right)$ and $f\left(\boldsymbol{x}^{(i,t)}\right)$ are the objective values for the trial vector $\bar{\bar{\boldsymbol{x}}}^{(i,t)}$ and the current target vector $\boldsymbol{x}^{(i,t)}$, respectively.

4 The Proposed Real-Integer-Discrete-Coded Differential Evolution Algorithm

The only difference between a real-coded DE and an integer-coded DE is the type of the generated mutant, the expression of which is given by Eq. (1). In most of the integer-coded versions of DE, different random schemes are implemented for converting the real value of the mutant into an integer value. However, in the actual sense of an "integer-coded algorithm", only some integer values should be employed to generate a new integer value without any conversion, as done in the well-known binary-coded genetic algorithm [8]. Such a binary-coded DE is proposed here, in which, based on some logic, only 0 or 1 is assigned to the mutant using some other binary values. Although the DE deals with binary variables only, it can be applied to work directly with integer (other than 0-1) and discrete variables also. This binary-coded DE is combined with the real-coded DE, addressed in Sect. 3, for forming a real-integer-discrete-coded DE, which can work with any type of variables (real, integer and/or discrete). In order to improve the performance of this real-integer-discrete-coded DE by preserving all efficient vectors of a generation, some changes are also proposed to the original acceptance operator of DE. Moreover, a new mechanism is also proposed for generating diversity among the vectors.

4.1 Vector Representation and Initialization

The total dimension of a vector is determined from the number and types of variables. Each real variable is represented by a single dimension. The number of dimensions required for an integer variable is determined from the upper limit of the variable, which is obtained as the maximum number of binary bits required to represent its upper limit. It is to be noted that a discrete-valued variable is also dealt with as an integer variable, an integer value of which represents the index of its actual discrete value, so that the lower limit of such a variable is 1 and the upper limit is the number of its allowable discrete values.

Once the total dimensions of a vector is determined, the population is initialized randomly before starting the DE operations. A real dimension is initialized by a random real value in the given range of the real variable which is represented by that dimension, while all the binary dimensions are initialized randomly by 0 or 1 with 50% probability.

4.2 Mutation Operator

In the mutation operation, Eq. (1) is employed for generating a real-valued mutant against a real dimension of a vector. The procedure for generating a binary-valued mutant against a binary dimension, which is the main contribution of the present work, is explained below in detail:

As given by Eq. (1), the mutation operator generates a mutant by employing three variables, one from each of the base vector and two random vectors ($x_j^{(\alpha,t)}$, $x_j^{(\beta,t)}$, $x_j^{(\gamma,t)}$). Since all these three variables have binary values (0 or 1) only, there are 8 distinct combinations of their values, which are given through columns (1)–(3) in Table 1. The corresponding expressions for the mutant $\bar{x}_j^{(i,t)}$ are given in column (4), in which rows (1)–(4) can be said as the "*don't caring rows*", meaning of which is that, whatever may be the value of F, $\bar{x}_j^{(i,t)} = x_j^{(\alpha,t)}$ as long as $x_j^{(\beta,t)} = x_j^{(\gamma,t)}$. Now, in order to have a balanced situation, the mutant $\bar{x}_j^{(i,t)}$ is expected to be 0 or 1 with 50% probability. Since two 0's and two 1's are already obtained in rows (1)–(4), the other four rows (5)–(8), which are functions of F, are also expected to give two 0's and two 1's. With the presumption that F lies in the range of (0,1), it is observed that the values of $\bar{x}_j^{(i,t)}$ in rows (5) and (8) are in the range of (0,1), and its values in rows (6) and (7) are outside the range of (0,1). Since the value of $\bar{x}_j^{(i,t)}$ should be either 0 or 1, a simple assumption is made at this juncture that "the binary value of $\bar{x}_j^{(i,t)}$ is 1 if its real value is in the range of (0,1), and 0 otherwise". The corresponding binary values of $\bar{x}_j^{(i,t)}$ against all the eight combinations of $x_j^{(\alpha,t)}$, $x_j^{(\beta,t)}$ and $x_j^{(\gamma,t)}$ are given in column (6), which contain four 0's and four 1's, thus fulfill the expected 50% probability for obtaining 0's or 1's. However, this simple transformation may sometime suffer from the following two major drawbacks:

1. Due to the fixed transformation, the vectors may lose diversity among them, resulting the population to converge at a sub-optimal point, and

Table 1. Different cases under the mutation operation to binary variables

Column →	$x_j^{(\alpha,t)}$	$x_j^{(\beta,t)}$	$x_j^{(\gamma,t)}$	$\bar{x}_j^{(i,t)}$				
Row ↓	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
(1)	0	0	0	0	0	0	0	if $r_1^{(t)} \geqslant p_m$
							1	if $r_1^{(t)} < p_m$
(2)	0	1	1	0	0	0	0	if $r_2^{(t)} \geqslant p_m$
							1	if $r_2^{(t)} < p_m$
(3)	1	0	0	1	1	1	1	if $r_3^{(t)} \geqslant p_m$
							0	if $r_3^{(t)} < p_m$
(4)	1	1	1	1	1	1	1	if $r_4^{(t)} \geqslant p_m$
							0	if $r_4^{(t)} < p_m$
(5)	0	1	0	F	(0,1)	1	1	if $r_5^{(t)} \geqslant p_m$
							0	if $r_5^{(t)} < p_m$
(6)	0	0	1	$-F$	< 0	0	0	if $r_6^{(t)} \geqslant p_m$
							1	if $r_6^{(t)} < p_m$
(7)	1	1	0	$1+F$	> 1	0	0	if $r_7^{(t)} \geqslant p_m$
							1	if $r_7^{(t)} < p_m$
(8)	1	0	1	$1-F$	(0,1)	1	1	if $r_8^{(t)} \geqslant p_m$
							0	if $r_8^{(t)} < p_m$

2. The proposed assumption may not always be true, i.e., the binary values of $\bar{x}_j^{(i,t)}$ along some dimensions of a vector may be 0 if its real values lie in the range of (0,1), and 1 otherwise.

Therefore, in order to avoid such situations, some randomness is imposed to $\bar{x}_j^{(i,t)}$ in obtaining its binary values. Under this consideration, a binary value of $\bar{x}_j^{(i,t)}$ in column (6) of Table1 is true if its random probability $r_k^{(t)}$ ($k \in \{1,\ldots,8\}$) is greater than or equal to a predefined probability p_m (named as the *mutation probability*), otherwise $\bar{x}_j^{(i,t)}$ will take the alternative binary value. The final binary values of the mutant $\bar{x}_j^{(i,t)}$ are given in column (7) along with the corresponding randomness conditions in column (8).

It is to be noted that, in this binary-coded mutation operation, the perturbance factor F is not required explicitly, but it is only based on the presumption that F lies in the range of (0,1). However, the place of F has been taken by the new user-defined mutation probability p_m. Extensive empirical studies have shown that better results are obtained for $p_m \leqslant 15\%$.

4.3 Crossover Operator

A crossover operator is applied for forming a trial vector against a target vector, in which either a dimension of the target vector or its mutant is adopted, with some predefined crossover probability, as the corresponding dimension of the

trial vector. The entire crossover operation is independent of the data-type of the dimensions (real or integer) of the target vector or their mutants. Therefore, the crossover operator, given by Eq. (2), is applied here without any modification.

4.4 Acceptance Operator

The role of the traditional acceptance operator, given by Eq. (3), is just to decide whether or not a target vector would be replaced by its trial vector. Thus, like the crossover operator, the acceptance operator is also independent of the data-type of the dimensions (real or integer) of a vector. Therefore, this acceptance operator also can be applied here without any modification.

However, Datta and Figueira [4] observe two drawbacks with this acceptance operator, given by Eq. (3). Firstly, a target vector, which is superior to many other vectors of the population, may get lost due to the generation of a better trial vector against it. Secondly, a trial vector, which is better than many vectors of the population, may be missed due to its inferior quality than that of the target vector against which it is generated. In order to overcome these drawbacks by preserving all the efficient vectors, this acceptance operator is replaced here by the "elite-preserving mechanism" proposed by Deb et al. [6]. This mechanism is applied to the proposed DE as follows: without taking any decision based on a pair of a target vector and its trial vector only, all the target and trial vectors are first combined, and then the combined vectors are sorted according to their qualities (objective values). Finally, the first 50% of the best vectors of the combined population are extracted as the population for the next generation.

4.5 Diversity Generating Mechanism

In spite of introducing some randomness in Table1 for preserving diversity among binary-valued vectors, it is observed in experimental trials that sometime the search still gets stuck at some local optimum. This might have happened due to the fact that the proposed randomness is not sufficient to preserve diversity among the vectors. Therefore, a new mechanism is proposed here for generating additional diversity among the vectors. In this mechanism, if all the vectors converge to a single point during the execution, the value of a binary variable is altered from 0 to 1 or from 1 to 0 with a small probability (say 5%).

5 Numerical Experiments and Discussions

The proposed DE is implemented in `C` programming language. In order to demonstrate its effectiveness, two numerical examples, one is the design of a gear train and other is the design of a coil spring, are considered here, which are non-linear engineering design problems involving real, integer and discrete variables (for detail of the problems, Sandgren [18] may be referred). These problems are similar with many other mechanical design problems. As the problems are clearly defined and fairly easy to understand, they form a suitable basis for comparing

alternative optimization methods. In the following two subsections, the applications of the proposed DE to the considered two examples are presented, where it is found successful in obtaining the known best solutions of the problems.

5.1 The Integer-Valued Gear Train Design Problem

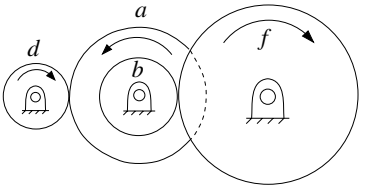
The gear train problem involves the optimization of the gear ratio of a compound gear train. As shown below, the gear train arrangement consists of two pairs of gearwheels, d-a and b-f, whose overall gear ratio can be expressed as:

$$\text{Gear ratio} = \frac{\text{Angular velocity of the output shaft } (w_0)}{\text{Angular velocity of the input shaft } (w_i)} = \frac{z_d z_b}{z_a z_f} \,,$$

where z_d, z_b, z_a and z_f denote the numbers of teeth on the gears.

It is required to find the numbers of teeth on the wheels so that a gear ratio, as close as possible to 1/6.931, can be obtained. The only constraint in the problem is that the number of teeth in each gear should be in range of [12,60]. Accordingly, the optimization problem can be expressed as below:

$$\begin{aligned} &\text{Determine} && \boldsymbol{x} = (z_d, z_b, z_a, z_f) \\ &\text{to minimize} && \varphi(\boldsymbol{x}) = \left(\tfrac{1}{6.931} - \tfrac{z_d z_b}{z_a z_f}\right)^2 \\ &\text{subject to:} && g(\boldsymbol{x}) \equiv 12 \leqslant z_d, z_b, z_a, z_f \leqslant 60 \end{aligned}$$



For solving the problem by using the proposed DE, the mutation probability for binary variables and the crossover probability are initially assigned the values of 0.15 and 0.80, respectively. Instead of working with a single fixed value, the value of each of these parameters is made generation-wise self-adaptive in a range from zero to its maximum assigned value. Such self-adaptation of a parameter reduces the dependency of an algorithm on that parameter, if any [3]. A value is made self-adaptive through the *polynomial mutation of a real number* [5], in which, for a random number r in (0,1) and a given polynomial distribution index $\eta > 0$, a real number x is evolved in the range of $(x^{(l)}, x^{(u)})$ as given below:

$$x \leftarrow x + \left(x^{(u)} - x^{(l)}\right) d_q \tag{4}$$

$$\text{where } d_q = \begin{cases} \left[(2r + (1-2r)\left(1 - \frac{x - x^{(l)}}{x^{(u)} - x^{(l)}}\right)^{\eta+1}\right]^{\frac{1}{\eta+1}} - 1\,, & \text{if } r < 0.5 \\ 1 - \left[2(1-r) + 2(r-0.5)\left(1 - \frac{x^{(u)} - x}{x^{(u)} - x^{(l)}}\right)^{\eta+1}\right]^{\frac{1}{\eta+1}}, & \text{otherwise}\,. \end{cases}$$

Moreover, the *penalty-parameter-less constraint handling approach*, proposed by Deb [5], is applied here for working with infeasible solutions. In this approach, all the infeasible solutions are first made inferior to any feasible solution through a fitness function, and then all the solutions are treated as feasible solutions only.

Then the problem is solved 30 times by randomly fixing the size of the DE population in a run in the range of [60,100]. A comparison of the results obtained by different optimization techniques is presented in Table 2, where it is

Table 2. Optimal solutions for the gear train design problem

Variables/ Functions	Sandgren [18]	Loh and Papa. [16]	Zhang and Wang [22]	Lin et al. [15]	Guo et al. [10]	Proposed DE
z_d	18	19	30	19	16	16
z_b	22	16	15	16	19	19
z_a	45	42	52	43	43	43
z_f	60	50	60	49	49	49
$\varphi(\boldsymbol{x})$	5.7×10^{-6}	0.23×10^{-6}	2.4×10^{-9}	2.7×10^{-12}	2.7×10^{-12}	2.7×10^{-12}
Gear ratio	0.146666	0.144762	0.144231	0.144281	0.144281	0.144281
Error (%)	1.65	0.334	0.033	0.0011	0.0011	0.0011

observed that the proposed DE also produces the known best solution of the problem. It is to be noted that, for solving the problem, Sandgren [18] applied the branch-and-bound method through sequential quadratic programming, Loh and Papalambros [16] applied a sequential linearization algorithm, Zhang and Wang [22] applied a simulated annealing algorithm, Lin et al. [15] applied a genetic algorithm, and Guo et al. [10] applied a hybrid swarm intelligence approach.

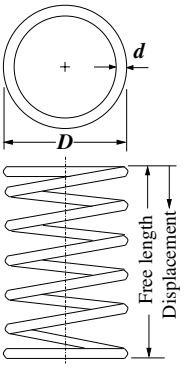
It is also to be noted that the proposed DE provides the same objective value (the best one) in each of its 30 runs. Moreover, against that best objective value, four distinct sets of variable values are obtained in different runs, or even in different solutions of a single run, depicting that the problem has multiple optimum solutions. The obtained variable sets are $(z_d, z_b, z_a, z_f) = (16,19,43,49)$, (19,16,43,49), (16,19,49,43) and (19,16,49,43).

In the case of computational cost, the numbers of function evaluation are found varying in the range of [1800,5900] over the 30 runs of the proposed DE, with the mean, median and standard deviation of 3994.67, 4180 and 36.72, respectively. When executed in the Fedora 8 Linux environment in a HCL Desktop with 2.67 GHz processor, 2.0 GB RAM and 160 GB HDD, the maximum computational time is found to be less than one second. It is to be noted that no computational issue is discussed in other publications given in Table 2.

5.2 The Real-Integer-Discrete-Valued Coil Spring Design Problem

This problem involves the design of a helical compression spring, where an axial and constant load is to be applied. It is required to minimize the wire volume of the spring (minimum weight), subject to some given constraints. The design variables are the integer-valued number of spring coils (N), the real-valued outside diameter of the spring (D), and the discrete-valued spring wire diameter (d). Accordingly, the problem can be formulated as below:

$$
\begin{aligned}
&\text{Determine} && \boldsymbol{x} = (N, D, d)\\
&\text{to minimize} && \varphi(\boldsymbol{x}) = \tfrac{\pi^2 D d^3 (N+2)}{4}\\
&\text{subject to:} && g_1(\boldsymbol{x}) \equiv \tfrac{8 C_f F_{\max} D}{\pi d^3} - S \leqslant 0\\
& && g_2(\boldsymbol{x}) \equiv l_f - l_{\max} \leqslant 0\\
& && g_3(\boldsymbol{x}) \equiv d_{\min} - d \leqslant 0\\
& && g_4(\boldsymbol{x}) \equiv D - D_{\max} \leqslant 0\\
& && g_5(\boldsymbol{x}) \equiv 3.0 - \tfrac{D}{d} \leqslant 0\\
& && g_6(\boldsymbol{x}) \equiv \sigma_p - \sigma_{pm} \leqslant 0\\
& && g_7(\boldsymbol{x}) \equiv \sigma_p + \tfrac{F_{\max} - F_p}{K} + 1.05(N+2)d - l_f \leqslant 0\\
& && g_8(\boldsymbol{x}) \equiv \sigma_w - \tfrac{F_{\max} - F_p}{K} \leqslant 0\\
&\text{where} && C_f = \tfrac{4(D/d)-1}{4(D/d)-4} + \tfrac{0.615d}{D}\\
& && K = \tfrac{Gd^4}{8ND^3}\\
& && \sigma_p = \tfrac{F_p}{K}\\
& && l_f = \tfrac{F_{\max}}{K} + 1.05(N+2)d
\end{aligned}
$$



The supplied numerical data for the problem are given below along with the allowable discrete values of the spring wire diameter (d) in Table 3.

$F_{\max}$	= Maximum working load	= 1000.0 lb
S	= Maximum allowable shear stress	= 189000.0 psi
$l_{\max}$	= Maximum free length	= 14.0 inch
$d_{\min}$	= Minimum wire diameter	= 0.2 inch
$D_{\max}$	= Maximum outside spring diameter	= 3.0 inch
F_p	= Preload compression force	= 300.0 lb
σ_{pm}	= Maximum allowable deflection under preload	= 6.0 inch
σ_w	= Deflection from preload position to maximum load position	= 1.25 inch
G	= Shear modulus of the material	= 11.5×10^6 psi.

Table 3. Allowable wire diameters (discrete values of d)

Allowable wire diameters (inch)						
0.0090	0.0095	0.0104	0.0118	0.0128	0.0132	0.0140
0.0150	0.0162	0.0173	0.0180	0.0200	0.0230	0.0250
0.0280	0.0320	0.0350	0.0410	0.0470	0.0540	0.0630
0.0720	0.0800	0.0920	0.1050	0.1200	0.1350	0.1480
0.1620	0.1770	0.1920	0.2070	0.2250	0.2440	0.2630
0.2830	0.3070	0.3310	0.3620	0.3940	0.4375	0.5000

For solving the problem by using the proposed DE, the perturbance factor for real variables, mutation probability for binary variables and crossover probability are assigned the initial values of 0.70, 0.15 and 0.80, respectively. As in the case of the problem of Sect. 5.1, the value of each of these parameters is also made

generation-wise self-adaptive in a range from zero to its maximum assigned value. Moreover, combining the constraints $g_3(\boldsymbol{x})$–$g_5(\boldsymbol{x})$, the limits of N and D are fixed as [1,70] and [0.6,3.0] inch, respectively. Since d is a discrete variable with 42 allowable values given in Table 3, its integer limits are set automatically as [1,42]. On the other hand, the *penalty-parameter-less constraint handling approach* [5] is applied, in this problem also, for working with infeasible solutions.

Then the problem is solved 30 times by randomly fixing the size of the DE population in a run in the range of [40,60]. In all the performed 30 runs, fixed values of 9 and 0.283 inch are obtained for N and d, respectively. The only variation is obtained in the values of D, which vary in the range of (1.223044,1.223147) inch. The corresponding objective values vary in the range of (2.658565,2.658790) inch^3. A comparison of the results obtained by different optimization techniques is presented in Table 4. It is to be noted that, for solving this problem, Sandgren [18]

Table 4. Optimal solutions for the mechanical coil spring design problem

Variables/ Functions	Sandgren [18]	Chen and Tsao [2]	Wu and Chow [21]	Lampinen-Zelinka [14]	Guo et al. [10]	Proposed DE
N	10	9	9	9	9	9
D $[in]$	1.180701	1.2287	1.227411	1.223041	1.223	1.223044
d $[in]$	0.283	0.283	0.283	0.283	0.283	0.283
$-g_1(\boldsymbol{x})$	543.09	415.969	550.993	1008.8114	1008.81	1008.6195
$-g_2(\boldsymbol{x})$	8.8187	8.9207	8.9264	8.94564	8.946	8.945628
$-g_3(\boldsymbol{x})$	0.08298	0.0830	0.0830	0.08300	0.083	0.083000
$-g_4(\boldsymbol{x})$	1.8193	1.7713	1.7726	1.77696	1.77696	1.776957
$-g_5(\boldsymbol{x})$	1.1723	1.3417	1.3371	1.32170	1.32170	1.321706
$-g_6(\boldsymbol{x})$	5.4643	5.4568	5.4585	5.46429	5.4643	5.464283
$-g_7(\boldsymbol{x})$	0.0	0.0	0.0	0.0	0.0	0.0
$-g_8(\boldsymbol{x})$	0.0	0.0174	0.0134	0.0	0.0	0.0
$\varphi(\boldsymbol{x})$ $[in^3]$	2.7995	2.6709	2.6681	2.65856	2.659	2.658565

applied the branch-and-bound method through sequential quadratic programming, Chen and Tsao [2] and Wu and Chow [21] applied two different versions of genetic algorithms, and Guo et al. [10] applied a hybrid swarm intelligence approach. On the other hand, Lampinen and Zelinka [14] solved the problem by using a real-integer-discrete DE, in which an integer value is obtained just by truncating a real value. It is observed in Table 4 that the solution produced by the proposed DE is equally as good as the best solution reported in the literature. Although a little difference is observed between the obtained value of D and that of the best solution reported by Lampinen and Zelinka [14], such a small difference in the sixth decimal place of an inch is insignificant from the practical point of view (the difference in the value of $\varphi(\boldsymbol{x})$ is caused because of the difference in the value of D).

In the case of computational cost, the numbers of function evaluation are found varying in the range of [539960,3711560] over the 30 runs of the proposed DE, with the mean, median and standard deviation of 2270994, 2489380 and

931.9, respectively. Such a huge number of function evaluation is required because of the highly non-linear constraints of the problem. When executed in the Fedora 8 Linux environment in a HCL Desktop with 2.67 GHz processor, 2.0 GB RAM and 160 GB HDD, the computational time is found varying in the range of [2,10] seconds. It is to be noted that no computational issue is discussed in other publications given in Table 4.

6 Conclusions

The success of differential evolution (DE) algorithms to different real-valued problem domains has encouraged to develop some integer-coded versions of DE for dealing with integer/discrete-valued problems. However, most of those versions actually convert a real-valued solution of a real-coded DE into an integer-valued solution by applying some forms of decoding mechanisms. Only a limited number of works are found attempting to develop an actual integer-coded DE. A new binary-coded DE is proposed here, which is then combined with a real-coded DE for directly working with real, integer and discrete variables of a problem without any conversion. In order to improve the performance of the proposed DE, some changes are also proposed to the acceptance operator of the traditional DE. Moreover, the proposed DE is found suffering from diversity among its solutions, which has been overcome by incorporating a diversity generating mechanism. The effectiveness of the DE is demonstrated through its application to two non-linear real-integer-discrete-valued engineering design problems, where it is found successful in obtaining the known best solutions of the problems. Further research will be carried out to exploit the potentialities of the proposed DE by applying it to other design problems. Moreover, attempts will also be made for further improvement of its performance.

Acknowledgement. The work is carried out under the first author's post-doctoral research grant (SFRH/BPD/34482/2006) offered by the *Fundação para a Ciência e a Tecnologia* (FCT), *Ministério da Ciência, Tecnologia e Ensino Superior*, Portugal. The second author acknowledges COST ACTION 0602 grant on *Algorithmic Decision Theory*. Both the authors are also thankful to the five unknown reviewers for their valuable suggestions in improving the article.

References

1. Balamurugan, R., Subramanian, S.: Hybrid integer coded differential evolution – dynamic programming approach for economic load dispatch with multiple fuel options. Energy Conversion & Management 49, 608–614 (2008)
2. Chen, J.L., Tsao, Y.C.: Optimal design of machine elements using genetic algorithms. J. Chinese Society of Mechanical Engineers 14(2), 193–199 (1993)
3. Datta, D.: Multi-Objective Evolutionary Algorithms for Resource Allocation Problems. PhD thesis, Department of Mechanical Engineering, Indian Institute of Technology Kanpur (IIT-Kanpur), India (2007)

4. Datta, D., Figueira, J.R.: Graph partitioning by multi-objective real-valued metaheuristics: A comparative study (Under submission)
5. Deb, K.: An efficient constraint handling method for genetic algorithms. Computer Methods in Applied Mechanics and Engineering 186, 311–338 (2000)
6. Deb, K., Agarwal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
7. Engelbrecht, A.P., Pampará, G.: Binary differential evolution strategies. In: IEEE Congress on Evolutionary Computation (CEC 2007), pp. 1942–1947 (2007)
8. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading (1989)
9. Gong, T., Tuson, A.: Differential evolution for binary encoding. Advance Soft Computing 39, 251–262 (2007)
10. Guo, C., Hu, J., Ye, B., Cao, Y.: Swarm intelligence for mixed-variable design optimization. J. Zhejiang Univ. SCI. 5(7), 851–860 (2004)
11. Holland, J.H.: Adaptation in natural and artificial systems. The University of Michigan Press, Ann Arbor (1975)
12. Kanlikilicer, A.E., Keles, A., Uyar, A.S.: Experimental analysis of binary differential evolution in dynamic enviornments. In: Genetic and Evolutionary Computation Conference (GECCO-2007), London, UK, pp. 2509–2514 (2007)
13. Lampinen, J., Zelinka, I.: Mixed-integer-discrete-continuous optimization by differential evolution, Part 1: The optimization method. In: Ošmera, P. (ed.) Fifth International Mendel Conference on Soft Computing (MENDEL 1999), Brno, Czech Republic, pp. 71–76 (1999)
14. Lampinen, J., Zelinka, I.: Mixed-integer-discrete-continuous optimization by differential evolution, Part 2: A practical example. In: Ošmera, P. (ed.) Fifth International Mendel Conference on Soft Computing (MENDEL 1999), Brno, Czech Republic, pp. 77–81 (1999)
15. Lin, S.S., Zhang, C., Wang, H.P.: On mixed-discrete nonlinear optimization problems: A comparative study. Engineering Optimization 23, 287–300 (1995)
16. Loh, H.T., Papalambros, P.Y.: A sequential linearization approach for solving mixed-discrete nonlinear design optimization problems. ASME J. of Mechanical Design 113, 325–334 (1991)
17. Pampará, G., Engelbrecht, A.P., Franken, N.: Binary differential evolution. In: IEEE Congress on Evolutionary Computation (CEC 2006), Vancouver, BC, Canada, pp. 1873–1879 (2006)
18. Sandgren, E.: Nonlinear integer and discrete programming in mechanical design optimization. ASME Transactions on Mechanical Design 112(2), 223–229 (1990)
19. Storn, R., Price, K.: Differential evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, Berkeley, CA (1995)
20. Storn, R., Price, K.: Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–354 (1997)
21. Wu, S.-J., Chow, P.-T.: Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. Engineering Optimization 24(2), 137–159 (1995)
22. Zhang, C., Wang, H.P.: Mixed-discrete nonlinear optimization with simulated annealing. Engineering Optimization 21, 277–291 (1993)

Fitness Distance Correlation and Search Space Analysis for Permutation Based Problems

Botond Draskoczy

University of Stuttgart, FMI,
Universitätstr. 38, 70569 Stuttgart, Germany
draskoczy@fmi.uni-stuttgart.de

Abstract. The fitness distance correlation (FDC) as a measure for problem difficulty was first introduced by Forrest and Jones. It was applied to many binary coded problems. This method is now applied to permutation based problems. As demanded by Schiavinotto and Stützle, the distance in a search space is calculated by regarding the steps of the (neighborhood) operator. In this paper the five most common operators for permutations are analyzed on symmetric and asymmetric TSP instances. In addition a local quality measure, the point quality (PQ) is proposed as a supplement to the global FDC. With this local measure more characteristics and differences can be investigated. Some experimental results are illustrating these concepts.

1 Introduction

Permutation based problems like TSP are well analyzed and successfully optimized by evolutionary algorithms (EA) and other optimization heuristics. The common way is to find well adapted operators and heuristic specific parameters by extensive experiments. Another way is to predict good operators by analyzing positive characteristics of the operator specific search space. The fitness distance correlation [6] was successfully applied to this task [3]. By classification of Thierens the FDC is more an analysis tool than a predictive tool [14]. Usually it is not straightforward to calculate the correct operator specific distances between solutions, therefore approximations are used ([8] and [10]). But Schiavinotto and Stützle pointed out the importance of correct distance algorithms in [12], but didn't applied them. One goal of our study is to apply the FDC analysis to permutation based problems with precisely calculated distances for five common operators. An example for such an analysis can be found in [5], but there only symmetric TSP instances for two operators are analyzed. One problem of the FDC is its oversimplifying nature, one scalar value is sufficient to compare operators, but not for analyzing the search space.

After the preliminary definitions in section two, the point quality (PQ) is defined. As a local quality measure, the PQ helps to analyze and visualize a search space. Section three specifies the implemented distance algorithms. Data generation and the inspected test problems are described in section four. Results of the FDC analysis and examples of PQ diagrams are finally presented in section five.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 47–58, 2010.

2 Definitions and Operators

Important for analysis of search space (synonymously used for landscape or search graph) is the concept of distance. This pairwise distance depends on the used operator and defines a neighborhood for each point in the search space.

Definition 1. *A (neighborhood) operator* OP *is defined over a set S_n (in this paper the set of all permutations of length n) by*

$$\mathrm{OP} : S_n \to 2^{S_n}$$

The (k-th) neighborhood of a subset $s \subseteq S_n$ we denote by

$$\mathcal{N}^0_{\mathrm{OP}}(s) = s, \quad \mathcal{N}^k_{\mathrm{OP}}(s) = \mathcal{N}_{\mathrm{OP}}(\mathcal{N}^{k-1}_{\mathrm{OP}}(s)) \;\; \textit{and} \;\; \mathcal{N}_{\mathrm{OP}}(s) = \bigcup_{\pi \in s} \mathrm{OP}(\pi) = \mathcal{N}^1_{\mathrm{OP}}(s)$$

The operator specific size of the neighborhood is denoted as $|OP(\pi)|$ or $|OP|$ (if it has the same size for all $\pi \in S_n$).

Definition 2. *The distance $d_{\mathrm{OP}}(\pi, \varphi)$ between two points (solutions) $\pi, \varphi \in S_n$ in search space is*

$$d_{\mathrm{OP}}(\pi, \varphi) = \min\left\{k \mid \varphi \in \mathcal{N}^k_{\mathrm{OP}}(\{\pi\})\right\}$$

Definition 3. *The operator specific diameter of the search space is*

$$\Phi_{\mathrm{OP}} = \max\left\{d_{\mathrm{OP}}(\pi, \varphi) | \pi, \varphi \in S_n\right\}$$

2.1 Operators

For the formal definition of the operators for permutations (from now S_n is the set of all permutations of length n) we denote permutations in common notation. For example $\pi = (3, 4, 1, 2)$ is a permutation of length $n = 4$ and is identical to the permutation $\pi = (1, 3)(2, 4)$ in cycle notation. The composition is calculated from right to left, for example $(4, 3, 2, 1) \circ (4, 3, 1, 2) = (1, 2, 4, 3)$.

Definition 4 (Neighbor Swap (NbrSwap)). *For a permutation $\pi \in S_n$ the* NbrSwap *operator is defined as*

$$\mathrm{NbrSwap}(\pi) = \{(\pi_1, \pi_2, \ldots, \pi_{k-1}, \pi_{k+1}, \pi_k, \pi_{k+2}, \ldots, \pi_n) \mid 1 \leq k < n\}$$

This operator swaps two adjacent elements in a permutation.

Definition 5 (Element Swap (Swap)). *For a permutation $\pi \in S_n$ the* Swap *operator is defined as*

$$\mathrm{Swap}(\pi) = \{(\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_j, \pi_{i+1}, \ldots, \pi_{j-1}, \pi_i, \pi_{j+1}, \ldots, \pi_n) \mid 1 \leq i < j \leq n\}$$

This is one of the most used operators, it interchanges two elements in a permutation.

Definition 6 (Element Shift (Shift)). *For a permutation* $\pi \in S_n$ *the* Shift *operator is defined as*

$$\begin{aligned}\mathrm{Shift}(\pi) = &\{(\pi_1, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_{i+k}, \pi_i, \pi_{i+k+1}, \ldots, \pi_n) \mid k > 0\} \cup \\ &\{(\pi_1, \ldots, \pi_{i+k-1}, \pi_i, \pi_{i+k}, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_n) \mid k < 0\}\end{aligned}$$

This operator moves an element to another position in the permutation.

Definition 7 (Substring Reversal (SStrRev)). *For a permutation* $\pi \in S_n$ *the* SStrRev *operator is defined as*

$$\mathrm{SStrRev}(\pi) = \{(\pi_1, \ldots, \pi_{i-1}, \pi_j, \pi_{j-1}, \ldots, \pi_{i+1}, \pi_i, \pi_{j+1}, \ldots, \pi_n) \mid 1 \leq i < j \leq n\}$$

This operator reverses a substring of the permutation.

Definition 8 (Substring Shift (SStrShift)). *For a permutation* $\pi \in S_n$ *the* SStrShift *operator is defined as*

$$\begin{aligned}\mathrm{SStrShift}(\pi) = \{&(\pi_1, \ldots, \pi_{i-1}, \pi_{j+1}, \ldots, \pi_{j+k}, \pi_i, \ldots, \pi_j, \pi_{j+k+1}, \ldots, \pi_n) \\ &\mid 1 \leq i + k < j + k \leq n\}\end{aligned}$$

This operator moves a substring of the permutation to another position. For some features of these five operators see table 1.

2.2 Fitness Distance Correlation and Point Quality

The FDC measures the correlation of fitness and distance to the nearest global optimum opt. Originally [6] it was applied to bit string coded optimization problems and in many cases ([3], [15]) this measure successfully predicts the performance of a genetic algorithm. The correlation coefficient for a sample M of size m is calculated as

$$\mathrm{FDC} = \frac{C_{\mathrm{FD}}}{s_f s_d} \quad \text{where} \quad C_{\mathrm{FD}} = \frac{1}{m} \sum_{i \in M} (f_i - \bar{f})(d(i, \mathrm{opt}_i) - \bar{d})$$

is the covariance of f and d and s_f, s_d, $\bar{f}$ and $\bar{d}$ are the standard deviations and means of f and d.

The FDC is a global measure as it calculates only one scalar value for a search space defined by problem instance and neighborhood operator. For analyzing and understanding the characteristics of a search space too many informations are lost. A possibility to remedy this shortage can be a local quality measure. Such a measure will provide more opportunities to study characteristics of a search space. Using the idea of FDC, with the two attributes fitness and distance, the neighborhood of any element in the search space is divided into four subsets. This can be illustrated with a 2-dimensional coordinate system and its four quadrants (figure 1).

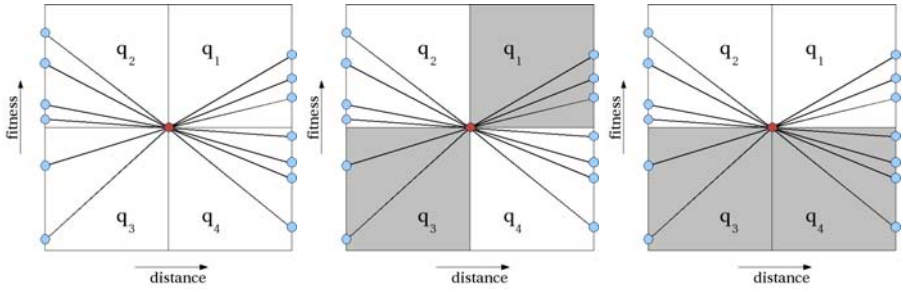


Fig. 1. All neighbors of a solution can be pictured as elements in quadrants $q_1, \ldots, q_4$ (left). In minimization problems (like TSP) neighbors in q_1 and q_3 are supporting optimization (middle), but elements in q_2 and q_4 are deceptive. As walking through search space of a minimization problem, heuristics are favoring elements in q_3 and q_4 (right).

Definition 9. *For each element $\pi \in S_n$ with known fitness value f_π and distance $d(\pi, \varphi)$ to a nearest optimal solution the point quality is defined as*

$$\mathrm{PQ}(\pi) = \begin{cases} \dfrac{\left|\left\{\pi' \mid (d(\pi',\varphi) \leq d(\pi,\varphi) \wedge f_{\pi'} \leq f_\pi) \vee (d(\pi',\varphi) \geq d(\pi,\varphi) \wedge f_{\pi'} \geq f_\pi)\right\}\right|}{|\mathrm{OP}(\pi)|} & \textit{if min. problem} \\ \dfrac{\left|\left\{\pi' \mid (d(\pi',\varphi) \leq d(\pi,\varphi) \wedge f_{\pi'} \geq f_\pi) \vee (d(\pi',\varphi) \geq d(\pi,\varphi) \wedge f_{\pi'} \leq f_\pi)\right\}\right|}{|\mathrm{OP}(\pi)|} & \textit{if max. problem} \end{cases}$$

An element with a high PQ value will support the optimizing process. If all elements has a perfect PQ, then the FDC is pefect, too.

3 Distance Algorithms

If a problem is binary coded, the Hamming-Distance is used for calculating the pairwise distance. It can be calculated in $\Theta(n)$. In genetic algorithms (GA) One-Bit-Flipping is used as (mutation) operator, therefore Hamming-Distance represents the correct distance in search space.

In permutation based optimization algorithms many different operators are used. The calculation of the distance needs to reflect the neighborhood of the operator used – otherwise the results are incorrect for most combinations of operator and approximation algorithms. This was shown in [12] by Schiavinotto and Stützle and they also listed correct distance algorithms for five operators. A distance algorithm for permutations (only) has to calculate the distance to identity $\imath$. Because for calculating the pairwise distance the property $d_{\mathrm{OP}}(\pi, \varphi) = d_{\mathrm{OP}}(\pi^{-1} \cdot \varphi, \imath)$ can be used (also shown in [12]).

Neighbor Swap (NbrSwap): The minimal number of neighbor swaps to sort a permutation is the inversion number of the permutation. In [12] an $O(n^2)$ algorithm is presented. Here a faster $O(n \log n)$ and easy to implement algorithm is used. The idea is a balanced tree which is implemented as an array like a heap.

Element Swap (Swap): With a swap operation the length $|c_i|$ of a cycle c_i can be maximally reduced to a cycle c_i' of length $|c_i'| = |c_i| - 1$ and a second cycle of length $|c_i''| = 1$. The identity $\imath$ has n cycles of length 1. Therefore the element swap distance is $d_{\text{Swap}}(\pi, \imath) = \sum_{c_i}(|c_i|-1)$. This can be calculated in linear time $O(n)$.

Element Shift (Shift): To calculate the element shift distance all elements that stay on their place have to be detected. These elements form the *longest common subsequence* (LCS) or, in the case of permutations, the *longest increasing subsequence*. Here the algorithm of [13] is used and optimized for permutations. This algorithm needs $O(n \log n)$ time for calculating $d_{\text{Shift}}(\pi, \imath) = n - \text{LCS}(\pi, \imath)$. With van Emde Boas' data structure [16] a complexity of $O(n \log \log n)$ can be achieved.

Substring Reversal (SStrRev): This problem is known as *sorting by reversals* (SBR) and it is $\mathcal{NP}$-hard as proved by Caprara [2]. Therefore an easy to implement 2-approximation algorithm was used to calculate the distance [7] with $O(n^2)$ time complexity. The currently best known approximation guarantee is of 1.375 [1].

Substring Shift (SStrShift): This problem is known as *sorting by transpositions* (SBT) and it is presumed to be $\mathcal{NP}$-hard as well. For calculating $d_{\text{SStrShift}}$ an easy to implement $O(n^2)$ greedy algorithm was implemented. This algorithm reduces the breakpoints between increasing strips in the permutation. The currently best known approximation guarantee is of 1.375 [4].

Table 1. Operator overview

Operator	|OP|	Φ_{OP}	other operator names	dist.-alg. complexity
NbrSwap	$n-1$	$\frac{n(n-1)}{2}$	APEX, Swap	$O(n \log n)$
Swap	$\frac{n(n-1)}{2}$	$n-1$	2-cycle, element change, EX	$O(n)$
Shift	$(n-1)^2$	$n-1$	transposition, DSH	$O(n \log n)$
SStrRev	$\frac{n(n-1)}{2}$	$n-1$	2-opt, 2-change, reversal, INV	$O(n^2)$
SStrShift	$\frac{n^3-n}{6}$	$\lceil \frac{n+1}{2} \rceil$ (conjecture)	transposition	$O(n^2)$

4 Data Generation and Test Problems

To simulate the behavior of optimization heuristics the data is generated by (stochastic) hillclimb walks through the search space, because an important problem of random walk or random selection of elements is the small variance in distance to the optimum. The term *hillclimb* is here used in the sense of *go to a better solution* in every step.

For calculating the FDC value a random element of the search space was selected to start a new hillclimb walk and then a random neighbor was calculated.

If this neighbor had a better fitness value this element was selected for the next step. If no better neighbor was found after calculating n^2 elements of the neighborhood, the hillclimb walk was stopped and a new walk was started. After each walk the FDC value was calculated from all elements of all walks. If the difference between old and new calculated FDC value was less than 0.01 for three times the analysis was finished (figure 2).

In order to calculate the data source for PQ analysis the same procedure as for FDC is used. With the only difference, that in each step all n^2 neighbors are calculated. For these n^2 neighbors fitness and distance values are calculated and then the number of elements in quadrants $q_1, \ldots, q_4$ are counted.

For asymmetric TSP instances the distances for identity was set to zero, therefore identity $\imath$ is optimal and has length zero. All other weights in the

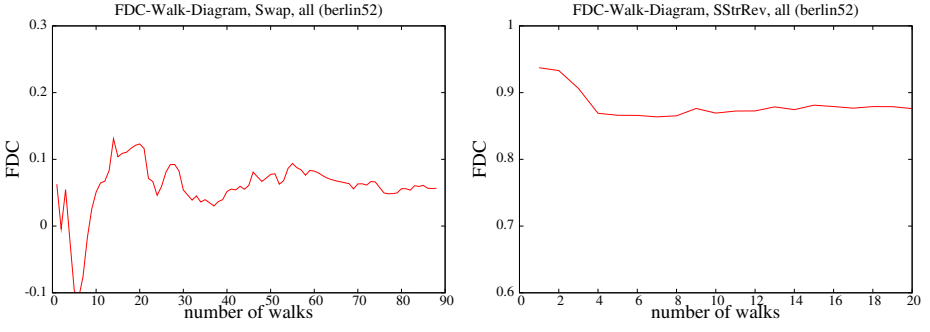


Fig. 2. Two examples for the convergence of FDC value by calculating it through hillclimb walks

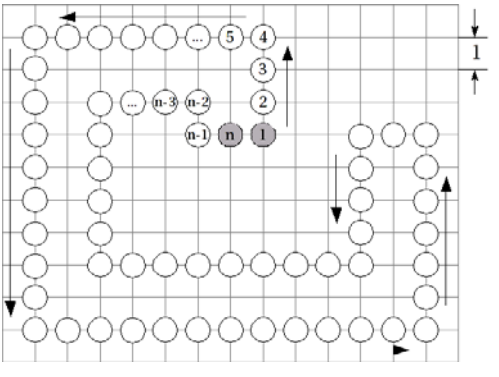


Fig. 3. The 2-dimensional euclidian TSP test instances are imported from well known TSPLIB [11]. To have more nontrivial test instances with known optimum, we developed the @-type euclidian TSP. All points are arranged on an equidistant grid with distance 1. The inner and outer side length are calculated in dependence of the length n. The distance to next city is 1 and the distance between inner and outer path is 2. Therefore the identity $\imath$ is the optimum with minimal length n.

distance matrix were randomly chosen from $\{1, 2, \ldots, n^{\exp}\}$. The parameter exp for asymmetric benchmark generation is used to analyze the impact of the ratio of weights and n to the FDC value.

5 Results

Extensive experiments for the five operators and applicable permutation based codings are performed. The FDC value for symmetric TSP was calculated for nine instances over three orders of magnitude. For asymmetric TSP ten instances are analyzed.

For symmetric TSP the SStrRev operator produces the highest FDC values (less tour length, less distance to optimum). The SStrRev operator performs the minimal change of two edges. The second best operators are the Shift or SStrShift operator (table 2). For this two operators the PQ diagrams reveals, that the Shift operator outclasses the SStrShift operator. This operator leads to better fitness elements with less distance to optimum. For greater n the Shift or SStrShift operators are inducing better FDC values. At instance `pr1002` their FDC values reach the FDC value of the SStrRev operator. This effect should be analyzed in further work.

For asymmetric TSP the SStrRev operator isn't a good choice, this arises from the asymmetric nature of the problem class. Here the SStrShift operator produces best FDC values (table 3). An explanation is that moving good optimized stages (substrings) of a tour to a better position leads to a search space which is better to optimize. The exponent for randomly generated weights in distance matrix does not influence the FDC value. The FDC values of SStrRev and Shift operator are comparable, but as shown in figure 5, the PQ diagrams reveals that the Shift operator leads to better fitness values (top and middle diagram on the left side).

Table 2. As expected, best choice for symmetric TSP instances is the SStrRev operator. Data from hillclimb walks induce highest FDC value in all experiments. This operator performs minimal change (two edges) on TSP tour. The Shift and SStrShift operators are changing three edges whereas the Shift operators specific neighborhood is a subset of the SStrShifts neighborhood.

problem instance	size n	Nbr Swap	Swap	Shift	SStr Rev	SStr Shift
@	26	0.01	0.16	0.26	**0.86**	0.11
@	50	0.01	0.05	0.43	**0.87**	0.37
eil51	51	0.0	0.07	0.39	**0.89**	0.37
berlin52	52	0.05	0.06	0.47	**0.88**	0.24
@	100	0.02	0.08	0.52	**0.87**	0.65
ch130	130	−0.01	0.06	0.52	**0.88**	0.64
tsp225	225	0.04	0.07	0.61	**0.85**	0.75
@	250	0.12	0.11	0.60	**0.84**	0.8
pr1002	1002	0.0	0.06	0.74	**0.75**	**0.76**

Table 3. For all tested asymmetric TSP instances the SStrShift operator leads to highest FDC values. The SStrRev and Shift operators have comparable FDC values, but PQ analysis shows that the SStrRev operator is inferior.

problem instance	size n	Nbr Swap	Swap	Shift	SStr Rev	SStr Shift
ATSP exp=1	26	0.03	0.06	0.36	0.30	**0.73**
ATSP exp=1	52	−0.02	0.01	0.27	0.22	**0.75**
exp=2	52	0.0	0.0	0.25	0.30	**0.74**
exp=3	52	0.02	0.02	0.32	0.27	**0.71**
exp=5	52	0.0	−0.01	0.24	0.27	**0.74**
ATSP exp=1	100	0.0	0.03	0.25	0.23	**0.71**
exp=5	100	0.01	0.02	0.21	0.19	**0.72**
ATSP exp=1	200	0.0	−0.01	0.14	0.18	**0.72**
ATSP exp=1	500	0.0	−0.02	−0.04	0.19	**0.67**
ATSP exp=1	1000	−0.03	0.0	−0.05	0.20	**0.60**

As a PQ diagram example the symmetric TSP (berlin52) instance is shown in figure 4. All five operators are plotted. Three operators, Swap, Shift and SStrRev, result in good fitness values. But only the SStrRev operator does reach the best fitness value and small distance. The Swap and Shift operator reach good fitness values, but these points are mostly far away from optimum.

The PQ diagrams for asymmetric TSP are shown in figure 5. The operators SStrShift and Shift lead to fitness values near to the optimum. The elements generated by SStrShift operator are closest to optimum but doesn't reach the optimum. This can be a symptom why asymmetric TSP are harder to optimize than symmetric TSP. FDC value gives a good hint for the selection of a convenient operator. But in some cases its oversimplifying nature is obscuring differences of the operator specific search space. In table 3 FDC values for Shift and SStrRev operator are comparable. But the PQ diagram in figure 5 (left side) shows, that SStrRev is clearly inferior, because worse fitness values are reached by hillclimb walks.

To analyze the distribution of local optima in search space and examine the difference between good and bad performing operators, the proportion of neighbors with worse fitness values (elements in q_1 and q_2) are plotted over distance. As can be seen in figure 6, the local optima with best performing operator SStrRev are heaping near the global optima. Local optima for operator Shift and SStrShift are uniformly distributed in search space. Local optima for operator NbrSwap are heaping at $\Phi_{\mathrm{OP}}/2$, but not in the direction of global optima.

6 Conclusion

To extensively analyze fitness distance correlation, we implemented algorithms for measuring the real (minimal) distance between permutations for five operators, as demanded by [12]. Instances for symmetric and asymmetric TSP were

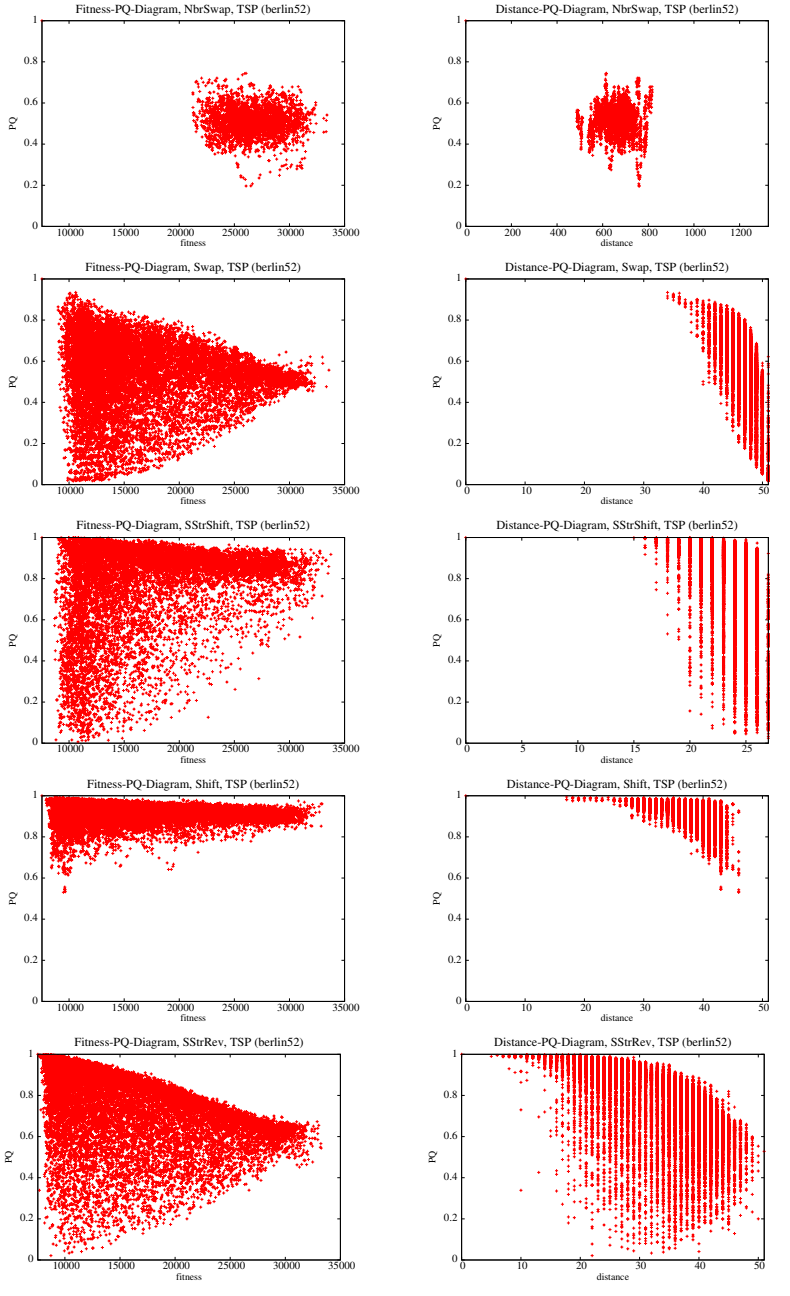


Fig. 4. An example for PQ analysis for all five operators (TSP berlin52). The diagrams are sorted from worst (top) to best (bottom) FDC value. On the left side the PQ is plotted over fitness (x-axis starting at optimum value) and on the right side over distance (x-axis ending at Φ_{OP}). The optimum located in the upper left corner of each diagram.

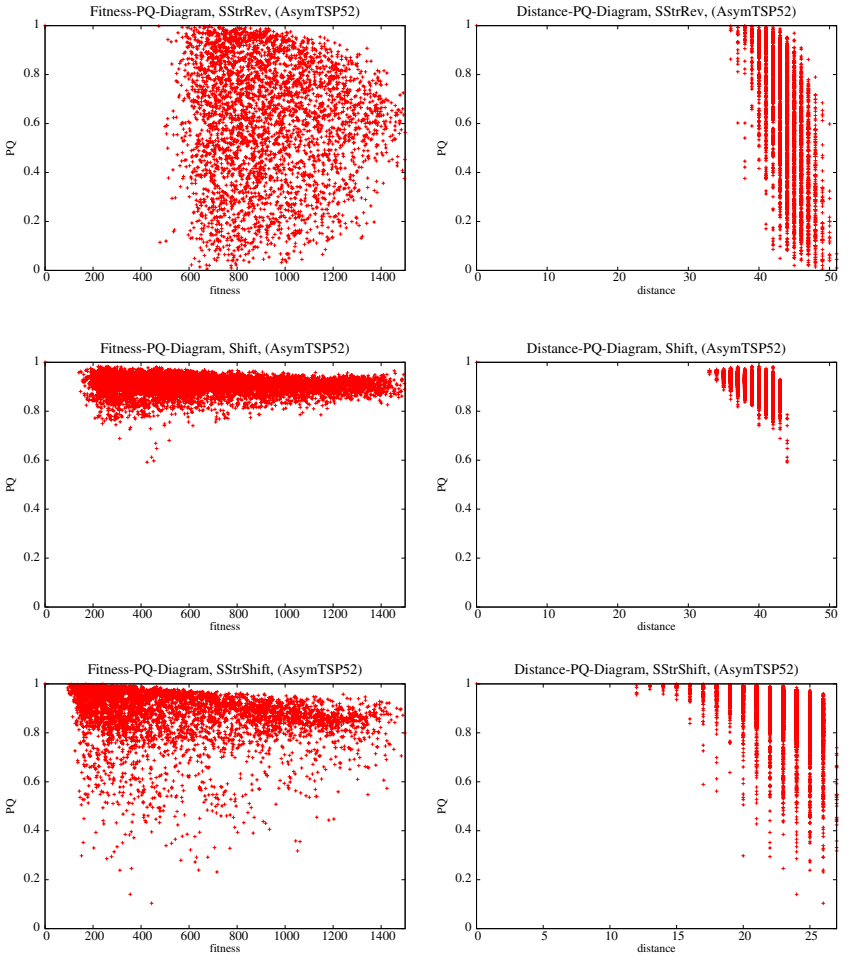


Fig. 5. Example PQ diagrams for AsymTSP with $n = 52$ and exp $= 1$. For asymmetric TSP instances the SStrShift operator produces the best FDC values. The SStrRev and Shift operators have approximately the same FDC values. But the fitness-PQ diagrams on the left side showing, that the Shift operator is outclassing the SStrRev operator.

analyzed. The best operator for symmetric TSP is the well known SStrRev operator. For asymmetric TSP the SStrShift operator leads to highest FDC values. We proposed PQ as an extension of FDC to measure local quality in search space. PQ diagrams can visualize additional characteristics of search space. The PQ diagrams can help to identify good operators even if the FDC values are approximately the same (examples shown in figure 4 and 5).

In future work we will use our framework to analyze further permutation based optimization problems, like bin packing or quadratic assignment problem (QAP). Additionally, more problem specific operators and their distance algorithms will be implemented to understand, what makes a search space easy to optimize

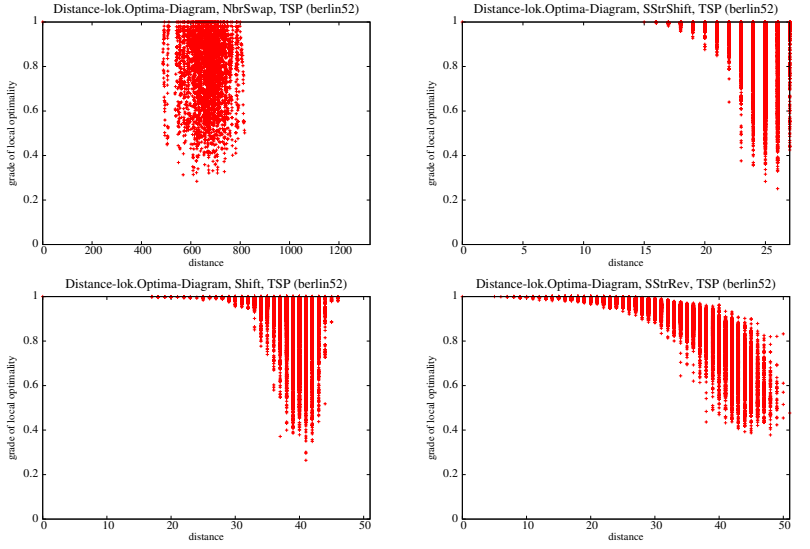


Fig. 6. Here the grade of local optimality is plotted (TSP berlin52). The y-axis indicates the proportion of neighbors which appear in quadrant q_1 or q_2. If all neighbors are in these quadrants a local optima is reached (the end of a hillclimb walk). It's visible how the local optima are distributed in search space.

and to identify characteristics of such operators. Possibly another way to gain knowledge of search spaces is to analyze PQ diagrams of artificial and misleading functions, like the so called ridge functions [9].

References

1. Berman, P., Hannenhalli, S., Karpinski, M.: 1.375-Approximation Algorithm for Sorting by Reversals. Electronic Colloquium on Computational Complexity (ECCC) 8(47) (2001)
2. Caprara, A.: Sorting by Reversals is Difficult. In: Proceedings of the first annual international conference on Computational molecular biology, pp. 75–83 (1997)
3. Collard, P., Gaspar, A., Clergue, M., Escazut, C.: Fitness Distance Correlation, as Statistical Measure of Genetic Algorithm Difficulty, Revisited. In: Proceedings of the European Conference on Artificial Intelligence, pp. 650–654 (1998)
4. Elias, I., Hartman, T.A.: 1.375-Approximation Algorithm for Sorting by Transpositions. IEEE/ACM Trans. Comput. Biol. Bioinformatics 3(4), 369–379 (2006)
5. Fonlupt, C., Robilliard, D., Preux, P.: Fitness Landscape and the Behavior of Heuristics. In: Evolution Artificielle, vol. 97 (1997)
6. Jones, T., Forrest, S.: Fitness Distance Correlation as a Measure of Problem Difficulty for Genetic Algorithms. In: Proc. of the 6th International Conference on Genetic Algorithms, pp. 184–192 (1995)

7. Kececioglu, J., Sankoff, D.: Exact and Approximation Algorithms for Sorting by Reversals, with Application to Genome Rearrangement. Algorithmica 13(1/2), 180–210 (1995)
8. Merz, P., Freisleben, B.: Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem. IEEE Trans. Evolutionary Computation 4(4), 337–352 (2000)
9. Quick, R.J., Rayward-Smith, V.J., Smith, G.D.: Fitness Distance Correlation and Ridge Functions. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, pp. 77–86. Springer, Heidelberg (1998)
10. Reeves, C.R.: Landscapes, Operators and Heuristic Search. Annals of Operations Research 86, 473–490 (1999)
11. Reinelt, G.: TSPLIB - A Traveling Salesman Problem Library. INFORMS Journal on Computing 3(4), 376–384 (1991)
12. Schiavinotto, T., Stützle, T.: A Review of Metrics on Permutations for Search Landscape Analysis. Comput. Oper. Res. 34(10), 3143–3153 (2007)
13. Szymanski, T., Hunt, J.: A fast Algorithm for Computing Longest Common Subsequences. Commun. ACM 20(5), 350–353 (1977)
14. Thierens, D.: Predictive Measures for Problem Representation and Genetic Operator Design (2002)
15. Tomassini, M., Vanneschi, L., Collard, P., Clergue, M.: A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming. Journal Evol. Comput. 13(2), 213–239 (2005)
16. van Emde Boas, P.: Preserving Order in a Forest in Less Than Logarithmic Time and Linear Space. Inf. Process. Lett. 6(3), 80–82 (1977)

A Genetic Algorithm to Minimize Chromatic Entropy

Greg Durrett[1], Muriel Médard[2], and Una-May O'Reilly[1]

[1] Computer Science and Artificial Intelligence Laboratory
[2] Research Laboratory for Electronics
Massachusetts Institute of Technology
{gdurrett,medard,unamay}@mit.edu

Abstract. We present an algorithmic approach to solving the problem of chromatic entropy, a combinatorial optimization problem related to graph coloring. This problem is a component in algorithms for optimizing data compression when computing a function of two correlated sources at a receiver. Our genetic algorithm for minimizing chromatic entropy uses an order-based genome inspired by graph coloring genetic algorithms, as well as some problem-specific heuristics. It performs consistently well on synthetic instances, and for an expositional set of functional compression problems, the GA routinely finds a compression scheme that is 20-30% more efficient than that given by a reference compression algorithm.

Keywords: chromatic entropy, functional compression, graph coloring.

1 Introduction

Chromatic entropy is a combinatorial optimization problem closely related to graph coloring, though it is much less well known because of its relative lack of applications. However, it has recently appeared in information theory as the bottleneck in a particular method of encoding data from two correlated sources. Specifically, a solution to chromatic entropy would allow the implementation of an improved scheme for data compression of correlated sources used as inputs to a function [1].

Chromatic entropy is known to be NP-hard, as shown by Cardinal et al. [2]. Given that this problem cannot be solved efficiently in theory, it is natural to consider finding solutions using genetic algorithms. In this paper, our goal is to introduce the problem and describe a genetic algorithm (GA) tailored to its specifics. In Section 2 we present the background on functional compression necessary to understand the application, then discuss the problem of chromatic entropy. Because our GA borrows heavily from previous algorithms for solving graph coloring, we proceed to briefly survey some of the relevant GA graph coloring literature. In Section 3, we describe the design choices for our GA. Section 4 analyzes the performance of the algorithm on synthetic probabilistic graphs and on graphs derived from certain instances of the functional compression problem, showing that the algorithm performs favorably on both. Section 5 concludes and discusses future work.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 59–70, 2010.

2 Background

2.1 Definitions

We use the standard definition of entropy throughout this paper.

Definition 1. *Let X be a discrete random variable with probability density function p that can take values x_i for $1 \leq i \leq n$. The entropy of X is given by*

$$H(X) = -\sum_{i=1}^{n}(p(x_i)\log_2 p(x_i)) \tag{1}$$

Conditional entropy will also be used frequently and is not quite a trivial extension of basic entropy.

Definition 2. *Let X and Y be discrete random variables, with a joint distribution $p(x, y)$, conditional distributions $p(x|y)$ and $p(y|x)$, and marginal distributions $p(x)$ and $p(y)$. If X can take values x_i for $1 \leq i \leq n$ and Y can take values y_j for $1 \leq j \leq m$, we define the conditional entropy $H(X|Y)$ as follows:*

$$H(X|Y) = \sum_{j=1}^{m}(p(y_j)H(X|Y = y_j)) = \sum_{j=1}^{m}\left(p(y_j)\left(-\sum_{i=1}^{n}(p(x_i|y_j)\log_2 p(x_i|y_j))\right)\right) \tag{2}$$

This can be thought of as an average of the conditional entropies $H(X|Y = y_j)$ weighted according to the probability that $Y = y_j$.

In addition to entropy, we require the notion of a coloring in a discrete, edge-node graph.

Definition 3. *Let $G = (V, E)$ be an undirected graph over the vertex set V with edge set E. A* k-coloring *of G is an assignment of "colors" to vertices of G using k colors such that no two adjacent vertices have the same color. More formally, we partition the vertices into k disjoint sets C_i such that if $x_1, x_2 \in C_i$, then $(x_1, x_2) \notin E$.*

Note that throughout this work, the graphs we are using are simple, undirected graphs. Henceforth, any graph not otherwise specified is simple and undirected.

2.2 Functional Compression Basics

Doshi et al. provide in [1] the main information-theoretic results relevant to the efforts of this paper. We provide a brief summary here of the key concepts presented in their paper.

Consider two correlated sources of data, X and Y, separated from each other and X separated from a decoder. Figure 1 shows the basic setup: we wish to encode the data from X in order to compute a function $f(X, Y)$ without loss at the decoder, and we want this encoding to be as efficient as possible. In the case where $f(X, Y) = (X, Y)$ (i.e. when we want to be able to recover both X and Y directly), the well-established rate result is the Slepian-Wolf bound described

in [3]. The theorem of [3] treats the more general problem when both X and Y are separate from the decoder, and states that X and Y can be sent at rates R_1 and R_2 satisfying $R_1 > H(X|Y)$, $R_2 > H(Y|X)$, and $R_1 + R_2 > H(X,Y)$. In our specific problem, when Y is available at the decoder, the optimal rate for the transmission of X is the conditional entropy of X given Y, $H(X|Y)$. Achieving this rate entails sending X using a more efficient encoding by taking advantage of its correlation with Y, knowing that the decoder can use its knowledge of Y to disambiguate the transmitted data. In the literature, this problem is known as the problem of functional compression with side information.

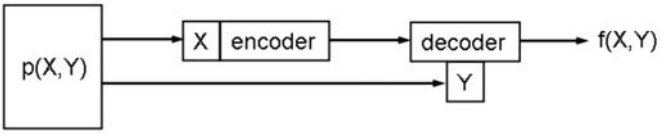


Fig. 1. The functional compression problem when Y is available as side information at the decoder

This analysis so far has ignored the fact that we are computing $f(X,Y)$ at the decoder, rather than reporting the two values directly. If f is a many-to-one function, we can do significantly better than the Slepian-Wolf bound. As an example taken from [1], consider when X and Y are integers and f is the function $X + Y \pmod 4$. Regardless of how large X and Y might be, it is sufficient to encode only the last two bits of each, rather than the entire integer. The Slepian-Wolf bound only optimizes the rate by taking advantage of the correlation between the two random variables; it does not take advantage of the properties of the function to further increase the efficiency of the encoding.

In general, it is not this simple to use our knowledge of f to improve our encoding. To analyze a given f, we construct something called the confusability graph.

Definition 4. *The confusability graph for X given Y, f, and $p(X,Y)$ is a graph $G = (V,E)$ where V contains a vertex v_x for each value x that X can take and E contains an edge between v_{x_1} and v_{x_2} if x_1 and x_2 are confusable. Two values of X are* confusable *if for some $y \in Y$, $p(x_1,y) > 0$, $p(x_2,y) > 0$, and $f(x_1,y) \neq f(x_2,y)$.*

This definition is slightly easier to grasp if one considers the complement of the graph. If two nodes are not confusable (i.e. there is no edge between them in the confusability graph), then for all possible values y of the random variable Y, either $p(x_1,y) = 0$ or $p(x_2,y) = 0$ (we can disambiguate the two values based on the fact that one of them can never appear with the given y), or $f(x_1,y) = f(x_2,y)$ (they yield the same value under the function, so there is no need to disambiguate them). See Section 4.1 for an example of using these rules to construct a confusability graph.

By this definition, two nodes that are disconnected on the confusability graph can safely be given the same encoding, since knowing Y and knowing that X takes one of these two values is all that we require to compute the value of f. This extends beyond pairs of nodes to independent sets of nodes. By coloring the confusability graph, we partition it into color classes, each of which is an independent set. To transmit a particular value, we can simply transmit the name of the color class that contains that value, without ambiguity as to what the value of $f(X, Y)$ will be. Using a Slepian-Wolf encoding on the distribution over the color classes yields a valid encoding scheme for the problem. This rate is precisely the conditional chromatic entropy of the confusability graph of X given Y, which we define mathematically in the next section. However, intuitively, this is appropriately analogous to the case when $f(X, Y) = (X, Y)$, for which the rate bound is the conditional entropy $H(X|Y)$. By coloring we have augmented the Slepian-Wolf method to compress data based not only on the correlation between X and Y, but on the properties of the function f, as captured by the confusability graph.

2.3 Chromatic Entropy

Armed with this intuition about the method, we can rigorously define chromatic entropy. Chromatic entropy is defined for simple graphs that have an associated probability distribution p over the set of nodes (i.e. $\sum_{v_i \in V} p(v_i) = 1$). We use the following definition from [2]:

Definition 5. *Let $G = (V, E)$ be a graph with a probability distribution p of a random variable X over the vertices and k color classes C_i. Set $p_{C_i} = \sum_{v \in C_i} p(v)$; this represents the probability that a vertex of G chosen according to p falls in the ith color class. The chromatic entropy of G given p and C is*

$$H_G^{\chi}(C, X) = -\sum_{i=1}^{k} p_{C_i} \log_2(p_{C_i}) \tag{3}$$

Essentially, this is the entropy associated with the probability distribution ϕ over the color classes C_i, where $\phi(C_i) = \sum_{v \in C_i} p(v)$.

Körner showed in [4] that by minimizing this quantity for a high enough "power" of a confusability graph, we can achieve an arbitrarily close approximation to the optimal rate for the case when we are transmitting a single value X to the decoder and the function $f(X)$ depends only on X, which is a simplified version of our compression problem with Y removed. Doshi et al. [1] extended this result to the side information case, where we have a conditional distribution $p(X|Y)$ rather than a univariate distribution. They define a quantity called conditional chromatic entropy, which by analogy with the definition of conditional entropy in Section 2.1, is given to be

$$H_G^{\chi}(C, X|Y) = \sum_{y \in Y} p(y) H_G^{\chi}(C, X|Y = y) \tag{4}$$

This conditional chromatic entropy is the optimal rate for encoding of X. It represents a rate optimized by taking advantage of both the correlation between the signals X and Y and the properties of the function.

For a discussion of the relationship between chromatic entropy and standard entropy, we refer the reader to Alon and Orlitsky's work in [5].

Heuristics. Given that the sum can be decomposed this way, it is helpful to consider heuristics for minimizing chromatic entropy motivated by the univariate chromatic entropy problem, as this is much easier to analyze than the conditional chromatic entropy problem.

Intuition suggests that reducing the number of colors also reduces chromatic entropy, but while this is a good heuristic, it is not necessary true that the coloring yielding the minimum chromatic entropy is a minimum coloring. This is proven in full depth by Cardinal et al. in [2], but we present here a simple example inspired by their proof. Consider the graph shown on the left in Figure 2. The graph is 3-colorable, and is clearly not 2-colorable by the existence of a 3-clique. Up to symmetries of vertices and color classes, there are only two three-colorings: $\{1,6\}$, $\{2,5\}$, $\{3,4\}$ (shown in Figure 2, middle) and $\{1,5,6\}$, $\{2,4\}$, $\{3\}$ (not shown). The first 3-coloring contains three color classes all with equal probability mass, and $H_G^\chi = 1.5849$. For the second 3-coloring, we can compute the chromatic entropy as follows:

$$H_G^\chi = -0.616 \log_2 0.616 - 0.333 \log_2 0.333 - 0.05 \log_2 0.05 = 1.1745 \qquad (5)$$

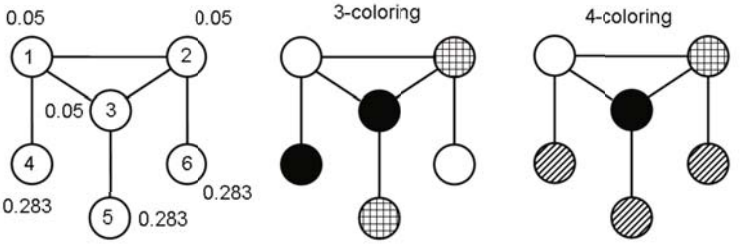


Fig. 2. Example graph to demonstrate that minimum colorings do not always yield minimum chromatic entropy values. One of the possible three-colorings and the optimal four-coloring are also shown.

However, the four-coloring with color classes $\{1\}$, $\{2\}$, $\{3\}$, and $\{4,5,6\}$ (Figure 2, right) has $H_G^\chi = 0.8476$. This example demonstrates that, as with the general entropy function, chromatic entropy is minimized when the terms in the entropy sum are as "uneven" as possible. The fourth color class provided "flexibility" to make the distribution ϕ over the color classes more uneven ($[0.85, 0.05, 0.05, 0.05]$ as opposed to $[0.6166, 0.3333, 0.05]$) thereby reducing the entropy.

One heuristic inspired by this example is to attempt to find independent sets with high probability mass and establish these as color classes, the idea being to unbalance the distribution as much as possible. There is another sense in which concentrating probability mass is beneficial: Lemma 2 of [2] tells us that, given a choice of color classes to assign a node to, chromatic entropy is increased the least by adding the node to the color class with the highest probability. This follows from the concavity of $f(x) = -x\log(x)$ in the entropy function. These two principles should inform our GA design in order to take full advantage of the problem structure.

Although we now know generally how to best distribute mass within color classes, we still do not know how many color classes to use. Theorem 6 in [2] states that the number of colors in a coloring that minimizes chromatic entropy for a particular graph cannot be bounded by any function of the number of colors in a minimum coloring for that graph, even for very specific, well-behaved types of graphs. Fortunately, there is a sense in which minimizing the number of color classes minimizes the chromatic entropy, as captured by the following lemma.

Lemma 1. *Given a graph G with distribution p and coloring C, if color classes C_1 and C_2 can be merged without violating the rules of coloring, then the coloring $C' = \{C_1 \cup C_2, C_3, ..., C_k\}$ has strictly lower chromatic entropy than the original coloring C.*

The proof follows directly from the definition of chromatic entropy, so we omit it. Because there are only constrained situations under which increasing the number of color classes decreases the chromatic entropy, minimizing the number of color classes is a powerful heuristic for minimizing chromatic entropy.

2.4 Genetic Algorithms for Graph Coloring

The connections between graph coloring and chromatic entropy indicate that a GA for one might be effectively adapted to the other, if the fitness function were to be changed appropriately. We can draw inspiration from the genomes used to solve graph coloring problems. Each individual in the population encodes a coloring, and in this sense chromatic entropy poses an interesting challenge because, for some representations, only a small fraction of the potential individuals encode legal colorings. The most naive representation might be to have a genome of length n for an n-node graph, and have each position store the color for the corresponding node. However, a mutation or crossover operation applied to an individual that represents a legal coloring will frequently produce an individual that represents an illegal coloring. In [6], Dorne and Hao successfully use this genome to solve graph coloring by taking their fitness function to be the number of edges that violate the coloring constraints. Their approach, though interesting, only works for a fixed number of colors, so it is unsuitable for chromatic entropy.

Eiben et al. [7] use a GA to minimize the number of violated coloring constraints, in order to find a valid k-coloring for a fixed k. They use a so-called

"order-based" representation. Each individual in the population is a permutation of the vertices of the graph in question. An individual is "decoded" into a coloring using a greedy algorithm as follows: the nodes are iteratively colored in the order specified by the permutation, and at each iteration, the current node is colored using the lowest-numbered color possible given the partial coloring of the graph. This genome admits a number of sensible mutation and crossover operators and every potential individual corresponds to a valid coloring. The reverse is not true: certain non-minimal colorings cannot be created by this greedy method. However, via a proof we omit for brevity, the optimal solution is assured to be attainable.

Sivanandam et al. [8] also use an order-based representation but their GA uses a different greedy algorithm for decoding. Their decoding introduces many more color classes than the decoding from [7]. For our problem this correlates with chromatic entropy values that are relatively high. We have elected to use the PX crossover of [8].

3 Algorithm Design

3.1 Genome and Objective

We use the order-based representation from [7] and [8] for our genome, as this sidesteps the issue of what to do when an invalid coloring is produced. For decoding, we use the greedy color assignment algorithm of [7] per Section 2.4.

Our objective is to minimize conditional chromatic entropy as defined in Section 2.3, given by

$$H_G^\chi(C, X|Y) = \sum_{y \in Y} p(y) H_G^\chi(C, X|Y = y) \tag{6}$$

Given a coloring (or a permutation that we decode into a coloring), we can evaluate this function directly given that we know the topology of the graph and the joint distribution.

Because we cannot derive a minimum objective value to use as a stopping condition, we run the GA for a fixed number of generations and report the best observed individual after these runs.

3.2 Mutation

For our mutation operator, we use the swap operator as described in Eiben et al. [7], which randomly swaps two adjacent vertices. We fix a mutation rate inversely proportional to the number of the vertices in the graph so as to swap a constant number of pairs of vertices in expectation. This parameter was tuned experimentally for each class of graphs that we considered. We also investigated varying the distance of the swap: rather than swapping two adjacent nodes, which may hardly change the overall coloring, we considered all pairs of nodes that are separated by a distance d, for fixed d. Experiments confirmed our intuition that

increasing d increased the amount of change the mutation operator induced. Higher swap distances were more likely to cause shakeups in the color classes, and they generally increase the number of color classes. However, this difference did not propagate to a difference in the performance of the GA. The performance difference between mutation operators with different swap distances is dwarfed by the variation among GA trials. Therefore, we fixed the swap distance at one and proceeded to examine crossover operators.

3.3 Crossover

We considered one crossover operator, the PX operator described in [8]. This operator produces a child by reordering a subset of the nodes in the first parent according to their order in the second parent. In our experiments, we observed that using PX appears to improve performance slightly, though again, the variance between individual samples is much more significant than this effect. However, the added computational cost of this operator is fairly small, so in light of the potential performance improvement, we incorporated the PX operator into our GA.

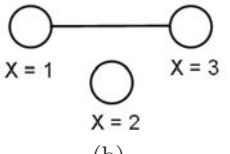
4 Results

4.1 Performance on Sample Problem Instances

In order to study the effectiveness of the overall functional compression scheme, we must start with a joint distribution and function of two variables, create the confusability graph, and compare the encoding from functional compression with the basic Slepian-Wolf encoding of the two correlated sources. Figure 3 shows a simple example of constructing a confusability graph from a table of function and probability values.

f, p	Y = 1	Y = 2	Y = 3
X = 1	0, $\frac{1}{7}$	1, $\frac{1}{7}$	1, $\frac{1}{7}$
X = 2	0, 0	1, $\frac{1}{7}$	1, $\frac{1}{7}$
X = 3	1, $\frac{1}{7}$	1, $\frac{1}{7}$	0, 0

(a)



(b)

Fig. 3. Example function f and joint distribution p over two random variables X and Y, each of which takes values in the set $\{1, 2, 3\}$. The figure shows the confusability graph of X given Y. There is an edge between $X = 1$ and $X = 3$ because they are confusable when $Y = 1$, but no other pair of X values is confusable.

Intuitively, the algorithm will be much more effective when there are relatively few edges in the graph and many nodes can be given the same coloring. We generated joint distributions and functions that would yield confusability graphs

with varying edge densities. The density of a confusability graph becomes quite high if the function takes a wide range of values and most combinations of X and Y occur with positive probability. This fact follows from the definition of the confusability graph. Therefore, we used $\{0,1\}$-valued functions and joint distributions with many "holes" (i.e. (x,y) pairs that occur with probability 0).

We randomly generate each joint distribution as follows: first, we set a parameter $\delta \in [0,1]$ to indicate the probability that a given entry in the table should be nonzero, then we choose each entry to be zero or nonzero according to this probability, and finally we normalize the table such that all nonzero entries are equiprobable and sum to one. If δ is set to a higher value, the graph will be denser, since the joint distribution is less helpful in allowing us to disambiguate values. The function table was generated in a similar fashion, with a parameter $\epsilon \in [0,1]$ chosen to be the probability that a given $f(x,y)$ would be 1. The most dense graph possible is achieved by $\epsilon = 0.5$, since a biased choice of function values will cause more of the (X,Y) pairs to be equal. By adjusting these parameters, we were able to create graphs of essentially any density for which we know $f(x,y)$ and $p(x,y)$.

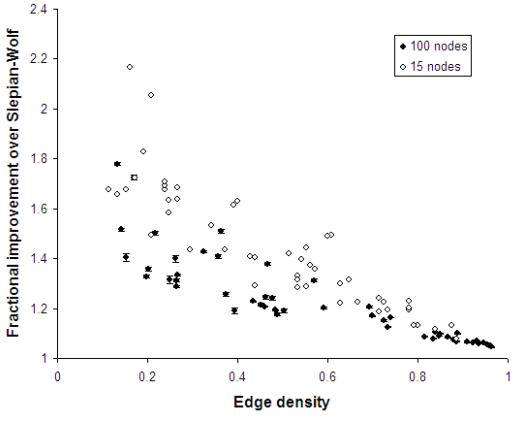


Fig. 4. Graph of the performance of the GA relative to the Slepian-Wolf bound for fifty 15-node and fifty 100-node graphs of varying edge densities, with 5 trials run on each graph. The y-axis shows the averaged values of $\frac{H_G^\chi(C,X|Y)}{H(X|Y)}$ for each graph, the fractional improvement over the Slepian-Wolf bound given by the best coloring C discovered by the GA. Error bars representing the standard deviation are shown for the 100-node graphs; in the 15-node case, the GA discovered identical (presumably optimal) values in all trials for all but two of the graphs.

In Figure 4, we display our results on graphs of varying densities created using this scheme, using for each graph a GA of 200 individuals iterated for 50 generations. We compare the encoding rate achieved by the GA (i.e. the lowest chromatic entropy value we could find) to the Slepian-Wolf bound, which can be thought of as the chromatic entropy of the graph where every node is colored

differently. For high densities, it is difficult to garner much improvement, as we need to use many relatively small color classes when coloring the graph. However, for low densities, we can improve on the Slepian-Wolf bound by 30% or more, which translates to a 30% reduction in the amount of data that needs to be transmitted in the original problem.

4.2 Performance on Random Graphs

Though these graphs do represent a simple class of problem instances that are theoretically appropriate, our particular method of constructing functions and joint distributions might be somehow restricting the range of instances we are considering. We would like to be able to demonstrate our algorithm's performance in a general setting and argue that it will do well at minimizing chromatic entropy for any given graph. To do this, we will construct graphs of all densities with completely random structures (i.e. all edges are have an equal, fixed probability of being included), and then experiment with a range of probability distributions on the nodes. Although these graphs will generally have no particular interpretation in the context of functional compression in which edge structure is linked to the joint distribution, it does allow us to convince ourselves of our algorithm's viability for applications other than confusability graphs.

We generated random 100-node graphs of various densities using Culberson's graph generator, described in [9]. In the generation of these graphs, edges are included or not included independently and with a fixed probability, so graphs produced in this manner will have no special structure. We then attached sensible joint distributions to these graphs. We induce an ordering on the n vertices and define a random variable X that takes values over the vertices $v_1, ..., v_n$. We also define a random variables Y that takes values $y_1, ..., y_n$ not associated with the vertices. Our joint distribution over these is as follows:

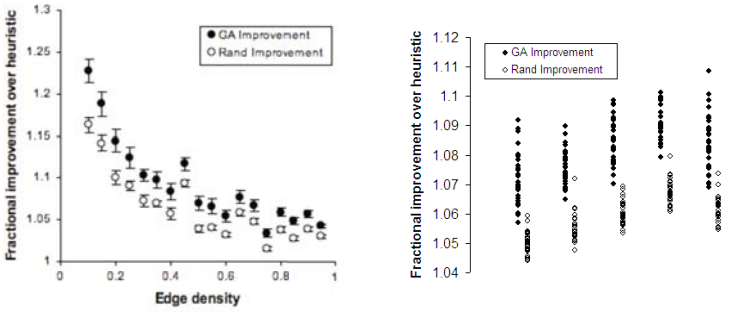
$$P(x_i, y_j) \propto \exp\left\{-\frac{1}{2}\left[i-\frac{n}{2}, j-\frac{n}{2}\right]\begin{bmatrix} s & -cs \\ -cs & s \end{bmatrix}\left[i-\frac{n}{2}, j-\frac{n}{2}\right]^T\right\} \qquad (7)$$

$s > 0$ and $c \in (0, 1)$ in the inverse covariance matrix term are constants so that we can tune the degree of variance and covariance in the parameters, respectively. The motivation behind this was to create distributions with varying degrees of correlation between X_i and Y_j in order to evaluate the GA on a variety of instances.

We will compare the GA performance to a heuristic and random search. The heuristic is a deterministic method that is independent of the any ordering of the nodes in the graph. Its goal is to maximize probability mass on high-entropy color classes as much as possible because this frequently minimizes chromatic entropy. It greedily colors the nodes in the graph in descending order of marginal probability mass $P(X)$. To each node, it assigns the color whose color class has the highest current weight in terms of $P(X)$ and that admits a legal coloring. The random search examines as many random individuals in the search space as the GA evaluated for fitness over an entire run, and returns the best among these.

Experiments showed that the performance of the GA relative to the heuristic and random search was independent of the covariance matrix, so we fix $s = 4$ and $c = 0.5$. We know from Section 4.1 that density is an important factor for the performance of the algorithm.

We fixed the number of individuals in the population at 200 and the number of generations at 50, to maintain consistency with the experiment in Section 4.1. The results are shown in Figure 5. Both the GA and random search consistently outperform the heuristic. In addition, we see that the GA consistently does modestly better than random search, with Wilcoxon sign-rank tests yielding p-values of less than 2×10^{-6} for every graph under investigaton. Given the small marginal computational cost to implement the machinery of the GA, we prefer the GA to the random search.



(a) Averaged GA vs averaged RAND on graphs of varying density

(b) Thirty trials of GA vs RAND on each of five graphs of density 0.5

Fig. 5. (a) shows the performance of the GA and RAND relative to the heuristic for 100-node random graphs of varying edge densities. Each point represents the average performance of the given method on the graph averaged over 30 trials, with error bars showing the standard deviation. The y-axis shows $\frac{H_G^X(C_{\text{Result}}, X|Y)}{H_G^X(C_{\text{Heuristic}}, X|Y)}$ for each graph, the fractional improvement in chromatic entropy for the coloring found by the method over the chromatic entropy of the heuristic coloring. Wilcoxon sign-rank tests established the statistical significance with a p-value of less than 2×10^{-6} for each point. (b) compares GA and RAND more closely to show that GA almost always performs better than RAND across multiple trials and multiple graphs of the same density, and therefore we are justified in using averages on one graph to contrast the two.

5 Conclusion

In this paper, we implemented a GA to minimize conditional chromatic entropy for a given graph with a joint probability distribution over its vertices. The GA employs an order-based representation combined with a greedy coloring heuristic. We applied this algorithm to the problem of functional compression with side information, wherein the minimization of the conditional chromatic entropy of a confusability graph supports approximation of the optimal encoding rate. This scheme allowed us to improve Slepian-Wolf encodings by around 30% for

relatively sparse confusability graphs, and furthermore, the algorithm routinely outperformed random search and a very inexpensive heuristic.

Future work along a theoretical avenue to pursue is fitness landscape analysis. Chromatic entropy is an interesting problem in that there is something of a continuous nature to it as a result of having a probability distribution over the vertices. Changes in the color class assignments of very low probability vertices have correspondingly small effects in the overall chromatic entropy calculation. This could yield a fitness landscape that is atypical of combinatorial optimization problems, which could in turn suggest even better algorithms to solve it efficiently.

References

1. Doshi, V., Shah, D., Medard, M., Jaggi, S.: Distributed functional compression through graph coloring. In: Data Compression Conference, DCC 2007, March 2007, pp. 93–102 (2007)
2. Cardinal, J., Fiorini, S., Van Assche, G.: A graph coloring problem with applications to data compression (2004)
3. Slepian, D., Wolf, J.: Noiseless coding of correlated information sources. IEEE Transactions on Information Theory 19(4), 471–480 (1973)
4. Körner, J.: Coding of an information source having ambiguous alphabet and the entropy of graphs. In: 6th Prague Conference on Information Theory, pp. 411–425 (1973)
5. Alon, N., Orlitsky, A.: Source coding and graph entropies. IEEE Trans. Inform. Theory 42, 1329–1339 (1995)
6. 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)
7. Eiben, A.E., Van Der Hauw, J.K., Van Hemert, J.I.: Graph coloring with adaptive evolutionary algorithms. Journal of Heuristics 4(1), 25–46 (1998)
8. Sivanandam, S.N., Sumathi, S., Hamsapriya, T.: A hybrid parallel genetic algorithm approach for graph coloring. Int. J. Know.-Based Intell. Eng. Syst. 9(3), 249–259 (2005)
9. Culberson, J.C., Luo, F.: Exploring the k-colorable landscape with iterated greedy. In: Dimacs Series in Discrete Mathematics and Theoretical Computer Science, pp. 245–284. American Mathematical Society, Providence (1995)

Evolutionary Approaches to the Three-dimensional Multi-pipe Routing Problem: A Comparative Study Using Direct Encodings

Marcus Furuholmen[1], Kyrre Glette[2], Mats Hovin[2], and Jim Torresen[2]

[1] Aker Solutions AS
Snaroyveien 36, P.O. Box 169, 1364 Fornebu, Norway
[2] University of Oslo, Department of Informatics,
P.O. Box 1080 Blindern, 0316 Oslo, Norway
{marcusfu,kyrrehg,jimtoer,matsh}@ifi.uio.no

Abstract. In this study, three Genetic Algorithms (GAs) are applied to the Three-dimensional Multi-pipe Routing problem. A Standard GA, an Incremental GA, and a Coevolutionary GA are compared. Variable length pipelines are built by letting a virtual robot move in space according to evolved, fixed length command lines and allocate pipe segments along its route. A relative and an absolute encoding of the command lines are compared. Experiments on three proposed benchmark problems show that the GAs taking advantage of the natural problem decomposition; Coevolutionary GA, and Incremental GA outperform Standard GA, and that the relative encoding works better than the absolute encoding. The methods, the results, and the relevant parameter settings are discussed.

1 Introduction

Inspired by Darwinian evolution in natural systems, Genetic Algorithms (GAs) [1] have been successfully applied as global optimizers to several large and complex industrial problems [2]. GAs are population based, stochastic algorithms which are especially suitable for locating near-optimal solutions to problems involving high dimensionality, multi modality, and discontinuous search spaces.

In Standard GA , a *single* population of candidate solutions, called *individuals*, is implemented, and each individual codes for an entire solution to the problem. Many real world problems, however, consist of more or less interdependent sub problems. Such problems may be solved more efficiently by taking advantage of the natural problem decomposition. One approach to such problems is inspired by *coevolution* in natural systems, where species interact and adapt to each other. Cooperative Coevolutionary GAs operate in a *cyclical* manner, with several separate populations, each population working on a particular part of a larger problem. CCGAs have been found to be able to solve complex problems involving interacting sub problems [3] [4] [5] by taking advantage of the natural problem decomposition. Another approach to interacting subproblems is *incremental evolution*, where each sub-problem is solved *sequentially*. The approach

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 71–82, 2010.

is similar to the waterfall model, commonly encountered in engineering systems development [6]. Incremental GAs have been suggested in literature [7] [8] [9] as an approach enabling the generation of complex hierarchical solutions by taking advantage of the natural problem decomposition and hence limiting the search space.

The Three-dimensional Multi-pipe Routing (3DMPR) problem is concerned with automating and optimizing pipe routing design involving several pipelines in three dimensions. In this paper, we compare the performance of a Standard GA (SGA), a Cooperative Coevolutionary GA (CCGA), and an Incremental GA (IGA) applied to the 3DMPR problem. Additionally; a random search (RS), and two different encodings; a relative and an absolute encoding, are implemented and compared. Our main interest in this study is to compare the different approaches, using *direct encodings* and a minimum amount of domain knowledge (uninformed). Future research will focus on learning, and taking advantage of heuristic knowledge (informed), by implementing a machine learning approach, and finally to compare the results. To the author's best knowledge, no benchmark problems have been proposed for the 3DMPR problem. We therefore propose three benchmarks for experimentation and future comparisons.

1.1 Pipe Routing Design

Pipe routing can be understood as to be a subset of assembly design and is important for several industrial applications such as process plant layout, circuit layout, aircraft and ship design. In ship design, pipe routing takes over 50% of the total detail-design man-hours [10] and can run as high as 80% of the purchased equipment cost or 20% or the fixed-capital investment in fluid-process plants [11]. Pipe routing is normally done by human experts following a piping and instrumentation diagram (P&ID), where the location of various equipment is predetermined. However, the design process is complex and time consuming, making it practically impossible for the designer to test several scenarios.

Pipelines may be subjected to several constraints and objectives to be optimized. E.g. pipelines must connect terminals at given locations and avoid obstacles, while the number of bends and the overall material cost should be minimized. Additionally, several other objectives may be specified such as to minimize the amount of pipe support, provide access for ease of maintenance, minimize stress and design for thermal expansion.

Theoretically, pipe routing can be seen as a special case of general path planning in robotics, in which there are two major families of approaches known as *cell decomposition* and *skeletonization* [12]. Each approach reduces the continuous path-planning problem to a discrete graph search problem by identifying some canonical states and paths within the free space. Cell decomposition involves generating a uniform grid and is often used in conjunction with a potential field function. A potential field function is defined over state space that is to be minimized; e.g. the function has minimum value at the goal cell and maximum values at cells occupied by obstacles. Skeletonization involves generating a nonuniform graph according to some heuristic, and is based on the idea of reducing the robot's free

space to a lower dimensional representation, for which the planning problem is easier. In order to find paths satisfying the constraints, and optimizing the objectives, the generated graphs are commonly searched by some search algorithm. Deterministic shortest path algorithms such as A* [13] and Dijkstra [14] guarantee an optimal solution given sufficient time (other methods are Lee's maze algorithm [15] and the continuous Line-search algorithm [16]).

Pipe routing belongs to a class of optimization problems with very large, multimodal search spaces, where one is more interested in finding feasible solutions in practical time than trying to find the absolute optimal solution. Since straight pipes are preferred, deterministic shortest path algorithms would require a skeleton to be provided. However, the skeletonization approach may prove problematic for multi-pipe routing due to the fact that once a pipeline is changed, it would require a new calculation of the skeleton, which may be a computationally demanding process. A cell decomposition approach is therefore implemented. Cell decomposition has the advantage that it is simple to implement, but it also suffers from the curse of dimensionality, suggesting the 3DMPR problem to be a good candidate for optimization by stochastic search algorithms, such as GAs.

1.2 Previous Approaches

Most of the pipe routing literature limits the problem to single pipelines or to two dimensions and evaluation mostly focuses on the accommodation of the physical constraints [17] such as avoiding collisions and reaching the goal cell.

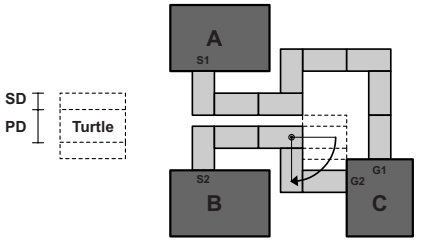
Ito optimized in [18] single pipelines in two dimensions by introducing a GA approach to support interactive pipe planning. The method was based on cell-decomposition and a potential energy function. Pipelines were represented as variable length genomes based on a character set representing absolute direction actions. Local heuristics were implemented in order to reduce the search space, and repair functions were used in order to handle non-feasible solutions.

Kim et al. optimized multiple two dimensional pipelines by minimizing pipe lengths in [19] by first generating a skeleton consisting of "Steiner points" and "escape graphs", and then to let search algorithms operate on chromosomes representing all connections. The authors found that GA outperformed both stochastic hill-climbing and simulated annealing in larger test cases. However, collision handling between pipelines was not addressed.

Sandurkar and Chen optimized single pipelines in three dimensions in [20] where the shape of obstacles were more realistically implemented by use of tessellated objects. A GA operated on a composite set of variables representing length of pipes, the direction cosines of each pipe, the angles of bends between successive pipes, and the number of bends along the pipeline.

Wang et al. optimized in [21] three pipelines in three dimensions by manually setting the number of bends, and letting a GA optimize the coordinates of the pipe segments by a floating number representation.

Park and Storch optimized multiple pipelines in three dimensions in [10] for a case study of a ship engine room. A cell generation method was introduced in



Pipeline AC Relative:
Ex1: F L F F L F R F F R F F # # # #
Ex2: # F # # L F F # L F R F F R F F
Pipeline AC Absolute:
Ex1: S E E N E E S S # # # # # # # #
Ex2: # S E # # E N E E # S S # # # #
Pipeline BC Relative:
Ex1: F R F F R F L F # # # # # # # #
Ex2: R L F # R # F F # R F L # F R L
Pipeline BC Absolute:
Ex1: N E E S E # # # # # # # # # # #
Ex2: # # # N # # E # E # # # S E # #

Fig. 1. Turtle building pipelines by interpreting evolved command lines

order to develop candidate pipeline solutions, and an algorithm was developed to select appropriate pipelines from a tree of combinations.

The next section explains the method, the experimental setup and the results are presented in section 3 and discussed in section 4 while section 5 concludes.

2 Method

This section explains the problem representation, the fitness evaluation, the various algorithms, and finally, the benchmark problems.

2.1 Problem Representation

A two dimensional representation of a multi-pipe routing problem with three modules, two pipelines, and several examples of valid genotypes, is illustrated in figure 1. Linear, fixed length genomes are composed out of symbols coding for specific commands. A pipeline is constructed by letting an imaginary robot, called a *turtle* (as in turtle graphics), move about in space from the start position S, by interpreting the commands in the genome and allocating pipe segments to visited grid cells. The diameter of the pipe segment PD, as well as the required safety distance SD, is represented by the turtle.

A command is either a *move* command, which moves the turtle in a specified direction, or a *turn* command, which changes the orientation of the turtle. In order to enable the construction of pipelines of different lengths, a special command # tells the turtle to "do nothing". Two different encodings were implemented and compared; in a *relative* encoding, the commands $C_r = [F, R, L, U, D, \#]$ codes for move one step forward, turn right/left/up/down and "do nothing"; in an *absolute* encoding, the commands $C_a = [N, S, E, W, U, D, \#]$ codes for (turn and) move north/south/east/west/up/down and "do nothing".

The size of the search space as defined by the representation of the genotype S_g can be determined by $S_g = C^{\alpha \times N}$ where C is the number of commands, α is the number of alleles coding for a single pipeline and N is the number of pipelines. However, the solution space as defined by the number of possible phenotypes S_p is drastically reduced because a large number of different genotypes codes for the same phenotype, and because several different phenotypes are equally fit.

2.2 Fitness Evaluation

A pipeline should connect its terminal to the goal position G and stay clear from obstacles, as well as minimize the number of bends and the overall length. By inspection of figure 1, it is e.g. evident that pipeline AC is not optimal due to an excessive number of bends. Notice also the added complexity introduced by the multi-pipe scenario; a pipeline must not collide with the obstacles, with itself, or with any other pipeline in the problem. A single pipe segment is checked for collision by determining if the corresponding turtle at that position collides with the "physical" part of any other object in the environment.

A set of N pipelines are evaluated by the following aggregate scalar error function to be minimized;

$$E = \sum_{i=1}^{N} (w_1 d_i + w_2 c_i + w_3 b_i + w_4 s_i), \tag{1}$$

where d_i is the Manhattan distance between the terminal of the last pipe segment of the i'th pipeline and the goal position. The variable c_i is the total number of collisions between pipeline i and the environment, while b_i is the number of bends and s_i is the number of segments of pipeline i. The different components are weighted by the values $w_1 = 150$, $w_2 = 100$, $w_3 = 10$ and $w_4 = 1$ which were set manually by trial and error.

2.3 Genetic Algorithms

This section explains the implementations of the Standard GA, the Cooperative Coevolutionary GA and the Incremental GA which were compared. The algorithms are illustrated in figure 2.

Standard GA (SGA) is by far the most frequently used EC algorithm, where each individual in a single population p codes for an entire set of N pipes, as illustrated in figure 2 a). This is done by letting the genome of each individual be composed out of $\beta(= N)$ genes, where each gene codes for a single pipeline. The fitness from an evaluated individual is fed back into the evolutionary process.

The Cooperative Coevolutionary GA (CCGA) approach takes advantage of the natural problem decomposition by implementing $P(= N)$ populations, as illustrated by figure 2 b). The genome of each individual consists of a single gene ($\beta = 1$) coding for a single pipeline. A cyclical process consisting of $S(= P)$ steps, involving selected members from all populations, is executed for each generation. For each step, all members of a specific population (colored white) are evaluated, while the other populations (colored gray) are frozen - representing only the best individual found so far (the *elite*). The approach is similar to the cooperative coevolutionary system proposed by Potter and De Jong in [4].

The Incremental GA (IGA) approach takes advantage of the natural problem decomposition by letting $P(= N)$ populations code for N pipelines respectively, as illustrated in figure 2 c). As compared to the CCGA, the complexity is increased in $S(= P)$ sequent steps which evolve for a fixed number of generations.

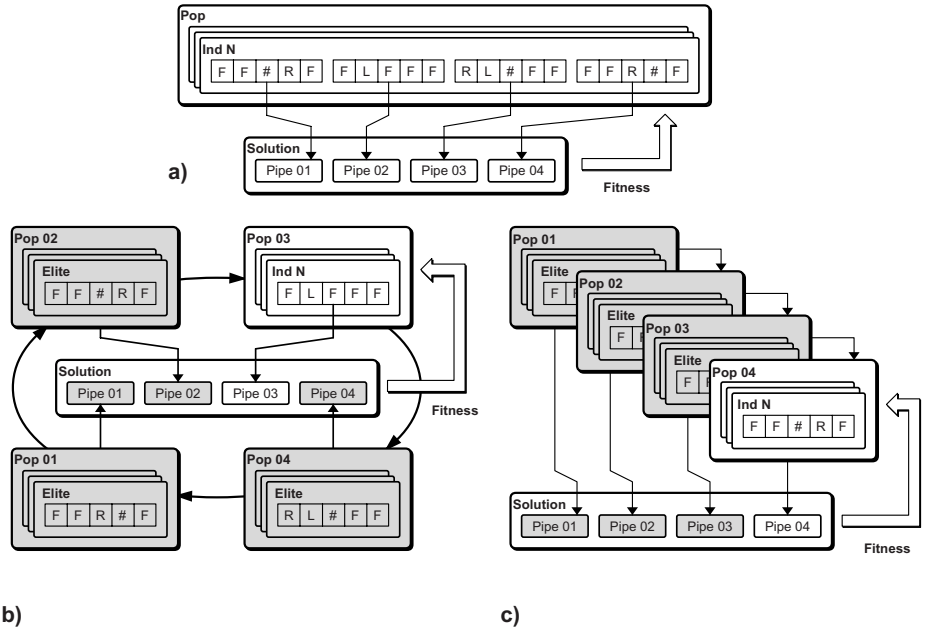


Fig. 2. a) Standard GA; b) Coevolutionary GA; and c) Incremental GA

For each step; a new pipeline is added to the problem (colored white), the previous pipes are fixed (colored gray) and will never be revisited.

2.4 Benchmark Problems

To the author's best knowledge, no benchmark problems exist for the 3DMPR problem; we therefore propose three benchmark problems, as illustrated in table 1, each problem consisting of four pipelines to be connected.

The "Square" problem consists of four equal sized modules to be interconnected, as illustrated in the P&ID in figure 3. The IO (in/out) points are evenly distributed in the horizontal plane; however, several fit solutions are in three dimensions due to the required interconnections and the upper bound on pipe length set by the maximum gene length, as illustrated in the evolved solution in figure 5 a). The "Twist" problem consists of two modules to be interconnected

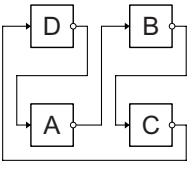


Fig. 3. The P&ID for the 3DMPR-P1 benchmark problem

Table 1. 3DMPR Benchmark Problems

Square						Twist						Hub					
Modules			Pipelines			Modules			Pipelines			Modules			Pipelines		
ID	Dim	Pos	ID	Start	Goal	ID	Dim	Pos	ID	Start	Goal	ID	Dim	Pos	ID	Start	Goal
A	4,4,4	-5,-5,0	AB	-3,-5,0	3,5,0	A	4,8,8	-5,0,0	AB1	-3,2,-2	3,-2,2	A	6,12,6	0,0,1	BA	-14,0,0	-3,-3,1
B	4,4,4	5,5,0	BC	7,5,0	3,-5,0	B	4,8,8	5,0,0	AB2	-3,-2,-2	3,2,2	B	4,4,4	-16,0,0	CA	-3,15,0	-3,3,1
C	4,4,4	5,-5,0	CD	7,-5,0	-7,5,0				AB3	-3,2,2	3,-2,-2	C	4,4,4	-5,15,0	DA	12.5,8,0	3,3,1
D	4,4,4	-5,5,0	DA	-3,5,0	-7,-5,0				AB4	-3,-2,2	3,2,-2	D	4,4,4	14.5,0,0	EA	16.5,0,0	3,-3,1
												E	4,4,4	10.5,8,0			

by four pipes. Each pipe must connect to the opposite side but at different locations, as illustrated by the evolved solution in 5 b). The "Hub" problem is a common scenario found in several industrial applications where several modules must connect to a central unit, as illustrated by the evolved solution in 5 c).

3 Experiments and Results

This section describes the experimental setup and the results from comparing the presented algorithms on the benchmark problems.

The experimental settings are reviewed in table 2, and the following explains the details. For each approach, P populations were randomly initiated with population size μ. The GA's used a generational model with $\lambda(=\mu)$ offspring and elitism enabled (one elite per population). The termination criteria was set to a fixed number of generations G per population. The number of genes β corresponds to the number of pipelines coded by a single individual, and the gene length α sets the maximum length/complexity of a pipeline. Selection was implemented using stochastic universal sampling [22]. Two-point crossover with crossover probability p_c, and bit-flip mutation with mutation probability p_m, was implemented. The mutation and crossover probabilities for each algorithm were set by trial and error.

In order to make a fair comparison between the evolutionary approaches, the evolutionary parameters were set so that an equal number of individual fitness evaluations $N_f = 400.000$ were executed for all approaches. However, for each approach, an unequal number of pipeline evaluations, and a corresponding number of (time consuming) collision detections, must be executed. E.g. in SGA, all pipes must be evaluated per generation, since we do not know in advance which pipe has changed since the last generation, as opposed to CCGA and IGA.

For each problem, 10 independent runs of the SGA, CCGA, and IGA were executed. The average performance of each algorithm, tracking the best individual called the *elite*, is plotted in figure 4. For the "Square" problem, a random search (RS), and a CCGA with absolute encoding (CCGA*) was also tested (10 runs) and compared. The random search (RS) was implemented by tracking the elite from generating $\mu \times G$ random solutions.

The experimental results, for each algorithm, applied to each problem, are listed in table 3. The table shows the error (according to error function 1) of the best found solution, the average error, and the standard deviation after G generations.

Table 2. Evolutionary parameter settings

GA	P	G	μ	β	α	p_m	p_c
SGA	1	1000	400	4	100	0.0025	0.3
CCGA	4	1000	100	1	100	0.01	0.3
CCGA*	4	1000	100	1	100	0.01	0.3
IGA	4	250	400	1	100	0.01	0.3
RS	1	1000	400	4	100	-	-

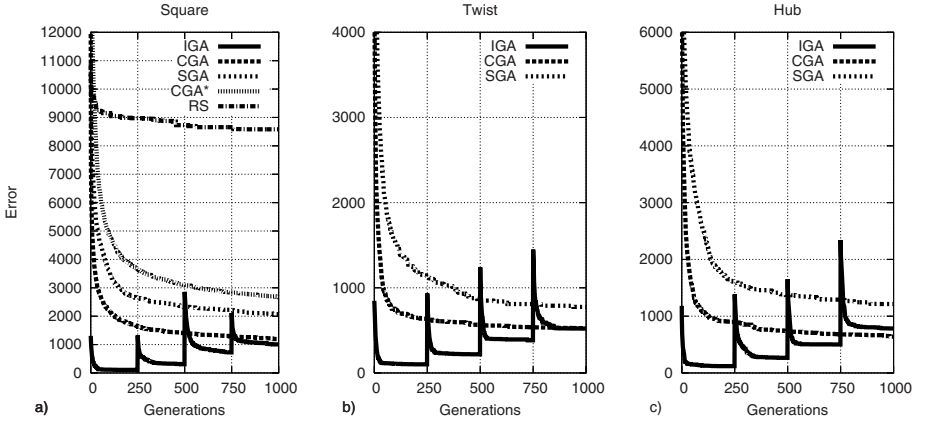


Fig. 4. Errorplots of the various GAs averaged over 10 runs for a) the "Square" problem, b) the "Twist" problem and c) the "Hub" problem

Table 3. Experimental results (error)

	Square			Twist			Hub		
GA	Best	Avg	Dev	Best	Avg	Dev	Best	Avg	Dev
SGA	1799	2065	177	562	773	168	878	1214	374
IGA	574	1007	215	428	521	87	578	784	136
CCGA	752	1195	229	436	526	78	410	600	139
CCGA*	2414	2669	197	-	-	-	-	-	-
RS	7616	8583	386	-	-	-	-	-	-

The results shows that the average performances of both IGA and CCGA are signifacantly better than SGA after a fixed number of generations. IGA found the best solution to the "Square" and the "Twist" problem, while CCGA found the best solution to the "Hub" problem. The phenotypes of the best found solutions are illustrated in figure 5.

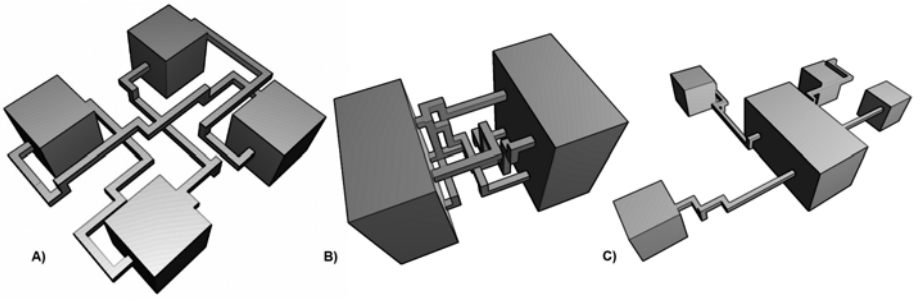


Fig. 5. The best found solutions to the 3DMPR benchmarks problems A) the "Square" problem (IGA) , B) the "Twist" problem (IGA) and C) the "Hub" problem (CCGA)

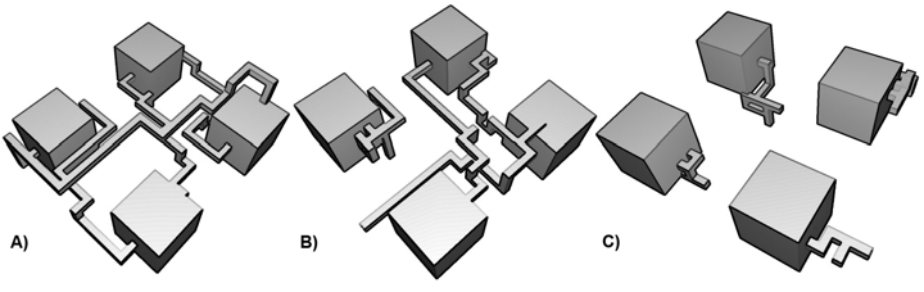


Fig. 6. The "Square" problem optimized by A) CCGA B) SGA C) RS

The CCGA implementing an absolute encoding (CCGA*) performs worse than CCGA with relative encoding, and all GA's performs significantly better than the random search. The best solutions obtained by CCGA, SGA, and RS, when applied to the "Square" problem, are illustrated in figure 6.

4 Discussion

The experiments showed that the evolutionary methods taking advantage of the natural problem decomposition (IGA and CCGA) obtained the best solutions. A common explanation to this is that the search space is reduced. We hypothesize, however, that the main reason for the better performance is that by decomposing the problem, a-priori knowledge is introduced which guides the search more strictly to specific regions of the search space. This lets each population specialize to a smaller part of the problem. E.g. in the case of CCGA; N smaller, incremental steps/decisions are taken during a single generation, introducing more knowledge to the global search algorithm, compared to SGA.

Potter et al. concluded in [4] that CCGAs could improve search when applied to independent parameters, but that it would be less suitable for interdependent parameters. This is confirmed in table 3, which shows that CCGA was 51% better than SGA on the "Hub" problem, consisting of rather independent pipelines,

and 32% better than SGA on the "Twist" problem, consisting of highly interdependent pipelines. However, CCGA was able to effectively converge to higher fit solutions as compared to SGA on all problems.

The IGA approach demonstrated the best performance for the "Square" and the "Twist" problem. We expected this approach to easier get stuck in local minima since once a pipe is optimized it can never be revisited. However, several parameter settings may influence the results, such as the sequence of pipes to optimize as well as properties of the specific benchmark problems. We believe further research is needed in order to determine the better approach.

It was found that the performance of the different approaches was largely influenced by the mutation probabilities. The best results were achieved setting $p_m = 0.025$ per allele for the SGA, and $p_m = 0.01$ for the CCGA and IGA. In both cases, the settings actually correspond to statistically letting one allele per individual per generation mutate so that a smallest possible change in each individual from one generation to the next is provided.

The relative encoding performed better than the absolute encoding. In the relative encoding, a mutation early in the genome is likely to have dramatic change on the phenotype, a feature similar to that of binary coding vs. gray coding [23]. On the other hand, as discussed in section 2.2, a relative encoding introduces a smaller search space. Further research is needed in order to understand why the relative encoding worked better, but we hypothesize that the reason is a better exploration of the search space.

The use of non-coding alleles enables different pipe lengths to evolve from fixed length genotypes. However, a maximum pipe length/complexity is set in advance, determining the size of the genotypic search space, as well as the required time for fitness evaluations. In problems where there is a large variation in the required lengths of pipelines, the suggested approach may be less suitable, as the largest distance defines the overall search space required by also the shorter pipes.

5 Conclusions and Further Work

Three different GAs; Standard GA, Cooperative Coevolutionary GA and Incremental GA were applied to the Three-dimensional Multi-pipe Routing problem, and their performances were compared. Variable length pipelines were built by letting a virtual robot move in space according to evolved, fixed length command lines and allocate pipe segments along its route. Two encodings; a relative and an absolute encoding, were compared. It was found that the approaches taking advantage of the natural problem decomposition, the Cooperative Coevolutionary GA and the Incremental GA, significantly outperformed the Standard GA, and that the relative encoding gave the better results.

This work has mainly focused on evaluating and comparing evolutionary approaches, as well as establishing benchmark problems for future reference. The representation and the operators were totally uninformed, in the sense that a minimum amount of domain knowledge was implemented in the search. In future work we will use an indirect approach, as suggested by Furuholmen et al.

in [24], and [25], in order to learn heuristics from which to build pipelines, and to compare the results using the suggested benchmark problems.

Further work should focus on increasing the realism of the application by developing more complex benchmark problems, by e.g. implementing different pipe diameters, minimize pipe support cost by collocating similar pipelines, minimize maintenance cost by providing accessibility, and addressing strain and thermal expansion during operation. We also suggest implementing a multi objective optimization approach, including the tradeoffs between the different objectives.

Acknowledgement

The authors wish to acknowledge the support of the Norwegian Research Council and the TAIL IO project for their continued funding and support for this research. The TAIL IO project is an international cooperative research project led by StatoilHydro and an R&D consortium consisting of ABB, IBM, Aker Solutions and SKF.

References

1. Eiben, A., Schoenauer, M.: Evolutionary computing. Arxiv preprint cs/0511004 (2005)
2. Oduguwa, V., Tiwari, A., Roy, R.: Evolutionary computing in manufacturing industry: an overview of recent applications. Applied Soft Computing Journal 5(3), 281–299 (2005)
3. Potter, M., Jong, K.: Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. Evolutionary Computation 8(1), 1–29 (2000)
4. Potter, M., De Jong, K.: A Cooperative Coevolutionary Approach to Function Optimization. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) PPSN 1994. LNCS, vol. 866, pp. 249–257. Springer, Heidelberg (1994)
5. Husbands, P., Mill, F.: Simulated co-evolution as the mechanism for emergent planning and scheduling. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 264–270. Morgan Kaufmann Publishers, San Francisco (1991)
6. Neill, C., Laplante, P.: Requirements engineering: The state of the practice. IEEE software, 40–45 (2003)
7. Torresen, J.: Incremental evolution of a signal classification hardware architecture for prosthetic hand control. International Journal of Knowledge-based and Intelligent Engineering Systems 12(3), 187–199 (2008)
8. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Adaptive Behavior 5(3), 317–342 (1997)
9. Garder, L., Høvin, M.: Robot gaits evolved by combining genetic algorithms and binary hill climbing, pp. 1165–1170 (2006)
10. Park, J., Storch, R.: Pipe-routing algorithm development: case study of a ship engine room design. Expert Systems with Applications 23(3), 299–309 (2002)
11. Guirardello, R., Swaney, R.: Optimization of process plant layout with pipe routing. Computers and Chemical Engineering 30(1), 99–114 (2005)
12. Norvig, P., Russell, S.: Artificial intelligence: a modern approach. Prentice-Hall, Englewood Cliffs (2003)

13. Hart, P., Nilsson, N., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics 4(2), 100–107 (1968)
14. Dijkstra, E.: A note on two problems in connexion with graphs. Numerische mathematik 1(1), 269–271 (1959)
15. Rubin, F.: The Lee path connection algorithm. IEEE Transactions on computers 100(23), 907–914 (1974)
16. Hightower, D.: A solution to line-routing problems on the continuous plane. In: Proceedings of the 6th annual conference on Design Automation, pp. 1–24. ACM, New York (1969)
17. Qian, X., Ren, T., Wang, C.: A survey of pipe routing design. In: Control and Decision Conference, CCDC 2008, pp. 3994–3998 (2008) (Chinese)
18. Ito, T.: A genetic algorithm approach to piping route path planning. Journal of Intelligent Manufacturing 10(1), 103–114 (1999)
19. Kim, D., Corne, D., Ross, P.: Industrial plant pipe-route optimisation with genetic algorithms. In: Ebeling, W., Rechenberg, I., Voigt, H.-M., Schwefel, H.-P. (eds.) PPSN 1996. LNCS, vol. 1141, pp. 1012–1021. Springer, Heidelberg (1996)
20. Sandurkar, S., Chen, W.: GAPRUS genetic algorithms based pipe routing using tessellated objects. Computers in Industry 38(3), 209–223 (1999)
21. Wang, H., Zhao, C., Yan, W., Feng, X.: Three-dimensional Multi-pipe Route Optimization Based on Genetic Algorithms. International Federation for Information Processing-publications-IFIP 207, 177 (2006)
22. Baker, J.: Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents, pp. 14–21. L. Erlbaum Associates Inc., Hillsdale (1987)
23. Eiben, A., Smith, J.: Introduction to evolutionary computing. Springer, Heidelberg (2003)
24. Furuholmen, M., Glette, K., Hovin, M., Torresen, J.: Scalability, generalization and coevolution–experimental comparisons applied to automated facility layout planning. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 691–698. ACM, New York (2009)
25. Furuholmen, M., Glette, K., Hovin, M., Torresen, J.: Coevolving Heuristics for The Distributors Pallet Packing Problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (2009)

A Tabu Search Heuristic for Point Coverage, Sink Location, and Data Routing in Wireless Sensor Networks

Evren Güney, İ. Kuban Altınel, Necati Aras, and Cem Ersoy

Boğaziçi University, İstanbul, Türkiye
{evren.guney,altinel,arasn,ersoy}@boun.edu.tr

Abstract. The point coverage, sink location, and data routing problems are considered in an integrated way and two new mixed-integer programming formulations are proposed. As these models are difficult to solve, a nested solution procedure is proposed. The best sensor locations are sought by tabu search in the upper level. For the fixed sensor locations, the remaining problem of determining sink locations and data routes are solved efficiently in the lower level. According to the experimental results performed on a number of test instances, the performance of the nested solution approach is quite satisfactory, and the proposed heuristic method brings considerable improvements over a two-stage solution approach.

Keywords: Wireless sensor networks, mixed-integer linear programming, data routing, point coverage, sink location.

1 Introduction

The increasing interest in Wireless Sensor Networks (WSN) is due to their usability in a wide range of real life applications such as surveillance and border protection, environmental and agricultural monitoring, factory automation, and smart home design [1]. A typical sensor in a WSN has the ability to observe its surroundings to collect data, and also contains an onboard radio to be used for sending the collected data to a base-station either directly or over a multi-hop path [2]. The energy source of a sensor is the limited battery power, and in most of the cases the batteries are not replaceable after the deployment. Thus it is desirable to design a WSN in such a way that its lifetime is as long as possible.

WSN can be homogeneous consisting of identical sensors, or heterogenous consisting of sensors with different technical characteristics and costs. A WSN is deployed over an area of interest which is called the sensor field. The main purpose of a WSN is to cover the sensor field considering certain objectives such as maximizing network lifetime or maximizing coverage quality. This is called the *Coverage Problem* (CP). As the number of sensors in a WSN increases, deploying the sensors effectively becomes a significant problem. Various strategies

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 83–94, 2010.

and techniques are surveyed in [2]. Also an interesting line of research emerged, where the point coverage problem is formulated as a mixed-integer linear program (MILP) [3–5].

In a WSN, each sensor collects and processes data and tries to send this information to a central unit called sink or base station. Since communication ranges of sensors are limited, the data packets usually have to follow multihop paths. The *Routing Problem* (RP), which is concerned with determining the most energy efficient sensor-to-sink routes for a given set of sensor and sink locations, is one of the fundamental problems in WSNs, because data transmission is an energy consuming task. Although sink locations are either fixed or assumed to be known a priori in some studies [6–8], the determination of optimal sink locations is also an important WSN design issue since the total amount of energy necessary to send the data to a sink is a crucial factor that is affected by sink locations. Hence determining optimum sink locations plays an important role for maximizing the network lifetime by decreasing the consumption of the communication energy [9]. Joint optimization of sink locations and routing has also received considerable attention. Gandam et *al.* [10] mentions that using multiple sinks and periodically changing their locations increases the network lifetime. Efrat et *al.* [11] study joint sink location and routing for a WSN with a single moving sink.

In this work we jointly solve the point coverage, sink location, and data routing problems in heterogeneous sensor networks. We refer to this integrated problem as the *Coverage, Location and Routing Problem* (CLRP). We propose two new mathematical programming formulations which consider the three design issues together under a single model. However, these formulations can be solved by commercial solvers only for small-sized problems. Therefore, for medium and large-sized problems we propose a nested heuristic solution procedure. In the upper level of this approach, we try to find good sensor locations that satisfy coverage and budget restrictions using Tabu Search. In the lower level, we solve the remaining sink location and routing problem using efficient methods. To the best of our knowledge, this approach is new in this research area.

The rest of the paper is organized as follows. In the next section, we introduce the new formulations. Solution techniques for CLRP are explained in Section 3, while experimental results are reported in Section 4. We conclude the paper in Section 5.

2 Point Coverage, Sink Location and Data Routing Problem

We propose two different mathematical programming formulations to solve the CLRP. The first formulation represents a network design model where routing energy is arc-based. That is, the energy spent on each arc is proportional to the amount of data flow on that arc. The second formulation is a median model, where the consumed energy is path-based where the sensor-to-sink assignment

cost is equal to the least energy path connecting them. Before getting into the details of the formulations, we mention the underlying scenario in which the WSN operates.

CLRP is defined on a field $\mathcal{N}$ with $|\mathcal{N}| = N$ points. Theoretically, this field may be in any dimension, but for the sake of simplicity, we consider a two dimensional field with $N = m \times n$ points. Any point of $\mathcal{N}$ is a candidate location for a sensor or sink. There exist K different types of sensors, i.e., $|\mathcal{K}| = K$. Sensors of different types have a different cost, sensing and transmission range. The amount of coverage at every point in the sensor field has to be above a certain threshold value. The coverage amount at a point is determined by a probability function of the distances between this point and sensor locations. A common approach is setting the probability of detecting a target at distance d from a sensor to $e^{-\alpha d}$, where parameter α determines the rate at which the detection probability of a sensor decreases with the distance. Sensors belonging to different types have different α values and the ones with higher α are of lower quality. An important assumption is that each sensor's coverage is independent from the others, which helps to easily compute the cumulative coverage amount [5].

The minimum required coverage at any point i is b_i, and parameter a_{jik} denotes the contribution of a type-k sensor at point j to the coverage of point i. Each sensor has a cost of h_k and the available budget for sensor deployment is equal to B. A sink can be installed at any point in the sensor field, and the total number of sinks is fixed at p. We note that a sensor and a sink can be placed at the same point. We define the following decision variables. Binary variables s_{jk} represent the locations of sensors, where $s_{jk} = 1$ if a type-k sensor is located at point $j \in \mathcal{N}$, $s_{jk} = 0$ otherwise. Binary z_j variables denote the sink locations where $z_j = 1$ if a sink is installed at point j, it is zero otherwise.

Both models that will be developed use the parameters and decision variables defined above. The difference arises with respect to variables used for data routing. In the first model, called the Single Commodity Flow Formulation of CLRP (CLRP-1), sensor-to-sensor and sensor-to-sink arcs are defined to represent data routes. The second formulation, referred to as the p-Median Formulation of CLRP (CLRP-2), makes use of binary variables for the assignment of sensors to sinks and another set of variables to represent the arcs used for each assignment.

2.1 Single Commodity Flow Formulation of CLRP

In CLRP-1 two sets of variables are defined to represent the flows. The flow variables, x_{ijkl}, represent the amount of sensor-to-sensor flows between a type-k sensor at point i and a type-l sensor at point j y_{ijk} denote sensor-to-sink flows from a type-k sensor at point i to sink at point j. The flow costs c_{ijk} are computed as $\gamma^k d_{ij}^{\theta}$, where d_{ij} is the distance between points i and j in the sensor field, γ^k is the amount of energy spent for one unit of data flow by a type-k sensor, and θ is the path loss factor which is usually assumed to be between 2 and 4 [12]. An MILP formulation for this model is given below.

CLRP-1:

$$\min \sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{\substack{j\in\mathcal{N}\\ j\neq i}}\sum_{l\in\mathcal{K}} c_{ijk}x_{ijkl} + \sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{N}} c_{ijk}y_{ijk} \tag{1}$$

$$\text{s.t.} \sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} a_{jik}s_{jk} \geq b_i \qquad i\in\mathcal{N} \tag{2}$$

$$\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} h_k s_{jk} \leq B \tag{3}$$

$$\sum_{\substack{j\in\mathcal{N}\\ j\neq i}}\sum_{l\in\mathcal{K}} x_{jilk} + \sum_{\substack{j\in\mathcal{N}\\ j\neq i}} y_{jil} + s_{ik} = \sum_{j\in\mathcal{N}} y_{ijk} + \sum_{\substack{j\in\mathcal{I}\\ j\neq i}}\sum_{k\in\mathcal{K}} x_{ijkl} \qquad i\in\mathcal{N},\, l\in\mathcal{K} \tag{4}$$

$$\sum_{i\in\mathcal{I}}\sum_{k\in\mathcal{K}} y_{ijk} \leq KNz_j \qquad j\in\mathcal{N} \tag{5}$$

$$\sum_{j\in\mathcal{N}} z_j = p \tag{6}$$

$$x_{ijkl}, y_{ijk} \geq 0 \qquad i,\ j\in\mathcal{N};\ j\neq i,\ k\in\mathcal{K},\ l\in\mathcal{K} \tag{7}$$

$$z_j, s_{ik} \in \{0,1\} \qquad i\in\mathcal{N},\ j\in\mathcal{N},\ k\in\mathcal{K} \tag{8}$$

The objective function (1) minimizes the sum of sensor-to-sensor and sensor-to-sink routing costs. Coverage constraints (2) ensure the placement of a sufficient number of sensors to satisfy the desired coverage quality for all points in the sensor field. The budget constraint (3) limits the number of sensors to be installed. Constraints (4) are the flow balance constraints, where the left-hand side of the equation is the sum of three quantities to point i: total inflow from other sensors to a type-k sensor at point i, the total inflow to the sink at point i, and the data generated by a type-k sensor at point i. The right-hand side of the equation is the sum of the total outflow to the sinks or to other sensors from point i. Feasibility constraints (5) ensure that there cannot be a sensor-to-sink flow to point j if there is no sink there. p-median constraint (6) determines the number of sinks to be installed in the sensor field. Finally, constraints (7) and (8) are the non-negativity and binary restrictions on the variables. Observe that there are $O(N^2K^2)$ variables $O(NK)$ constraints in the formulation.

2.2 *P*-Median Formulation of CLRP (CLRP-2)

The formulation of CLRP-2 is in such a way that each type-k sensor at point i is assigned to a sink at point j. We use binary variables x_{ijk} for this purpose. To determine the routing cost of each assignment we need to know the arcs are used in that assignment. Therefore, we introduce binary variable u_{lm}^{ijk} which is one if arc (l,m) is used in the assignment of a type-k sensor at point i to a

sink at point j. Similar to the previous formulation, c_{lmk} represents the flow cost between points l and m when there is a type-k sensor at point l. The mathematical model is given below.

CLRP-2:

$$\min \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}\sum_{l\in\mathcal{N}}\sum_{m\in\mathcal{N}}\sum_{k\in\mathcal{K}} c_{lmk}u_{lm}^{ijk} \tag{9}$$

$$\text{s.t.} \sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} a_{jik}s_{jk} \geq b_i \quad i\in\mathcal{N} \tag{10}$$

$$\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{N}} h_k s_{jk} \leq B \tag{11}$$

$$\sum_{j\in\mathcal{N}} x_{ijk} = s_{ik} \qquad i\in\mathcal{N},\ k\in\mathcal{K} \tag{12}$$

$$x_{ijk} \leq z_j \qquad i,j\in\mathcal{N},\ k\in\mathcal{K} \tag{13}$$

$$\sum_{j\in\mathcal{N}} z_j = p \qquad i,j\in\mathcal{N},\ k\in\mathcal{K} \tag{14}$$

$$\sum_{m\in\mathcal{N}} u_{lm}^{ijk} - \sum_{m\in\mathcal{N}} u_{ml}^{ijk} = \left\{ \begin{array}{l} x_{ijk},\ l=i \\ 0,\ l\in\mathcal{N}\backslash\{i,j\} \\ -x_{ijk},\ l=j \end{array} \right\} \quad i,j,l\in\mathcal{N}, i\neq j,\ k\in\mathcal{K} \tag{15}$$

$$\sum_{m\in\mathcal{N}} u_{lm}^{ijk} \leq s_{lk} \qquad i,j,l\in\mathcal{N}, i\neq j,\ k\in\mathcal{K} \tag{16}$$

$$u_{lm}^{ijk}, x_{ijk} \geq 0 \qquad i,j,l,m\in\mathcal{N},\ k\in\mathcal{K} \tag{17}$$

$$s_{ik},\ z_i \in \{0,1\} \qquad i\in\mathcal{N},\ k\in\mathcal{K} \tag{18}$$

The objective function (9) minimizes the sum of routing costs over all arcs. Coverage constraints (10) and budget constraint (11) are the same as in the previous formulation. Assignment constraints (12) ensure that each sensor is assigned to only one sink. Feasibility constraints (13) prohibit any assignment if there is no sink at point j. This is the stronger version of this constraint set. The weaker version, $\sum_{i\in\mathcal{N}}\sum_{k\in\mathcal{K}} x_{ijk} \leq NKz_j$, has fewer constraints, but it yields a lower linear programming (LP) relaxation. p-median constraint (14) sets the number of sinks to be installed in the sensor field to p. Constraint set (15) are the flow balance constraints. If there is a type-k sensor at point i and a sink at point j, and there is an assignment x_{ijk}, then there must be a unit outflow from the sensor and a unit inflow to the sink. For all other intermediate points, the sum of the total inflow and outflow should be zero. Constraints (16) ensure that an arc beginning at point l cannot be used if there is no sensor at that point. Constraints (17) and (18) are non-negativity and binary restrictions on the variables.

Although we define x_{ijk} and u_{lm}^{ijk} as binary variables, they can be relaxed to be continuous nonnegative variables since they will be equal to either zero

or one at optimality. Given fixed sensor locations, this formulation is reduced to the classical *p-median* problem, which is known to be NP-hard. In CLRP-2 formulation, the number of variables is $O(N^4K)$ and the number of constraints is $O(N^3K)$.

3 Solving CLRP-1 and CLRP-2

Available commercial solvers for MILP problems can find optimal solutions to CLRP-1 and CLRP-2 for only small-sized instances. Since CLRP-1 has much less variables and constraints than CLRP-2, we can obtain optimal solutions for relatively larger problems in less time compared to CLRP-2 using commercial solvers. Nevertheless, as the problem size increases, it becomes very hard to obtain even good solutions in reasonable time for both models. As a result, alternative techniques should be developed.

3.1 A Nested Solution Approach

We propose a nested solution approach to solve CLRP based on the formulation of CLRP-2. In the upper level we place sensors in such a way that coverage and budget restrictions are satisfied. In the lower level, for the fixed sensor locations we solve the remaining sink location and routing problem (LRP-2).

Given a feasible sensor location set $\mathcal{I} \subseteq \mathcal{N}$ with $|\mathcal{I}| = I$ sensors satisfying coverage and budget requirements (i.e., s_{ik} are fixed), CLRP-2 can be re-written in a less complicated fashion since we can drop variables s_{ik} as well as constraints (10) and (11). The resulting formulation LRP-2 is given below:

LRP-2:

$$\min \sum_{i\in\mathcal{I}} \sum_{k\in\mathcal{S}_i} \sum_{j\in\mathcal{N}} g_{ijk} x_{ijk} \tag{19}$$

$$\text{s.t.} \sum_{j\in\mathcal{N}} x_{ijk} = 1 \qquad i \in \mathcal{I},\ k \in S_i \tag{20}$$

$$x_{ijk} \le z_j \qquad i \in \mathcal{I},\ k \in S_i,\ j \in \mathcal{N} \tag{21}$$

$$\sum_{j\in\mathcal{N}} z_j = p \tag{22}$$

$$x_{ijk}, z_j \in \{0,1\} \quad i \in \mathcal{I},\ k \in S_i,\ j \in \mathcal{N} \tag{23}$$

When the sensor locations are known, we can execute a preprocessing step to determine the shortest paths from each sensor to every candidate sink location. The minimum costs obtained for these paths are designated as g_{ijk}. They are simply the sum of the arc costs c_{ijk} on the shortest path between a type-k sensor at point i and a sink at point j. They can be determined by solving a shortest path problem for every i, k and j, which can effectively be carried out using a many-to-many shortest path algorithm such as Floyd-Warshall's algorithm [13]. In other words, in the preprocessing step we consider a complete graph with N nodes over the sensor field $\mathcal{N}$, and determine one-to-one shortest paths between the I nodes of $\mathcal{I}$ and the N nodes of $\mathcal{N}$ before we solve LRP-2.

3.2 Tabu Search to Determine Sensor Locations

Tabu Search (TS) is a metaheuristic algorithm that guides the local search to prevent it from being trapped in premature local optima or cycling [14]. This is achieved by prohibiting the moves that cause to return to previously visited solutions throughout a certain number of iterations. We use TS to make a search in the solution space of feasible sets of sensor locations and try to find one such that when LRP-2 is solved with this set, a near-optimal solution is found for the CLRP-2.

The TS heuristic begins with a feasible sensor set. The basic steps of the algorithm is carried out as long as the termination criterion is not satisfied. At each iteration of TS, r neighbors (feasible sensor sets) are generated from the current solution by using various neighborhood structures (moves). While generating these neighbors, those moves that are in the tabu list are not allowed for certain number of iterations unless the corresponding objective value is better than the objective value of the best solution obtained so far, i.e., the incumbent.

Initial Sensor Locations. We apply an easy and quick strategy to find the initial sensor set. It is based on a greedy heuristic [5], at each step of which a sensor is deployed at the minimum cost without violating the budget constraint so as to reduce the maximum undercoverage level.

Moves of Tabu Search. At each step of TS we create candidate neighborhood sets by using the following Add, Drop, and Swap moves: 1-Add, 2-Add, 1-Drop, 2-Drop, 1-Swap, and 2-Swap. We randomly add (drop) one and two sensors in the 1-Add (1-Drop) and 2-Add (Drop) moves, respectively. In the 1-Swap (2-Swap) move, on the other hand, we randomly drop one sensor (two sensors), and then add one new sensor (two new sensors) so that the total number of sensors is unchanged. If the new sensor set obtained by any move is feasible, the objective value associated with it is computed by solving the corresponding LRP-2 model. In a heterogeneous sensor network, where there exist sensors with different sensing ranges and costs, the above-mentioned moves may cause an infeasibility by violating the coverage and/or budget constraints. When such an infeasibility is encountered, LRP-2 is not solved.

The total number of neighbors generated by all the moves is equal to r. In our experiments we use different r values to work with different neighborhood size that may effect the quality of the solutions obtained by the proposed heuristic.

Determining a Local Minimum. When we are given a feasible set of sensor locations, we can easily reduce CLRP-2 to LRP-2, and evaluate the corresponding objective value by solving it. TS starts with a feasible initial sensor set and at each step a local search around this set is performed. We generate r neighbors (sensor sets) by using the moves described above, and the objective value of each of these r neighbors is computed by solving the corresponding LRP-2 which helps to determine the sink locations and data routes. If we find a better solution than the incumbent, we update our best feasible sensor set. This process continues until one of the two termination criteria is satisfied. These are

the maximum number of total iterations and the maximum number of iterations without any improvement in the best objective value.

While solving LRP-2 we employ two methods: Lagrangean Heuristic (LH) and Nested-Dual Heuristic (NDH). Notice that these are two well-known efficient methods to solve LRP-2 [15]. Since NDH is a faster algorithm compared to LH, it can solve more instances of LRP-2 within the same time interval. As a consequence, we increase the neighborhood size parameter r to $2r$ for NDH. Our experimental analysis shows that increasing the neighborhood size considerably improves the quality of the heuristic solution.

4 Experimental Results

In this section, we report the solution quality and CPU time requirement of our heuristic method. We compare our heuristic results with the solutions generated by a two-stage solution approach. In the first stage of this approach, we solve the point coverage problem to find the optimal sensor locations. Given the sensor locations, in the second stage we optimally solve the data routing and sink location problem. The objective value provided by the two-stage approach is denoted by z_{2S}. It is noteworthy to mention that in both stages a commercial solver is used. We measure the accuracy of the solutions by the percent deviations from z_{2S} according to the formula:

$$100 \times \frac{|z_{2S} - X|}{z_{2S}}, \tag{24}$$

where $X \in \{z_{IP}, z_{LH}, z_{NDH}$. z_{IP} is the objective value obtained by solving CLRP-1 using a commercial solver. z_{LH} and z_{NDH} are the best objective values of our nested approach, which use Lagrangean Heuristic (LH) and Nested Dual Heuristic (NDH) to solve LRP-2, respectively. We also compare the average CPU times of these three solution methods and that of the two-stage approach. We use CPLEX 11.2 [16] integrated with GAMS 2.50 [17] algebraic modeling system as our commercial solver.

Our test bed consists of 24 test problems which are generated for sensor fields consisting of $n \times n$ square grids with $n \in \{5\text{–}15, 20\}$. This means that there is a total of $N = n^2$ points in the grid. We consider an open area without many obstacles and choose a path loss factor of $\theta = 2$. In addition, we use the Euclidean distance as the distance measure. All experiments are carried out on a Dell PowerEdge 2400 computer with two 64-bit, 2.66-GHz Xeon 5355 Quad Core processors and 28GB memory.

We only have routing costs in the objective function and they are directly related to the energy spent per unit data flow. A sensor typically spends energy for sensing, processing and transmission. We consider only the transmission energy because it is on several magnitudes higher than the others. A typical AA type of battery contains 2200 mAh of energy and to transmit one packet of data 10 nAh energy is spent for unit distance [18]. Therefore, we assume connection weights are simply equal to the energy spent per unit data flow, and take $\gamma = 10$ nAh. We

consider two different types of sensors with different costs: $h_1 = 10 and h_2 = 20$. The coverage parameters are $\alpha_1 = 0.5$ and $\alpha_2 = 0.4$, and energy consumption parameters are $\gamma^1 = 10$ nAh and $\gamma^2 = 20$ nAh for the two types of sensors. The minimum coverage requirement threshold parameter $b_i = 0.99$ and is equal for all i.

There is no fixed cost associated with sink installation and the number of sinks is an input parameter. First, we consider the case with one sink ($p = 1$). We solve each test problem three times and report the best results in Table 1. The first column includes the number of points in the sensor field N for each test problem. The second column shows the amount of energy spent in the network that is computed by the two-stage approach. The third column displays the objective value obtained by solving the CLRP-2 formulation using CPLEX. We set a time limit of four hours and CPLEX cannot find the optimal solutions within this period for the problems that have $N \geq 100$ points, which is represented by an "*". The fourth and fifth columns contain the results obtained by our nested solution method when CLRP-2′ is solved by Lagrangean Heuristic and Nested Dual Heuristic, respectively. The last three columns give the percent energy improvements provided by our solution methods with respect to the two-stage solution approach. On the average, 74.3%, 71.2% and 72.1% less energy is spent for data routing when the WSN is operated using the deployment strategies obtained by our methods. This implies that the lifetime of the WSN can be increased considerably.

The results for $p = 2$ are displayed in Table 2. It can be observed that 69.9%, 65.5% and 68.8% less energy is spent on the average when the WSN is constructed by each of our solution methods, respectively. As is the case with $p = 1$, CPLEX cannot find the optimal solutions within the allowed time of four hours when solving CLRP-2 for problem instances with $N \geq 100$ points. We compare the CPU performances of the two versions of our TS heuristic (i.e., LH and

Table 1. Comparison of the Solution Methods when $p = 1$

	Routing Energy (nAh)				% Improvement		
N	z_{2S}	z_{IP}	z_{LH}	z_{NDH}	z_{IP}	z_{LH}	z_{NDH}
25	480	200	200	200	58.3	58.3	58.3
36	1520	370	370	370	75.7	75.7	75.7
49	2230	500	610	570	77.6	72.6	74.4
64	3120	760	890	810	75.6	71.5	74.0
81	4680	1020	1110	1110	78.2	76.3	76.3
100*	6550	1440	1660	1590	78.0	74.7	75.7
121*	9350	1910	2050	2070	79.6	78.1	77.9
144*	9780	2510	2920	2640	74.3	70.1	73.0
169*	12680	3240	3520	3340	74.4	72.2	73.7
196*	16580	3960	4580	4640	76.1	72.4	72.0
225*	19000	47900	5720	5640	74.8	69.9	70.3
400*	38440	12110	14240	13990	68.5	63.0	63.6
Avg.	10367.5	2734.2	3154.2	3080.8	74.3	71.2	72.1

Table 2. Comparison of the Solution Methods when $p = 2$

	Routing Energy (nAh)				% Improvement		
N	z_{2S}	z_{IP}	z_{LH}	z_{NDH}	z_{IP}	z_{LH}	z_{NDH}
25	330	130	130	130	60.6	60.6	60.6
36	700	230	240	230	67.1	65.7	67.1
49	1160	330	380	330	71.6	67.2	71.6
64	1610	500	560	500	68.9	65.2	68.9
81	2420	670	870	760	72.3	64.0	68.6
100*	3560	960	1120	1050	73.0	68.5	70.5
121*	5010	1300	1520	1450	74.1	69.7	71.1
144*	6260	1670	2090	1880	73.3	66.6	70.0
169*	8040	2140	2580	2250	73.4	67.9	72.0
196*	9280	2610	3320	2810	71.9	64.2	69.7
225*	11500	3310	4120	3650	71.2	64.2	68.3
400*	27260	10660	10250	8850	60.9	62.4	67.5
Avg.	6427.5	2042.5	2265.0	1990.8	69.9	65.5	68.8

Table 3. Comparison of CPU Times

	$p = 1$				$p = 2$			
N	2S	IP	LH	NDH	2S	IP	LH	NDH
25	1.3	2.2	54.0	21.0	1.8	13.2	44.7	11.1
36	142	7.8	91.9	46.5	243.0	168.8	82.7	25.3
49	4372	31.5	119.3	88.9	6118.0	407.8	188.2	52.6
64	18000	341.2	182.1	161.1	14400.0	1443.5	339.6	131.0
81	18000	344.4	391.0	323.2	14400.0	3694.7	450.1	262.5
100	18000	14400.0	979.2	874.0	14400.0	14400.0	673.4	547.6
121	18000	14400.0	1330.6	1736.7	14400.0	14400.0	1252.5	741.5
144	18000	14400.0	1783.4	3667.5	14400.0	14400.0	1326.6	1263.6
169	18000	14400.0	3994.9	8556.2	14400.0	14400.0	2369.7	2463.0
196	18000	14400.0	5520.2	14400.0	14400.0	14400.0	3730.0	3653.7
225	18000	14400.0	7756.8	14400.0	14400.0	14400.0	4503.8	4546.0
400	18000	14400.0	14400.0	14400.0	14400.0	14400.0	14400.0	14400.0
Avg.	11176.3	8460.6	3050.3	4889.6	11330.2	8877.3	2446.8	2341.5

NDH) with that of the two-stage solution approach. The results are displayed in Table 3. For both $p = 1$ and $p = 2$, LH and NDH are faster than the other two methods for most of the instances. The exceptions are the smallest three problems with $N = 25, 36, 49$ for $p = 1$ and the smallest two problems with $N = 25, 36$ for $p = 2$. Solving CLRP-2 as an MILP by CPLEX requires on the average three to four times as much time as needed by NDH and LH.

When we compare NDH and LH, NDH performs better especially when $p = 2$. This behavior is expected because of two reasons. First, NDH is a specially tailored method to solve the p-median type of problems. Second, the neighborhood size of NDH is twice as large as that of LH.

Table 4. Effect of b_i on the Performance of the Methods

	$b_i = 0.99$				$b_i = 0.90$			
N	2S	IP	LH	NDH	2S	IP	LH	NDH
25	480	58.3	58.3	58.3	340	82.4	82.4	82.4
36	1520	75.7	75.7	75.7	790	82.3	82.3	82.3
49	2230	77.6	72.6	74.4	980	83.7	83.7	83.7
64	3120	75.6	71.5	74.0	1320	83.3	81.8	82.6
81	4680	78.2	76.3	76.3	1860	81.2	79.6	80.6
100	6550	78.0	74.7	75.7	3520	82.7	81.3	82.1
Avg.		73.9	71.5	72.4		82.6	81.8	82.3

Finally, we examine the effect of the deployment density of the sensors on the total energy spent. This is achieved by changing the minimum coverage requirement b_i. In the new set of experiments, we reduce b_i from 0.99 to 0.90, and compare the results with those obtained with $b_i = 0.99$ when $p = 1$. As can be observed in Table 4, the total energy spent decreases when $b_i = 0.90$ since the number of deployed sensors is reduced. According to the preliminary results, the performance of our solution method has increased, which implies that solving CLRP in an integrated way in WSNs with a smaller density has more potential for increasing the network lifetime.

5 Conclusions

In this work, we study the joint optimization of point coverage, sink location, and data routing problems in wireless sensor networks. Two new mixed-integer linear programming formulations are proposed. The first one (CLRP-1) involves data routing variables defined on arcs, while the second one (CLRP-2) uses data routing variables based on sensor-to-sink assignments. As the solution of these formulations using commercial solvers is inefficient for even small-sized problems, we develop a nested solution procedure based on Tabu search metaheuristic using the CLRP-2 formulation. The best sensor locations are sought by tabu search, and for the fixed locations, a reduced model of CLRP-2 is solved by Lagrangean Heuristic or Nested Dual Heuristic to give the sink locations and data routes. Experimental results show that the nested heuristic can produce better solutions than a two-stage solution approach within the same amount of computation time.

Acknowledgments. This research has been supported by TÜBİTAK Grant no: 107M250 and Boğaziçi University Research Fund Grant no: 05HA303.

References

1. Yick, J., Mukherjee, B., Ghosal, D.: Wireless Sensor Network Survey. Comput. Netw. 52, 2292–2330 (2008)
2. Younis, M., Akkaya, K.: Strategies and Techniques for Node Placement in Wireless Sensor Networks: A Survey. Ad Hoc. Netw. 6, 621–655 (2008)

3. Dhillon, S.S., Chakrabarty, K., Iyengar, S.S.: Sensor Placement for Grid Coverage under Imprecise Detections. In: Proc. International Conference on Information Fusion, pp. 1581–1587 (2002)
4. Cardei, M., Thai, M., Li, Y., Wu, W.: Energy-Efficient Target Coverage in Wireless Sensor Networks. In: IEEE INFOCOM 2005, Miami, USA (2005)
5. Altınel, İ.K., Aras, N., Güney, E., Ersoy, C.: Binary Integer Programming Formulations and Heuristics for Differentiated Coverage in Heterogeneous Sensor Networks. Computer Networks 52, 2419–2431 (2008)
6. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: Energy-efficient Communication Protocol for Wireless Microsensor Networks. In: Proc. International Conference on System Sciences (2000)
7. Kim, H., Seok, Y., Choi, N., Choi, Y., Kwon, T.: Optimal Multi-sink Positioning and Energy-Efficient Routing in Wireless Sensor Networks. In: Kim, C. (ed.) ICOIN 2005. LNCS, vol. 3391, pp. 264–275. Springer, Heidelberg (2005)
8. Lin, L., Shroff, N.B., Srikant, R.: Energy-aware Routing in Sensor Networks: A Large System Approach. Ad Hoc Netw. 5, 818–831 (2007)
9. Ciciriello, P., Mottola, L., Picco, G.P.: Efficient Routing from Multiple Sources to Multiple Sinks in Wireless Sensor Networks. In: Langendoen, K.G., Voigt, T. (eds.) EWSN 2007. LNCS, vol. 4373, pp. 34–50. Springer, Heidelberg (2007)
10. Gandam, S.R., Dawande, M., Prakash, R., Venkatesan, S.: Energy Efficient Schemes For Wireless Sensor Networks With Multiple Mobile Base Stations. In: Proceedings of the IEEE Global Telecommunications Conference (GLOBECOM 2003), pp. 377–381 (2003)
11. Efrat, A., Har-Peled, S., Mitchell, J.: Approximation Algorithms For Location Problems in Sensor Networks. In: Proceedings of the IEEE/Creat-Net BROADNETS–Wireless Networking Symposium, pp. 767–776 (2005)
12. Callaway, E.H.: Wireless Sensor Networks: Architectures and Protocols. Auerbach Publications, Boca Raton (2004)
13. Floyd, R.W.: Algorithm 97: Shortest Path. Commun. ACM 5, 345 (1962)
14. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Dordrecht (1997)
15. Güney, E., Altınel, İ.K., Aras, N., Ersoy, C.: Efficient Integer Programming Formulations for Optimum Sink Location and Routing in Wireless Sensor Networks. In: ISCIS, Istanbul, Turkey (2008)
16. CPLEX 11.0 User's Manual, ILOG (2008)
17. GAMS 2.50 User's Manual, GAMS Development Co. (2006)
18. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, S., Anderson, J.: Wireless Sensor Networks for Habitat Monitoring. In: Proc. WSNA 2002, Atlanta, Georgia (2002)

Ant Colony Optimization for Tree Decompositions

Thomas Hammerl and Nysret Musliu

Institute of Information Systems, Vienna University of Technology, Austria
thomas.hammerl@gmail.com, musliu@dbai.tuwien.ac.at

Abstract. Instances of constraint satisfaction problems can be solved efficiently if they are representable as a tree decomposition of small width. Unfortunately, the task of finding a decomposition of minimum width is NP-complete itself. Therefore, several heuristic and metaheuristic methods have been developed for this problem. In this paper we investigate the application of different variants of Ant Colony Optimization algorithms for the generation of tree decompositions. Furthermore, we extend these implementations with two local search methods and we compare two heuristics that guide the ACO algorithms. Our computational results for selected instances of the DIMACS graph coloring library show that the ACO metaheuristic gives results comparable to those of other decomposition methods such as branch and bound and tabu search for many problem instances. One of the proposed algorithms was even able to improve the best known upper bound for one problem instance. Nonetheless, for some larger problems the best existing methods outperform our algorithms.

1 Introduction

Many constraint satisfaction problems (CSPs) can be solved efficiently if they have a tree decomposition of small width. Each tree decomposition has a characteristic called *width* and each CSP problem can be transformed to many different valid decompositions. The smaller a decomposition's width the faster the solution to the problem can be computed.

To illustrate the application of tree decomposition for solving CSP problems suppose that we have to find solutions for the the graph coloring problem (GCP), which is a well known CSP in the literature. For this problem we have to find a coloring of vertices of a given graph in such a way that no two vertices connected by an edge share the same color. An instance of the GCP is shown on the left side of Figure 1. The task is now to find a valid coloring just using the colors red, green, and blue.

One naive approach to solve this problem might be to try out all possible combinations of variable assignments and see which ones are valid. There are d^n possible combinations in general where d is the number of available colors and n is the number of vertices.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 95–106, 2010.

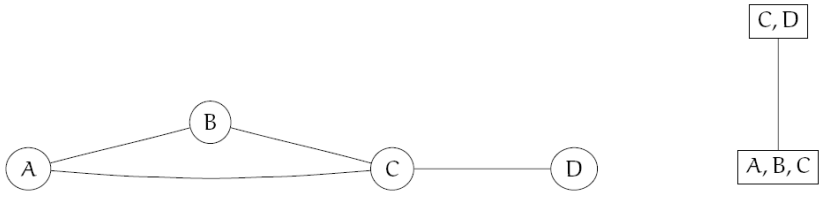


Fig. 1. Instance of the graph coloring problem and one possible tree decomposition

To solve this problem by tree decomposition, we should first generate the tree decompositon of the corrosponding problem graph. The concept of tree decomposition has been introduced first by Robertson and Seymour [14]:

Definition 1. *(see [14], [10]) Let $G = (V, E)$ be a graph. A tree decomposition of G is a pair (T, χ), where $T = (I, F)$ is a tree with node set I and edge set F, and $\chi = \{\chi_i : i \in I\}$ is a family of subsets of V, one for each node of T, such that*

1. $\bigcup_{i \in I} \chi_i = V$,
2. *for every edge $(v, w) \in E$, there is an $i \in I$ with $v \in \chi_i$ and $w \in \chi_i$, and*
3. *for all $i, j, k \in I$, if j is on the path from i to k in T, then $\chi_i \cap \chi_k \subseteq \chi_j$.*

The width of a tree decomposition is $max_{i \in I} |\chi_i| - 1$. The treewidth of a graph G, denoted by $tw(G)$, is the minimum width over all possible tree decompositions of G.

One possible tree decomposition for our graph coloring problem is shown on the right side of Figure 1. This tree decomposition fulfills all conditions of the previous definition. If we want to solve the graph coloring problem based on this tree decomposition, we can start out by solving the subproblems given by each vertex in the tree decomposition. Using our naive approach of trying out all possible combinations of variable assignments we generate 3^3 (27) different solution candidates for the vertex containing A, B, and C. Because of the constraints $A \neq B$, $A \neq C$, and $B \neq C$ only six of them are valid. For the subproblem containing the vertices C and D we generate 3^2 (9) solution candidates and rule out three of them because of the constraint $C \neq D$. We can now get all solutions to the whole problem by joining the subproblem solutions. Therefore, we will take a look at the variables both subproblems have in common. In this case, that is the variable C. Each solution for the subproblem A, B, C is joined with the solutions for the subproblem C, D sharing the same color for the vertex C. By using the tree decomposition we had to generate 36 combinations of variable assignments in order to determine all solutions compared to the 81 combinations we had to generate without the tree decomposition. This difference increases very quickly with the size of the graph coloring problem and constraint satisfaction problems in general. The smaller the subproblems in the tree decomposition the more efficient we can solve a particular problem. This is why we are interested in finding tree decompositions of small *width*.

Tree decompositions can be generated from a given graph by successive elimination of graph vertices. Each time a vertex is eliminated a new tree node is created that contains the eliminated vertex and its neighbors. Additionally, the neighbors of the eliminated node are connected in the remaining graph. It is guaranteed that there is a so-called *optimal elimination ordering* that yields the tree decomposition with the minimum width of all valid tree decompositions for the given constraint graph. Therefore, one way to generate a tree decomposition of small width is to search for a good ordering of the vertices of the graph. Unfortunately, there are $n!$ different elimination orderings. For that reason not only exact methods but also many approximation algorithms have been applied to the problem of finding tree decompositions of small width.

A complete algorithm for tree decompositions is proposed by Gogate and Dechter [7]. This algorithm applies different pruning techniques, and provides anytime solutions, which are good upper bounds for tree decompositions. Heuristic techniques for the generation of tree decompositions with small width are mainly based on searching for a good elimination ordering of graph nodes. Several heuristics that run in polynomial time have been proposed for finding a good elimination ordering of nodes. These heuristics select the ordering of nodes based on different criteria, such as the degree of the nodes (min degree heuristic), the number of edges to be added to make the node simplicial (min-fill heuristic), connectivity with the vertices previously selected in the elimination ordering (Maximum Cardinality Search (MCS) [17]) etc. For other types of heuristics based on the elimination ordering of nodes see [10].

Metaheuristic approaches have also been used for tree decompositions. Simulated annealing was applied by Kjaerulff [9]. Applications of genetic algorithms for tree decompositions are presented in [11] and [13]. A tabu search approach for generation of the tree decompositions has been proposed by Clautiaux et al [2]. The authors reported good results for DIMACS vertex coloring instances. Their approach improved the previous results in literature for 53% of instances. Some of the results in [2] have been further improved by Gogate and Dechter [7]. Upper bounds for several other problems have been improved by iterated local search algorithm [12].

Ant Colony Optimization (ACO) has not been applied yet for tree decompositions. In this paper we investigate the following variants of ACO algorithms for finding tree decompositions of small width: Simple Ant System ([3], [6]), Elitist Ant System ([3], [6]), Rank-Based Ant System [1], Max-Min Ant System ([15], [16]), and Ant Colony System [4]. We propose two different pheromone update strategies and implement two stagnation measures that indicate the degree of diversity of the solutions constructed by the ants. Furthermore, we implement two constructive heuristics (Min-Degree, Min-Fill) that can be incorporated alternatively into every ACO variant as a guiding function and investigated the combination of ACO with two existing local search methods: Hill Climbing and Iterated Local Search [12]. Finally we compare the results achieved by Ant Colony System for 62 DIMACS graph coloring instances with the results of other state of the art heuristic and exact algorithms.

2 Generation of Tree Decompositions by Ant Colony Optimization

Ant Colony Optimization (ACO) is a population-based metaheuristic introduced by Marco Dorigo [3] in 1992. As the name suggests the technique was inspired by the behaviour of "real" ants. Ant colonies are able to find the shortest path between their nest and a food source just by depositing and reacting to pheromones while they are exploring their environment. The basic principles driving this system can also be applied to many combinatorial optimization problems. For a detailed description of different ACO algorithms and their applications the reader is referred to the book *"Ant Colony Optimization"* [5] written by Dorigo and Stützle.

In this section we propose the application of ACO to the problem of finding tree decompositions of small width. We have implemented different ACO variants and combined these variants with different guiding heuristics, local search methods and pheromone update strategies that we will discuss after giving an explanation of the basic structure of the algorithm.

A simple constraint graph and the corresponding ACO construction tree are shown in Figure 2. The construction tree can be obtained from the constraint graph as follows: (1) Create a root node s that will be the starting point of every ant in the colony; (2) For every vertex of the constraint graph append a child node to the root node s; (3) To every leaf node append a child node for every vertex of the constraint graph that is neither represented by the leaf node itself nor by an ancestor of this node; (4) Repeat step 3 until there are no nodes left to append.

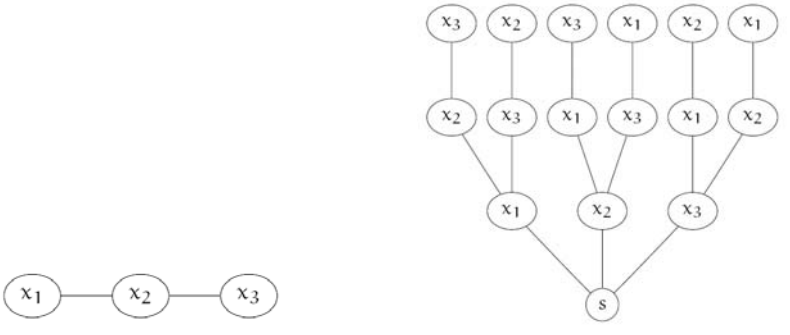


Fig. 2. Constraint graph $\mathcal{G}$ and the ACO construction tree

All possible elimination orderings for the constraint graph can now be represented as a path from the root node s to one of the leaf nodes in the construction tree. Therefore each of the ants finds such a path and at each node on its way the ant decides where to move next probabilistically based on the pheromone trails and a heuristic value both associated with the outgoing edges.

2.1 Pheromone Trails

A pheromone trail constitutes the desirability to eliminate a certain vertex x after another vertex y. The more pheromone is located on a trail the more likely the corresponding vertex will be chosen by the ant. A way to represent the pheromone trails of our construction tree is the matrix as shown below:

$$\mathcal{T} = \begin{pmatrix} \tau_{x_1x_1} & \tau_{x_1x_2} & \tau_{x_1x_3} \\ \tau_{x_2x_1} & \tau_{x_2x_2} & \tau_{x_2x_3} \\ \tau_{x_3x_1} & \tau_{x_3x_2} & \tau_{x_3x_3} \\ \tau_{sx_1} & \tau_{sx_2} & \tau_{sx_3} \end{pmatrix} \tag{1}$$

Each row contains the amounts of pheromone located on the trails connecting a certain node with all the other nodes. For example, the first row contains the pheromone levels related to the node x_1 describing the desirability of eliminating x_2 ($\tau_{x_1x_2}$) respectively x_3 ($\tau_{x_1x_3}$) immediately after x_1. The last row is dedicated to the root node s that is the starting point for every ant.

All pheromone trails are initialized to the same value in the beginning of the algorithm that is computed according to the following equation:

$$\tau_{ij} = \frac{m}{W_\eta} \qquad \forall \tau_{ij} \in \mathcal{T} \tag{2}$$

W_η is the width of the decomposition obtained using the guiding heuristic (min-degree or min-fill) while m is the size of the ant colony.

2.2 Heuristic Information

The ants make their decision which vertex to eliminate next not solely based on the pheromone matrix but also consider a guiding heuristic. We have implemented two different heuristics. In order to compute both of these heuristic values we need to maintain a separate graph in addition to the construction tree. We will call this graph the *elimination graph* because this graph is obtained from the original constraint graph by successively eliminating the vertices traversed by the ant in the construction tree. Further, we will denote this graph as $E(\mathcal{G}, \sigma)$ where $\mathcal{G}$ is the original constraint graph and σ is a partial elimination ordering.

Min-Degree. The value for the min-degree heuristic is computed according to this equation:

$$\eta_{ij} = \frac{1}{d(j, E(\mathcal{G}, \sigma)) + 1} \tag{3}$$

The expression $d(j, E(\mathcal{G}, \sigma)$ represents the degree of vertex j in the elimination graph $E(\mathcal{G}, \sigma)$.

Min-Fill. The value for the min-fill heuristic is computed according to this equation:

$$\eta_{ij} = \frac{1}{f(j, E(\mathcal{G}, \sigma)) + 1} \tag{4}$$

The expression $f(j, E(\mathcal{G}, \sigma)$ represents the number of edges that would be added to the elimination graph due to the elimination of vertex j.

2.3 Probabilistic Vertex Elimination

We will now take a more detailed look on how exactly the ants move from node to node on the construction tree. All of the ACO variants with the exception of Ant Colony System use Equation 5 alone to compute the probability p_{ij} of moving from a node i to another node j where α and β are parameters that can be passed to the algorithm in order to weight the pheromone trails and the heuristic values.

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha}\,[\eta_{ij}]^{\beta}}{\sum_{l \in E(\mathcal{G},\sigma)} [\tau_{il}]^{\alpha}\,[\eta_{il}]^{\beta}}, \qquad \text{if } j \in E(\mathcal{G},\sigma) \tag{5}$$

This probability is computed for each vertex left in the elimination graph. According to these probabilities the ant decides which vertex to eliminate next.

Ant Colony System introduces an additional parameter q_0 that constitutes the probability that the ant moves to the "best" node instead of making a probabilistic decision:

$$j = \begin{cases} \arg\max_{l \in E(\mathcal{G},\sigma)} \{[\tau_{il}]^{\alpha}[\eta_{il}]^{\beta}\}, \text{ if } q \leq q_0; \\ \text{Equation 5, otherwise;} \end{cases} \tag{6}$$

If a randomly generated number q in the interval of $[0, 1]$ is less or equal q_0 then the ant moves to the node that otherwise would have the highest probability to be chosen. Ties are broken randomly.

Ant Colony System also introduces a so-called local pheromone update. After an ant has constructed its solution it removes pheromone from the trails belonging to its solution according to the following equation whereas ξ is a variant-specific parameter and τ_0 is initial amount of pheromone:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0 \tag{7}$$

The motivation for this is to diversify the search so that subsequent ants will more likely choose other branches of the construction tree.

2.4 Pheromone Update

After each of the ants has constructed an elimination ordering (that optionally has been improved by a local search thereafter) the values in the pheromone matrix are updated reflecting the quality of the constructed solutions which will enable the subsequent ants in the following iteration to make decisions in a more informed manner. Moreover, pheromone is removed from the pheromone trails so poor solutions can be forgotten that might have been the best known solutions in earlier iterations of the algorithm.

Pheromone Deposition. Given an elimination ordering σ_k that was constructed by an ant k we need to determine for each subsequent elimination (i, j)

in σ_k the amount of pheromone that will be deposited on the corresponding pheromone trail τ_{ij}. We implemented an edge-independent and an edge-specific pheromone update strategy. The first adds the same amount of pheromone to all trails belonging to σ_k while the latter adds more or less pheromone to individual trails depending on the quality of a certain elimination.

The edge-independent pheromone update strategy adds the reciprocal value of the tree decomposition's width to all pheromone trails that are part of σ_k:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{W(\sigma_k)}, \text{ if } (i,j) \text{ belongs to } \sigma_k; \\ 0, \text{ otherwise;} \end{cases} \tag{8}$$

In contrast to the edge-independent update strategy the edge-specific update strategy deposits different amounts of pheromone onto the trails belonging to the same elimination ordering:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{d(j,E(\mathcal{G},\sigma_{kj}))/|E(\mathcal{G},\sigma_{kj})|} \cdot \frac{1}{W(\sigma_k)}, \text{ if } (i,j) \text{ belongs to } \sigma_k; \\ 0, \text{ otherwise;} \end{cases} \tag{9}$$

This amount depends on the ratio between the degree of the vertex j when it was eliminated $d(j, E(\mathcal{G}, \sigma_{kj}))$ and the number of vertices left in the elimination graph $|E(\mathcal{G}, \sigma_{kj})|$ at that time.[1]

Which ants are allowed to deposit pheromone and how this pheromone is weighted varies between the different ACO variants. The reader is referred to [5] for description of these variants.

Pheromone Evaporation. After the pheromone has been added to the trails a certain amount of pheromone is removed. This amount is determined based on the pheromone evaporation rate ρ:

$$\tau_{ij} = (1-\rho)\tau_{ij} \qquad \forall\tau_{ij} \in \mathcal{T} \tag{10}$$

While all the other ACO variants remove pheromone from every pheromone trail Ant Colony System only removes pheromone from the trails belonging to the best known elimination ordering σ_{bs}:

$$\tau_{ij} = (1-\rho)\tau_{ij} \qquad \forall(i,j) \in \sigma_{bs} \tag{11}$$

2.5 Local Search

All ACO variants can optionally be extended with one of two local search methods we implemented for tree decompositions. Both of these algorithms try to improve the quality of the solutions that were constructed by the ant colony by changing the positions of certain vertices in the elimination orderings. We used two local search techniques in this paper: an hill climbing algorithm and an iterated local search similar to the algorithm proposed by Musliu [12]. Both of these algorithms are discussed in detail in Section 5.4 of the master's thesis [8] this paper is based on.

[1] σ_{kj} is the partial elimination ordering that is obtained from σ_k by omitting j and all vertices that are eliminated after j.

2.6 Stagnation Measures

If the distribution of the pheromone on the trails becomes too unbalanced due to the pheromone depositions, the ants will generate very similar solutions causing the search to stagnate. In order to enable the algorithm to detect such situations we have implemented two stagnation measures (Variation Coefficient and $\overline{\lambda}$ Branching Factor) proposed by Dorigo and Stützle [5] that indicate how explorative the search behaviour of the ants is. A detailed description of stagnation measures is given in [8] (page 67).

3 Computational Results

In order to evaluate and compare the performance of the different ACO algorithms, a series of experiments were performed. All of these experiments were performed for the ten representative instances of the DIMACS Graph Coloring Challenge.

We experimented with variant-independent parameters and parameters of each ACO variant. After setting of all parameters to values that were obtained through trial and error the following experiments were performed (time-limit of 200 seconds was set for each run):

- $[(\alpha, \beta), \ldots] = [(1, 10), (2, 20), (3, 30), (5, 40), (2, 50)]$: the combination of $\alpha = 2$ and $\beta = 50$ outperformed the other combinations (considering best average width over five runs) in 27 of 50 experiments.
- Guiding Heuristics: we compared min-fill and min-degree heuristics. The results clearly indicated that the min-fill heuristic gives better results. Nonetheless, the min-degree heuristic is much more time-efficient. For instance, the ACO algorithms were only able to complete one iteration within 200 seconds for the problem instance le450_5a using the min-fill heuristic while 144 iterations could be performed using the min-degree heuristic. This is why we decided to use the min-degree heuristic for all remaining experiments since we would otherwise be unable to investigate the impact of the pheromone trails on the search behaviour of the ants due to the small number of iterations.
- Number of ants: 5, 10, 20, 50, 100. Best results were obtained by using ant colonies consisting of 100 ants for Simple and Elitist Ant System (SES, EAS), 50 ants for Rank-Based Ant System(RAS), 20 ants for Max-Min Ant System (MAS) and five ants for Ant Colony System (ACS).
- Weight e for Elitist Ant System: 2, 4, 6 and 10. Elitist weight of 10 gave best results.
- Number of ants w to deposit pheromone in every iteration in Rank-Based Ant System: 3, 5, 7 and 10. Best results were obtained with $w = 10$.
- Max-Min Ant System: $[(a, f), \ldots] = [(10, 2), (3, 5), (10, 5), (3, 2)]$. $(3, 5)$ parameter combination gave best results.
- Ant Colony System: $[(q_0, \xi), \ldots] = [(0.1, 0.3), (0.5, 0.05), (0.1, 0.05), (0.5, 0.3)]$. $(0.5, 0.3)$ gives the best results among all of these combinations.

Note that we initialize τ_0 the pheromone trails to m/W_η because according to [5] it is a good heuristic to initialize the pheromone trails to a value that is slightly higher than the expected amount of pheromone deposited by the ants in one iteration. Additionally we set the pheromone evaporation rate ρ to 0.1 for our experiments because that also worked well for the travelling salesman problem according to [5].

After we had found good parameter settings for each ACO variant we were now ready to compare them. Therefore, five runs were performed by each variant for every instance of the experimental set with $\alpha = 2$, $\beta = 50$, $\rho = 0.1$ and a time-limit of 500 seconds. Min-degree was used as the guiding heuristic. Additionally, the parameter settings for each variant were applied based on the results of the prior experiments.

Table 1. Comparison of minimum and average widths achieved by all ACO variants over 5 runs. Given in bold are those values that represent single-best solutions among all variants.

Instance	Minimum Width					Average Width				
	SAS	EAS	RAS	MAS	ACS	SAS	EAS	RAS	MAS	ACS
DSJC125.1	65	65	65	64	**63**	65.6	65.4	65.4	64	**63.8**
games120	37	38	38	37	37	38.8	38.8	38.6	37.4	**37**
le450_5a	311	312	**303**	308	309	313.6	314	**310.2**	312.8	311.4
le450_5b	313	314	311	**307**	312	313.6	315.8	314.8	**313.2**	313.4
miles500	25	25	25	25	25	25.4	25.2	25.8	**25**	25.2
myciel6	35	35	35	35	35	35	35	35	35	35
myciel7	68	68	69	68	69	68.8	68.8	69	68.8	69
queen12_12	114	113	112	112	**111**	114.6	114	114.4	113.2	**112**
queen8_8	48	48	48	47	47	48	48	48.2	47	47
school1	237	231	232	**228**	232	238	235.2	235.4	233.4	**233.2**

Table 1 lists the minimum and the average width achieved by each ACO variant for each problem instance. According to these results Max-Min Ant System and Ant Colony System performed slightly better than the other variants. Only once, for the problem instance le450_5a, Rank-Based Ant System was able to achieve better results than Max-Min Ant System and Ant Colony System. For all other problem instances one of these two variants delivered the best minimum and average width. Since Ant Colony System more often gave the single best solution among all variants, we decided to focus our remaining investigations on this ACO variant.

In Section 2.4 we presented two different pheromone update strategies. In order to compare them we have applied Ant Colony System with each of them. The edge-specific pheromone update strategy gave slightly better results than the edge-independent strategy.

Our final experiments dealt with the combination of Ant Colony System with the iterated local search and the hill climbing algorithm. Ant Colony System in combination with the iterated local search clearly outperformed the hybrid algorithm consisting of Ant Colony System and the hill climbing local search. ACS+HC was only able to give better results than ACS+ILS for two out of the ten problem instances (see page 91 in [8]).

4 Comparison with the Results in the Literature

Based on the results of our experiments we compared our Ant Colony System (ACS) variants with the results from the literature for 62 well known DIMACS graph coloring instances. All results reported in this paper have been obtained on a machine equipped with 48GB of memory and two Intel(R) Xeon(R) CPUs (E5345) having a clock rate of 2.33GHz. We performed 10 runs for each example with our ACS and ACS+ILS algorithms. Each run was performed with a time-limit of 1000 seconds. The best results from a set of algorithms proposed by Koster, Bodlaender and Hoesel in [10] (KBH) were obtained with a Pentium 3 800MHz processor. Results of Tabu Search (TabuS) algorithm proposed in [2] were obtained with a Pentium 3 1 GHz processor. Experiments for the branch and bound (BB) algorithm presented by Gogate and Dechter in [7] were performed on a Pentium 4 2.4 GHz processor using 2 GB of memory. The results with the genetic algorithm (GA) in [13] were obtained in a Intel(R) Xeon(TM) 3.2 GHz processor and 4 GB of memory. Results of iterative heuristic algorithm (IHA) [12] were obtained with a Pentium 4 processor 3 GHz and 1 GB of RAM.

Figure 3 visualizes for how many problem instances ACS respectively ACS+ILS gave a better, equal or worse minimum width compared (regarding the best found solution) with each of the other decomposition methods . As can be seen, both algorithms outperformed KBH on more instances than vice versa but only ACS+ILS also managed to outperform BB.

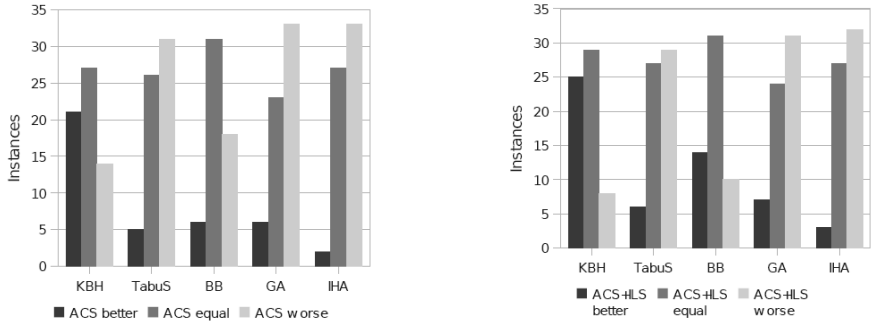


Fig. 3. Comparison of ACO algorithms with other decomposition methods

Algorithms GA, IHA and TabuS outperform our algorithms considering the number of found better upper bounds. However, the time limit of our algorithm was set to 1000 seconds, whereas other algorithms were executed for much longer time. Unfortunately, due to the space limitation of this paper we can not present these results here, but the reader is referred to [12] (this paper presents the execution times of KBH, BB, TabuS, GA and IHA). Based on these results it is clear that for large examples the execution time of these algorithms was much longer compare to our algorithms. Note that our ACS+ILS was able to find an improved upper bound of 30 for the problem instance *homer.col*. By

applying the ACS+ILS algorithm with the min-fill heuristic we could further improve the upper bound for this instance to 29. Considering comparison of ACS and ACS+ILS, for 25 problem instances ACS+ILS gave a better minimum width than ACS on its own. ACS was never able to outperform ACS+ILS with the exception of the problem instances *inithx.i.2* and *inithx.i.3* for which ACS achieved a better average width than ACS+ILS.

5 Conclusions

In this paper we have applied the ant colony optimization metaheuristic to the problem of finding tree decomposition of small width. Our experiments suggested that the ACO variants Max-Min Ant System and Ant Colony System give the best results for tree decompositions. We have applied Ant Colony System with and without the iterated local search to 62 benchmark graphs. The hybrid algorithm turned out to give better results than Ant Colony System on its own. It could improve the best known upper bound of the problem instance *homer.col* from 31 to 29. For 28 instances the algorithm was able to return a width equal to the best known upper bound. Nevertheless, especially for more complex problem instances both algorithms gave worse results than the best methods in literature. However, the time limit of our algorithm was set to 1000 seconds, whereas other algorithms were executed for longer time.

Subject of future research is the investigation of self-adaptive parameter settings. The algorithm could make use of the stagnation measures in order to adjust parameters such as the evaporation rate ρ autonomously. Another viable extension worth of further investigation is the application of ant colonies consisting of a number of ants proportional to the number of vertices in the constraint graph. That possibly could help to improve the quality of the pheromone updates and therefore could also improve the convergence behaviour of the algorithm.

Acknowledgments. The research herein is partially conducted within the competence network Softnet Austria (http://www.soft-net.at/) and funded by the Austrian Federal Ministry of Economics (bm:wa), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in terms of the center for innovation and technology (ZIT). Additionally, this work was partially supported by the Austrian Science Fund (FWF), project P20704-N18.

References

1. Bullnheimer, B., Hartl, R.F., Strauss, C.: A New Rank Based Version of the Ant System: A Computational Study. Central European Journal for Operations Research and Economics 7(1), 25–38 (1999)
2. Clautiaux, F., Moukrim, A., Négre, S., Carlier, J.: Heuristic and meta-heurisistic methods for computing graph treewidth. RAIRO Oper. Res. 38, 13–26 (2004)
3. Dorigo, M.: Optimization, Learning and Natural Algorithms [in Italian]. PhD thesis, Dipartimento die Elettronica, Politecnico di Milano, Milan (1992)

4. Dorigo, M., Gambardella, L.M.: Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation 1(1), 53–66 (1997)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. Bradford Book (2004) ISBN 0262042193
6. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B 26, 29–41 (1996)
7. Gogate, V., Dechter, R.: A complete anytime algorithm for treewidth. In: Proceedings of the 20th Annual Conference on Uncertainty in Artificial Intelligence, UAI 2004, pp. 201–208 (2004)
8. Hammerl, T.: Ant Colony Optimization for Tree and Hypertree Decompositions. Master's Thesis, Vienna University of Technology (2009)
9. Kjaerulff, U.: Optimal decomposition of probabilistic networks by simulated annealing. Statistics and Computing 1, 2–17 (1992)
10. Koster, A., Bodlaender, H., van Hoesel, S.: Treewidth: Computational experiments. Electronic Notes in Discrete Mathematics, vol. 8. Elsevier Science Publishers, Amsterdam (2001)
11. Larranaga, P., Kujipers, C.M.H., Poza, M., Murga, R.H.: Decomposing bayesian networks: triangulation of the moral graph with genetic algorithms. Statistics and Computing (UK) 7(1), 19–34 (1991)
12. Musliu, N.: An iterative heuristic algorithm for tree decomposition. In: Cotta, C., van Hemert, J. (eds.) Recent Advances in Evolutionary Computation for Combinatorial Optimization, vol. 153, pp. 133–150 (2008)
13. Musliu, N., Schafhauser, W.: Genetic algorithms for generalised hypertree decompositions. European Journal of Industrial Engineering 1(3), 317–340 (2007)
14. Robertson, N., Seymour, P.D.: Graph minors. II. algorithmic aspects of tree-width. Journal Algorithms 7, 309–322 (1986)
15. Stützle, T., Hoos, H.: Max-min Ant System and local search for the traveling salesman problem. In: IEEE International Conference on Evolutionary Computation, pp. 309–314 (1997)
16. Stützle, T., Hoos, H.: Max-min Ant System. Future Gener. Comput. Syst. 16(9), 889–914 (2000)
17. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithm to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13, 566–579 (1984)

Iterated Local Search with Path Relinking for Solving Parallel Machines Scheduling Problem with Resource-Assignable Sequence Dependent Setup Times

Edmar Hell Kampke, José Elias Claudio Arroyo, and André Gustavo Santos

Departamento de Informática, Universidade Federal de Viçosa
Avenida P.H. Rolfs s/n, Campus UFV, 36570-000 Viçosa, MG, Brasil
edmar.kampke@ufv.br, jarroyo@dpi.ufv.br, andre@dpi.ufv.br

Abstract. This paper addresses the unrelated parallel machine problem with machine and job sequence dependent setup. In this problem, the amount of the setup time does not only depend on the machine and job sequence, but also on a number of resources assigned, which can vary between a minimum and a maximum. The goal is to find a schedule that minimizes the linear combination of the total resources assigned and the total completion time. The problem is NP-hard in the strong sense. The NP-hardness of the problem motivates us to develop a new Iterated Local Search (ILS) heuristic to obtain near-optimal solutions. The heuristic uses an intensification strategy based on the Path Relinking technique which generates new solutions by exploring trajectories that connect high-quality solutions. Computational tests are carried out on a set of benchmark instances and the results obtained by the proposed ILS improve the best known results from the literature by a significant margin.

Keywords: Metaheuristics, Parallel Machine Scheduling, Setup Time, ILS, Path Relinking.

1 Introduction

In today's complex manufacturing setting, with multiple lines of products, each requiring many different steps and machines for completion, the decision maker for the manufacturing plant must find a way to successfully manage resources in order to produce products in the most efficient way possible. The decision maker needs to design a production schedule that promotes on-time delivery, and minimizes objectives such as the flow time of a product. Unrelated parallel machines can be characterized as machines that perform the same function but have different capabilities or capacities. A company may invest in similar machines that have different capabilities, taking into consideration the capital cost, operation cost and variability in demand. A bank of machines in parallel is a situation that is important from both a theoretical and a practical point of view [9]. From a theoretical point of view, it is a generalization of the single machine and a special case of the flexible flow shop. From a practical point of view, it is important because the occurrence of resources in parallel is common in the real world. Also, techniques for studying machines in parallel are often used in decomposition procedures for multistage systems.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 107–118, 2010.

This paper addresses the problem of scheduling jobs in unrelated parallel machines with sequence dependent setup times. In this problem (denoted by PMSDST), the processing time of each job depends on the machine to which it is assigned and setup times are incurred on the machines after having processed a job *i* and before processing other job *j*. A setup is a non-productive period of time which usually models operations to be carried out on machines after processing a job to leave them ready for processing the next job in the sequence. Setup times frequently represent cleaning production lines and/or adding/removing elements from a line when changing from one type of job another. The goal of the problem is to find a schedule that minimizes some criteria, such as makespan, total flow time and total tardiness relative to job due dates.

The literature on unrelated parallel machine scheduling with sequence dependent setup times is quite limited. In [5], a list scheduling heuristic to construct an initial solution is developed, followed by a local search involving pair wise exchange of jobs to minimize the total tardiness. [14] proposes a mixed integer programming (MIP) model for earliness/tardiness objective. The minimization of total weighted flowtime is studied in [13]. In [6] is proposed a simulated annealing heuristic to minimize total tardiness. Kim and Shin [7] propose an extension of the ATCS (apparent tardiness cost with setups) dispatching rule in order to minimize the maximum lateness. [3] applies a genetic algorithm for minimizing the makespan, total weighted completion time and total weighted tardiness. More recently, [2] studied a problem of optimal scheduling and lot-sizing a number of products on unrelated parallel machines to satisfy given demands, and [10] proposed several GRASP-like heuristics for the makespan criterion.

In this work we study the PMSDST problem considering machine setup times that depend on available resources. The new characteristic of this problem is that the setup time does not only depend on the machine and job sequence, but also on the number of resources assigned, which can vary between a minimum and a maximum. The setup times can be reduced or extended according to the number of resources assigned to carry out the setup operations. The scheduling problem that results from the addition of resource-assignable setups was formulated by Ruiz and Andrés [11] and it is denoted by PMRASDST. In [11] are presented a MIP model and 14 fast heuristics based on three dispatching rules. Ruiz and Andrés carry out careful and comprehensive statistical analyses to study what characteristics of the problem affect the MIP model performance and they also study the effectiveness of the different heuristics. The best results obtained by the 14 heuristics are available.

To solve the PMRASDST problem, we propose an Iterated Local Search (ILS) heuristic with path relinking technique as an intensification strategy. ILS iteratively applies local search to perturbations of the current search point, leading to a randomized walk in the space of local optima. Path relinking technique explores trajectories that connect high-quality solutions. The proposed ILS heuristic is tested on 720 problems generated by Ruiz and Andrés and the results are compared with the best results obtained by the 14 heuristics presented in [11].

2 The Unrelated Parallel Machine Resource-Assignable Sequence Dependent Setup Times Problem (PMRASDST)

This problem can be described as follows [11]. Consider n jobs to be processed on a set of m continuously available parallel machines. Each machine can process only one job at a time, and each job can be processed on any machine. Each job i has a deterministic processing time p_{ik} on the machine k, $i = 1,\ldots,n$, $k = 1,\ldots,m$. S_{kij} is the setup time to be carried out on machine k after having processed job i and before processing job j, and assume that $S_{kij} \neq S_{kji}$, i.e. setup times are machine and sequence dependent.

R_{kij} is the the amount of resources devoted to carrying out setup S_{kij} between job i and job j on machine k. R_{kij} varies between two values, the minimum and maximum resource (R_{kij}^- and R_{kij}^+). For example, some setups might require removing certain heavy tooling from the machine and therefore a minimum amount of workers (resources) are needed. Similarly, at a give moment it might not be possible to assign more resources due to physical or operational constraints. The relation between the actual setup time and the number of resources assigned is assumed to be linear. Therefore, if the minimum resources R_{kij}^- are assigned, the resulting setup time will be the largest possible (S_{kij}^+). Conversely, if the maximum resources R_{kij}^+ are assigned, the minimum setup (S_{kij}^-) is assigned. This relation is shown in Figure 1.

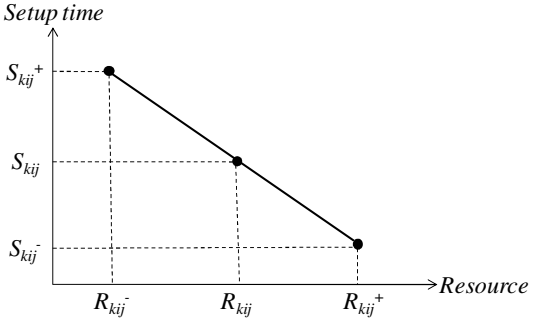


Fig. 1. Relation between setup time S_{kij} and number of assigned resources R_{kij}

Therefore, from the linear relation, the expression that results the amount of the setup time S_{kij} as a function of the assigned resources R_{kij} is the following:

$$S_{kij} = S_{kij}^+ - \frac{S_{kij}^+ - S_{kij}^-}{R_{kij}^+ - R_{kij}^-}(R_{kij} - R_{kij}^-) = S_{kij}^+ + \frac{S_{kij}^+ - S_{kij}^-}{R_{kij}^+ - R_{kij}^-}R_{kij}^- - \frac{S_{kij}^+ - S_{kij}^-}{R_{kij}^+ - R_{kij}^-}R_{kij}$$

Using K_1 for the first term and K_2 for the R_{kij} multiples, respectively we have as a result the slope-intercept form of the line relating S_{kij} and R_{kij}:

$$S_{kij} = K_1 - K_2 R_{kij}$$

Therefore, K_2 is the slope of the line that relates R_{kij} with S_{kij} as shown in Figure 1. In other words, K_2 indicates how much the setup time S_{kij} is reduced by each additional resource.

The objective of the problem is to minimize the linear combination of the total number of resources assigned and the total completion time (flow time) [11]:

$$Z = \lambda \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} R_{kij} + \delta \sum_{i=1}^{m} \sum_{j=1}^{n} C_{ik}$$

where, R_{kij} is the resources assigned to the setup between job i and job j on machine k. C_{ik} is the completion time of job i at machine k, λ and δ represent the weight or importance of each unit of resource and flow time, respectively. $R_{kij} = 0$ if job i is not processed before job j in the machine k. Similarly, if job i is not processed on machine k, then $C_{ik} = 0$.

Figure 2 illustrates an example of a schedule with $n = 4$ jobs and $m = 2$ machines. The jobs 4 and 2 are processed, in this order, on machine 1, finishing at times 43 and 155, respectively. The setup time between these jobs is 61, and 3 resources were used. Similarly, the jobs 3 and 1 are processed in machine 2, and they are completed at times 27 and 100, respectively. The setup time between these jobs is 28, and 4 resources were used. Assuming $\lambda = 50$ and $\delta = 1$, then $Z = 50\times(3+4) + 1\times(100+155+27+43) = 675$.

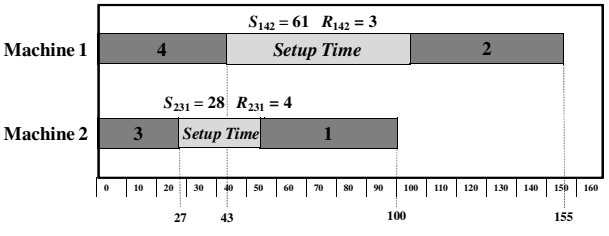


Fig. 2. Example of Solution

In this paper, a solution of the problem is represented by an array of size $n+m-1$, containing n jobs and $m-1$ separators (-1) between sequences for each machine. The schedule shown in Figure 2 is represented by the array [4, 2, -1, 3, 1]. Note that the sequences of jobs [4, 2] and [3, 1] correspond to the machines 1 and 2, respectively.

3 Proposed Algorithm

The Iterated Local Search (ILS) [8] is a simple and generally applicable heuristic that iteratively applies local search to modifications of a current solution s. Four basic procedures or operators are needed to derive an ILS algorithm: a procedure "Generate-Initial-Solution", which returns an initial solution s, a procedure "Perturbation" that perturbs the current solution s leading to some intermediate solutions s_1, a procedure "LocalSearch" that takes s_1 to a local optima s_2, and an acceptance criterion that decides from which solution the next perturbation step is applied. The Perturbation, LocalSearch and acceptance criterion are executed until the stop condition is satisfied. The best solution s^* generated over all iterations is returned at the end of the algorithm. In Figure 3 shows the pseudocode for a basic ILS algorithm.

In this paper we add an intensification procedure based on the Path Relinking technique which generates new solutions by exploring trajectories that connect the solution returned by the local search, with an elite solution. In the next subsections, we described each procedure of the ILS heuristics applied for the PMRASDST problem.

```
Procedure ILS
1     s ← Generate-Initial-Solution;  s* ← s;
2     while StopCondition do
3        s1 ← Perturbation(s);
4        s2 ← LocalSearch(s1);
5        if Z(s2) < Z(s*) then
6           s* ← s2;
7        end-if;
8        AcceptanceCriterion(s, s2);
9     end-while;
10    return s*;
end;
```

Fig. 3. Pseudocode of the iterated local search procedure (ILS), proposed in [8]

3.1 Initial Solution Construction

To construct an initial solution, we use the best dispatching heuristics proposed by Ruiz and Andrés [11]. This heuristic is called *Dynamic Job Assignment with Setups Resource Assignment* (DJASA) and runs as shown in Figure 4.

```
Procedure DJASA
1     L ← {j1,…,jn};
2     sequence ← { };
3     while |sequence| < n do
4        for each job of L, calculate the value of Z by adding ji in sequence;
5        take the job ji that provided the smallest increment of Z;
6        sequence ← sequence ∪ { ji};
7        L ← L – { ji};
8     end-while;
9     s ← LocalSearch(sequence);
10    return s;
end;
```

Fig. 4. Pseudocode of the *DJASA* heuristic used in the procedure of the initial solution construction

In [11] was also tested the assignment of the minimum and the average resources. The maximum resources assignment produced the best results.

The solution built by the DJASA heuristic is improved using a local search procedure described in 3.3.

3.2 Perturbation

The perturbation procedure should be chosen *strong enough* to allow to leave the current local minimum and to enable the local search to find new, possibly better local minima. At the same time, the perturbation should be *weak enough* to keep enough characteristics of the current local minimum.

In this work, we use a greedy perturbation procedure similar to the presented in [12]. This procedure is composed of two phases: destruction and reconstruction. In the destruction phase, d different jobs are chosen randomly to be removed from the current sequence s. These jobs are stored on a set S_R. In the reconstruction phase, all jobs in S_R are reinserted in s. The first job j_1 of S_R is inserted in all possible positions of s generating a set of partial sequences that include job j_1. The sequence with the best objective value is kept for the following iteration. The process ends when S_R is empty and therefore s contains all the n jobs.

3.3 Local Search

The goal of the local search is to improve the solutions s obtained in the perturbation procedure. Local search methods begin with a solution, and generate a neighborhood of this solution. The neighborhood contains all the solutions reached through single moves made in the current solution. In this neighborhood a solution that is better than the current solution is picked up. The chosen solution becomes a new solution (or current) and the process continues until a local optima is reached. Remember that, a solution (or schedule) is represented by a sequence of $N = n+m$-1 elements (n jobs and m-1 elements -1).

In this study we use two neighborhood structures. The first neighborhood is defined by the insertion move, which consists of removing an element from its original position and inserting it in the on the $N-1$ remaining positions. This move generates a neighborhood of size $(N-1)^2$. If an improvement is obtained, then the process is repeated.

The second neighborhood is defined by the swap move, which consists of interchanging all pairs of elements. The second neighborhood has size $N(N-1)/2$.

The best results were obtained using the insertion neighborhood. Therefore, the numerical experiments were performed using this neighborhood.

3.4 Acceptance Criterion

The acceptance criterion is used to decide to which solution the next perturbation should be applied. One important aspect of the acceptance criterion is to introduce a bias between intensification and diversification of the search. One solution is accepted if it improves the best current solution. In the ILS algorithm, the solutions that are worse than the best current solution can also be accepted with a small probability. Thus, we avoid the constant use of the best current solution and a fast stagnation of the solutions evaluated. This work employs an acceptance criterion similar to that used in the metaheuristic Simulated Annealing [6]. However, in this case, the value of the temperature is constant:

$$Temperature = 0.5 \times \frac{\sum_{i=1}^{n} \sum_{k=1}^{m} p_{ik}}{n \times m \times 10}$$

The temperature value depends on the processing times (p_{ik}) of the jobs, the number of jobs n and the number of machines m, in the same way as used in [12]. If the solution s_2 returned by the local search is worse than the current solution s, it can be accepted with a probability defined by $exp\{-(Z(s_2) - Z(s))/Temperature\}$,where, exp is the exponential function.

3.5 Path Relinking

Path relinking was first introduced in the context of Tabu Search [4], as an approach to integrate intensification and diversification strategies. It consists of exploring trajectories that connect high-quality solutions, by starting from an *initial solution* and generating a path in the neighborhood of this solution towards another solution, called the *guiding solution*. This path is generated by selecting movements that introduce in the initial solution attributes of the guiding solution. At each step, all movements that incorporate attributes of the guiding solution are analyzed and the movement that best improves (or least deteriorates) the initial solution is chosen.

In this paper, as in [1], the path relinking maintains a set of E_{Size} elite solutions (high-quality solutions). The procedure begins with a random selection of the *guiding solution* $s_g \in E$. The initial solution s_i is the solution returned by the local search procedure ($s_i \neq s_g$). The solutions in E are local optima and the path relinking tries to find better solutions that are not in the insertion neighborhood of s_i and s_g, analyzed by local search procedure. In the path relinking we use swap moves, i.e., the procedure performs the role of crossover. The crossover is widely used in evolutionary algorithms frameworks.

In each step, swap moves is made in s_i, incorporating attributes of s_g. The new solutions are analyzed and, if applicable, the best solution is updated. The procedure ends when s_i and s_g become equal. Figure 5 shows an example in which the guide solution is obtained in two steps from the initial solution.

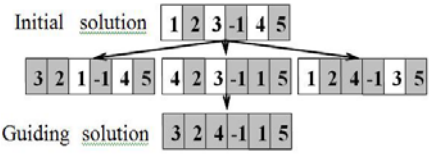


Fig. 5. Example of path relinking technique

3.6 ILS with Path Relinking for PMRASDST Problem

In the proposed algorithm, path relinking is embedded in the ILS mechanism, being used as an intensification procedure.

In the implementation of the ILS with path relinking algorithm, denoted by ILSPR, the size of the elite solutions set was calibrated and defined by $E_{Size} = 10$. The parameter d used in the perturbation procedure was calibrated and the best results were obtained for $d = 4$ (for $6 \leq n \leq 10$) and $d = 10$ (for $n > 10$). A pseudocode description of the proposed ILSPR algorithm is presented in Figure 5.

```
Procedure ILSPR (E_Size, StopCondition, d)
1    E = ∅;
2    s ← GetLocalOptimaSolution;
3    s* ← s;
4    while not StopCondition do
5        s1 ← Perturbation(s, d);
6        s2 ← LocalSearch(s1);
7        if |E| = E_Size then
8            s3 ← Choose randomly a elite solution of E;
9            s4 ← Path_Relinking(s2, s3);
10           s5 ← Path_Relinking(s3, s2);
11           s6 ← Best_Solution (s4, s5);
12           if s6 ∉ E and s6 ≠ s2 then
13               s2 ← LocalSearch(s6);
14           else
15               s2 ← s6;
16           end-if
17           UpdateEliteSet(s2);
18       else
19           E ← E ∪ {s2};
20       end-if
21       AcceptanceCriterion(s, s2);
22   end-while;
23   s* ← Best_Solution (E);
24   return s*;
end;
```

Fig. 6. Pseudocode of the proposed ILSPR Algorithm

4 Numerical Experiments

This paper tests the performance of the ILSPR heuristic algorithm. For that purpose, the same problems generated by Ruiz and Andrés [11] were used. They generated all the data: processing times p_{ik} and the minimum and maximum resources and setups (R_{kij}^{-}, R_{kij}^{+}, S_{kij}^{-} and S_{kij}^{+}). A total of 720 problems were generated, divided into two groups: small and large (each one with 360 problems). The small group contains problems with $n \in \{6, 8, 10\}$ jobs and $m \in \{3, 4, 5\}$ machines. In the large group of problems, the number of jobs (n) and machines (m) belong respectively to the sets: {50, 75, 100} and {10, 15, 20}. For all problems, the parameters λ and δ (that represent the importance or cost of each unit of resource and flow time, respectively), were the same used by Ruiz and Andrés [11]: λ=50 and δ=1.

The ILSPR algorithm was implemented in Java (version 1.6) and executed using the JDK 6.0 compiler. The tests were conducted on an Intel® Core™ Quad with a 2.4 GHz processor and 3GB of RAM.

4.1 Results

The results obtained by the algorithm are compared with the best results available in literature [11]. The ILSPR efficiency is evaluated by the relative percentage improvement relative to the MIP model [11] solved by CPLEX (for small problems) and to the constructive heuristics proposed in [11] (for large problems). This improvement is calculated using the following formula:

$$Percentage\ Improvement = \frac{100 \times (Z_C - Z_{ILSPR})}{Z_C} \%$$

where Z_C is the best objective value obtained in [11] and Z_{ILSPR} is the objective value obtained by the ILSPR heurístic.

The solutions available for the small problems were obtained by the software CPLEX 9.1, solving a MIP model [11]. In [11], the execution of CPLEX was limited to 300 seconds for problems with $n = 6$ jobs and 3600 seconds for problems with $n = 8$ and $n = 10$ jobs. For all problems with $n = 6$ and $n = 8$ jobs, MIP model solved by CPLEX found the optimal solution. For the problems with $n = 10$ jobs, CPLEX found only approximated solutions, because of the limited execution time.

Table 1. Improvement percentage of ILSPR relative to software CPLEX

Small problems					
n*×*m	**Average of *Z***		**Improvement Percentage (%)**	**Time (s) ILSPR**	**Time (s) CPLEX**
	CPLEX	**ILSPR**			
6 × 3	**7383**	**7383**	**0.00**	9	300
6 × 4	**4742**	**4742**	**0.00**	12	300
6 × 5	**2872**	**2872**	**0.00**	15	300
8 × 3	**13.531**	13.537	-0.04	12	3600
8 × 4	**9433**	9469	-0.38	16	3600
8 × 5	**6829**	6830	-0.01	20	3600
10 × 3	21.165	**20.790**	**1.77**	15	3600
10 × 4	15.029	**14.812**	**1.44**	20	3600
10 × 5	11.168	**10.783**	**3.45**	25	3600
Average	-	-	**0.69**	**16**	**2500**

Table 1 presents, for each set of problems of $n \times m$ size, the averages of the relative percentage improvement of ILSPR relative to the MIP model solved by CPLEX. Each value in the fourth column of the Table 1 represents the average improvement on a total of 40 problems. We observed that the ILSPR found the optimal solution for all problems with $n = 6$. For problems with $n = 8$ it did not find the optimal solution for all problems. In problems with $n \times m = 10 \times 5$, the ILSPR method on average yielded a 3.45% over the MIP model results. The less values of improvement of the algorithm (for problems with $n = 10$) were for the problems with $n \times m = 10 \times 4$ where the average improvement was 1.44%. For the total of 360 problems tested of the "small group", the ILSPR algorithm obtained a total improvement of 0.69% (on the average).

The fifth and sixth columns of Table 1 show the computational times spent by the ILSPR and MIP Model. Note that the time spent by ILSPR increases according to the number $n \times m$.

The solutions available for the large problems were obtained by the constructive heuristics, proposed by Ruiz and Andrés [11]. Table 2 presents, like Table 1, the averages of the relative percentage improvement of ILS (without path relinking) and ILSPR relative to results obtained by the constructive heuristics. The improvement is calculated using the same formula of Table 1.

Table 2. Improvement percentage of ILSPR relative to constructive heuristics in the literature

Large problems						
n*×*m	**Average of *Z***			**Improvement Percentage ILS (%)**	**Improvement Percentage ILSPR (%)**	**Time (s) ILSPR**
	Constructive Heuristics	**ILS**	**ILSPR**			
50 × 10	128.276	116.739	**116.628**	9.34	9.43	250
50 × 15	86.561	**79.008**	79.175	9.14	8.95	375
50 × 20	61.953	56.886	**56.884**	8.42	8.44	500
75 × 10	260.992	**232.361**	232.778	11.37	11.23	375
75 × 15	182.903	168.263	**168.026**	8.54	8.62	562
75 × 20	140.067	**128.547**	128.554	8.78	8.77	750
100 × 10	415.174	374.882	**374.482**	10.13	10.22	500
100 × 15	303.162	**277.066**	277.328	9.06	8.99	750
100 × 20	237.082	220.627	**220.021**	7.51	7.79	1000
Average	**-**	**-**	**-**	**9.14**	**9.16**	**562.44**

According to the results presented in the fifth and sixth column of Table 2, it may be seen that for the set of $n \times m = 75 \times 10$ problems, the ILS algorithm improved the constructive heuristics in 11.37%. The lower values of improvement of the ILSPR algorithm were for the problems with $n \times m = 100 \times 20$ where the on average improvement was 7.79%. For a total of 360 problems, the total average improvement of the ILSPR algorithm with respect to the constructive heuristics was 9.16%. The ILSPR obtained better results than the ILS in 188 problems, while the ILS obtained better results in 169 problems. In only 3 problems, the results obtained by the algorithms ILS and ILSPR were identical.

The seventh column of Table 2 shows the computational time spent by the ILSPR algorithm for each set of problems of $n \times m$ size. The computational times spent by the constructive heuristics are not available.

For the stop condition we use a computational time, based on the number of jobs (n) and the number of machines (m) of each problem. The computational time used is $(n \times m)/2$ seconds. For example, the problem with $n = 100$ jobs and $m = 20$ machines have a runtime of 1000 seconds.

From the large problems, a small sample of 30 problems were chosen randomly, 10 from each of the following three sets of problems of $n \times m$ size: 50×10; 75×15 and 100×20. The problems were solved by the ILSPR algorithm with a stop condition defined as $n \times m$ seconds. At each $(n \times m)/10$ seconds, the best solution found until

that time was printed. At the end of the execution of each problem, the 10 printed solutions show the decline in output solutions along the execution. Figure 7 shows the graph of the averages of the solutions obtained by the average of time spent (in seconds) for the solved problems.

In the analysis of the chart in Figure 7, we note that the average of the solutions has a steep decline in the first half of the graph. In the second half, we note a lower decline with a stabilization trend. In fact, analysing the results used to build the graphic we can note that among the 30 problems selected, 23 failed to improve the solution in the second half of the graph. This analysis was important to determine the stop condition of the algorithm in $(n \times m)/2$ seconds.

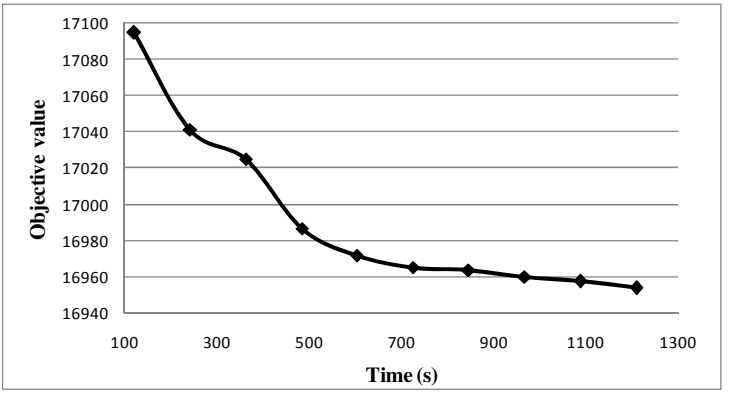


Fig. 7. Graphic showing the average of the solutions by the average of time spent

5 Conclusion

This paper proposes an Iterated Local Search with Path Relinking (ILSPR) algorithm for a parallel machines scheduling problem. The performance of the proposed heuristics was tested on 720 problems, including large size instances. The solutions obtained by these heuristics were compared to the best solutions found in literature [11]. For the large size problems, the ILSPR algorithm had an average improvement of 8.72% compared to the best solutions available in from other heuristics in literature [11]. For some instances the improvement was more than 10%. In the small group, in several instances, was obtained the optimal solution, also found by MIP model solved by CPLEX and in instances where MIP model did not found the optimal solution, the proposed algorithm found better solutions in a short time.

Acknowledgements. This work is supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) and the Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG).

References

1. Aiex, R.M., Resende, M.G.C., Pardalos, P.M., Toraldo, G.: Grasp with path relinking for three-index assignment. INFORMS Journal on Computing 17(2), 224–247 (2005)
2. Dolgui, A., Eremeev, A.V., Kovalyov, M.Y., Kuznetsov, P.M.: Multi-product lot-sizing and scheduling on unrelated parallel machines to minimize makespan. In: Dolgui, A., Lototsky, V.A., Nof, S.Y. (eds.) Proceedings of the 13th IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2009), Moscow, Russia, pp. 832–837. Elsevier Science, Amsterdam (2009)
3. Fowler, J.W., Horng, S.M., Cochran, J.K.: A hybridized genetic algorithm to solve parallel machine scheduling problems with sequence-dependent setups. International Journal of Industrial Engineering 10, 232–243 (2003)
4. Glover, F.: Tabu search and adaptive memory programming - advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kenngington, J.L. (eds.) Interfaces in Computer Science and Operations Research, pp. 1–75 (1996)
5. Guinet, A.: Textile production systems: A succession of non-identical parallel processor shops. Journal of the Operational Research Society 42, 655–671 (1990)
6. Kim, D.W., Kim, K.H., Jang, W., Chen, F.F.: Unrelated parallel machine sched-uling with setup times using simulated annealing. Robotics and Computer Integrated Manufacturing 18, 223–231 (2002)
7. Kim, C.O., Shin, H.J.: Scheduling jobs on parallel machines: a restricted tabu search approach. International Journal of Advanced Manufacturing Technology 22, 278–287 (2003)
8. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G.A. (eds.) Kochenberger Handbook of metaheuristics, pp. 321–353. Kluwer Academic, Boston (2003)
9. Pinedo, M.: Scheduling: theory, algorithms, and systems. Prentice-Hall, New Jersey (1995)
10. Rabadi, G., Moraga, R.J., Al-Salem, A.: Heuristics for the unrelated parallel machine scheduling problem with setup times. Journal of Intelligent Manufacturing 17(1), 85–97 (2006)
11. Ruiz, R., Andrés, C.: Unrelated parallel machines scheduling with resource-assignable sequence dependent setup times. In: Baptiste, P., Kendall, G., Munier-Kordon, A., Sourd, F. (eds.) Proceedings of the 3rd Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA), Paris, France, pp. 439–446 (2007)
12. Ruiz, R., Stützle, T.: An iterated greedy heuristic for the sequence dependent setup times flow shop problem with makespan and weighted tardiness objectives. European Journal of Operational Research 187(3), 1143–1159 (2008)
13. Weng, M.X., Lu, J., Ren, H.: Unrelated parallel machines scheduling with setup consideration and a total weighted completion time objective. International Journal of Production Economics 70(3), 215–226 (2001)
14. Zhu, Z., Heady, R.: Minimizing the sum of earliness/tardiness in multi-machine scheduling: a mixed integer programming approach. Computers & Industrial Engineering 38(2), 297–305 (2000)

Enhancing a Tabu Algorithm for Approximate Graph Matching by Using Similarity Measures

Segla Kpodjedo, Philippe Galinier, and Giulio Antoniol

SoccerLAB, Ecole Polytechnique de Montreal
{segla.kpodjedo,philippe.galinier,giuliano.antoniol}@polymtl.ca

Abstract. In this paper, we investigate heuristics in order to solve the Approximated Matching Problem (AGM). We propose a tabu search algorithm which exploits a simple neighborhood but is initialized by a greedy procedure which uses a measure of similarity between the vertices of the two graphs. The algorithm is tested on a large collection of graphs of various sizes (from 300 vertices and up to 3000 vertices) and densities. Computing times range from less than 1 second up to a few minutes. The algorithm obtains consistently very good results, especially on labeled graphs. The results obtained by the tabu algorithm alone (without the greedy procedure) were very poor, illustrating the importance of using vertex similarity during the early steps of the search process.

Keywords: Approximate Graph Matching, similarity measure, tabu search.

1 Introduction

Graph representations are well suited to all kinds of real life objects. In many applications, a frequent task is to find a matching between the vertices of two given objects represented as graphs while optimizing a particular criterion. Depending on the application domain, the graphs to be matched can represent images [13], molecules [14] or software artifacts [1,10], etc.

In graph theory, the Graph isomorphism [11] problem is a well-known graph matching problem. However, this exact matching is generally not appropriate because it imposes a strict correspondence between nodes and edges of the two graphs. Graphs to be matched in many applications are not isomorphic: as a result, matchings of interest are not necessarily complete (somes vertices may remain unmatched) and should tolerate errors of correspondances. Other more flexible graph matching problems include Maximum Common Edge Subraph (MCES) [12] problem and Error-Tolerant Graph Matching (ETGM) [4]. The MCES[1] consists in finding a common partial subgraph with a maximum number of edges. In ETGM, one defines a set of graph edit operations, each with assigned cost, and the goal of the problem is to find a series of edit operations (transforming the first graph into the second one) with a minimum cost.

[1] Different from the Maximum Common Induced Subgraph (MCIS) problem where one aims at finding a common induced subgraph with a maximum number of vertices.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 119–130, 2010.

Various kinds of techniques have been proposed in order to solve graph matching problems such as MCES and ETGM. There are exact algorithms such as the RASCAL algorithm for MCES [12]. But given that most graph matching problems are NP-hard [5], there are also heuristics, including metaheuristics such as simulated annealing [6], deterministic annealing [9], genetic algorithms [2], tabu search [15] etc. Notice that, while benchmarks were proposed for exact matching (graph isomorphism and MCIS) [7], there are no standard benchmarks for approximate graph matching. In this paper, our goal is to investigate heuristics for Approximate Graph Matching (AGM). Preliminary tests showed that a tabu algorithm exploiting a simple neighborhood can not efficiently treat most of problem instances even with graphs containing as few as 20 vertices. In order to improve the efficiency of a tabu algorithm, we propose to guide it at the early stage of the search by using similarity measures between the vertices of the two graphs. This measure is calculated by taking into account the adjacent edges of the two vertices. A greedy procedure which exploits such similarity measures is used in order to initialize the tabu algorithm.

The remaining of the paper is organized as follows. In Section 2, we present the definition of the AGM problem. In Section 3, we describe our algorithm. In particular, we introduce the proposed measure of similarity, and describe both the greedy and the tabu procedures. Results obtained by our algorithm are presented in Section 4. Finally, we conclude in Section 5 with some general remarks and perspectives.

2 Problem Definition

In many applications, there is the need to find a good matching between two objects represented as graphs. In the following, we first give preliminary definitions and notations about graphs and matchings. Then, we analyze what a "good" matching may mean. Finally, we propose the formal definition of the AGM problem we use in this paper.

2.1 Preliminary Definitions

The graphs we consider are directed, and they have labels on their vertices and/or edges. Note that undirected graphs can be treated as symetric directed graphs.

Let Σ represent a finite set of symbols. A graph (labeled on alphabet Σ) is defined as a quadruple (V, E, l_V, l_E) where: V is the finite set of vertices (or nodes); $E \subseteq V \times V$ is the set of edges; $l_V : V \rightarrow \Sigma$ is the node labeling function; and $l_E : E \rightarrow \Sigma$ is the edge labeling function.

For practical reasons, we also define another function $L_E : V \times V \rightarrow \Sigma_+ = \Sigma \cup \{\#\}$ as follows: $L_E(x, y) = l_E(x, y)$ if $(x, y) \in E$, otherwise $L_E(x, y) = \#$. In other words, L_E coincides with l_E on E, and the symbol $\#$ is used in order to represent the absence of edge between two vertices.

Let us consider two graphs G_1=$(V_1, E_1, L_{V1}, L_{E1})$ and G_2=$(V_2, E_2, L_{V2}, L_{E2})$ labeled on the alphabet Σ. A matching between the two graphs is any relation

$\mu \subseteq V_1 \times V_2$ such that each vertex is matched to at most one vertex in the other graph (the one-to-one constraint): $\forall x, y \in V_1, \forall z, t \in V_2, (x, z), (x, t) \in \mu \Rightarrow z = t$ and $(x, z), (y, z) \in \mu \Rightarrow x = y$. In the following, an element of μ (a couple) will be named a *node match*.

2.2 What Is a Good Matching?

Let us consider two graphs G_1 and G_2 and a matching μ between them. A node match (x_1, x_2) in μ matches a vertex $x_1 \in V_1$ to a vertex $x_2 \in V_2$. Therefore, there are two possible cases depending on the labels of these two vertices:

- An *exact vertex label correspondance* occurs if $l_V(x_1) = l_V(x_2)$.
- A *vertex label error* occurs if $l_V(x_1) \neq l_V(x_2)$.

A couple of node matches $((x_1, x_2), (y_1, y_2)) \in \mu \times \mu$ matches a pair (x_1, y_1) of vertices of V_1 to a pair (x_2, y_2) of vertices of V_2. Therefore, there are four possible cases depending on (i)the presence of edges $(x_1, y_1) \in E_1$ and $(x_2, y_2) \in E_2$ (ii) the labels assigned to these (potential) edges:

- An *exact edge label correspondance* occurs if $(x_1, y_1) \in E_1$, $(x_2, y_2) \in E_2$, and $l_E(x_1, y_1) = l_E(x_2, y_2)$.
- An *edge label error* occurs if $(x_1, y_1) \in E_1$, $(x_2, y_2) \in E_2$, but $l_E(x_1, y_1) \neq l_E(x_2, y_2)$.
- A *structural error* occurs if $(x_1, y_1) \in E_1$ and $(x_2, y_2) \notin E_2$, or if $(x_1, y_1) \notin E_1$ and $(x_2, y_2) \in E_2$.
- Finally, the last possibility is that $(x_1, y_1) \notin E_1$ and $(x_2, y_2) \notin E_2$.

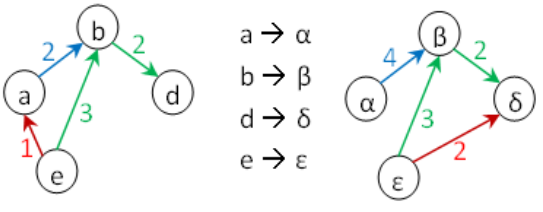


Fig. 1. Example of graph matching

Figure 1 represents two graphs with labels on their edges and a matching $\{(a, \alpha), (b, \beta), (d, \delta), (e, \epsilon)\}$ between them. In this matching, there are: 2 exact edge label correspondences (between (b, d) and (β, δ) and between (e, b) and (ϵ, β)); one edge label error (between (a, b) and (α, β)); two structural errors (edges (e, a) in the first graph and (ϵ, δ) in the second graph).

Although the notion of "good matching" may vary from one context to the other, it seems to be generally admitted that a good matching is characterized by the presence of exact correspondences (*exact vertex and edge label correspondences*), and the absence (or a small number) of errors (*inexact vertex and edge label correspondences* and *structural errors*).

We propose to define the score of a matching as the algebric sum of bonuses credited for exact correspondences and penalties assigned for errors. This definition is that of a graph matching which tolerates errors, is simple and generic enough to model, given appropriate weights for bonuses and maluses, problems such as graph isomorphism, MCES, MCIS, etc. We use two matrices M_V and M_E of real numbers. For any $(a, b) \in \Sigma \times \Sigma$, $M_V[a][b]$ represents the score assigned for matching a vertex which label is a to a vertex with label b. Similarly, for any $(a, b) \in \Sigma_+ \times \Sigma_+$, $M_E[a][b]$ represents the score assigned for matching a pair of vertices which label is a to a pair of vertices which label is b.

2.3 Definition of the AGM Problem

An AGM problem instance is defined by a quintuplet $(\Sigma, G_1, G_2, M_V, M_E)$ with two graphs $G_1 = (V_1, E_1, L_{V1}, L_{E1})$ and $G_2 = (V_2, E_2, L_{V2}, L_{E2})$ labeled on Σ and two functions $M_V : \Sigma \times \Sigma \rightarrow \mathbb{R}$ and $M_E : \Sigma_+ \times \Sigma_+ \rightarrow \mathbb{R}$ used in order to represent the score function. The score of a matching μ between G_1 and G_2 is defined by:

- $f(\mu) = f_V(\mu) + f_E(\mu)$, where
- $f_V(\mu) = \Sigma_{(x_1,x_2)\in\mu} M_V(l_V(x_1), l_V(x_2))$ and
- $f_E(\mu) = \Sigma_{(x_1,x_2),(y_1,y_2)\in\mu} M_E(L_E(x_1, y_1), L_E(x_2, y_2))$.

The goal of the problem is to find a matching with a maximum score.

3 Algorithm

The AGM problem is NP-hard since there is a simple reduction of the subgraph isomorphism problem to the AGM problem. Therefore, the only algorithms that are able to provide optimal solutions have an exponential worst-case complexity (if $P \neq NP$). For this reason, large problem instances are likely to be intractable for exact algorithms and to necessitate the use of meta-heuristics.

In this Section, we present a tabu search algorithm named Tabu Search for Approximated Graph Matching (TS-AGM) to treat the AGM problem. This algorithm is enhanced by the use of an initialization procedure named *GreedySim* which is based on structural similarity measures between the vertices of the two graphs. Both procedures are described in the following.

3.1 The Tabu Procedure

Starting from an initial configuration in the search space, a tabu algorithm moves iteratively from the current configuration to a neighboring one. On each iteration, the algorithm chooses the best neighbor of the current configuration (the one with the smallest cost), while avoiding returning toward configurations recently visited, by using a short-term diversification technique named tabu list [8].

Neighborhood definition. The search space of the TS-AGM algorithm is the set of matchings. The evaluation function is simply the objective function. A move applied to the current configuration S consists in (1) inserting a new node

match into S, while respecting the 1-to-1 constraint, or (2) removing a node match from S. An insertion move is denoted by $< +, (x_1, x_2) >$, where (x_1, x_2) represents the node match inserted into the configuration. Similarily, a removal move is denoted by $< -, (x_1, x_2) >$. Each move mv is evaluated by its impact $\delta(mv)$ on the evaluation function: $\delta(mv) = f(S \oplus mv) - f(S)$, where $S \oplus mv$ represents the configuration obtained by applying move mv to configuration S.

The tabu algorithm. The tabu mechanism is two-fold: just inserted node matches are forbidden to leave the current configuration for a given number of iterations (they are inserted into the so-called tabu-out list). Similarily, just removed node matches are forbidden to re-enter the configuration for a given number of iterations (they are inserted in the tabu-in list).

The TS-AGM procedure has four parameters: S_0 is the initial configuration transmitted to the procedure; parameter *max_fail_iter* specifies the stopping criterion; parameters *lgtl_in* and *lgtl_out* are used in order to set the tabu tenure.

The pseudo-code of the TS-AGM procedure is as follows.

Algorithm **TS-AGM**(S_0)

Set $S := S_0$;
do
mv := find the non-tabu move with a maximum value of $\delta(.)$;
  If $mv =< +, x_1, x_2 >$ (mv is an insertion move);
    Insert (x_1, x_2) into S;
    Insert (x_1, x_2) for *lgtl_out* iterations into the *tabu_out* list;
  Else $mv =< -, x_1, x_2 >$ is a removal move);
    Remove (x_1, x_2) from S;
    Insert (x_1, x_2) for *lgtl_in* iterations into the *tabu_in* list;
Until the stop criterion is met;
Return the best matching found during the search.

The current configuration is denoted by S. The procedure is initialized by using configuration S_0. Then, on each iteration, all potential moves (both insertion and removal moves) are evaluated and the best non tabu move (the one with a maximum value of δ) is selected (ties are broken randomly). After that, the selected move is applied to S and the tabu list is updated. The algorithm stops when it has performed *max_fail_iter* iterations without improvement over the best configuration found so far. It returns the best configuration generated during the search.

Difficulties encountered by the tabu algorithm. The most straightforward way to build the initial configuration is to use an empty set of node matches. In our experiments, we have observed that the Tabu algorithm does not perform well when it is initialized by using an empty configuration. It may wander in

the search space for a very large number of iterations without finding anything else but (very) poor configurations. The same problem occurs if the algorithm is initialized by using a random configuration (a configuration made of node matches randomly chosen, while respecting the 1-to-1 constraint).

In order to improve the efficiency of our TS procedure, we initialize it by using a configuration produced by the greedy constructive procedure, named *GreedySim*, described in the next Section.

3.2 The Initialization Procedure

The *GreedySim* procedure builds step by step a matching by inserting iteratively a new node match into the configuration. The choice of the node match to be inserted into the configuration (the greedy criterion) is based on similarity measures.

The similarity measure. Considering a graph $G = (V, E, ...)$, for any vertex $x \in V$ and any label $l \in \Sigma$, we denote by $f_+(x, l)$ and $f_-(x, l)$ the number of ingoing and outgoing edges adjacent to x and whose label is l: $f_+(x, l) = |\{y \in V : L_E(x, y) = l\}|$ and $f_-(x, l) = |\{y \in V : L_E(y, x) = l\}|$.

Consider two graphs G_1 and G_2 and a matching μ between them. Given $(x_1, x_2) \in V_1 \times V_2$, we denote by $potential(x_1, x_2)$ the maximum number of edges adjacent to x_1 and x_2 that can be correctly matched (assuming that x_1 would be matched to x_2). Clearly we have: $potential(x_1, x_2) = \Sigma_{l \in \Sigma}(min(f_+(x_1, l), f_+(x_2, l)) + min(f_-(x_1, l), f_-(x_2, l)))$.

The similarity $simil(x_1, x_2)$ between two nodes x_1 and x_2 is normalised as a real number between 0 and 1 and computed as:

$$simil(x_1, x_2) = \frac{2 \times potential(x_1, x_2)}{deg(x_1) + deg(x_2)} \times \frac{potential(x_1, x_2)}{maxPotential}$$

where $maxPotential = max_{(x_1,x_2) \in V_1 \times V_2} potential(x_1, x_2)$.

The first factor of the formula represents a raw similarity between the nodes. The second factor of the formula is meant to discriminate against vertices x_1 and x_2 with high similarity but low degrees. Indeed, if two vertices have low degrees, their high similarity can be simply due to mere chance. This is why we decrease their similarity score.

The GreedySim() procedure. The *GreedySim* procedure builds greedily a matching by using similarity scores between the vertices of the two graphs.

The procedure first computes the similarity for all pairs of vertices in $V_1 \times V_2$. Then, it performs a series of iterations. On each iteration, the greedy score $gr(x_1, x_2)$ of each legal move (x_1, x_2) is computed and the pair with the best greedy score is inserted into the configuration (ties are broken randomly). The procedure stops when all the nodes of the smallest graph are matched.

The greedy score is computed as follows:

$$gr(x_1, x_2) = \delta_0(x_1, x_2) + B \times simil(x_1, x_2)$$

where $\delta_0(x_1, x_2)$ is the number of new perfect edge correspondances; and B ($B \geq 1$) is a parameter used to weight the similarity of x_1 and x_2. At the beginning, the similarity is the more reliable information about the number of perfect edge correspondances a node match might ultimately bring. Thus B is maximal but as the solution is being built, B should decrease to the point that δ_0 becomes the main contributor to the score.

Our tests determine that setting B to $maxPotential$ then decreasing it linearily up to 1 (where the similarity is only used to untie ex-aequo moves) was an efficient way to apply the principles above.

4 Experimental Results

In this section, we present experiments performed in order to evaluate the proposed tabu algorithm. In these experiments, we have generated pairs of graphs with various features and we considered two different score functions. We have tested our tabu algorithm initialized with the greedy procedure; for comparison purposes, experiments performed with the tabu algorithm initialized with an empty solution have also been conducted.

4.1 A Generator of Random Instances

A graph database is proposed for exact matching (graph isomorphism and MCIS) in [7]. However, datasets used in the litterature for approximate graph matching are often not available and, when they are, the graphs are usually very small. Thus, to perform our experiments, we have developed our own generator. For the sake of simplicity, the graphs used in our experiments have labels on their edges but not on their nodes. Notice that having information (labels) on the nodes tends to make easier the matching process. Our generator is controlled by the following parameters:

- Parameter n represents the number of vertices of the two graphs.
- Parameter d is function of the expected density of the graphs.
- Parameter nl indicates the number of labels. A uniform distribution is assumed.
- Parameter $q (0 \leq q \leq 1)$ is used in order to control the similarity between the two graphs. The larger the value of q, the most similar the two graphs. In particular: if $q = 0$, the two graphs will be built independantly; and if $q = 1$, the two graphs will be isomorphic.

Given a quadruplet (n, d, nl, q) of parameters, the generators builds the two graphs G_1 and G_2 as follows:

Procedure **Generate-Graphs**(n, d, nl, q)

Let v_i^1 and v_i^2 $(i = 1..n)$ represent the nodes of the first
and the second graph, respectively;
$p = \frac{d}{n}$ is the probability of having an arrow between any couple of vertices
For $i = 1..n$, and $j = 1..n$ **do**
If $i \neq j$
Choose lb_1 and lb_2 such that $Prob(lb = \#) = 1 - p$
and $Prob(lb = k) = \frac{p}{nl}$ for $k = 1..nl$
Let $l(v_i^1, v_j^1) := lb_1$;
Choose $idem$ in $\{true, false\}$ such that $Prob(idem = true) = q$;
If $idem$ **then** $l(v_i^2, v_j^2) := lb_1$; **else** $l(v_i^2, v_j^2) := lb_2$;
Build a random permutation μ on $[1..n]$;
Apply μ to the vertices of G_2;
Return (G_1, G_2, μ).

For each couple of vertices in G_1, a label is assigned with respect to the values of parameters d and nl. The correspondent couple in G_2 is assigned, with probability q, the same label. Otherwise, its label is also assigned with respect to the values of parameters d and nl. We notice that the procedure returns, in addition to the two graphs, the permutation μ used to reorder the vertices of G_2. This permutation can be seen as a matching between G_1 and G_2. Notice that μ is an isomorphism when $q = 1$, and is generally a good matching when the value of q is high. It will be used in the following in order to obtain a good lower bound of the optimal score of a matching.

4.2 Datasets Used in the Experiments

Our graphs are large to ensure that our approach is scalable as this is a major concern for most of available algorithms. They are relatively sparse since graphs from most real-life applications are not very dense; plus preliminary experiments showed that the more dense the graphs, the easier it was to match them with our approach. Both unlabeled and labeled graphs were considered to assess the efficiency of our approach with respect to the number of labels on edges. Finally the graphs to be matched are highly similar to ensure a high confidence in our computed lower bound.

For our experiments, we use the following values for the parameters:

- Number of vertices: $n = 300, 1000$ or 3000;
- Expected mean of in and out degree of a vertex: $d = 2$ or 5;
- Number of labels: $nl = 1$ or 4;
- Similarity parameter: $q = 0.8, 0.9$ or 1;

We have generated 36 pairs of graphs, one for each possible combination of parameters n, d, nl and q. The graphs are available online [3]. In addition, we consider two different evaluation functions denoted by f_0 or f_1. Function f_0 is suited for the MCES problem and corresponds to the number of edges correctly

matched in a configuration, while f_1 penalizes errors and represents the number of edges correctly matched, minus the number of incorrect edge correspondences (label and structural errors).

4.3 Results Obtained by the Enhanced Tabu Algorithm

With each pair of graphs and each evaluation function, we have performed a series of 20 runs by using the tabu procedure initialized with the solution returned by the greedy procedure (GS+TS). The parameters of the TS algorithm were set as follows: *max_iter_fail*=$n/2$, *lgtl_in*=10, and *lgtl_out*=5. The algorithms coded in C++ was compiled with g++ and run on a Linux Dual Processor Opteron 64-bit with 16 Gb RAM running Redhat Advanced Server version 4. Results obtained during these experiments are presented in Tables 1 and 2.

Table 1. Results obtained with labeled graphs ($nl = 4$)

name	$GS(f_0)$			$GS+TS(f_0)$					$GS(f_1)$			$GS+TS(f_1)$				
	f_{avg}	f_{std}	cpu	iter	f_{avg}	f_{std}	**succ**	cpu	f_{avg}	f_{std}	cpu	iter	f_{avg}	f_{std}	**succ**	cpu
300_2_0.8_4	*99*	0	0	0	*99*	0	**0**	1	*99*	0	0	73	*124*	0	**20**	1
300_2_0.9_4	*100*	0	0	0	*100*	0	**20**	1	*100*	0	0	7	*103*	0	**20**	1
300_2_1_4	*100*	0	0	0	*100*	0	**20**	1	*100*	0	0	0	*100*	0	**20**	1
300_5_0.8_4	*100*	0	1	0	*100*	0	**20**	1	*100*	0	1	49	*116*	0	**20**	2
300_5_0.9_4	*100*	0	1	0	*100*	0	**20**	1	*100*	0	1	4	*100*	0	**20**	1
300_5_1_4	*100*	0	1	0	*100*	0	**20**	1	*100*	0	1	0	*100*	0	**20**	1
1000_2_0.8_4	*99*	0	4	0	*100*	0	**0**	5	*100*	1	4	322	*124*	0	**20**	10
1000_2_0.9_4	*100*	0	4	0	*100*	0	**7**	6	*100*	0	4	84	*104*	0	**20**	7
1000_2_1_4	*100*	0	4	0	*100*	0	**14**	6	*100*	0	4	4	*100*	0	**18**	6
1000_5_0.8_4	*99*	0	10	0	*99*	0	**0**	11	*96*	0	10	167	*110*	0	**20**	17
1000_5_0.9_4	*100*	0	9	0	*100*	0	**20**	11	*100*	0	9	6	*100*	0	**20**	12
1000_5_1_4	*100*	0	9	0	*100*	0	**20**	11	*100*	0	9	0	*100*	0	**20**	11
3000_2_0.8_4	*100*	0	45	87	*100*	0	**0**	52	*98*	1	45	1757	*133*	0	**20**	106
3000_2_0.9_4	*100*	0	43	12	*100*	0	**0**	50	*100*	0	43	346	*105*	0	**20**	63
3000_2_1_4	*100*	0	44	0	*100*	0	**20**	52	*100*	0	44	6	*100*	0	**20**	65
3000_5_0.8_4	*99*	0	87	0	*99*	0	**0**	93	*97*	0	87	471	*112*	0	**20**	157
3000_5_0.9_4	*100*	0	89	0	*100*	0	**20**	95	*100*	0	89	26	*100*	0	**20**	113
3000_5_1_4	*100*	0	89	0	*100*	0	**20**	96	*100*	0	90	0	*100*	0	**20**	102

Table 1 presents the results obtained with labeled graphs ($nl = 4$). Each line in the table corresponds to an instance. The left part of the table corresponds to $f = f_0$ and the right part to $f = f_1$. In each part, we display information about the greedy procedure (GS), and then about the whole algorithm (GS+TS).

Results are evaluated relatively to the lower bound and are expressed in percentages of the lower bound. A score f such that $f < 100$ means that the lower bound has not been reached. We consider a run to be a "success" if the score of the solution returned by the algorithm is not worse than the lower bound ($f \geq 100$). *succ* indicates the number of successes within 20 runs of the algorithm. f_{avg} and f_{std} are respectively the mean and standard deviation of the scores on the 20 runs. *cpu* indicates the mean of the cpu time spent on the algorithm. In the case of GS+TS, we present an additional information *iter* which is the number of iterations of the tabu before it finds the best solution returned. When *iter* is 0, it means the tabu could not improve on the solution returned by the greedy procedure.

Let us first consider the left part of the table ($f = f_0$). For all 36 pairs of graphs, the displayed value of f_{avg} varies between 99 and 100. Notice that, for simplicity, the values are rounded. As a result, a value of 100 in column "f_{avg}" means in fact that $99.5 \leq f_{avg} < 100.5$. It explains why, in several cases, the displayed value of f_{avg} is 100 but $succ = 0$. When we observe the f_{avg} reached by the greedy procedure, we can see that these values are very close to the final value of f_{avg}. In addition, the value of *iter* equals 0 in most cases: this indicates that the tabu algorithm could not improve on the quality of the solution obtained by the greedy procedure.

In summary, the results (normalized average value of the score) obtained by the GS+TS algorithm are generally very close to 100, but it was already true for the solution found by the greedy procedure.

Let us turn to the right part of the table ($f = f_1$). In this case, the lower bound is often far from the optimum, which explains that the average score sometimes takes values that are much higher than 100 (up to 133). We can notice that the number of successes equals 20 in most cases. We notice that the tabu algorithm was in many cases able to improve significantly over the solution produced by the greedy procedure. This is because although the solution returned by the greedy procedure contains most exact edge correspondences, it also contains many errors which are penalised by the f_1 score function.

When $q = 1$ (isomorphic graphs), we know that the lower bound is in fact the optimum. In this case, we can observe that our algorihm obtained consistently optimal solutions. When $q < 1$, the lower bound may not be equal to the optimum. However, we notice that the standard deviation of the score function is always smaller than 0.5. This may suggest that the scores of the solutions produced by our algorithm are very close to the optimum.

Table 2 presents the results obtained with unlabeled graphs ($nl = 1$). These results are far from being as good as the ones obtained with labeled graphs

Table 2. Results obtained with unlabeled graphs ($nl = 1$)

name	$GS(f_0)$			$GS+TS(f_0)$					$GS(f_1)$			$GS+TS(f_1)$				
	f_{avg}	f_{std}	cpu	iter	f_{avg}	f_{std}	**succ**	cpu	f_{avg}	f_{std}	cpu	iter	f_{avg}	f_{std}	**succ**	cpu
300_2_0.8_1	*59*	6	1	52	*60*	6	**0**	1	*-181*	40	1	640	*95*	34	**8**	6
300_2_0.9_1	*85*	10	0	3	*85*	10	**0**	1	*44*	37	0	73	*104*	0	**20**	1
300_2_1_1	*100*	0	0	0	*100*	0	**20**	1	*100*	0	0	0	*100*	0	**20**	1
300_5_0.8_1	*96*	0	1	0	*96*	0	**0**	1	*77*	0	1	51	*109*	0	**20**	2
300_5_0.9_1	*100*	0	1	0	*100*	0	**20**	1	*100*	0	1	3	*101*	0	**20**	1
300_5_1_1	*100*	0	1	0	*100*	0	**20**	1	*100*	0	1	0	*100*	0	**20**	1
1000_2_0.8_1	*60*	13	33	203	*60*	13	**0**	35	*-130*	74	34	2178	*73*	40	**7**	144
1000_2_0.9_1	*91*	11	5	21	*91*	11	**0**	6	*66*	43	5	315	*99*	17	**19**	18
1000_2_1_1	*99*	1	4	4	*99*	1	**5**	5	*97*	3	4	11	*100*	0	**14**	6
1000_5_0.8_1	*50*	24	10	23	*50*	24	**0**	11	*-202*	146	10	1779	*57*	50	**9**	171
1000_5_0.9_1	*42*	23	10	114	*42*	23	**0**	12	*-124*	89	10	1576	*58*	47	**5**	145
1000_5_1_1	*100*	0	7	0	*100*	0	**20**	8	*100*	0	7	0	*100*	0	**20**	8
3000_2_0.8_1	*54*	5	83	243	*54*	5	**0**	390	*-108*	17	83	1817	*64*	31	**4**	385
3000_2_0.9_1	*90*	3	43	144	*91*	3	**0**	50	*62*	14	43	1283	*106*	0	**20**	75
3000_2_1_1	*99*	0	39	4	*99*	0	**0**	45	*98*	1	40	17	*100*	0	**17**	292
3000_5_0.8_1	*36*	21	127	100	*36*	21	**0**	135	*-305*	102	127	2231	*27*	46	**4**	432
3000_5_0.9_1	*88*	3	72	212	*88*	3	**0**	79	*55*	13	72	286	*100*	0	**20**	103
3000_5_1_1	*100*	0	65	0	*100*	0	**7**	72	*99*	1	66	4	*100*	0	**20**	78

when using score function f_0. Notice however that the result obtained with isomorphic graphs are still very good. Results with $q = 0.9$ are also generally good while $q = 0.8$ gave bad results. When using the score function f_1, the algorithm obtained much better results than with f_0 but these results were less consistent than those obtained with labeled graphs.

4.4 Results Obtained without Using the Greedy Procedure

In addition to the results presented above, we have also tested the tabu procedure when it is initialized by using an empty solution. All the results obtained by using the same problem instances were very poor. Therefore, we have tested the procedure by using the same parameters except the number n of vertices in the two graphs. The tested values were 20, 50 and 100. While there were a few successes for $q = 1$ and $d = 5$, the results were overwhelmingly very poor with averages mostly under 50%.

5 Conclusion

In this paper, we propose a new technique aimed at enhancing a local search heuristic for the approximated graph matching (AGM) problem. The principle of this technique is to guide the search by using a similarity measure computed between the nodes of two graphs. We observe that a tabu search heuristic using the objective function obtains very poor results even on small graphs. We thus propose a new similarity measure which is used intensively in the early stages of the matching process in order to get the search in a good area. We use a greedy procedure based on that principle to initialize the tabu search. We tested our algorithm on a large number of random graphs of various sizes, densities (up to 3000 vertices and 15000 edges). Results appear consistently good, especially on labeled graphs and the computing times are relatively low (from less than 1s to 200s).

Future work will be devoted to (i) investigating other similarity measures and ways of using them (iii) applying the algorithm on real-life graphs taken from different research fields (iii) extending our graph generator in order to produce a database for approximate graph matching.

References

1. Abi-Antoun, M., Aldrich, J., Nahas, B., Schmerl, N., Garlan, D.: Differencing and merging of architectural views. Automated Software Engineering 15(1), 35–74 (2008)
2. Barecke, T., Detyniecki, M.: Memetic algorithms for inexact graph matching. In: IEEE Congress on Evolutionary Computation, CEC 2007, pp. 4238–4245 (2007)
3. Benchmark for AGM, http://web.soccerlab.polymtl.ca/~sekpo/
4. Bunke, H.: Error-tolerant graph matching: a formal framework and algorithms. In: Proc. Advances in Pattern Recognition, pp. 1–14 (1998)

5. Crescenzi, P., Kann, V.: Approximation on the web: a compendium of np optimization problems. In: Rolim, J.D.P. (ed.) RANDOM 1997. LNCS, vol. 1269, pp. 111–118. Springer, Heidelberg (1997)
6. Eshera, A.A., Fu, K.S.: A similarity measure between attributed relational graphs for image analysis. In: Proc. 7th Intl Conf. on Pattern Recognition, pp. 75–77 (1984)
7. Foggia, P., Sansone, C., Vento, M.: A database of graphs for isomorphism and subgraph isomorphism benchmarking. In: Proc. 3rd IAPR TC-15 Intl Workshop Graph-Based Representations in Pattern Recognition, pp. 176–187 (2001)
8. Glover, F.: Tabu search-part i. ORSA Journal on Computing 1(3), 190–206 (1989)
9. Gold, S., Rangarajan, A.: A graduated assignment algorithm for graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence 18(4), 377–388 (1996)
10. Kpodjedo, S., Ricca, F., Galinier, P.: Recovering the evolution stable part using an ecgm algorithm: Is there a tunnel in mozilla? In: European Conf. on Software Maintenance and Reengineering, vol. 0, pp. 179–188 (2009)
11. Miller, G.L.: Graph isomorphism, general remarks. Journal of Computer and System Science 18(2), 128–142 (1979)
12. Raymond, J.W., Gardiner, E.J., Willett, P.: Rascal: calculation of graph simi- larity using maximum common edge subgraphs. Computer Journal 45(6), 631–644 (2002)
13. Toshev, A., Jianbo, S., Daniilidis, K.: Image matching via saliency region correspondences. In: CVPR 2007, IEEE Conf. on Computer Vision and Pattern Recognition, pp. 33–40 (2007)
14. Wang, Y., Makedon, F., Ford, J., Huang, H.: A bipartite graph matching framework for finding correspondences between structural elements in two proteins. In: Proc. Int. Conf. of the IEEE Engineering in Medecine and Biology Society, pp. 2972–2975 (2004)
15. Williams, M.L., Wilson, R.C., Hancock, E.R.: Deterministic search strategies for relational graph matching. In: Pelillo, M., Hancock, E.R. (eds.) EMMCVPR 1997. LNCS, vol. 1223, pp. 261–275. Springer, Heidelberg (1997)

Characterizing Fault-Tolerance of Genetic Algorithms in Desktop Grid Systems

Daniel Lombraña González[1], Juan Luís Jiménez Laredo[2], Francisco Fernández de Vega[1], and Juan Julián Merelo Guervós[2]

[1] University of Extremadura
Avda. Sta. Teresa de Jornet n°38 06800, Mérida, Spain
{daniellg,fcofdez}@unex.es
[2] University of Granada. ATC-ETSIIT
Periodista Daniel Saucedo Aranda s/n 18071, Granada, Spain
{jmerelo,juanlu}@geneura.ugr.es

Abstract. This paper presents a study of the fault-tolerant nature of Genetic Algorithms (GAs) on a real-world Desktop Grid System, without implementing any kind of fault-tolerance mechanism. The aim is to extend to parallel GAs previous works tackling fault-tolerance characterization in Genetic Programming. The results show that GAs are able to achieve a similar quality in results in comparison with a failure-free system in three of the six scenarios under study despite the system degradation. Additionally, we show that a small increase on the initial population size is a successful method to provide resilience to system failures in five of the scenarios. Such results suggest that Paralle GAs are inherently and naturally fault-tolerant.

1 Introduction

Genetic Algorithms (GAs) are a sub-class of Evolutionary Algorithms mainly used to solve optimization problems. As the complexity of the problems increases, GAs require a larger amount of computing resources. The more complex the instance, the larger the computing requirements. This fact leads to a sometimes prohibitively long time to solution that happens, for example, when tackling many real-world problems. In order to reduce the execution time of GAs, many research efforts have been focused during the last decades in approaching GAs in a parallel fashion. There are two main adventages of exploiting the inherent parallelism of GAs: (i) the computing load is balanced among different processors speeding-up the execution time, and (ii) the structural changes that the algorithm suffers when deployed in parallel are able to outperform the sequential counterpart (see for instance [1]).

Parallel algorithms, and thus parallel GAs, must be executed on platforms that consists of multiple computing elements or *processors*. In that sense, one of the most popular distributed systems are the Desktop Grid Systems (DGSs). The term "desktop grid" is used to refer to distributed networks of heterogeneous single systems that contribute spare processor cycles for computing. Perhaps the

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 131–142, 2010.

most well known desktop grid system is the Berkeley Open Infrastructure for Network Computing (BOINC) [2], which has among other projects the successful one: SETI@Home[1] . These systems are also known as *volunteer grids* because they aggregate the computing resources (desktop computers from offices or homes) that volunteers worldwide willingly donate to different research projects like SETI. One of the most important features of DGS is that they provide large-scale parallel computing capabilities, only for specific types of applications, at a very low cost.

Therefore DGSs can provide parallel computing capabilities for running demanding parallel Genetic Algorithms (PGAs), as in the case of e.g. Milky-Way@Home project [3]. But with large scale comes a higher likelihood that processors suffer a failure, interrupting the execution of the algorithm or crashing the whole system (in this paper we use the term "failure" and do not make the subtle distinction between "failure" and "fault", which is not necessary for our purpose).

Failures usually occur on large-scale distributed or parallel systems [4]. In DGSs, computers join the system, contribute some resources and leave it afterwards causing a collective effect known as churn [5]. Churn is an inherent property of DGSs and has to be taken into account in the design of applications. From the point of view of the application in a DGS, a failure occurs each time the CPU becomes busy (DGSs only use the idle cycles of CPU) or the computer is powered off or restarted.

To cope with failures, researchers have studied and developed different mechanisms to circumvent the failures or restore the system once a failure occurs. These techniques are known as *Fault-Tolerance mechanisms* and enforce that an application behave in a well-defined manner when a failure occurs [6]. Nevertheless, to the best of our knowledge, not many efforts have been applied to study the fault tolerance features of PEAs in general, and of PGAs in particular.

In previous works [7,8] we firstly analyzed the fault-tolerance nature of Parallel Genetic Programming (PGP) under several simplified assumptions. These initial results suggested that PGP exhibit a fault-tolerant behavior by default, encouraging to go a step further and run PGP on large-scale computing infrastructures that are subject to failures without requiring the employment of any fault-tolerance mechanism. This work was lately improved [9] by studying the fault-tolerance nature of PGP using real data from one of the most high churn distributed systems: the Desktop Grids. The results again showed that PGP can cope with failures without using any fault-tolerance mechanism, concluding that PGP is fault tolerant by nature since it implements by default the fault-tolerance mechanism called *graceful degradation* [10].

This paper builds on top of the previous work, and extends the study of fault-tolerance in EA to the PGAs in order to know if PGAs can be run in parallel or distributed systems without having to implement any fault-tolerance mechanism. To this aim, we have chosen a fine-grained master-worker model of parallelization [1]. A server, "the master", runs the main algorithm and hosts

[1] http://setiathome.berkeley.edu

the whole population. The server is in charge of sending non-evaluated individuals to workers in order to obtain the fitness value of them. This approach is effective because one of the most time-consuming steps of GAs is the evaluation –fitness computation– phase. The master waits until all individuals in generation n are evaluated before going to the next generation $n + 1$ and run the genetic operations. We assume that the system only suffers from omission failures [10]: (i) the master sends N individuals with $N > 0$ to a worker, and the worker never receives them, e.g., due to network transmission problems; or (ii) the master sends N individuals with $N > 0$ to a worker, the worker receives them but never returns them. This can occur because the worker crashes or the returned individuals are lost during the transmission.

In order to study the behavior of PGAs under the previous assumptions, we are going to simulate the failures using real-world traces of host availability from three DGSs. We have chosen Desktop Grid availability data because these systems exhibit large amounts of failures, and thus if it is possible to run inside them PGAs without using any fault-tolerance mechanism, PGAs will be able to exploit any parallel or distributed systems to its maximums.

The rest of the paper is organized as follows. Section 2 reviews related work; section 3 presents the setup of the different scenarios and experiments; section 4 shows the obtained results and their analysis; and, finally, Section 5 concludes the paper with a discussion of the results and future directions.

2 Related Work

Parallel Evolutionary Algorithms (PEAs) have been used by researchers as a common approach to reduce the large time to solutions requirements of hard problems. For instance, Trujillo et al. propose a computer vision problem addressed using GP in [11] that needs more than twenty four hours in order to obtain a solution in a single computer. Times to solution can be even worst, lasting days, weeks or even months. For this reason, researchers have studied the application of parallel computing techniques and distributed computing platforms in conjunction with Spatially Structured EAs to reduce the time to solution [1,12,13].

A distributed system can be defined based on its logical or functional distribution of processing capabilities [10]. The logical distribution is typically based on the following set of criteria:

- *Multiple processes.* The system consists of one or more sequential processes, each with an independent thread of control.
- *Interprocess communication.* Processes can communicate via messages.
- *Disjoint address spaces.* Processes have disjoint address spaces.
- *Collective goal.* Processes interact among each other in order to meet a common goal.

PEAs structured like the previous criteria have been developed and used for decades (i.e. [1]).

So far, EA researchers have not employed massively DGSs. Nevertheless, there are several projects using DGSs like the MilkyWay@Home project [3] which uses GAs to create an accurate 3D model of the Milky way, a ported version of LilGP [14] (a framework for GP [15]) to one of the most employed DGSs, BOINC [2], or the *custom execution environment* facility proposed and implemented by Lombraña et. al. [16,17] for BOINC.

Other EA researchers have focused their attention on P2P systems [18], which are very similar to DGSs because the computing elements are also desktop computers in its majority. However these systems are different because there is not a central server as in DGSs.

In all the described proposals –to the best of our knowledge– none of them have specifically addressed the problem of failures within PGAs. Nevertheless, some of those solutions internally employ some fault-tolerance mechanisms. In this sense, only Laredo et al. have analyze the resilience to failures of a parallel Genetic Algorithm in [19], following the Weibull degradation of a P2P system (failures are the host-churn behavior of these systems as well as DGSs) proposed by Stutzbach and Rejaie in [5]. Therefore, PGAs have not been analyzed before under real host availability traces (a.k.a. host-churn). Hence, this paper assesses fault tolerance in PGAs using host-churn data collected in three real-world DGSs [20]. The key contribution of this paper is the fully characterization of PGAs from the point of view of fault-tolerance with the aim of studying if PGAs can be run in parallel or distributed systems without using any fault-tolerance mechanism.

3 Experimental Setup

In order to study the fault-tolerance nature of PGAs, we are going to simulate the failures on a distributed system using three different traces from three real-world DGSs. The use of simulations allows tractability of the results making possible the statistical analysis. Furthermore, the experiments are repeatable thanks to the employment of host availability traces [20], allowing a fair comparison among experiments.

To this aim, we conduct experiments in a 3-trap instance [21]. According to [22], 3-trap lies on the region between the deceptive 4-trap and the non-deceptive 2-trap having, therefore, intermediate population size requirements that Thierens estimates in 3000 for the instance under study in [23]. A trap function is a piecewise-linear function defined on unitation (the number of ones in a binary string). There are two distinct regions in the search space, one leading to a global optimum and the other one to the local optimum (see Eq. 1). In general, a trap function is defined by the following equation:

$$trap(u(\overrightarrow{x})) = \begin{cases} \frac{a}{z}(z - u(\overrightarrow{x})), & \text{if} \quad u(\overrightarrow{x}) \leq z \\ \frac{b}{l-z}(u(\overrightarrow{x}) - z), & \text{otherwise} \end{cases} \tag{1}$$

where $u(\overrightarrow{x})$ is the unitation function, a is the local optimum, b is the global optimum, l is the problem size and z is a slope-change location separating the attraction basin of the two optima.

For the following experiments, 3-trap was designed with the following parameter values: $a = l - 1$, $b = l$, and $z = l - 1$. Tests were performed by juxtaposing $m = 10$ trap functions in binary strings of length $L = 30$ and summing the fitness of each sub-function to obtain the total fitness. All settings are summarized in Table 1.

Table 1. Parameters of the experiments

Trap instance	
Size of sub-function (k)	3
Number of sub-functions (m)	10
Individual length (L)	30

GA settings	
GA	GGA
Population size	3000
Selection of Parents	Binary Tournament
Recombination	Uniform crossover, $p_c = 1.0$
Mutation	Bit-Flip mutation, $p_m = \frac{1}{L}$

In order to achieve our goal of characterizing the fault-tolerant nature of PGAs two type of experiments were performed: (i) using an error-free assumption which implies that the system does not suffer failures during the execution of the PGA and (ii) an error-prone environment simulated by means of real-world traces from three different real DG systems. In this last scenario, the PGA does not employ any fault-tolerance mechanism to cope with failures and uses the parallelism at individual level. If the quality of solutions in the error-prone environment is similar to the error-free case, it will be a strong indication that PGAs are fault-tolerant by nature.

Simulations use three different traces: [20]: *ucb*, *entrfin*, and *xwtr*. The traces are time-stamped observations of the host availability. The *ucb* trace was obtained in a graduate student lab in the EE/CS Department at UC Berkeley for about 1.5 months with 85 hosts. The *entrfin* trace was collected at the San Diego Supercomputer Center for about 1 month with 275 hosts, and finally the *xwtr* trace was obtained at the Universitè Paris-Sud for about 1 month with 100 computers (check [20] for a full description of the measurement methods and traces).

Fig. 1 shows an example of the availability data from the *ucb* trace: the number of available computers in the system versus time over 24 hours. The figure shows the host-churn phenomena of these systems: hosts becoming available after being unavailable for a period of time. The experiments were carried out over such 24-hour periods of the availability traces. We fixed the maximum number of generations to 30 (which corresponds to 3 hours of execution in a failure-free system).

We have considered two scenarios for the error-prone case: (i) firstly we run the experiments under a stringent assumption (lost resources never become

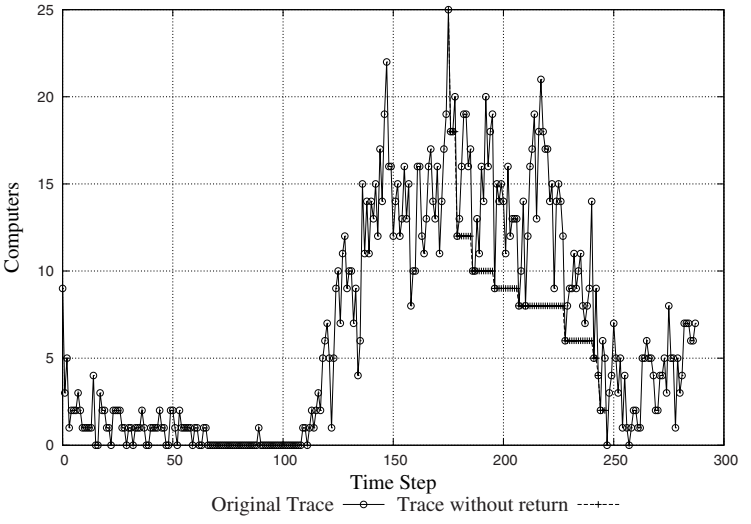


Fig. 1. Host availability for 1 day of the *ucb* trace

available again); (ii) and secondly we consider the full host-churn phenomena (host availability) allowing to re-acquire the lost hosts.

For both cases, we use two different 24-hour periods from each trace. For the first case, the worst-case scenario, we arbitrarily select the time within each 24-hour trace segment where there are the maximum number of available hosts, starting at that point the execution of the algorithm (see Fig. 1 labeled with "trace without return"). For the second case, we simply chose a random point in each of the 24-hour periods for starting the execution of the algorithm.

The server divides its population into the number of available hosts at the first generation, and sends I individuals to each worker. When a worker fails, I individuals are lost and will not take part in the next generations due to our worst-case assumption.

For all the experiments the server does not detect failures and no fault tolerance mechanisms are employed (e.g., no replication of individuals over multiple workers). The server waits a time T per generation based on the time required for a generation in the failure-free scenario, and proceeds to the next generation with the available individuals at that time. Bear in mind that the execution, which may obtain worse results in comparison with the error-free environment because of the lost individuals, becomes progressively less computation-demanding as population size gets smaller progressively (see Fig. 2).

The execution times per generation in both experiments, failure-free and failure-prone, are identical: with P individuals to be evaluated at a given generation and W workers, we send $\frac{P}{W} = I$ individuals to each worker; when a worker fails $\frac{P}{W}$ individuals are lost; given that those individuals will not participate in the next generation, the remaining workers will continue evaluating I individuals each, regardless of the number of failures.

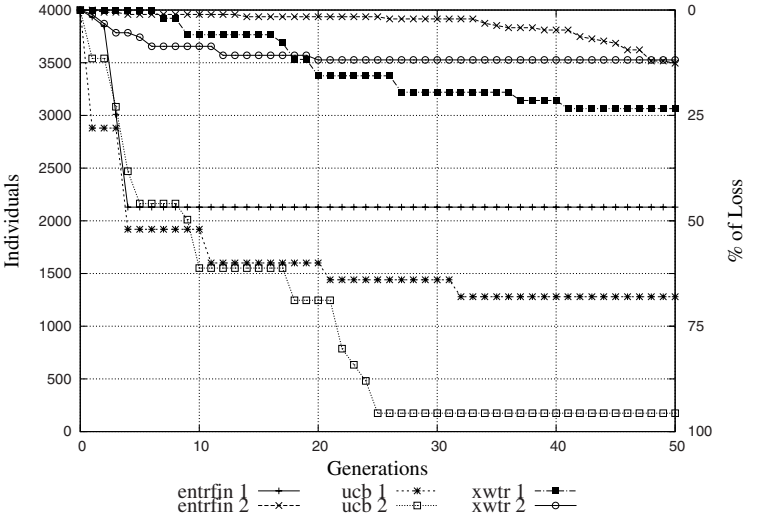


Fig. 2. Population size vs. generation

4 Analysis of Results

In order to analyze the results with confidence, data has been statistically analyzed (each experiment has been run 100 times). Firstly, we analyzed the normality of the data using the Kolgomorov-Smirnov and Shapiro-Wilk tests [24], obtaining as a result that all data are non-normal. Thus, to compare two samples, the error-free case with each trace, we used the Wilcoxon test (Tab. 2 shows the Wilcoxon analysis of the data).

Fig. 2 shows, for the worst-case scenario, how the population decreases as failures occur in the system. As explained before, two different 24-hours periods randomly selected are shown, denoted by Day 1 and Day 2, for the three employed traces of the experiments. Thus, a total of 6 different experiments, one per trace and day period, were run with the 3-Trap function problem.

Tab. 2 shows a summary of the obtained results for the experiments. From all the traces, the *ucb* has obtained the worst fitness 23.22 and 23.09 (respectively for both periods Day 1 and Day 2). The reason is that this trace in the first day loses a 64% of the population and in the second day it loses more or less the whole population 95.83% (see Fig. 2). Consequently it is very difficult for the algorithm to obtain a solution with a similar quality to the error-free scenario.

The second worst case of all the experiments is the *entrfin* trace for the first period (Day 1). This trace loses more or less half of the population in the first 5 generations (see Fig. 2), making really difficult to obtain a good solution even though the population size is steady the rest of the generations. Thus, the obtained fitness for this period is not comparable to the error-free case.

Table 2. 3-Trap fitness comparison between error-prone and error-free cases using Wilcoxon test (*Day 1 and 2*) – "not significantly different" means fitness quality comparable to the error-free case

Error Free fitness = 23.56

	Trace	Fitness	Wilcoxon Test	Significantly different?	Fitness	Wilcoxon Test	Significantly different?
Day 1	Entrfin	23.3	W = 6093, p-value = 0.002688	yes	**23.57**	**W = 4979.5, p-value = 0.9546**	**no**
	Entrfin 10%	**23.47**	**W = 5408.5, p-value = 0.2535**	**no**	**23.69**	**W = 4397.5, p-value = 0.07682**	**no**
	Entrfin 20%	**23.48**	**W = 5360, p-value = 0.3137**	**no**	**23.67**	**W = 4522.5, p-value = 0.1645**	**no**
	Entrfin 30%	**23.49**	**W = 5283.5, p-value = 0.4271**	**no**	**23.70**	**W = 4405, p-value = 0.08086**	**no**
	Entrfin 40%	**23.57**	**W = 4923.5, p-value = 0.8286**	**no**	**23.69**	**W = 4453.5, p-value = 0.11**	**no**
	Entrfin 50%	**23.59**	**W = 4910.5, p-value = 0.7994**	**no**	23.75	W = 4162.5, p-value = 0.01234	yes
	Ucb	23.22	W = 6453, p-value = 6.877e-05	yes	23.09	W = 6672.5, p-value = 7.486e-06	yes
	Ucb 10%	23.27	W = 6098.5, p-value = 0.002753	yes	23.12	W = 6826, p-value = 6.647e-07	yes
	Ucb 20%	23.37	W = 5837.5, p-value = 0.02051	yes	23.14	W = 6654, p-value = 7.223e-06	yes
	Ucb 30%	**23.40**	**W = 5664, p-value = 0.06588**	**no**	23.26	W = 6371, p-value = 0.0001507	yes
	Ucb 40%	**23.51**	**W = 5186.5, p-value = 0.6004**	**no**	23.37	W = 5893.5, p-value = 0.01316	yes
	Ucb 50%	**23.42**	**W = 5623, p-value = 0.08335**	**no**	23.32	W = 6108, p-value = 0.002166	yes
	Xwtr	**23.56**	**W = 5056, p-value = 0.8748**	**no**	**23.60**	**W = 4806, p-value = 0.5791**	**no**
	Xwtr 10%	**23.57**	**W = 4923.5, p-value = 0.8286**	**no**	**23.62**	**W = 4765, p-value = 0.5002**	**no**
	Xwtr 20%	**23.68**	**W = 4474, p-value = 0.1245**	**no**	**23.69**	**W = 4453.5, p-value = 0.11**	**no**
	Xwtr 30%	23.73	W = 4259.5, p-value = 0.02812	yes	**23.60**	**W = 4806, p-value = 0.5791**	**no**
	Xwtr 40%	**23.68**	**W = 4502, p-value = 0.1466**	**no**	**23.63**	**W = 4688.5, p-value = 0.3695**	**no**
	Xwtr 50%	**23.71**	**W = 4356.5, p-value = 0.05817**	**no**	23.77	W = 4065.5, p-value = 0.004877	yes
	Results with Host Churn						
Day 1	Entrfin	**23.52**	**W = W = 5222, p-value = 0.5322**	**no**	**23.58**	**W = 4931, p-value = 0.8452**	**no**
	Ucb	21.31	W = 9708.5, p-value < 2.2e-16	yes	23.03	W = 7038.5, p-value = 4.588e-08	yes
	Xwtr	**23.64**	**W = 4640, p-value = 0.2982**	**no**	**23.7**	**W = 4405, p-value = 0.08086**	**no**

Finally, the *xwtr* trace in both periods obtains solutions with similar quality to the error-free environment (23.56 and 23.6 respectively for each day). In both periods, the *xwtr* trace does not lose more than a 20% for Day 1 and 12% for the second day. Consequently, we conclude that for the 3-Trap function problem, it is possible to tolerate a gradual loss of up to 20% of the individuals without sacrificing solution quality and more importantly without using any fault-tolerance mechanism. Nevertheless, if the loss of individuals is too high, above the 45%, the solution quality is significantly diminished. Since real-world DGSs experience such large amount of failures, we attempt to address this problem. Our simple idea is to increase the initial population size (a 10%, 20%, 30%, 40% and 50%) and run the same simulations using the same traces. The aim is to compensate the loss of the system by providing more individuals at the first generation.

Tab. 2 shows the obtained results for Day 1 and Day 2 periods of the three traces with the increased population. For the *entrfin* trace, the first period (Day 1) with a loss rate of 45.3%, a 10% extra individuals is enough to obtain solutions of similar quality to the error-free case. In the second period, Day 2, the trace obtains similar solutions to the error-free case and when adding an extra 50% the obtained solution is even better than in the error-free case.

For the *ucb* trace, the first period (Day 1) increasing a 30% the size of the population is sufficient to obtain solutions with similar quality to the error-free case. The second period, Day 2, even though an extra 50% of individuals is added at the first generation it is not enough to cope with the high loss rate of this period: 95.83%.

Finally, the *xwtr* trace for both periods obtains solutions with similar quality to the error-free case and in some cases it improves it. For this trace, the increased population would have not been necessary because the PGA tolerates, without any extra individual, the rate loss of both periods.

It is important to remark that by adding more individuals to the initial population, we are increasing the computation time since more individuals have to be evaluated per generation. Nevertheless, this extra time is similar to the extra time that would be required by standard fault tolerance mechanisms (e.g. failure detection and re-send lost individuals for fitness evaluation). Thus, we conclude that increasing the population size, accordingly to the failure rate, is enough to improve the PGA quality of solutions when the failure rate is known.

Up to now, we have only considered the worst-case scenario: lost resources never become available again. Nevertheless, real-world DG systems does not behave like this assumption, and thus we are going to use the traces with the possibility of re-acquiring the lost resources (see Fig. 1).

When using the full churn traces of the three DGSs (*entrfin*, *ucb* and *xwtr*) an important question arises: what work is assigned to the new available workers? We have assumed that when workers become available again the master node creates I new random individuals and increases the size of the population accordingly. Thus, the size of the population can be changed dynamically as individuals are added and removed along generations. In this scenario it could happen that new workers nodes appear during the execution of the algorithm increasing the population over its optimum size. Hence the master node is not allowed to create more individuals than the optimum population size leaving several workers idle. In order to avoid idle workers, it would be interesting to adjust the number of I individuals to evaluate accordingly to the number of available hosts. Nevertheless, we leave such load-balancing study for a future work.

On the other hand, due to the loss of resources, the population can be emptied because all the workers have disappeared. If this situation occurs, the server node proceeds to the next generation by waiting the specified time T (based on the required time per generation in the failure-free environment) for new workers.

Tab. 2 shows the obtained results for the three traces with the host-churn phenomena (*entrfin*, *ucb* and *xwtr*) and the previous corresponding two periods: Day 1 and Day 2. We used the same periods as in the worst-case scenario, but now choosing a random point in the 24-hours period as the starting point for the algorithm. Tab. 3 shows the obtained fitness of the 3-Trap function problem and the host churn of each trace represented by the minimum, median, mean, maximum, and variance of the number of available worker nodes.

If the variance of the number of available hosts is zero, then the execution is obviously the same as in the error-free case because the number of hosts is steady along generations. In this case, the obtained fitness should be similar to the error-free case. This situation is present within the second period (Day 2) of the *xwtr* trace (variance equal to zero) and thus the obtained fitness is similar to the error-free case (see Tab. 2). The other period of the *xwtr* trace has also a very small variance, 0.11, resulting in a similar solution quality in comparison with the error-free scenario. The *entrfin* trace for both periods obtains solutions of similar quality to the error-free environment, even though the large variance observed in the Day 1 period ($s^2 = 305.59$). Despite the large variance,

Table 3. Obtained fitness for 3-Trap function with host churn

Trace	Hosts					Fitness
	Min.	Median	Mean	Max.	Var. (s^2)	3-Trap
Error free	-	-	-	-	-	23.56
entrfin (*Day 1*)	92	161.5	156.8	177	305.59	23.52
entrfin (*Day 2*)	180	181	180.9	182	0.6	23.58
ucb (*Day 1*)	0	2	1.9	9	3.12	21.31
ucb (*Day 2*)	0	4	3.7	7	2.7	23.03
xwtr (*Day 1*)	28	29	28.87	29	0.11	23.64
xwtr (*Day 2*)	86	86	86	86	0	23.70

the number of available hosts is high in comparison with the other traces, so the PGA tolerates better the failures and provides solutions of similar quality to the error-free case. Finally, the *ucb* trace obtains the worst results due to in both periods the minimum number of available hosts is zero. Consequently, the population is emptied, making very difficult to obtain solutions of similar quality to the error-free environment.

4.1 Summary of Results

We have studied the fault-tolerance nature of PGA for the 3-Trap function problem with parallelism at individual level using real-world traces from Desktop Grid Systems. We have simulated the failures using real data from three different traces under two different 24-hour periods and compared the obtained solutions with the error-free case. For all the experiments the PGA has not used any fault-tolerance mechanism.

Firstly, we studied the fault-tolerance nature of PGAs under a stringent assumption: resources cannot be re-acquired after a failure. In this scenario the obtained results suggest that PGAs for the 3-Trap function problem tolerates a loss rate up to 20% of individuals. As DGSs can exhibit larger amounts of failures, we introduce a simple solution to cope with failures when the failure rate is known. The method consists in increasing the population size in the first generation in order to compensate the individuals that are going to be lost along generations. The method showed that thanks to this approach it is possible to cope with failures, obtaining solutions of similar quality to the error-free environment. For this first scenario we realized that there is an approximately linear degradation of solution quality as host losses increases.

Secondly, we analyzed the behavior of PGAs for the 3-Trap function under the same traces but using the host-churn phenomena: resources can become available after being unavailable. In this case results exhibit a dynamic degradation due to the number of available hosts variates along generations. In both cases PGA shows to be fault-tolerant by exhibiting a *graceful degradation* [10].

5 Conclusions

In this work we have analyzed the behavior of Parallel Genetic Algorithms (PGAs) running in Desktop Grid Systems with high failure rates in order to fully characterize the fault tolerant features of PGAs. We have employed a well-known problem, 3-trap function, and real world traces from three different desktop grid systems. Our main conclusion is that PGAs inherently provides *graceful degradation* without the requirement of implementing any fault-tolerance mechanism. Additionally, we have also presented a simple method for coping with errors that consist in increasing the initial population size (accordingly to the errror rate of the system) to compensate the gradual loss of individuals along generations when the error rate is known. To the best of our knowledge this is the first time that PGAs are fully characterized from the point of view of fault-tolerance.

Acknowledgments

This work was supported by University of Extremadura, regional government Junta de Extremadura, National Nohnes project TIN2007-68083-C02-01 Spanish Ministry of Science and Education and Junta de Andalucia CICE project P06-TIC-02025.

References

1. Tomassini, M.: Spatially Structured Evolutionary Algorithms. Springer, Heidelberg (2005)
2. Anderson, D.: Boinc: a system for public-resource computing and storage. In: Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, 2004, pp. 4–10 (2004)
3. Desell, T., Szymanski, B., Varela, C.: An asynchronous hybrid genetic-simplex search for modeling the Milky Way galaxy using volunteer computing. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 921–928. ACM, New York (2008)
4. Schroeder, B., Gibson, G.A.: A Large-Scale Study of Failures in High-Performance Computing Systems. In: Proc. of the International Conference on Dependable Systems, pp. 249–258 (2006)
5. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proceedings of the 6th ACM SIGCOMM on Internet measurement (IMC 2006), pp. 189–202. ACM Press, New York (2006)
6. Gartner, F.C.: Fundamentals of fault-tolerant distributed computing in asynchronous environments. ACM Computing Surveys 31(1), 1–26 (1999)
7. Lombraña, D., Fernández, F.: Analyzing fault tolerance on parallel genetic programming by means of dynamic-size populations. In: Congress on Evolutionary Computation, Singapore, September 2007, vol. 1, pp. 4392–4398 (2007)
8. Hidalgo, I., Fernández, F., Lanchares, J., Lombraña, D.: Is the island model fault tolerant? In: Genetic and Evolutionary Computation Conference, London, England, July 2007, vol. 2, p. 1519 (2007)

9. González, D.L., de Vega, F.F., Casanova, H.: Characterizing fault tolerance in genetic programming. In: Workshop on Bio-Inspired Algorithms for Distributed Systems, Barcelona, Spain, pp. 1–10 (2009)
10. Ghosh, S.: Distributed systems: an algorithmic approach. Chapman & Hall/CRC, Boca Raton (2006)
11. Trujillo, L., Olague, G.: Automated Design of Image Operators that Detect Interest Points, vol. 16, pp. 483–507. MIT Press, Cambridge (2008)
12. Cantu-Paz, E.: A survey of parallel genetic algorithms. Calculateurs Paralleles, Reseaux et Systems Repartis 10(2), 141–171 (1998)
13. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5), 443–462 (2002)
14. de la, O.F.C., Guisado, G.L., Lombraña, D., Fernández, F.: Una herramienta de programación genética paralela que aprovecha recursos públicos de computación. In: V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, Tenerife, Spain, February 2007, vol. 1, pp. 167–173 (2007)
15. Bill Punch, D.Z.: Lil-gp, http://garage.cse.msu.edu/software/lil-gp/index.html
16. Lombraña, D., Fernández, F., Trujillo, L., Olague, G., Cárdenas, M., Araujo, L., Castillo, P., Sharman, K., Silva, A.: Interpreted applications within boinc infrastructure. In: Ibergrid 2008, Porto, Portugal, May 2008, pp. 261–272 (2008)
17. González, D.L., de Vega, F.F., Trujillo, L., Olague, G., Araujo, L., Castillo, P., Merelo, J.J., Sharman, K.: Increasing gp computing power for free via desktop grid computing and virtualization. In: Proceedings of the 17th Euromicro Conference on Parallel, Distributed and Network-based Processing, Weimar, Germany, February 2009, pp. 419–423 (2009)
18. Laredo, J.L.J., Eiben, A.E., van Steen, M., Merelo, J.J.: On the run-time dynamics of a peer-to-peer evolutionary algorithm. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 236–245. Springer, Heidelberg (2008)
19. Laredo, J.L.J., Castillo, P.A., Mora, A.M., Merelo, J.J., Fernandes, C.: Resilience to churn of a peer-to-peer evolutionary algorithm. Int. J. High Performance Systems Architecture 1(4), 260–268 (2008)
20. Kondo, D., Fedak, G., Cappello, F., Chien, A., Casanova, H.: Characterizing resource availability in enterprise desktop grids, vol. 23, pp. 888–903. Elsevier, Amsterdam (2007)
21. Ackley, D.H.: A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, Norwell (1987)
22. Deb, K., Goldberg, D.E.: Analyzing deception in trap functions. In: Whitley, L.D. (ed.) FOGA, pp. 93–108. Morgan Kaufmann, San Francisco (1992)
23. Thierens, D.: Scalability problems of simple genetic algorithms. Evolutionary Computation 7(4), 331–352 (1999)
24. Crawley, M.J.: Statistics, An Introduction using R. Wiley, Chichester (2007)

The Office-Space-Allocation Problem in Strongly Hierarchized Organizations

Rui Lopes and Daniela Girimonte

Infrastructures and Facilities Management Division,
European Space Research and Technology Center

Abstract. The office-space-allocation (OSA) problem will be introduced for strongly hierarchized organizations. In several organizations the relation between the entities[1] is strongly hierarchized, affecting the design and handling of constraints and algorithms used to solve the problem. Moreover there is also an increase in the constraint number when compared to the common test instances. Several well known meta-heuristics were used, new constraints developed, and some variations to the local search algorithms were studied. This article describes the work done and its application to a particular case study, the European Space Research and Technology Center (ESTEC).

1 Introduction

Very dynamic organizations such as the European Space Agency (ESA) are faced with continuous restructuring of their human resources: new employees arrive, others leave or change their work appointment, and possibly also their position in the hierarchy. The facility managers are asked to have these changes reflected on the allocation of the office space. Moreover in many organizations the increase in workforce is not always accompanied by an enhancement and/or adaptation of the infrastructures. The scope of the activity of this organization is quite broad and it is extremely important to keep the diverse teams working on different matters/subjects coherently grouped. ESTEC is the biggest site of ESA, presenting a problem with very high dimensions (around 2000 employees and 1500 offices). These characteristics make the problem of office space allocation a hard and long process for a human being. State of the art optimization techniques can be used to save resources and optimize the space usage. The problem is common in several organizations and consists of allocating office space amongst the entities, satisfying a given set of constraints. The goal is to optimize the space usage, satisfying a set of hard constraints and as many soft constraints as possible[1,2].

[1] The term "entities" is used to designate every kind of entity (employees, printers, laboratories,etc); In the test instance in the scope of this article all the entities are employees and so this term will also be used throughout the text; the reader should understand "entities" when such happens.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 143–153, 2010.

The approach taken was to implement four well known meta-heuristics similarly to [2]. To improve the performance of the algorithms, based on the qualitative and quantitative analysis of the obtained solutions, variations to local search, and new constraints handling procedures were developed. Experiments were carried out to compare the performance of the original algorithms with the new versions.

The remaining of the article is organized as follows. In Sect. 2 the general formulation of the problem is given and the problem domain and test instance are described. Section 3 discusses the implementation of the algorithms, followed by the modifications that were implemented, presenting and comparing the results of the different approaches. Finally, a summary and some remarks are given in Sect. 4.

2 The OSA Problem

The OSA problem is a highly constrained combinatorial optimization problem. It consists in allocating a set of resources (rooms) to a number of entities (employees, printers, laboratories and/or others) [3], and it may arise in any institution/organization, whenever the resources are scarce and there is a need to optimize the space usage, having into account a, usually heavy, set of constraints. The goal is then to find an allocation that optimizes the space usage satisfying a set of hard constraints and as many soft constraints as possible. These constraints restrict the positioning and grouping of the entities and are detailed in Sect. 2.2. In this domain, constraints of proximity/adjacency, grouping, and sharing are common. The first type is set when one entity should be adjacent to a determined room or entity; the second type refers to a group of entities that should be allocated close to each other; finally, the sharing constraints indicate whether an entity should or not share an office with another entity.

The following subsections present the problem formulation and the case study description.

2.1 General Formulation

This problem is common in the literature and its formulation is similar to other combinatorial optimization problems like the bin packing, the general assignment problem and others [4]. This is a multi-objective optimization problem, commonly formulated as a bi-objective problem. Following the work of [2], the problem is formulated as follows.

There are n entities, with sizes s_1, s_2, ..., s_n, and m rooms with capacities c_1, c_2, ...,c_m. A matrix X of $[x_{\mathrm{i,j}}]$ values is defined to represent the solution, where $x_{\mathrm{i,j}}= 1$ if the entity j (e_j) is assigned to room i (r_i). The constraints are distinguished between hard and soft, having h hard constraints that must be satisfied, and g soft constraints, which should be satisfied.

The problem's goal is to minimize, over x, the function

$$F(x) = f_1(x) + f_2(x) \tag{1}$$

given that for each entity there is only one assigned resource and all the hard constraints are satisfied. The functions $f_1(x)$ and $f_2(x)$ are, respectively, the space-misuse, and the violation-of-soft-constraints functions, as described in [2]:

$$f_1(x) = \sum_{i=1}^{m} WP_i + \sum_{i=1}^{m} OP_i \tag{2}$$

$$f_2(x) = \sum_{r=1}^{g} SCP_r \tag{3}$$

where WP_i *and* OP_i represent, respectively, the amount of space wasted or overused for each room i; SCP_r the penalty for violating the r^{th}soft constraint. For each room only one of WP_i *or* OP_i has a value greater than zero, and the overusing of a room is considered worse than the waste of its space:

$$WP_i = max(\,0,\, c_i - \sum_{j=1}^{n} x_{ij} s_j\,) \tag{4}$$

$$OP_i = max(\,0,\, 2(\sum_{j=1}^{n} x_{ij} s_j - c_i)\,) \tag{5}$$

A candidate solution to this problem can be modeled as a n-dimensional vector $Y = [y_1, y_2, ..., y_n]$ where the j^{th}position in the vector corresponds to e_j and the value $y_j = i$ allocates e_j to the r_i room. This structure reduces the amount of memory needed to represent the allocation solutions and also assures that each entity is assigned only one room.

2.2 ESTEC Case Study

In ESA, as mentioned before, there is a well defined and strong hierarchization amongst the entities. These have two properties related to hierarchy: standard and address code. The standard refers to the category of the entity, based on which size and sharing properties are defined. For instance, an employee may have the category "Head of Section", which makes him entitled to a single office (non sharing property) of 18 m^2. The address code reflects the hierarchy of the business units of the agency , and defines the grouping logic for the entities. As an example, if an employee has address code "TIA-ABC" he/she belongs to section ABC of TIA directorate. The remaining of this section explains in detail how these properties relate and affect the office space allocation problem. The problem instance used in this article is a subset of the ESTEC instance, correspondent to the TIA directorate of ESA. The following paragraphs describe the test instance.

Entities and Resources Data
The data set of this instance offers 109 resource units (m = 109), with different characteristics, to be allocated to 105 entities (n = 105).

The address code property is a character sequence, from five to seven characters size, of the form "TIA-ABC". "TIA" defines the directorate the entity belongs to[2], "TIA-A" the department, "TIA-AB" the division and, finally, "TIA-ABC" the section. Every entity is associated to one and only one address code, at any of the described levels, and shall be grouped with all the other entities of the same address code. To obtain a coherent allocation the grouping must work also when one goes up the abstraction. That is, the sections of some division, the divisions of the same department, and the departments belonging to a directorate should be close to each other.

The existent standards, which define the entities size and sharing attributes, are summarized in Table 1.

Table 1. ESA standards and corresponding office space and share attributes

Standard	Space (m^2)	Share
Head of Directorate	39	No
Head of Department	29	No
Head of Division	24	No
Head of Section	18	No
Staff A Senior	16	No
Staff A, B and B Sr.	12	No
Others	8	Yes

The resources are identified by a character sequence with fixed length of five characters, in the form "BA301". The first character identifies the building, the second the region of the building (usually from A to F), the third the floor of the building, and, at last, the remaining two characters give the room number. Each room in the data set has two lists of adjacent rooms; one is a regular adjacency list comprising all the offices in the close surroundings of the office; the other, which will be called "strict adjacency list" has only the rooms that are literally adjacent, that is, which share a wall with the current room. The latter was introduced because there is a constraint imposed by the stakeholders that the secretaries must stay immediately adjacent to their chiefs.

Constraints

In this section the constraints present in the test instance will be described[3]. There is a clear distinction between "global constraints" and "particular constraints", where a global constraint applies to every entity and a particular constraint applies to a specific subject and/or target. Table 2 summarizes the particular constraints that can be found in the data set and Table 3 the global

[2] The entities employed at the directorate level are exceptions and have address code "D/TIA".

[3] all the types described in [2] can be used, but this instance does not have a constraint of each type.

Table 2. Particular constraints applied to the test instance

Constraints	Hard	Penalty (SCP)	Description
be located in	1	10	entity allocated in the designated room
be adjacent to	0	50	entity adjacent to other entity

Table 3. Global constraints applied to the test instance

Constraints	Hard	Penalty (SCP)	Description
Grouping (5)	0	1	Group entities at department level
Grouping (6)	0	3	Group entities at division level
AvoidSpreading (5)	0	50	Avoid splitting of the department over several buildings/floors
AvoidShareAcross (5)	0	3	Avoid sharing of office between entities of different departments
Sharing	1	1	All the entities whose standard states not to share, **must be** allocated alone

constraints. The penalty values were determined by trial end error. The paragraphs that follow detail the implementation.

The description of the particular constraints is self-explanatory but few remarks must be added about the adjacency constraint. This constraint uses the strict adjacency list, because this is what corresponds to the decision makers expectations. As explained before, this constraint is mostly used to place the secretaries *immediately adjacent* to their administrators. Moreover, the adjacency constraints defined in the data are soft, when ideally they would be hard constraints. This was decided because, given the usual local search algorithms, the entities involved in such constraints would have the same allocation from the initial solution until the final, making it too rigid for in-field usage of the algorithms[4].

The global constraints were developed for the domain here described and need some more detail. As mentioned before, the standard property of the entities defines whether or not an entity should share an office. In order to have a feasible solution, according to ESTEC rules, all entities which are entitled to a single office must not share and the remaining must share. That is why the "Sharing"constraint is hard. However, because we could have an odd number of sharing entities, or to prevent worse grouping, the latter are not forced to share. If the constraint is used as soft (Hard = 0) then a penalty is applied for any entity whose share condition does not correspond to its standard.

All the other global constraints applied have a number characterizing it. These constraints compute based on the address code of the entities and can operate at different levels of the character string, corresponding to the different levels of

[4] The analysis of the generated solutions suggests that the best algorithms comply with the soft adjacency constraints almost all the times, and results in better grouping when compared to the usage of such constraints as hard.

the agency hierarchy (see the previous section). To have a coherent grouping, by the agency standards, all the entities in one division should be close to each other (Grouping(6), i.e.at division level). The Grouping(5), i.e. at department level, is used to enforce grouping between divisions, trying to avoid the mixing of divisions from different departments. In some tests that were made, although the total grouping penalty was smaller when compared to a manual allocation, it would always split the departments through a bigger number of floors/buildings, which pleases less the decision makers. In order to improve this aspect the constraint AvoidSpreading(5), i.e. at department level, is used, applying a penalty proportional to the number of different floors/buildings used by a department. Finally, the optimization of the space usage would make the algorithm pair entities from different departments. However, for the allocation to be coherent it is preferable to have some space wastage (free space) than to allocate two entities from different departments. For this reason, the constraint AvoidShareAcross(5), i.e. at department level, applies a penalty to these situations.

Some new constraints, specific to the present problem domain, have been implemented to generate solutions closer to the stakeholders expectations and demands. Some common constraints to OSA domains were also applied. The global constraints apply to all entities, so one could say that a global constraint corresponds to n particular constraints, where n is the number of entities. The next section reveals the algorithms that were used and the improvement attempts over these, providing a comparison of the results.

3 Algorithms and Results

Four known meta-heuristics were implemented: Hill-Climbing or Iterative Improvement (II) [5], Simulated Annealing (SA) [6], Tabu Search (TS) [7], and the Hybrid Meta-heuristic (HMH) [1]. These meta-heuristics perform the optimization using local search operators, namely swap, relocate, and interchange[5]. This approach is based on the work developed in [2], and its test instances were used to verify the base library used to implement the solvers. The results obtained with these algorithms, as they are described in the literature, are presented in Table 4, for different run lengths. The values presented were obtained by averaging 10 runs. It is not a goal to describe here the algorithms that can be found in several published articles.

Similarly to results found on the literature, the Hybrid Meta-heuristic outperforms the other algorithms if the run length is large enough. For the shortest run length it is outperformed by the Tabu Search, which has a faster convergence. The Simulated Annealing performance is comparable to the Iterative Improvement, up to 5000 iterations. When the run is larger the SA will continue improving, opposite to the II which will get stuck (expected due to the SA algorithm).

[5] The swap operator, swaps the allocated room between two entities; the relocate operator relocates an entity to a different room; the interchange changes all the entities in one room to another room and vice-verse.

Table 4. Average results obtained with the four algorithms, without variations, for different run lengths. The best results are emphasized.

Length	**10000 iterations**			
Algorithms	II	SA	TS	HMH
Space misuse (f_1)	449.37	440.26	430.08	416.81
Soft constraints (f_2)	203.57	201.87	206.22	209.52
Total (f)	652.93	642.13	636.30	**626.33**
Length	**5000 iterations**			
Algorithms	II	SA	TS	HMH
Space misuse (f_1)	436.08	440.11	450.11	424.42
Soft constraints (f_2)	209.42	213.17	209.82	207.88
Total (f)	645.50	653.28	659.93	**632.30**
Length	**1050 iterations (10*n)**			
Algorithms	II	SA	TS	HMH
Space misuse (f_1)	442.36	451.93	438.87	432.68
Soft constraints (f_2)	242.51	231.40	219.54	244.19
Total (f)	684.87	683.32	**658.40**	676.87
Best of all	597.15	589.74	**585.49**	604.30

The next sections detail the variations developed, their purpose, and the results achieved.

3.1 Variations to Local Search (LS) Operators

When analyzing the obtained solutions to evaluate the grouping quality, several divisions could be found to be split because of two or three sequential entities (see Sect. 2.2) that were allocated in a different zone, although also grouped with each other. This suggests that a variation to the Local Search operators, which are used by all the mentioned meta-heuristics, allowing the recombination of sequences of entities/resources, could be beneficial. In order to verify this, two variations of the swap and relocate operators were implemented to allow recombination of sequences. A property, *length*, was added to the local search and experiments were run in two modalities: fixed length and variable length. In the former all the operations will involve a sequence of *length* entities and resources, and in the latter the length of the involved sequences is randomly generated every iteration between 1 and *length*.

The following experiments were carried on: modified relocate operator with fixed and variable length (see Table 5 on the following page); modified swap operator with fixed and variable length (see Table 6 on page 151); both variations to local search at the same time, with variable and fixed length (see Table 7 on page 151). Finally, a local search with five operators - the standard three plus both variations to relocate and swap with variable length 2 - was attempted (see Table 8 on page 152). The next paragraphs comment on the obtained results.

Both the TS and HMH algorithms with the fixed length relocate (Table 5) operator perform worse than the original. The Simulated Annealing performance on the other hand has increased mainly because of better space use. The results show that the most robust algorithms perform worse with the fixed length extended operator but the SA algorithm outperforms the original results.

With the variable length operator the results are very similar to those obtained with the original versions.

Table 5. Average results obtained with the four algorithms for the variation relocate operator with fixed and variable length 2. The best results are emphasized.

Operator	**Relocate Fixed Length 2**				**Relocate Variable Length 2**			
Run Length	10000							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	438.52	420.6	441	423.75	449.09	438.78	426.59	404.94
Soft constraints (f_2)	206.67	206.89	212.12	256.4	225.04	206.05	212.48	218.37
Total (f)	645.2	**627.48**	653.12	680.15	674.13	644.83	639.07	**623.30**
Run Length	5000							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	443.56	425.56	425.68	418.17	440.77	436.47	431.74	425.78
Soft constraints (f_2)	203.99	215.02	233.46	288.9	223.78	208.56	216.57	219.15
Total (f)	647.55	**640.58**	659.14	707.07	664.55	645.03	648.31	**644.93**
Run Length	1050 (10*n)							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	438.64	441.2	449.68	430.9	439.80	443.86	444.83	435.51
Soft constraints (f_2)	221.77	251.07	219.11	304.03	238.28	246.07	213.84	266.53
Total (f)	**660.42**	692.27	668.78	734.93	678.08	689.93	**658.67**	702.03
Best of all	602.81	**597.06**	604.29	643.33	584.67	590.77	601.66	597.74

The fixed length swap operator (Table 6) worsened the performance of every meta-heuristic. Despite this, the SCP(f_2) achieved is worse than the original, but the space misuse is better. The HMH outperforms every algorithm with this variation.

The variable length swap operator achieves better results that the fixed. In comparison to the original algorithms, HMH modified version is slightly better and TS kept its original performance. Again the SA algorithm is the one that deals better with the modified operator since it had the biggest quality increase. HMH outperforms all other algorithms for the largest run, even the original versions and the modified variable relocate operator (see Table 5).

When both fixed length relocate and swap operators (Table 7 on page 151) are used, the algorithms perform worse (as expected, since by them selves it was not better). Apparently, none of the advantages of the operators alone (better soft penalty for relocate and better space usage for swap) is maintained when put to work at the same time - because they complement each other. The most affected with decreasing performance were TS and, surprisingly, also HMH.

Table 6. Average results obtained with the four algorithms for the variation operator Swap with fixed and variable length 2. The best results are emphasized.

Operator	**Swap Fixed Length 2**				**Swap Variable Length 2**			
Run Length	10000							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	428.79	429.74	429.41	408.67	435.79	419.02	423.46	403.71
Soft constraints (f_2)	220.2	216.77	224.07	229.69	213.14	195.81	214.89	207.92
Total (f)	648.99	646.52	653.48	**638.36**	648.93	614.83	638.35	**611.63**
Run Length	5000							
Algorithms	II	SA	TS	633.67	II	SA	TS	HMH
Space misuse (f_1)	428.66	433.43	434.62	412.64	441.83	423.31	436.85	409.24
Soft constraints (f_2)	225.64	227	219.12	234.83	214.17	225.74	210.95	209.30
Total (f)	654.3	660.42	653.74	**647.47**	655.99	649.05	647.80	**618.54**
Run Length	1050 (10*n)							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	436.4	447.73	430.32	434.16	436.90	425.92	435.05	431.07
Soft constraints (f_2)	250.34	245.41	234.05	270.72	235.90	227.50	223.32	272.29
Total (f)	686.74	693.13	**664.36**	704.88	672.80	**653.43**	658.37	703.36
Best of all	**599.29**	600.33	609.18	611.78	602.16	**572.06**	595.11	589.04

Table 7. Average results obtained with the four algorithms for the variation operator Relocate with fixed length 2. The best results are emphasized.

Operator	**S&R Fixed Length 2**				**S&R Variable Length 2**			
Run Length	10000							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	427.27	421.07	447.05	425.94	424.13	432.65	413.22	411.76
Soft constraints (f_2)	225.53	233.34	230.42	286.16	223.82	201.86	226.42	221.10
Total (f)	**652.80**	654.41	677.47	712.11	647.95	634.51	639.64	**632.86**
Run Length	5000							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	436.43	438.96	440.33	441.52	428.05	433.19	435.05	417.91
Soft constraints (f_2)	221.50	231.12	216.12	316.97	213.31	208.67	218.41	232.06
Total (f)	657.94	670.08	**656.44**	758.49	**641.36**	641.86	653.46	649.97
Run Length	1050 (10*n)							
Algorithms	II	SA	TS	HMH	II	SA	TS	HMH
Space misuse (f_1)	435.28	440.76	435.87	428.97	440.03	441.31	424.72	430.61
Soft constraints (f_2)	248.14	263.51	250.61	356.10	229.46	248.67	243.43	282.91
Total (f)	**683.42**	704.27	686.48	785.07	669.50	689.98	**668.15**	713.52
Best of all	610.42	629.15	**591.33**	681.16	**585.11**	593.75	597.24	604.30

The results for the variable length operators suggest that TS and II reach a local optimum before 5000 iterations, and its performance is comparable to the original. The SA and HMH perform better than the original, combining

Table 8. Average results obtained with the four algorithms for the local search with five operators: the three typical relocate, swap, and interchange, and both variations to operator relocate and swap with variable length 2. The best results are emphasized.

Algorithms	II	SA	TS	HMH
Run Length	10000			
Space misuse (f_1)	438.31	418.66	442.76	417.06
Soft constraints (f_2)	210.10	205.85	217.95	215.42
Total (f)	648.40	**624.51**	660.71	632.48
Run Length	5000			
Space misuse (f_1)	424.19	427.34	427.03	425.41
Soft constraints (f_2)	217.17	221.42	213.95	209.36
Total (f)	641.36	648.76	640.98	**634.77**
Run Length	1050 (10*n)			
Algorithms	II	SA	TS	HMH
Space misuse (f_1)	440.16	435.64	422.38	448.85
Soft constraints (f_2)	230.82	258.67	260.41	254.27
Total (f)	**670.98**	694.31	682.79	703.12
Best of all	609.15	**587.00**	608.36	631.74

benefits of both modified operators by improving the grouping (specially the HMH) and the space usage. The HMH outperforms every algorithm and the original versions.

Joining the best performing operators variations (Table 8 on page 152) with the standard local search, an improvement relative to the original results is obtained. This improvement is significant in the SA case. However, it does not outperform the results with the variations as replacement for the standard operators. The benefits from the swap variation (better space usage) can be seen on the results, but a better grouping should be expected when using the variable length relocate. Also the best solution found is slightly better than the original version, but its quality decreased when compared to the experiments with the variation operators as replacement for the standard ones.

As can be seen with such results, the local search can be improved, based on the recombination of solutions. As in the genetic algorithms concept, parts of the chromosomes (solutions) can be recombined, based on an extended local search, improving the quality of the obtained results. The next section summarizes the work described so far and provides guidelines for further investigation.

4 Summary and Final Remarks

Organizations such as the European Space Agency (ESA) are extremely dynamic and are faced with continuous restructuring of their human resources. The facility managers are asked to have these changes reflected on the allocation of the office space. State of the art optimization techniques can be used to save resources and optimize the space usage. Following an approach similar to [2], despite of known

results in this domain when using genetic algorithms with recombination being weaker than local search [8], it is shown here that it is possible to extend the local search operators towards recombination operators, improving the obtained results, and outperforming the original local search.

However, the application of such changes is not straightforward. Several approaches were tried and the different results compared. First of all, fixed length operators yielded bad quality results . When testing the swap and relocate variable length operators individually, one can see an improvement at different levels, and the final quality of the solution is better than the original one.Individually, both variable length operators improve on the original results specially by reducing, respectively, the soft constraints penalty and the space usage. When both modifications are applied together (Table 7 on page 151) there is no further improvement in comparison with the individual trials for most of the algorithms. If the original local search operators are used in conjunction with both variations the obtained results are similar. One can notice that there is a better space usage in SA and HMH, which is compensated with a not so good soft constraints penalty. This suggests that other combinations of operators should be tried.

To conclude, the extension of the local search operators used in meta-heuristic search has the potential to outperform the original local search, by recombining bigger, variable length, parts of solutions during the process. Future work should also include an extension to the interchange operator, combinations of this with the remaining operators, and different combinations of the original local search operators with the extended operators.

References

1. Burke, E.K., Cowling, P., Silva, J.D.L.: Hybrid population-based metaheuristic approaches for the space allocation problem. In: Proceedings of the 2001 Congress on Evolutionary Computation CEC 2001, COEX, World Trade Center, 159 Samseongdong, Gangnam-gu, Seoul, Korea, 27-30, 2001, pp. 232–239. IEEE Press, Los Alamitos (2001)
2. Silva, J.D.L., Burke, E.K.: Asynchronous cooperative local search for the office-space-allocation problem. INFORMS Journal on Computing 19(4), 575–587 (2007)
3. Lee, J.: A First Course in Combinatorial Optimization. Cambridge University Press, Cambridge (2004)
4. Hansen, M.P.: Metaheuristics for multiple objective combinatorial optimization. PhD thesis (1998)
5. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach. Prentice Hall, Englewood Cliffs (2003)
6. Kirkpatrick, S., Gelatt, C.D., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
7. Glover, F., Laguna, M.: Tabu search (1997)
8. Burke, E.K., Cowling, P., Landa Silva, J.D., McCollum, B.: Three methods to automate the space allocation process in UK universities. In: Burke, E., Erben, W. (eds.) PATAT 2000. LNCS, vol. 2079, pp. 254–273. Springer, Heidelberg (2001)

A Study of Memetic Search with Multi-parent Combination for UBQP

Zhipeng Lü[1], Jin-Kao Hao[1], and Fred Glover[2]

[1] LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France
[2] OptTek Systems, Inc., 2241 17th Street Boulder, CO 80302, USA
lu@info.univ-angers.fr, hao@info.univ-angers.fr, glover@opttek.com

Abstract. We present a multi-parent hybrid genetic–tabu algorithm (denoted by GTA) for the Unconstrained Binary Quadratic Programming (UBQP) problem, by incorporating tabu search into the framework of genetic algorithm. In this paper, we propose a new multi-parent combination operator for generating offspring solutions. A pool updating strategy based on a quality-and-distance criterion is used to manage the population. Experimental comparisons with leading methods for the UBQP problem on 25 large public instances demonstrate the efficacy of our proposed algorithm in terms of both solution quality and computational efficiency.

Keywords: UBQP, Memetic Algorithm, Tabu Search, Genetic Algorithm, multi-parent combination.

1 Introduction

The unconstrained binary quadratic programming problem may be written

$$\text{UBQP: Maximize } f(x) = x'Qx$$
$$x \text{ binary}$$

where Q is an n by n matrix of constants and x is an n-vector of binary (zero-one) variables.

The formulation of UBQP is notable for its ability to represent a wide range of important problems, including those from financial analysis [1], computer aided design [2], traffic management [3], machine scheduling [4]), cellular radio channel allocation [5] and molecular conformation [6]. Moreover, many combinatorial optimization problems pertaining to graphs such as determining maximum cliques, maximum cuts, maximum vertex packing, minimum coverings, maximum independent sets, maximum independent weighted sets are known to be capable of being formulated by the UBQP problem as documented in [7]. A review of additional applications and formulations can be found in [8].

Given the interest in the UBQP and its NP-hard nature [9], a large number of solution procedures have been reported in the literature. Some representative examples include exact algorithms (such as [7,10]), local search based approaches (such as direct local search [11], Simulated Annealing [12,13,14] and Tabu Search

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 154–165, 2010.

[13,15,16,17,18]) and population-based approaches (such as Evolutionary Algorithms [19,20,21,22], Scatter Search [23] and Memetic Algorithms [24]).

In the current paper, we study a memetic algorithm for the UBQP, which integrates a tabu search procedure with a genetic search approach. The proposed algorithm is characterized by several original features. First, we introduce a "logic" combination operator using multiple parents called MSX to produce a combination scheme that more fully exploits the problem structure within the present context. Second, the proposed MSX operator is jointly employed with the conventional uniform crossover to generate diversified new solutions. Finally, our algorithm relies on a quality-and-distance replacement strategy for population updates to maintain the population diversity.

To assess the performance and the competitiveness of our memetic algorithm in terms of both solution quality and efficiency, we provide computational results on the 10 largest benchmark instances with 2 500 variables from ORLIB as well as 15 larger instances with up to 5 000 variables, comparing our outcomes with the best results of the literature.

The remaining part of the paper is organized as follows. In Section 2, our memetic algorithm is described, including the tabu search procedure, the multi-parent combination operator and the pool updating rule. Sections 3 is dedicated to the computational results and concluding remarks are given in Section 4.

2 Hybrid Genetic–Tabu Algorithm

2.1 Main Scheme and Initial Population

Memetic algorithms such as hybrid evolutionary algorithms are known to be highly effective for solving a large number of constraint satisfaction and optimization problems [25]. By combining the more global recombinant search and the more intensive local search, the memetic framework is expected to offer a better balance between the exploration and exploitation of the search space.

In principle, our genetic-tabu algorithm (GTA) repeatedly alternates between a combination operator that is used to generate new offspring solutions and a tabu search procedure that optimizes the newly generated offspring solutions. As soon as an offspring solution is improved by tabu search, the population is accordingly updated based on two criteria: the solution quality and the diversity of the population.

The general framework of our GTA algorithm is described in Algorithm 1. GTA contains four main components: population initialization, a tabu search procedure, a multi-parent combination operator and population updating. Starting from an initial random population, GTA uses the tabu search procedure to optimize each individual to reach a local optimum (see Sect. 2.2, lines 4-6 in Algorithm 1). Then, a combination operator is employed to generate new offspring solutions (see Sect. 2.3, line 10 in Algorithm 1), whereupon a new round of tabu search is launched to improve the new solutions. Subsequently, the population updating rule will decide whether an improved solution should be inserted into

Algorithm 1. Pseudo-code of the GTA algorithm for the UBQP problem

```
1: Input: matrix Q
2: Output: the best solution x* found so far
3: P = {x^1, ..., x^p} ← Population_Initialization( )
4: for i = {1, ..., p} do
5:    x^i ← Tabu_Search(x^i)
6: end for
7: x* = arg max{f(x^i)|i = 1, ..., p}
8: repeat
9:    randomly choose a subset of individuals E from P
10:   x^0 ← Combination_Operator(E)
11:   x^0 ← Tabu_Search(x^0)
12:   if f(x^0) > f(x*) then
13:      x* = x^0
14:   end if
15:   {x^1, ..., x^p} ← Pool_Updating(x^0, x^1, ..., x^p)
16: until a stop criterion is met
```

the population and which existing individual should be replaced (line 15 in Algorithm 1). Throughout the search process, x^* records the best solution found (lines 7, 12-14 in Algorithm 1).

The individuals of the initial population are generated randomly (i.e., each variable x_i of the n-vector x receives a value of 0 or 1 with equal probability). To build a diversified initial population, a new individual is added to the population only if it is not too *close* to any of the existing solutions of the population. The *distance threshold* for executing this rule is discussed in Section 2.3.

2.2 Tabu Search Procedure

As demonstrated in [15] and more recently in [16,17,18], TS is one of the more successful approaches for the UBQP. Our tabu search procedure uses a *neighborhood* defined by the simple *one-flip move*, which is widely used in local search algorithms for binary problems such as the UBQP problem and the satisfiability problem [26]. The one-flip move consists of changing (flipping) the value of a single variable x_i to its complementary value $1 - x_i$. The implementation of this neighborhood uses a fast incremental evaluation technique [15,27] to calculate the cost (move value) of transitioning to each neighboring solution.

More formally, let $N = \{1, \ldots, n\}$ denote the index set for components of the x vector. We preprocess the matrix Q to put it in lower triangular form by redefining (if necessary) $q_{ij} = q_{ij} + q_{ji}$ for $i > j$, which is implicitly accompanied by setting $q_{ji} = 0$ (though these 0 entries above the main diagonal are not stored or accessed). Let Δ_i be the move value of flipping the variable x_i, and let $q_{(i,j)}$ be a shorthand for denoting q_{ij} if $i > j$ and q_{ji} if $j > i$. Then each move value can be calculated in linear time using the formula:

$$\Delta_i = (1 - 2x_i)(q_{ii} + \sum_{j \in N, j \neq i, x_j = 1} q_{(i,j)}) \quad (1)$$

In addition, once a move is performed, one needs just to update a subset of move values affected by the move. Specifically, it is possible to update the move values upon flipping a variable x_i by performing the following abbreviated calculation:

1. $\Delta_i = -\Delta_i$
2. For each $j \in N - \{i\}$,
   $\Delta_j = \Delta_j + \sigma_{ij}\ q_{(i,j)}$
   where $\sigma_{ij} = 1$ if $x_j = x_i$, $\sigma_{ij} = -1$ otherwise.

We employ the convention that x_i represents x_i's value before being flipped.

TS typically incorporates a *tabu list* as a "recency-based" memory structure to assure that solutions visited within a certain span of iterations, called the tabu tenure, will not be revisited [28]. In our implementation, each time a variable x_i is flipped, a value is assigned to an associated record $TabuTenure(i)$ (identifying the "tabu tenure" of x_i) to prevent x_i from being flipped again for the next $TabuTenure(i)$ iterations. For the current study, we elected to set

$$TabuTenure(i) = tt + rand(10) \quad (2)$$

where tt is a given constant and rand(10) takes a random value from 1 to 10.

Our TS algorithm then restricts consideration to variables not currently tabu (by the criterion established by (2)), and selects a variable to flip that produces the best (largest) Δ_i value. In the case that two or more moves have the same best move value, a random best move is selected. Meanwhile, a simple aspiration criterion is applied that permits a move to be selected in spite of being tabu if it leads to a solution better than the current best solution.

Our TS method stops when the best solution cannot be improved within a given number α of moves and we call this number the *improvement cutoff*.

2.3 Combination Operator

In our GTA algorithm, we jointly use two kinds of combination operators to generate suitable offspring: one is the uniform crossover widely used in the literature; the other is a "logic" multi-parent combination operator proposed in this paper. At each iteration, we randomly choose one of these two operators with equal probability to generate new offspring solutions.

The main idea of uniform crossover is to assign values to the variables of offspring that represent assignments made in common by both parents, and to randomly assign values to remaining variables of the offspring solution [29]. In our case, the application of uniform crossover is controlled by the Hamming distance d_{ij} between two parent solutions x^i and x^j (i.e., d_{ij} equals the number of variables that receive different values in the parents. We require that two solutions chosen as parents must satisfy $d_{ij} > \overline{d}$, where $\overline{d}$ denotes the average

distance between pairs of solutions in the population. Therefore, we have $\overline{d} = \frac{2}{p(p-1)} \sum_{i=1}^{p} \sum_{j=i+1}^{p} d_{ij}$, where p denotes the population size.

Our "logic" multi-parent combination operator, called MSX, relies on information extracted from diversified and elite solutions. Let $E = \{x^{(1)}, \ldots, x^{(s)}\}$, where $x^{(i)} = (x_1^{(i)}, \ldots, x_n^{(i)})$ and the solutions in E are ordered in terms of their quality, i.e., $x^{(1)}$ is the best solution in E and $x^{(s)}$ is the worst. The value s giving the number of solutions in E is allowed to vary randomly between 4 and 8. E itself is generated by randomly selecting elements from the pool one at a time, subject to the restriction that each new element added to E must be separated by a distance of at least $\overline{d}$ from all elements of E previously added, as a basis for assuring the diversity of E. (In some cases, this requirement may compel E to have fewer than s elements.)

Associated with each $x^{(i)}$ in E, we identify the value

$$sum(i) = \sum_{j=1}^{n} x_j^{(i)} \tag{3}$$

and define a weight $w(i)$ for the solution $x^{(i)}$ as the inverse of $sum(i)$:

$$w(i) = 1/sum(i) \tag{4}$$

The weighted quantity $w(i)x_j^{(i)}$, which equals $w(i)$ if $x_j^{(i)} = 1$ and equals 0 otherwise, may be interpreted as the "relative contribution" of setting $x_j = 1$ in the solution $x^{(i)}$. In other words, if $x_j^{(i)} = 1$ for all $j \in N$ then each assignment $x_j = 1$ contributes only $1/n$ to the weighted quantity, whereas if $x_j^{(i)} = 1$ for only two $j \in N$ then each $x_j = 1$ assignment contributes $1/2$, disclosing that the relative contribution of any given assignment $x_j = 1$ in the latter solution is significantly greater than in the former.

For the elite set E ($|E| = s$), define the value $Strength(j)$ to be the weighted sum of the values $x_j^{(i)}$ (hence of the values $x_j^{(i)} = 1$) over the solutions $x^{(i)}$ in E:

$$Strength(j) = \sum_{i=1}^{s} w(i)x_j^{(i)} \tag{5}$$

The value $Strength(j)$ gives a relative indication of the tendency of the solutions in E to favor $x_j = 1$ or $x_j = 0$. That is, we may say that the larger the value of $Strength(j)$, the greater is the degree that "E favors $x_j = 1$". We allow the use of different weights for different solutions to reflect the fact that some solutions may deserve greater influence in determining the strength assigned a given variable than other solutions. The use of different weights also permits the use of strategies that amend the emphasis placed on various solutions as a function of search history, though we have not exploited this feature in the present study.

For the goal of generating a solution x from the set of solutions E, the value $Strength(j)$ by itself is not enough to determine that x_j should be 1

or 0. To make this determination, we need to order the vector $Strength$ ($= (Strength(1), \ldots, Strength(n))$) from its largest to smallest component:

$$Strength(j_1) \geq Strength(j_2) \geq \ldots \geq Strength(j_n) \tag{6}$$

To complete the determination of a suitable vector x to be derived from the set E, we select for the number of components x_j of x that should receive a value $x_j = 1$. We take an average of the $sum(i)$ values over E to get a value for the number of x_j components that should be 1 in an "average" solution. Specifically, let

$$Avg = \sum_{i=1}^{s} sum(i)/s \tag{7}$$

Making use of these elements, we can now compute the vector x created from combining the solutions of E as follows:

1. For each $j = j_k$, $k \leq Avg - r_1$, set $x_j = 1$;
2. For each $j = j_k$, $k \geq Avg + r_2$, set $x_j = 0$;
3. For each $j = j_k$, $Avg - r_1 < k < Avg + r_2$, randomly set $x_j = 1$ or $x_j = 0$.

where r_1 or r_2 denotes a randomly generated number from 3 to 10.

This idea is inspired from the intuition that it is preferable to shift Avg slightly in one direction or another to make the generation of offspring solutions more varied. In such a way, an offspring solution x is generated based on the elite set E, to which the tabu search procedure can be applied to further optimize the solution.

2.4 Pool Updating

In our algorithm, when an offspring x^0 is obtained by the combination operator, we improve x^0 by the tabu search algorithm and then decide whether the offspring should be inserted into the population, replacing the *worst* solution in the population. For this purpose, we define a quality-and-distance goodness score of the offspring x^0 with respect to the population.

The main idea is to favor the inclusion of x^0 in the population if x^0 is "good enough" (in terms of its objective function evaluation) and is not too *similar* to any solution currently in the population. In order to make things clearer, we make use of the following definitions:

Definition 1. Distance Between a solution and a Population: Given a population $P = \{x^1, \ldots, x^p\}$ and the distance d_{ij} between any two solutions x^i and x^j $(i, j = 1, \ldots, p, i \neq j)$, the distance between a solution x^i $(i = 1, \ldots, p)$ and the population P is defined as the minimum distance between x^i and any other solution in P, denoted by $D_{i,P}$:

$$D_{i,P} = min\{d_{ij} | x^j \in P, j \neq i\} \tag{8}$$

Definition 2. Goodness Score of a solution for a Population: Given a population $P = \{x^1, \ldots, x^p\}$ and the distance $D_{i,P}$ for any solution x^i ($i = 1, \ldots, p$), the goodness score of solution x^i for population P is defined as:

$$g(i, P) = \beta \widetilde{A}(f(x^i)) + (1 - \beta)\widetilde{A}(D_{i,P}) \tag{9}$$

where $f(x^i)$ is the objective function value of solution x^i and $\widetilde{A}(\cdot)$ represents the normalized function:

$$\widetilde{A}(y) = \frac{y - y_{min}}{y_{max} - y_{min} + 1} \tag{10}$$

where y_{max} and y_{min} are respectively the maximum and minimum values of y in the population P. The number "1" is used to avoid the possibility of a 0 denominator. β is a constant parameter and we empirically set $\beta = 0.6$ in this paper.

It is reasonable that the greater the goodness score $g(i, P)$, the better solution x^i, since we should not only maintain a pool of good quality solutions but also emphasize the importance of the diversity of the solutions to avoid a premature convergence of the population. Therefore, if the goodness score of the offspring solution is good enough, it will have high probability to replace the worst solution in the population. Interested readers are referred to [30] for more details about this quality-and-distance based pool updating strategy.

3 Experimental Results

3.1 Instances and Experimental Protocol

To assess the efficiency of our proposed GTA algorithm, we carry out experiments on two sets of benchmarks. The first set of benchmarks is composed of the 10 largest instances of size $n = 2\,500$ introduced in [13] and available in the ORLIB [31]. These instances are used in the literature by many authors (e.g., [13,14,16,17,18,24]). Note that the small test instances from the ORLIB whose sizes range from n=50 to 1 000 present no challenge for our GTA algorithm, since all their best known results can be obtained within 2 seconds by our algorithm. The second set of benchmarks consists of a set of 15 randomly generated large problem instances named p3000.1,. . .,p5000.5 with sizes ranging from n=3 000 to 5 000 [16,17]. These instances are available at: http://www.soften.ktu.lt/∼gintaras/ubqop_its.html.

Our algorithm is programmed in C and compiled using GNU GCC on a PC running Windows XP with Pentium 2.66GHz CPU and 512M RAM. Given the stochastic nature of our GTA procedure, each problem instance is independently solved 20 times. For the instances with 2 500, 3 000, 4 000 and 5 000 variables, the CPU time limit is set to be 40, 500, 800 and 1 500 seconds, respectively.

3.2 Computational Results and Comparisons

Based on preliminary testing, we observed that the following parameter settings give satisfying results: population size $p = 20$, tabu tenure constant $tt = n/150$, tabu search improvement cutoff $\alpha = 2n$ and goodness score constant parameter $\beta = 0.6$. The calibrated parameter values are kept constant for all the experiments. It is possible that better solutions would be found by using a set of instance-dependent parameters.

Our first experiment aims to evaluate the overall performance of our GTA algorithm on the tested instances. The results of this experiment are summarized in Table 1, showing the computational statistics of our GTA algorithm. Columns 2 and 3 respectively give the density (dens) of the Q matrix and the previous best known objective values (f_{prev}). Columns 4 to 10 give our results: the best objective value (f_{best}), the best solution gap to the previous best known values g_{best} ($= f_{best} - f_{prev}$), the average solution gap to the previous best known value g_{avr} ($= f_{avr} - f_{prev}$) (where f_{avr} represents the average objective value over 20 runs), the standard deviations of the solution gaps over 20 runs (σ), the number of success runs (suc) for reaching the best known results f_{prev}, the best and the average CPU time (seconds) for reaching the best results f_{best} (t_{best} and t_{avr}) over 20 independent runs. Furthermore, for each set of benchmarks, the summary of our algorithm's average performance is indicated in the row "Average". Note that the previous best known objective values f_{prev} are extracted from [17] and [18], which are obtained by allowing a time limit of up to *several hours*. These reference algorithms are among the best performing algorithms for the tested instances.

The results shown in Table 1 disclose that our GTA algorithm can stably reach the previous best known results within a very short CPU time, demonstrating the high efficiency of our method. For the 10 medium size ORLIB instances with 2 500 variables, our algorithm can easily reach all the previous best known objective values within 4 seconds on our computer. For the 15 remaining large and difficult instances, our algorithm can also easily reach the previous best known objective values within the given time limit. The average CPU time to obtain the best known objective values is only 352 seconds and the average number of success runs is about 15 out of 20 runs for this set of benchmarks.

As indicated in Section 2, one of the original features of our approach is the multi-parent "logic" combination operator and its joint use with the uniform crossover. In order to check the effect of this strategy, we conducted our second experiment to compare GTA with a variant of GTA, where the multi-parent combination operator is disabled and the remaining components are kept unchanged. We denote this algorithm by GTA$_a$, where we disable our multi-parent combination operator MSX and keep only the uniform crossover.

We run this second experiment using GTA$_a$ under exactly the same conditions as before and the results are reported in Table 2 together with those of GTA extracted from Table 1. Once again, the following information is provided for each instance: the best solution gap to the previous best known objective values

g_{best}, the average solution gap to the previous best known objective values g_{avr}, the standard deviations of the solution gaps over 20 runs (σ), and the number of success runs (suc) for reaching the best known objective values f_{prev} over 20 runs.

One observes that GTA performs better than GTA_a in terms of all the performance criteria. In particular, for the large instance p5000.4, GTA_a failed to reach the best known objective value $f_{prev} = 12252318$ within the given time limit, while GTA reaches this best solution 3 times out of 20 runs. The average gap to the previous best known objective value is 214.3 for GTA against 301.1 for GTA_a. Moreover, GTA obtains the previous best known objective values more often than GTA_a does (15.3 versus 13.7 over 20 independent runs). For the best objective values obtained over 20 runs, we performed a 95% confidence t-test to assess the difference between these two algorithms and found that GTA is statistically superior to GTA_a in 6 instances while it is inferior to GTA_a only in 1 instance. For all the 18 remaining cases, there is no clear difference between these two algorithms.

Table 1. Overall performance of our GTA algorithm over 20 runs

instance	dens	f_{prev}	GTA						
			f_{best}	g_{best}	g_{avr}	σ	suc	t_{best}	t_{avr}
b2500.1	0.1	1515944	1515944	0	0.0	0.0	20	0.45	2.52
b2500.2	0.1	1471392	1471392	0	9.9	43.2	19	3.87	26.7
b2500.3	0.1	1414192	1414192	0	0.0	0.0	20	0.60	6.70
b2500.4	0.1	1507701	1507701	0	0.0	0.0	20	0.33	1.32
b2500.5	0.1	1491816	1491816	0	0.0	0.0	20	0.41	3.50
b2500.6	0.1	1469162	1469162	0	0.0	0.0	20	0.71	4.14
b2500.7	0.1	1479040	1479040	0	0.0	0.0	20	1.06	12.3
b2500.8	0.1	1484199	1484199	0	0.0	0.0	20	0.55	5.83
b2500.9	0.1	1482413	1482413	0	0.0	0.0	20	0.57	11.0
b2500.10	0.1	1483355	1483355	0	0.0	0.0	20	1.44	13.2
Average				0	0.99	4.32	19.9	0.999	8.72
p3000.1	0.5	3931583	3931583	0	0.0	0.0	20	8.83	42.8
p3000.2	0.8	5193073	5193073	0	0.0	0.0	20	5.39	38.6
p3000.3	0.8	5111533	5111533	0	7.7	33.6	19	13.0	82.2
p3000.4	1.0	5761822	5761822	0	0.0	0.0	20	36.1	79.8
p3000.5	1.0	5675625	5675625	0	298.2	373.9	14	16.3	85.6
p4000.1	0.5	6181830	6181830	0	0.0	0.0	20	4.92	52.0
p4000.2	0.8	7801355	7801355	0	194.1	471.5	17	186.3	276.7
p4000.3	0.8	7741685	7741685	0	0.0	0.0	20	42.9	208.4
p4000.4	1.0	8711822	8711822	0	3.0	13.1	19	43.7	168.5
p4000.5	1.0	8908979	8908979	0	260.2	455.1	15	171.3	420.6
p5000.1	0.5	8559680	8559680	0	507.4	313.4	4	137.7	636.3
p5000.2	0.8	10836019	10836019	0	425.6	250.1	11	81.6	562.8
p5000.3	0.8	10489137	10489137	0	356.4	164.3	9	264.7	726.0
p5000.4	1.0	12252318	12252318	0	1035.6	456.6	3	826.7	1326.1
p5000.5	1.0	12731803	12731803	0	126.3	401.6	18	423.9	568.3
Average				0	214.3	195.5	15.4	150.9	351.6

Table 2. Performance comparison of GTA algorithm with GTA_a

instance	f_{prev}	GTA				GTA_a			
		g_{best}	g_{avr}	σ	suc	g_{best}	g_{avr}	σ	suc
b2500.1	1515944	0	0.0	0.0	20	0	0.0	0.0	20
b2500.2	1471392	0	**9.9**	**43.2**	**19**	0	35.7	87.0	17
b2500.3	1414192	0	0.0	0.0	20	0	0.0	0.0	20
b2500.4	1507701	0	0.0	0.0	20	0	0.0	0.0	20
b2500.5	1491816	0	0.0	0.0	20	0	0.0	0.0	20
b2500.6	1469162	0	0.0	0.0	20	0	0.0	0.0	20
b2500.7	1479040	0	0.0	0.0	20	0	0.0	0.0	20
b2500.8	1484199	0	0.0	0.0	20	0	0.0	0.0	20
b2500.9	1482413	0	0.0	0.0	20	0	0.0	0.0	20
b2500.10	1483355	0	0.0	0.0	20	0	0.0	0.0	20
Average		0	**0.99**	**4.32**	**19.9**	0	3.57	8.70	19.7
p3000.1	3931583	0	0.0	0.0	20	0	0.0	0.0	20
p3000.2	5193073	0	0.0	0.0	20	0	0.0	0.0	20
p3000.3	5111533	0	7.7	33.6	19	0	**0.0**	**0.0**	**20**
p3000.4	5761822	0	0.0	0.0	20	0	0.0	0.0	20
p3000.5	5675625	0	**298.2**	373.9	**14**	0	387.3	**311.3**	6
p4000.1	6181830	0	0.0	0.0	20	0	0.0	0.0	20
p4000.2	7801355	0	**194.1**	**471.5**	**17**	0	286.5	649.9	14
p4000.3	7741685	0	0.0	0.0	20	0	0.0	0.0	20
p4000.4	8711822	0	**3.0**	**13.1**	19	0	6.0	26.2	19
p4000.5	8908979	0	**260.2**	**455.1**	**15**	0	865.2	1174.9	12
p5000.1	8559680	0	507.4	313.4	**4**	0	**415.8**	**132.1**	3
p5000.2	10836019	0	**425.6**	**250.1**	**11**	0	673.8	313.8	8
p5000.3	10489137	0	**356.4**	**164.3**	**9**	0	552.1	872.1	6
p5000.4	12252318	**0**	**1035.6**	456.6	**3**	608	1205.3	**280.6**	0
p5000.5	12731803	0	126.3	**401.6**	18	0	**124.5**	652.7	18
Average		**0**	**214.3**	**195.5**	**15.4**	40.5	301.1	294.2	13.7

Let us finally comment that we also carried out a similar experiment on another important feature of our GTA algorithm — the quality-and-distance based pool updating strategy. We compare GTA with another variant of GTA, where the pool updating strategy is disabled and is replaced by one that randomly takes the place of one of the parent individuals in the population. The remaining components are kept unchanged. Once again we observe that GTA performs better than this variant of GTA relative to all the four criteria shown in Table 2, implying the importance of our population updating strategy.

These results provide evidence of the benefit of our multi-parent combination operator and quality-and-distance based pool updating strategy.

4 Conclusions and Discussion

In this paper, we have presented the GTA algorithm, a hybrid genetic–tabu algorithm for solving the UBQP problem. The proposed algorithm integrates

a "logic" multi-parent combination operator for generating offspring solutions and an effective Tabu Search procedure. GTA uses also a pool updating strategy considering both solution quality and diversity. Tested on two sets of 25 well-known benchmark instances with 2 500 to 5 000 variables, we have shown that this hybrid algorithm obtains highly competitive outcomes in comparison with the previous best known results from the literature.

There are several directions to extend this work. One immediate possibility is to examine other dedicated combination operators by considering more detailed semantic information of the UBQP problem. Furthermore, more advanced adaptive memory strategies from tabu search afford opportunities for creating improvements of the local search part. Finally, given that the multi-parent combination introduced in this paper is independent of the UBQP problem, it is worthwhile to verify its effectiveness on other problems and to compare it with other conventional recombinant operators.

Acknowledgement

We would like to thank the referees for their helpful comments. The work is partially supported by a "Chaire d'excellence" from "Pays de la Loire" Region (France) and regional MILES (2007-2009) and RaDaPop projects (2008-2011).

References

1. McBride, R.D., Yormark, J.S.: An implicit enumeration algorithm for quadratic integer programming. Management Science 26, 282–296 (1980)
2. Krarup, J., Pruzan, A.: Computer aided layout design. Mathematical Programming Study 9, 75–94 (1978)
3. Gallo, G., Hammer, P., Simeone, B.: Quadratic knapsack problems. Mathematical Programming 12, 132–149 (1980)
4. Alidaee, B., Kochenberger, G.A., Ahmadian, A.: 0-1 quadratic programming approach for the optimal solution of two scheduling problems. International Journal of Systems Science 25, 401–408 (1994)
5. Chardaire, P., Sutter, A.: A decomposition method for quadratic zero-one programming. Management Science 41(4), 704–712 (1994)
6. Phillips, A.T., Rosen, J.B.: A quadratic assignment formulation of the molecular conformation problem. Journal of Global Optimization 4, 229–241 (1994)
7. Pardalos, P., Rodgers, G.P.: Computational aspects of a branch and bound algorithm for quadratic zero-one programming. Computing 45, 131–144 (1990)
8. Kochenberger, G.A., Glover, F., Alidaee, B., Rego, C.: A unified modeling and solution framework for combinatorial optimization problems. OR Spectrum 26, 237–250 (2004)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1979)
10. Boros, E., Hammer, P.L., Sun, R., Tavares, G.: A max-flow approach to improved lower bounds for quadratic 0-1 minimization. Discrete Optimization 5(2), 501–529 (2008)

11. Boros, E., Hammer, P.L., Tavares, G.: Local search heuristics for Quadratic Unconstrained Binary Optimization (QUBO). Journal of Heuristics 13, 99–132 (2007)
12. Alkhamis, T.M., Hasan, M., Ahmed, M.A.: Simulated annealing for the unconstrained binary quadratic pseudo-boolean function. European Journal of Operational Research 108, 641–652 (1998)
13. Beasley, J.E.: Heuristic algorithms for the unconstrained binary quadratic programming problem. In: Working Paper, The Management School, Imperial College, London, England (1998)
14. Katayama, K., Narihisa, H.: Performance of simulated annealing-based heuristic for the unconstrained binary quadratic programming problem. European Journal of Operational Research 134, 103–119 (2001)
15. Glover, F., Kochenberger, G.A., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Management Science 44, 336–345 (1998)
16. Palubeckis, G.: Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. Annals of Operations Research 131, 259–282 (2004)
17. Palubeckis, G.: Iterated tabu search for the unconstrained binary quadratic optimization problem. Informatica 17(2), 279–296 (2006)
18. Glover, F., Lü, Z., Hao, J.K.: Diversification-driven tabu search for unconstrained binary quadratic problems. 4OR (2010); doi: 10.1007/s10288-009-0115-y
19. Merz, P., Freisleben, B.: Genetic algorithms for binary quadratic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), pp. 417–424. Morgan Kaufmann, San Francisco (1999)
20. Lodi, A., Allemand, K., Liebling, T.M.: An evolutionary heuristic for quadratic 0-1 programming. European Journal of Operational Research 119(3), 662–670 (1999)
21. Katayama, K., Tani, M., Narihisa, H.: Solving large binary quadratic programming problems by an effective genetic local search algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), pp. 643–650. Morgan Kaufmann, San Francisco (2000)
22. Borgulya, I.: An evolutionary algorithm for the binary quadratic problems. Advances in Soft Computing 2, 3–16 (2005)
23. Amini, M., Alidaee, B., Kochenberger, G.A.: A scatter search approach to unconstrained quadratic binary programs. In: New Methods in Optimization, pp. 317–330. McGraw-Hill, New York (1999)
24. Merz, P., Katayama, K.: Memetic algorithms for the unconstrained binary quadratic programming problem. BioSystems 78, 99–118 (2004)
25. Moscato, P.: Memetic algorithms: a short introduction. In: New Ideas in Optimization, pp. 219–234. Mcgraw-Hill Ltd., Maidenhead (1999)
26. Hoos, H., Stützle, T.: Stochastic Local Search Foundations and Applications. Morgan Kaufmann / Elsevier (2004)
27. Glover, F., Hao, J.K.: Efficient evaluations for solving large 0-1 unconstrained quadratic optimization problems. To appear in International Journal of Metaheuristics 1(1) (2009)
28. Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Boston (1997)
29. Syswerda, G.: Uniform crossover in genetic algorithms. In: Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 2–9 (1989)
30. Lü, Z., Hao, J.K.: A memetic algorithm for graph coloring. European Journal of Operational Research 203(1), 241–250 (2010)
31. Beasley, J.E.: Obtaining test problems via internet. Journal of Global Optimization 8, 429–433 (1996)

Bicriteria Scheduling Problem on the Two-Machine Flowshop Using Simulated Annealing

Mohammad Mesgarpour[1], Nureddin Kirkavak[2], and Hakan Ozaktas[3]

[1] School of Mathematics, University of Southampton, Highfield, Southampton, SO17 1BJ, United Kingdom
[2] Department of Industrial Engineering, Eastern Mediterranean University, Famagusta, Mersin 10, Turkey
[3] Department of Industrial Engineering, Atilim University, Incek, Ankara, 06836, Turkey
m.mesgarpour@soton.ac.uk, nureddin.kirkavak@emu.edu.tr, hozaktas@atilim.edu.tr

Abstract. Real life scheduling problems require the decision maker to consider a number of criteria before arriving at any decision. The trade-offs involved in considering several different criteria provide useful insights for the decision maker. Surprisingly, research in the field of multi-objective scheduling has been quite limited when compared to research in single criterion scheduling. The subject of this paper is the bicriteria scheduling problem in a two-machine flowshop. The objective is to find a job sequence that minimizes sum of weighted total flowtime and total tardiness. Based on the problem characteristics, a Simulated Annealing algorithm is developed. The proposed meta-heuristic is compared with the branch and bound enumeration algorithm of the integer programming model as well as a modified version of the well-known NEH algorithm. During these evaluations, the experimental design approach and careful statistical analysis have been used to validate the effectiveness of the simulated annealing approach.

Keywords: Bicriteria Scheduling, Flowshop Scheduling, Simulated Annealing.

1 Introduction

In the early 1950s, the first scheduling publications containing the results by Johnson [15], Smith [29], and Jackson [13] have appeared in the production research literature. Lewis and El-Rewini [19] have mentioned that the scheduling problem in its general form is known to be NP-complete, as it is the creation of the optimal execution schedule under a number of conditions. This is due to the presence of numerous tasks. Consequently, many heuristics and meta-heuristics have been developed to find adequate (but sub-optimal) schedules. Until the late 1980s, the majority of the scheduling research has been concentrated on single criterion problems. In the recent years, multicriteria scheduling problems have received more attention [23].

The flowshop scheduling problem is considered as one of the production scheduling problems in which n different jobs must be processed by m machines sequentially [2]. Since the publication of the Johnson's paper [15], there has been a tremendous accumulation of research papers on flow shop scheduling problem (more than 1,300

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 166–177, 2010.

[17]). Rajendran and Ziegler [28] have specified that most flowshop scheduling problems are NP-complete and hence the study of heuristics and enumerative approaches for solving the problem is quite often undertaken.

The two-machine problem with the objective of minimizing mean flowtime or total flow time is known to be NP-hard [9]. Therefore, researchers have developed either implicit enumeration techniques such as branch-and-bound algorithms or dynamic programming to find the optimal solution for a limited number of jobs or local search algorithms for larger number of jobs. The two-machine problem with the objective of minimizing total tardiness is NP-complete [18] and has been solved by using different techniques such as branch-and-bound procedure based on elimination criterion and a lower bound to find a sequence that minimizes the tardiness value. A detailed review of the literature is provided in [10] and [17].

It is well-known that the minimization of sum of the flowtime and tardiness of jobs is a significant bicriteria objective for many real-life situations, especially with respect to minimization of inventory holding costs, contractual penalty costs for the belated supply and loss of customer goodwill ([1] and [8]). It involves maintaining work-in-process inventory at low levels along with minimal average throughput time spent by each job in the shop. Minimizing tardiness involves reducing penalty cost incurred for late jobs. The former represents internal efficiency while the latter represents external efficiency.

For many real-life situations jobs may have unproportional holding and tardiness costs, hence they cannot be treated equivalently. Therefore, the objective of minimizing the weighted sum of flowtime and tardiness of jobs is a significant and relevant objective for real-life problems [28]. Panwalker and Iskander [26] have pointed out that flowtime and tardiness are the most prominent measures among the scheduling objectives in industrial applications. Two-machine flowshop problems have rarely been discussed on the literature. Additional complexity is the primary reason for the lack of research on flowshop scheduling with two or more machines.

In this research, we study the problem of minimizing weighted total flowtime and total tardiness for the two-machine flowshop ($F2 \mid\mid F_w(\sum_{j=1}^{n} C_j,\ \sum_{j=1}^{n} T_j)$). Since, for the two-machine flowshop scheduling problem, minimizing total completion time is strongly NP-hard and minimizing total tardiness is NP-complete, our problem is either NP-hard or NP-complete. Therefore, a metaheuristic approach has been used to solve this problem.

The rest of the paper is organized as follows. In the next Section, the problem definition and the integer programming (IP) formulation are given. Simulated Annealing (SA) is briefly explained in Section 3. The algorithms which are used to solve the problem are discussed in Section 4. The experimental results are given in Section 5. Finally, the last Section summarizes our conclusions.

2 Problem Definition

To succinctly define the $F2 \mid\mid F_w(\sum_{j=1}^{n} C_j,\ \sum_{j=1}^{n} T_j)$ problem, let $n = \{J_1, J_2, \ldots, J_n\}$ be the set of jobs to be scheduled and $m = \{M_1, M_2\}$ be the two machines which are used in the flowshop. The processing times for job j on M_1 and M_2 are denoted by p_{j1} and p_{j2}, respectively, and the corresponding due date for job j is d_j. The objective is to find the schedule to minimize the weighted sum of total completion time and total tardiness for a two-machine flowshop.

The following assumptions are considered in the problem:

- Each job is to be processed first on machine-1 and then on machine-2.
- Each machine can process at most one job at a time and each job is processed at most by only one machine at a time.
- Both machines are available at time zero.
- It is also assumed that all jobs are available at time zero. So, with this assumption, the flowtime and the completion time of job *j* are equivalent for a given schedule (both terms are used interchangeably in the forthcoming discussions).
- Pre-emption is not allowed. This means that when an operation is started, it must be completed before another operation may be started on the same machine.
- There are no sequence-dependent set-up times.
- Machines are stable and available throughout the scheduling period.

In order to solve the two-machine flowshop scheduling problem considered in this paper, an IP model is given. This model is based on the IP models of Chou and Lee [4] and Eren and Guner [6]. Chou and Lee [4] have presented an IP model to minimize a weighted sum of total flowtime and makespan for the two-machine flowshop scheduling problem with n^2+3n variables and $5n$ constraints. Eren and Guner [6] have proposed an IP model to minimize total tardiness for the two-machine flowshop scheduling problems with n^2+5n variables and $6n$ constraints.

In this proposed model, there are n^2+3n+2 variables and $6n+2$ problem constraints, where n denotes the number of jobs. Here, the constant term, 2, has been referred to as the availability of two machines at the beginning of the scheduling period that can be reduced by incorporating the values into the corresponding constraints. So the total number of constraints and variables are less than those proposed by the IP model of Eren and Guner [6], respectively. The definition of parameters and variables are given afterwards.

Parameters:

n:	number of jobs,	j=1, 2, …, n,
m:	number of machines,	i=1, 2,
p_{ji}:	processing time of job j on machine i,	j=1, 2, …, n, i=1, 2,
d_j:	due date of job j,	j=1, 2, …, n,
w_1:	weight for the total completion time,	
w_2:	weight for the total tardiness (equal to 1-w_1).	

Decision Variables:

$Z_{j,r}$: $\begin{cases} 1 & \text{If job } j \text{ is scheduled to be processes at the } r^{th} \text{ position,} \\ 0 & \text{Otherwise,} \end{cases}$ j =1,2,…, n, r =1,2,…,n,

$W_{r,i}$: Completion time of the r^{th} positioned job at machine i, i =1,2, r =1,2, …,n,

$W_{0,1}$: The starting time of the scheduling period for the first machine (Assume $W_{0,1}=0$),

$W_{0,2}$: The starting time of the scheduling period for the second machine (Assume $W_{0,2}=0$),

T_r: Tardiness for the r^{th} positioned job, r =1, 2, …, n.

So, the proposed IP model is as follows.

$$\text{Minimize} \quad w_1 \sum_{r=1}^{n} W_{r,2} + w_2 \sum_{r=1}^{n} T_r, \tag{1}$$

$$\textit{Subject to} \quad \sum_{j=1}^{n} Z_{j,r} = 1, \qquad r = 1, 2, \ldots, n, \tag{2}$$

$$\sum_{r=1}^{n} Z_{j,r} = 1, \qquad j = 1, 2, \ldots, n, \tag{3}$$

$$W_{r,1} \geq W_{r-1,1} + \sum_{j=1}^{n} Z_{j,r} p_{j,1}, \qquad r = 1, 2, \ldots, n, \tag{4}$$

$$W_{r,2} \geq W_{r-1,2} + \sum_{j=1}^{n} Z_{j,r} p_{j,2}, \qquad r = 1, 2, \ldots, n, \tag{5}$$

$$W_{r,2} \geq W_{r,1} + \sum_{j=1}^{n} Z_{j,r} p_{j,2}, \qquad r = 1, 2, \ldots, n, \tag{6}$$

$$T_r \geq \sum_{j=1}^{r} Z_{j,r} d_j - W_{r,2}, \qquad r = 1, 2, \ldots, n. \tag{7}$$

$W_{r,i}$ and T_r are non-negative decision variables and $Z_{j,r}$ are binary decision variables. The objective function (1) minimizes the sum of weighted total flowtime and total tardiness for a complete permutation schedule. Equation (2) specifies that only one job should be scheduled at the r^{th} position. Each job should be scheduled only once which is defined in equation (3). The starting time of the r^{th} positioned job should be greater than or equal to the previous job completion time at the first machine and second machine which are indicated by constraint (4) and constraint (5), respectively. Also, the start time of the first positioned job at the second machine should be greater than or equal to its completion time at the first machine which is specified by constraint (6). Constraint (7) defines that the tardiness of the r^{th} positioned job should be greater than or equal to the difference between the due date and the completion time of the job at the second machine.

By considering $(T_r^+ + T_r^-)$ instead of T_r, and replacing inequality with equality in (7), we can also obtain the earliness and tardiness (just-in-time) measure. However, the size of the problem increases in terms of number of variables and number of constraints by n in the standard form. One can also include the makespan criterion (as the third objective) by adding $W_{n,2}$ to the objective function with an appropriate weight. Achieving a global optimal solution for medium and large sized problems (based on our experiments) in a reasonable amount of time is not possible. This study aims at devising a Simulated Annealing (SA) model to minimize the weighted sum of total flowtime and total tardiness to compare with a well-known optimal seeking software package called LINGO ([20]) in a limited number of branch-and-bound algorithm iterations.

Hoogeveen [12] and Chen and Bulfin [3] studied the complexity of the single machine bicriteria and multicriteria problems. They have proved that the only problem which can be solved in polynomial time is the hierarchical objective model of minimizing flowtime and maximum tardiness with flowtime being the primary objective. All the other problems are either NP-hard or remain open as far as computational complexity is concerned. Obviously, the problems including more than one machine and two criteria are more difficult to solve.

3 Simulated Annealing

Since the SA model is a randomized search procedure, it is started with an initial seed sequence to improve upon it. The rationale of starting with a good seed sequence is observed by Johnson et al. [14], especially to reduce the computational time.

The modified NEH (Nawaz, Enscore and Ham) algorithm is used as an initial seed sequence in SA.

For many years the NEH algorithm, a well-known heuristic for *m*-machine, *n*-job flowshop proposed by Nawaz et al. [24], has been widely accepted to be the best heuristics for solving the classical permutation flowshop problem ([16]). It is a constructive flowshop scheduling technique based on the assumption that a job with a high total processing time on all the machines should be given a higher priority than a job with a low total processing time. Then, the jobs are sorted in decreasing order of their total processing time requirements. The final sequence is built in a constructive way, adding a new job at each step and finding the best partial solution.

The NEH algorithm is developed for minimizing the makespan in flowshop scheduling ($Fm \mid\mid C_{max}$) in three steps: (i) initial construction, (ii) sorting jobs, and (iii) final construction. Since the sum of weighted total flowtime and weighted total tardiness are the objectives of this study, step (iii) of NEH algorithm is modified by replacing the weighted sum of total flowtime and total tardiness instead of the makespan.

The way of generating a neighbor sequence *s* for a specific candidate sequence has significant effects on the performance of the simulated annealing algorithm. Tian et al. [30] introduced the six perturbation schemes which can be summarized as:

1. Two adjacent jobs are interchanged,
2. Two random jobs are interchanged,
3. A single job is moved,
4. A sub-sequence of jobs is moved,
5. A sub-sequence of jobs is reversed,
6. A sub-sequence of jobs is reversed and/or moved.

Based on the experiments that have been done by Nearchou [25] for flowshop scheduling problems, the random exchange scheme (two jobs in a candidate sequence are randomly selected and interchanged) performs better results than the other schemes, so it is chosen as the first move of the SA model. According to Loukil et al. [21] for multi-objective production scheduling problems, shifting a single job is selected as the second move in the SA model. Finally, by performing some experiments for small and large sized problems, we have observed that exchanging the place of the latest job and the earliest job can be a good scheme in flowshop scheduling problem to minimize the sum of weighted total flowtime and weighted total tardiness. Therefore, it is considered as the third move in the SA algorithm.

The first two moves are the general and basic moves used widely in metaheuristic approaches for scheduling problems. By including the third move which is specific for the objective function, the efficiency of the model has been increased.

4 Proposed Simulated Annealing Model

The body of the proposed SA algorithm to minimize the objective function for *n* jobs is given as below:

1. GET an initial solution s ;
2. SET T_stop = EXP(0.4), $r = 0.985$, $temp\_step = 500$, $no\_improvement = 5000$;
3. SET $T = 40$, $counter = 1$;

4. FOR *iteration* = 1 TO 100,000 DO:
   4.1. SET *random* = RAND[0,1] ;
   4.2. IF *random* $\leq$ 0.4 THEN GOTO Move 1;
   ELSE IF *random* $\leq$ 0.8 THEN GOTO Move 2;
   ELSE GOTO Move 3;
   4.3. SET Δ = objective(s') - objective(s) ;
   4.4. IF $\Delta \leq 0$ THEN SET $s = s'$, *counter* = 1;
   ELSE IF RAND[0,1] $\leq$ EXP($-\Delta/T$) THEN SET $s = s'$, *counter* = 1;
   ELSE SET *counter* = *counter* + 1;
   4.5. IF (*iteration* MOD *temp_step*) = 0 THEN SET $T = r \times T$;
   4.6. IF (*counter* = *no_improvement*) AND ($T < T\_stop$) THEN GOTO 5;
5. END

The acceptance probability is the basic element of the search mechanism in SA. If the temperature is reduced sufficiently slowly, then the system can reach equilibrium (steady state) at each iteration [11]. The Metropolis acceptance criterion ([22]) is applied in our SA algorithm which is given as below:

$$P[\text{Accepting } s'] = \begin{cases} e^{-\frac{[\text{objective}(s')-\text{objective}(s)]}{T}} & \text{if objective}(s')-\text{objective}(s) > 0, \\ 1 & \text{if objective}(s')-\text{objective}(s) \leq 0. \end{cases} \quad (8)$$

where "*objective (s)*" and "*objective (s')*" denote the energies (objective function values) associated with solution s and s', respectively.

To set the initial temperature, the cooling parameter, and the number of steps in the cooling schedule some experiments are implemented and the parameter setting is tuned to obtain their best performance. Consequently, 40, 0.985 and 500 are chosen as the initial temperature (T), the cooling parameter (r), and the length of the temperature step in the cooling schedule (*temp_step*), respectively. Also, a counter (*no_improvement*) is set to 5,000 at the beginning of the SA algorithm to count the number of iterations without improvement.

As the algorithm parameters have been specified for all the instances of the problems and because of the stochastic nature of the problem, the SA algorithm is totally replicated h times (which is equal to the number of jobs, h= 10, 20, 30, and 50) to increase the chance of converging to the global optimal solution [27]. So that, the algorithm selects the best result among all replications as the SA result.

5 Computational Design and Experimentation

All the experimental work have been implemented on 20 PCs with the same hardware and software configurations (Pentium D, 3GHz, 960MB RAM). For the experimental design, problem size can be categorized into three classes of (i) small size (10 and 20 jobs), (ii) medium size (30 jobs), and (iii) large size (50 jobs). The test problems have been generated using a model that allows control to be exercised over various factors generally believed to affect the problem difficulty. The test problem generation is mainly based on the methods presented in Daniels and Chambers [5] and Fisher [7].

The processing time of the jobs on the first machine (M_1) and the second machine (M_2) in the flowshop are chosen as uniformly distributed integers, within the ranges of [$10c_1$, $30c_1$] and [$10c_2$, $30c_2$], respectively. The constants c_1 and c_2 control the degree of machine dominance. When $c_1>c_2$, processing times are on average larger on the first machine, representing the dominant or bottleneck processor in the system. The bottleneck machine is usually the machine in which operation times are proportionally larger than any other machine. Three pairs of (c_1, c_2) values are considered as {(1.00, 0.50), (0.75, 0.75) and (0.50, 1.00)}, roughly corresponding to flowshops with machine-1 dominance, no dominance, and machine-2 dominance.

After having obtained the processing time distribution, due dates of jobs are assigned using a uniform distribution within the range of [ABP(1-τ - R/2), ABP(1-τ + R/2)], where

$$\text{ABP (Average Busy Period)} = \begin{cases} n\bar{p}_1+\bar{p}_2 & \text{if } c_1 \geq c_2, \\ \bar{p}_1+n\bar{p}_2 & \text{if } c_1 < c_2. \end{cases} \tag{9}$$

and $\bar{p}_j$ denotes the average processing time on machine j (j=1, 2). The average busy period serves as an approximation of the completion time for the last job in any schedule. This value is adjusted by two parameters which are given as below:

- Tardiness factor (τ): It is the fraction of jobs which are late in an optimal solution (which affects the average number of tardy jobs).
- Due date range factor (R): It controls the variances of the due dates.

Three values of τ (0.25, 0.50, and 0.75) and three values of R (0.25, 0.50, and 0.75) have been included in the experimental design. If the values of τ and R, and their combinations are not selected properly, negative values can be generated as lower bounds for the due dates. Most researchers disregard this fact simply by ignoring the negative values and proceeding with the problem solution. However, this procedure affects the uniformity and variance of the due date distribution. We have overcome this problem by choosing the appropriate generation scheme (Table 1).

Five samples are randomly generated for each combination of machine dominance, τ, and r, resulting in a total of 90 instances of the problem for each of the four problem sizes. Weights of the objective function criteria are varied from 0 to 1 in step of 0.25. By considering zero as the weight of total flowtime or total tardiness, the problem is converted to a single criterion problem. Therefore, two single criterion problems and three bicriteria problems have been used in the experimental design. Totally, 5400 experiment runs have been performed.

Experimental results are divided into three parts. The first part gives the optimal solution or the upper bound value (if the allowable iteration branch limit is reached) as the basis to test the effectiveness of the heuristic scheduling algorithm. The second part shows the result of the modified NEH algorithm as a benchmark. The third part provides the solution obtained by using the proposed SA model.

Table 1. Different combinations of the processing time and due date factors

#*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
τ	0.75	0.75	0.75	0.75	0.75	0.75	0.5	0.5	0.5	0.5	0.5	0.5	0.25	0.25	0.25	0.25	0.25	0.25
R	0.5	0.5	0.5	0.25	0.25	0.25	0.5	0.5	0.5	0.25	0.25	0.25	0.5	0.5	0.5	0.25	0.25	0.25
c_1	1	0.75	0.5	1	0.75	0.5	1	0.75	0.5	1	0.75	0.5	1	0.75	0.5	1	0.75	0.5
c_2	0.5	0.75	1	0.5	0.75	1	0.5	0.75	1	0.5	0.75	1	0.5	0.75	1	0.5	0.75	1

* Each combination of parameters can be referred to by its corresponding column number

The IP model is used to find the optimal solution of the considered problem using Extended LINGO 9.0 ([20]). LINGO can find the global optimal solution up to 20 jobs for all the test problems. Since the computational time increases exponentially as the number of jobs become higher, LINGO cannot find the global optimal solution for problems with 30 and 50 jobs. By performing some experiments on larger sample problems, 50 million branches in the branch-and-bound tree have been defined as an upper limit for the number of iterations of LINGO. Executing LINGO to run more than 50 million iterations has no significant effect on improving the objective values.

Table 2. LINGO runtime (in second) for each set of 90 instances

Weights	10 Jobs	20 Jobs	30 Jobs	50 Jobs	Average
w_1=1.00 , w_2=0.00	1.5	105.9	1947.9	7660.6	**2429.0**
w_1=0.75 , w_2=0.25	1.4	298.4	4108.8	19807.3	**6054.0**
w_1=0.50 , w_2=0.50	1.5	187.8	4634.0	19213.8	**6009.3**
w_1=0.25 , w_2=0.75	1.7	199.2	3452.7	17305.4	**5239.8**
w_1=0.00 , w_2=1.00	1.3	91.1	1960.2	12119.7	**3543.1**
Average	**1.5**	**176.5**	**3220.7**	**15221.4**	

The average runtime of the LINGO model for the each set of 90 instances by considering 50 million iterations as an upper limit for different sample size and criteria is provided in Table 2 (w_1 and w_2 are the weights of total flowtime and total tardines, respectively). Also, it should be mentioned that the runtime of the modified NEH algorithm and SA model are less than a second for all cases.

The performance of the SA model, modified NEH and LINGO model are compared by calculating the average error percentage. The error percentage of optimally solved problem is defined as below.

$$\text{Error Percentage (optimal)} = \frac{\text{(SA result – LINGO Global optimal)}}{\text{LINGO Global optimal}} \times 100. \tag{10}$$

Similarly, for a large sized problem in which the optimality is not reached by LINGO, the relative error percentage is defined as the following:

$$\text{Error Percentage (non-optimal)} = \frac{(\text{SA result} - \text{LINGO Best solution})}{\text{LINGO Best solution}} \times 100. \quad (11)$$

Whenever the LINGO model can find the global optimal solution, the error percentage is always greater than or equal to zero (non-negative). The zero value means that the SA model finds the global optimal solution if the LINGO solution is global optimum. If the LINGO model cannot find the global optimal solution, the reported solution is the best objective obtained from the branch-and-bound algorithm after 50 million iterations and the percentage error can be any real number. The zero value of error percentage cannot guarantee that the SA model finds the global optimal solution; it only means that the LINGO solution and SA model solution are equivalent. The negative value shows that the SA model solution is better than the LINGO solution after 50 million iterations.

Summary of the average error percentage of five randomly generated samples for each combination of machine dominance for the SA model has been shown in Table 3. It has to be mentioned that the average error percentages of the SA model decreases as the number of jobs increases to 50 for all different weight objectives. This might be due to the complexity of the problem.

Table 3. Average error in percentage of SA algorithm with respect to LINGO model

#	(w_1=1.00 , w_2=0.00)			(w_1=0.75 , w_2=0.25)			(w_1=0.50 , w_2=0.50)			(w_1=0.25 , w_2=0.75)			(w_1=0.00 , w_2=1.00)		
	***n*=20**	***n*=30**	***n*=50**	***n*=20**	***n*=30**	***n*=50**	***n*=20**	***n*=30**	***n*=50**	***n*=20**	***n*=30**	***n*=50**	***n*=20**	***n*=30**	***n*=50**
1	0.000	0.000	-0.006	0.000	0.000	-0.084	0.000	0.010	-0.433	0.334	0.483	-1.168	0.013	0.331	-1.923
2	0.463	0.403	0.682	0.297	0.377	0.057	0.405	0.164	-0.159	0.005	0.033	-0.994	0.267	0.251	-2.087
3	0.117	0.044	0.000	0.048	0.000	0.184	0.011	0.050	-0.454	0.006	0.000	0.005	0.148	0.123	-0.724
4	0.000	0.000	0.000	0.000	0.000	0.000	0.003	0.000	-0.002	0.361	0.243	0.117	0.000	0.000	0.014
5	0.305	0.293	0.460	0.511	0.268	0.351	0.417	0.477	0.093	0.000	0.003	0.000	0.357	0.388	0.387
6	0.069	0.000	0.000	0.000	0.001	-0.001	0.004	0.023	0.000	0.000	0.010	-1.744	0.000	0.060	0.020
7	0.000	0.000	0.000	0.000	0.000	-0.118	0.005	0.006	-0.534	0.000	-0.083	-1.944	0.000	-0.049	-0.693
8	0.251	0.549	0.375	0.109	0.159	-1.197	0.083	-0.093	-0.707	0.000	0.000	-1.845	0.075	-0.050	-1.368
9	0.012	0.033	0.002	0.060	0.006	0.003	0.000	-0.002	-0.548	0.000	0.000	0.001	0.028	-0.041	-0.218
10	0.006	0.003	0.002	0.005	0.002	0.001	0.000	0.000	0.001	0.295	0.406	0.368	0.000	0.054	0.000
11	0.198	0.308	0.488	0.237	0.360	0.180	0.207	0.404	0.031	0.000	0.000	0.036	0.174	0.147	0.111
12	0.000	0.000	0.033	0.000	0.000	0.034	0.000	0.000	0.005	0.096	0.000	-0.051	0.000	0.000	0.000
13	0.044	0.000	0.001	0.167	0.000	-0.001	0.094	0.000	-0.004	0.279	0.202	-1.014	0.000	0.000	0.000
14	0.307	0.387	0.528	0.248	0.312	0.250	0.236	0.258	-0.414	0.020	0.043	-0.036	0.000	0.078	-10.321
15	0.000	0.463	0.108	0.000	0.018	0.050	0.000	0.609	-0.118	0.000	0.000	0.001	0.000	0.000	-0.952
16	0.000	0.003	0.000	0.000	0.000	0.000	0.000	0.002	0.000	0.172	0.275	0.252	0.000	0.000	0.000
17	0.153	0.283	0.474	0.293	0.541	0.423	0.111	0.302	0.178	0.315	0.141	0.139	0.164	0.000	-0.602
18	0.363	0.121	0.002	0.354	0.113	0.017	0.347	0.114	0.070	0.347	0.114	0.070	0.000	0.000	0.000
Max.	**0.463**	**0.549**	**0.682**	**0.511**	**0.541**	**0.423**	**0.417**	**0.609**	**0.178**	**0.361**	**0.483**	**0.368**	**0.357**	**0.388**	**0.387**
Ave.	**0.127**	**0.161**	**0.175**	**0.129**	**0.120**	**0.008**	**0.107**	**0.129**	**-0.166**	**0.124**	**0.104**	**-0.434**	**0.068**	**0.072**	**-1.020**
Min.	**0.000**	**0.000**	**-0.006**	**0.000**	**0.000**	**-1.197**	**0.000**	**-0.093**	**-0.707**	**0.000**	**-0.083**	**-1.944**	**0.000**	**-0.050**	**-10.321**
STD	**0.151**	**0.194**	**0.245**	**0.158**	**0.174**	**0.333**	**0.149**	**0.199**	**0.270**	**0.151**	**0.157**	**0.785**	**0.109**	**0.130**	**2.428**

The LINGO average runtime usually increases as the number of jobs increases and for some test problems (after 7 or 8 hours of computation time) LINGO model cannot find the global optimal solution, whereas, for the SA model runtime does not exceed one second for all test problems and the average error percentage of the SA model solutions are better than the LINGO model solutions for large test problems. Moreover, the average error percentage of the SA model solutions does not exceed 0.7% for all test problems. It means that the proposed SA model is a good metaheuristic model for the large sized problems in terms of runtime and quality of the results.

By increasing the number of jobs, the average error percentage of the modified NEH algorithm has increased for all the five different weighted criteria. The average error percentage of the SA model has increased for the first weighted criteria by increasing the number of jobs and by increasing the weight of the total flowtime. The average error percentage of the SA model has increased in most of the 18 combinations if the total flowtime has been considered as an objective. When two machines are balanced and the number of jobs is not equal to 50, the average error percentage increases. This indicates how the complexity of the problems increases for balanced machines. For other cases there is no specific observation.

Table 4. Number of best solutions obtained by each model out of 90 problems

	10 Jobs			20 Jobs			30 Jobs			50 Jobs		
Weights	**LINGO**	**NEH**	**SA**	**LINGO**	**NEH**	**SA**	**LINGO**	**NEH**	**SA**	**LINGO**	**NEH**	**SA**
w_1=1.00 , w_2=0.00	90	30	78	90	25	61	90	16	53	89	9	53
w_1=0.75 , w_2=0.25	90	9	75	90	3	60	89	0	55	66	1	58
w_1=0.50 , w_2=0.50	90	1	80	90	0	62	83	0	55	47	0	71
w_1=0.25 , w_2=0.75	90	1	83	90	0	69	85	0	52	46	0	67
w_1=0.00 , w_2=1.00	90	1	87	90	0	75	84	0	68	59	0	78
Total	**450**	**42**	**403**	**450**	**28**	**327**	**431**	**16**	**283**	**307**	**10**	**327**

In Table 4, number of the best solutions obtained by the LINGO model, the modified NEH and the SA model have been tabulated. The modified NEH algorithm has a good efficiency to solve the minimizing total flowtime in two-machine flowshop scheduling problems and small-sized problems relative to other objective criteria. The LINGO software package is not strong enough to optimally solve the mixed integer programming formation of the large-sized scheduling problems by branch-and-bound algorithm. Furthermore, total number of optimal and the number of equal solutions which have been found by LINGO and SA are near to each other for small sized and large sized problems. Moreover, the practical limitation of 50 million iterations seems to be sufficient to solve optimally for problems with up to 30 jobs. When insufficiency of this limitation becomes significant, SA results become relatively better.

The investigation for evaluating the difference between effects of SA model and the LINGO model has been done. Paired *t*-test shows that the difference in the objective

values for 10, 20, 30 and 50 jobs generated between the LINGO model and the SA model are meaningful at a significance level of 0.99.

6 Conclusions

This research has addressed the scheduling problem in a two-machine flowshop with the bicriteria of minimizing the sum of weighted total flowtime and total tardiness. The problem is NP-hard (or NP-complete) and this means that a good heuristic or a metaheuristics algorithm is essential to solve the discussed problem.

A branch-and-bound enumeration algorithm which tries to solve the integer programming model is proposed to be used for testing the efficiency of the SA model. Also the NEH algorithm is modified to serve as an initial solution of the SA model which is developed in this study. The SA model has been coded, evaluated and compared by computational analysis of experimental results. The analysis shows that the SA model is quite efficient. It also shows that for most of the test problems, the SA model has the desirable property in terms of the average error percentage as an appropriate technique. Moreover, for practical application purpose the SA model is preferable to the LINGO model considering runtime.

Acknowledgement. The authors would like to thank the Department of Industrial Engineering at Eastern Mediterranean University, North Cyprus, for supporting this research as a MSc. thesis study and providing required facilities in "Simulation & Optimization Laboratory".

References

1. Baker, K.R., Trietsch, D.: Principles of Sequencing and Scheduling. John Wiley and Sons, New York (2009)
2. Brucker, P.: Scheduling Algorithms. Springer, Heidelberg (2007)
3. Chen, C.L., Bulfin, R.L.: Complexity of a Single Machine Multi-criteria Scheduling Problems. Eur. J. Oper. Res. 70, 115–125 (1993)
4. Chou, F.D., Lee, C.E.: Two-Machine Flowshop Scheduling with Bicriteria Problem. Comput. Ind. Eng. 36, 549–564 (1999)
5. Daniels, R.L., Chambers, R.J.: Multiobjective Flowshop Scheduling. Nav. Res. Log. 37, 981–995 (1990)
6. Eren, T., Guner, E.: A Bicriteria Flowshop Scheduling with a Learning Effect. Appl. Math. Model. 32, 1719–1733 (2007)
7. Fisher, M.L.: A Dual Algorithm for the One-Machine Scheduling Problem. Math. Program. 11, 229–251 (1976)
8. French, S.: Sequencing and Scheduling: An Introduction to the Mathematics of the Jobshop. Ellis Horwood, Chichester (1982)
9. Gonzales, T., Sahni, S.: Flowshop and Jobshop Schedules. Oper. Res. 26, 36–52 (1978)
10. Gupta, J.N.D., Stafford Jr., E.F.: Flowshop Scheduling Research after Five Decades. Eur. J. Oper. Res. 169, 699–711 (2006)

11. Henderson, D., Jacobson, S.H., Johnson, A.W.: The Theory and Practice of Simulated Annealing. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics. Kluwer Academic publisher, Dordrecht (2003)
12. Hoogeveen, H.: Multicriteria scheduling. Eur. J. Oper. Res. 167, 592–623 (2005)
13. Jackson, J.R.: Scheduling a Production Line to Minimize Maximum Tardiness. Research Report 43, Management Research Project, UCLA (1955)
14. Johnson, D.S., Aragon, C.R., McGooch, L., Schevon, C.: Optimization by Simulated Annealing: Experimental Evaluation Part 1. Graph partitioning. Oper. Res. 37, 865–891 (1989)
15. Johnson, S.M.: Optimal Two- and Three- Stage Production Schedules with Set-up Times Included. Nav. Res. Log. Quart. 1, 61–68 (1954)
16. Kalczynski, P.J., Kamburowski., J.: On the NEH Heuristics for Minimizing the Makespan in Permutation Flowshops. Omega-Int. J. Manage. S. 35, 53–60 (2007)
17. Lei, D.: Multi-objective Production Scheduling: a Survey. Int. J. Adv. Manuf. Tech. 43, 926–938 (2009)
18. Lenstra, J.K., Rinnooy kan, A.H.G., Brucker, P.: Complexity of Machine Scheduling Problems. Ann. Discrete Math. 4, 281–300 (1977)
19. Lewis, T.G., El-Rewini, H.: Introduction to Parallel Computing: Chapter Scheduling Parallel Programs. Prentice-Hall Inc., Englewood Cliffs (1992)
20. LINDO Systems Inc. Extended LINGO Release 9.0. Chicago, IL (2005)
21. Loukil, T., Teghem, J., Tuyttens, D.: Solving Multi-Objective Production Scheduling Problems Using Metaheuristics. Eur. J. Oper. Res. 161, 42–61 (2005)
22. Metropolis, N., Rosenbluth, A.W., Teller, A.H., Teller, E.: Equation of State Calculation by Fast Computing Machines. J. Chem. Phys. 21, 1087–1091 (1953)
23. Nagar, A., Haddock, J., Heragu, S.: Multiple and Bicriteria Scheduling: A Literature Survey. Eur. J. Oper. Res. 81, 88–104 (1995)
24. Nawaz, M., Enscore Jr., E.E., Ham, I.: A Heuristic Algorithm for the m-Machine, n-Job Flowshop Sequencing Problem. Omega-Int. J. Manage. S. 11, 91–95 (1983)
25. Nearchou, A.C.: A Novel Metaheuristic Approach for the Flowshop Scheduling Problem. Eng. Appl. Artif. Intel. 17, 289–300 (2004)
26. Panwalker, S.S., Iskander, W.: A Survey of Scheduling Rule. Oper. Res. 25, 45–61 (1977)
27. Pham, D.T., Karaboga, D.: Intelligent Optimization Techniques: Genetic Algorithms. In: Tabu Search, Simulated Annealing and Neural Networks. Springer, London (2000)
28. Rajendran, C., Ziegler, H.: Scheduling to Minimize the Sum of Weighted Flowtime and Weighted Tardiness of Jobs in a Flowshop with Sequence-Dependent Setup Times. Eur. J. Oper. Res. 149, 513–522 (2003)
29. Smith, W.E.: Various Optimizers for Single Stage production. Nav. Res. Log. Quart. 3, 59–66 (1956)
30. Tian, P., Ma, J., Zhang, D.-M.: Application of the Simulated Annealing Algorithm to the Combinatorial Optimization Problem with Permutation Property: An Investigation of Generation Mechanism. Eur. J. Oper. Res. 118, 81–94 (1999)

A Memetic Algorithm for Workforce Distribution in Dynamic Multi-Skill Call Centres

David Millán-Ruiz[1] and J. Ignacio Hidalgo[2]

[1] Telefonica Research & Development,
28043 Madrid, Spain
dmr@tid.es
[2] Complutense U. of Madrid,
28040 Madrid, Spain
hidalgo@fis.ucm.es

Abstract. In this paper, we describe a novel approach for workforce distribution in dynamic multi-skill call centres. Dynamic multi-skill call centres require quick adaptations to a changing environment that only fast greedy heuristics can handle. The use of memetic algorithms, which are more complex than ad-hoc heuristics, can guide us to more accurate solutions. In order to apply memetic algorithms to such a dynamic environment, we propose a reformulation of the traditional problem, which combines predictions of future situations with a precise search mechanism, by enlarging the time-frame considered. Concretely, we propose a neural network for predicting call arrivals and the number of available agents, and a memetic algorithm to carry out the assignment of incoming calls to agents, which outperforms classical approaches to this dynamic environment. We also test our method on a real-world environment within a large multinational telephone operator.

Keywords: Memetic Algorithms, Dynamic Multi-Skill Call Centre.

1 Introduction

Over the last years, we perceive a tendency to tackle increasingly complex problems and application domains which frequently involve the processing of continuous, dynamic data flows. These arduous environments are usually hard to be efficiently maintained by conventional techniques. A classical, well-suited problem for studying dynamic systems is the workload distribution in multi-skill call centres.

The basic variant of a workforce distribution problem requires the assignment of tasks to agents who have the required skills to handle them over the time, satisfying a given set of additional constraints and respecting the dependencies among individual tasks and differences in the execution skills of the agents. This problem has multiple variants but, depending on the dynamism of the system, we can principally distinguish two main scenarios:

- On the one hand, we can find short-term planning environments in which a continuous planning is needed due to the high dynamism of the system. These solutions attempt

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 178–189, 2010.

to distribute the workload among agents by applying "basic" ad-hoc heuristics, looking at the current situation. This feature can be easily seen in workload distribution within a dynamic multi-skill call centre [1].

- On the other hand, we can find long-term planning systems in which the list of tasks is predefined and known by all agents like in the classic scheduling problem [2]; or environments in which a single task is assigned to each agent for a long period of time, similarly to the job assignment problem [3]. In other cases, agents are assigned to patterns of tasks, instead of specific tasks (such as in pattern-based scheduling [2]). Analogously, stable multi-skill call centres [1] can be also included in this group. These solutions consider stable behaviour over the time and apply more complex algorithms to match agents and tasks. When having a dynamic system, these approaches cannot be efficiently applied, since an adaptive method is required.

In this way, we put forward an alternative approach to traditional solutions which relies on a middle-term time-frame, instead of a short-term or long-term one. In other words, we reformulate the initial problem by enlarging the time-frame considered and provide the required mechanisms to implement this more efficient solution.

Table 1 summarises some fundamental characteristics of the previously described environments in relation to the time-frame considered.

Table 1. Comparison of the time-frame considered for the workforce distribution problem

Time-frame	*Complexity*	*Response time*	*Adaptability*	*Performance*
Short-term planning	low	low	medium	medium
Middle-term planning	high	medium	high	high
Long-term planning	medium	high	low	low

In this work, we focus on the application of this model to a multifarious, dynamic real-world problem, the workforce distribution within a dynamic multi-skill call centre, considering real-time data flows during one-day campaign. Our main contribution is the presentation of a novel approach which coalesces forecasting with optimisation. Besides, our approach enhances workforce distribution by additionally injecting real-time knowledge of the task, individual skill sets, and availability and utilisation of the workforce, allowing for dynamic and active distribution of tasks to match load peaks over the time. Our method also provides additional clearness on customer service level agreements, and endows with insights into optimisation and the ability to offer outstanding customer service. In addition, our approach enables us to work at a lower level of granularity than short-term algorithms do, because the search algorithm has more time to find a solution. We can then work at agent's profile level instead of predefined sets of agents as other methods impose. We also propose a partial fitness function in order to speed-up the evaluations of candidate solutions.

The rest of this document is organised as follows: Section 2 describes the problem definition both intuitively and formally, highlighting the complexity of this application domain. Section 3 discusses the state-of-the-art. Section 4 proposes a novel approach to the workforce distribution problem in a dynamic multi-skill call centre. Section 5 provides an evaluation of the model and conducts an analysis of the

results. Finally, Section 6 concludes our work with a summary of major contributions and points out prospects for future work.

2 Problem Description

A Call Centre (CC) is a centralised place used for receiving and transmitting large volumes of requests by telephone [4]. In a CC, the flow of calls is often divided into outgoing and incoming traffic. *Incoming calls* are those that go from the client to the CC to contract a service, ask for information or report a problem. These calls are modelled and then classified into several Call Groups (CGs) in relation to the nature of each call. Once these CGs have been modelled (call types), each call is only assigned to a CG. Conversely, *outgoing calls* are significantly different to incoming calls. These calls are those the agents make to clients with commercial pretensions.

In a Multi-Skill CC (MSCC) there are k types of calls, n customer calls and m agents that may have up to l skills ($l \leq k$). This implies that each agent can attend different types of calls and, given a type of call, it can be answered by several agents that have that skill. Obviously, this scenario can be simpler in some special CCs in which agents have a single skill. These CCs can be modelled with q single queues working in parallel. In other cases, every agent has all possible skills; hence all customers are queued in a single queue that can be handled by any agent. The system is noticeably easier to analyse in these two extreme cases. With all agents having all skills, the system is also more efficient (shorter waiting times, fewer abandonment rates) when the service time distribution for a given call type does not depend on the agent's skill set. However, this assumption turns out to be wrong in practice: agents are usually faster when they handle a smaller set of call types (even if their training gives them more skills). Agents with more skills are also more expensive as their salaries depend on their skill sets. Thus, for large volumes of call types, it makes sense to dedicate a number of single-skill agents (specialists) to handle most of the load. A small number of agents, proportional to the calls of each type, with two or more skills can cover potential fluctuations in the arriving load. To address this point, the skills are grouped in profiles. Figure 1 illustrates the relationship among clients' calls, queues and agents.

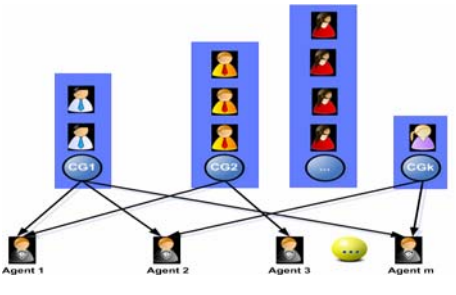


Fig. 1. Inbound scheme in MSCCs

More formally speaking, the following parameters can be found in an MSCC:

- a finite set of *n* customer calls $C = \{ c_1, c_2, ..., c_n \}$.
- a finite set of *k* CGs (call groups/types) $CG = \{ cg_1, cg_2, ..., cg_k \}$, where $k \leq n$ when every CG has, at least, one call queuing.
- a finite set of *m* agents $A = \{ a_1, a_2, ..., a_m \}$.
- a finite set of *k* agent-skills $S = \{ s_1, s_2, ..., s_k \}$ in which each agent-skill, s_i, represents the ability to handle the associated CG, cg_i, with the corresponding sub-index in CG: $s_1 \sim cg_1, s_2 \sim cg_2, ..., cg_k \sim cg_k$.
- a finite set of *d* agent-skill profiles $P = \{ P_1, P_2, ..., P_d \}$ in which each agent-skill profile P_i can be any subset of $S = \{ s_1, s_2, ..., s_k \}$.
- a finite set of *n* operations (execution or processing of each customer call, c_i) $O = \{ o_1, o_2, ..., o_n \}$ in which each operation, o_i, has associated a processing time which depends on its CG: $\{\tau_1, \tau_2, ..., \tau_k\}$.

Moreover, the solution must fulfil the following descriptions:

- on *O* define *A*, a binary relation which represents the precedence among operations. If $(o_1, o_2) \in A$ then o_1 has to be performed before o_2.
- each agent, a_i, has associated a finite non-null subset of *P*, containing his skills to handle different customer CGs.
- the same profile P^i can be assigned to several agents. In other words, several agents may have some skills in common (or even all of them).
- each agent, a_i, may have several profiles assigned but only one can be performed at a given instant *t*, $\langle a_i, P_j \rangle_t$. In other words, an agent cannot process two (or more) operations at the same instant.
- each solution must respect diverse (hard and soft) constraints given by business rules defined by business units or agents' regulations.

The purpose is to obtain the right assignment $\langle a_i, P_j \rangle_t$ every instant over a long period of time in a dynamic system, handling hard and soft constraints.

There are very significant metrics to measure the quality of a CC such as the abandonment and service rates. These metrics somehow hinge on the (customer) *service level* [5] which is defined as the percentage of customer calls that have to queue shorter than a specified amount of time. Our work is led by this metric.

The complexity of this problem is huge because we are not only dealing with an NP-hard problem like in the job assignment problem [3], but also considering high dynamism, massive incoming customer calls and large number of agents having multiple skills. Besides, since customer calls are not planned, this makes the call assignment a very laborious task.

3 Related Work

Reviewing the state-of-the-art, one can realise that many algorithms for workload distribution in Single-Skill CCs are available (e.g. [6]) because, in the past, agents

were commonly allocated to a single customer call group. Nevertheless, not much work has been conducted to workload distribution in MSCCs which is the typical situation in nowadays CCs. In the rest of this section, we discuss the main contributions to workforce distribution in MSCCs.

Workload distribution in MSCCs has been broadly faced by a Skill-Based Routing algorithm (SBR [7]). SBR is a call-assignment strategy used in CCs to assign incoming customer calls to the most suitable agent, instead of simply choosing the next existing agent. The need for SBR has arisen, as CCs have become larger and deals with a wider variety of call types. The major handicap of this approach is that online (ad-hoc) routing heuristics cannot be very complex in view of the fact that a very short response time is required. These fast, unplanned decisions may imply suboptimal task assignments to existing agents.

Conversely, Thompson [8] proposes an integer programming model which differentiates minimum acceptable service levels per time-frame from a constraint on the mean service level over the planning horizon. Although this approach considers prospective situations, it is less dynamic to changes than SBR.

Other approaches consider dependent planning intervals (see [9]). Most methods perform well enough within separate intervals but their performance decreases when moving to the next one, giving much trouble in prospective time-frames.

Other authors take into consideration overflow routing in multi-skill blocking systems with randomisation parameters by applying a branch-and-bound algorithm (see [10]) or cutting planes (see [11]). These techniques are only appropriate for stable environments because they need long response times and their performance highly decreases in large instances.

Finally, we can find one of the most representative algorithms of the state-of-the-art (Koole et al., 2008 [12]). It presents a heuristic, which considers the costs of agents and a service-level condition, to optimise the separation of agents among different CGs. This algorithm is faster than most of the previous approaches but deals with specific types of MSCCs in which customer calls arrive according to a Poisson process with deterministic rate. However, note that inbound flow in MSCCs is usually not a stationary Poisson process [13] and, the service times do not increase exponentially. Since calls arrive randomly according to a stochastic process, agents must be well-distributed to handle the calls as soon as possible. Besides, the previous techniques often consider a high granularity and need to work at agent groups' level instead of an agent's profile level. This setback does not enable us to offer more accurate configurations.

To conclude, we have seen in this section that some approaches employ "basic" heuristics to dynamically distribute incoming customer calls to agents while others cope with stable inbound flows and longer stability over the time. In this context, a large-time-frame planning cannot be carried out because of the continuous changeability of all variables involved. Moreover, "basic" heuristics based on the current situation (online routing strategies) may work under certain cases, e.g. stable workload, but daily use of these techniques will guide us to appalling solutions.

4 A Novel Approach

In this section, we propose a novel approach for workforce distribution in MSCCs by reformulating the original problem definition. We have illustrated in Section 3 how "basic" techniques distribute incoming customer calls as these come by employing greedy heuristics while others contend with a stable incoming customer call flow and a longer stability over the time which is not the archetypal situation in nowadays MSCCs. However, if we could forecast the real situation of an upcoming time-frame with a very high-confidence, then we would be able to apply more complex techniques which can outperform both short-term and long-term planning strategies. There, a need of an exact prediction of a middle-term situation comes out. Afterwards, we need to obtain a fair balance between diversity (exploration) and intensity (exploitation) to meet with success, making the most of the forecasting provided in the previous step.

The main purpose of the rest of this section is to provide a solution for this new problem definition which combines predictions for middle-term time-frames in changing systems and a powerful search mechanism founded on a memetic algorithm.

4.1 Forecast Module

Forecasting refers to the estimation of unknown variables in prospective situations. An exact prediction allows us to accurately balance workload among agents, giving enhanced service levels and optimising resources. Conventionally, customer call arrivals have been approximated according to a Poisson process. Assuming pure-chance arrivals and terminations leads to the following probability distribution (1):

$$P(n) = \left(\frac{\mu^n}{n!} \right) e^{-\mu} \tag{1}$$

Much literature (e.g. [12,14]) considers that the number of customer call arrivals at a given time follows a Poisson distribution, where n is the number of call arrivals in an interval T, and μ is the mean of call arrivals in that interval. However, the prediction of customer call arrivals in an MSCC does not often adjust to a Poisson distribution with deterministic rate. In all studies, the arrival process agrees with a Poisson process only if the arrival rate of the Poisson process is a stochastic process itself. Typically, the variance of the incoming calls in a given interval is much larger than the mean. However, it should be equal to the mean for Poisson distributions.

In [15], it is explained how to model an Improved Backpropagation Neural Network, step-by-step, to forecast unknown variables, bearing in mind the nature of a real MSCC rather than following a Poisson distribution when predicting. In this paper, we consider the Neural Network described in [15] to predict incoming customer calls in our experiments. The number of agents is also forecasted in line with the same mechanism, considering their timetables (break, start and leave times).

4.2 Search Module

To perform the search module, a Memetic Algorithm (MA) is proposed. Fundamentally, this combines GAs' operators with a Local Search (LS) heuristic to refine candidate solutions.

4.2.1 Methodology

Given the predictions from the forecast module; the search module, implemented as a steady-state MA, optimises the assignment among tasks and agents. Our MA maintains a set (population) of abstract representations (chromosomes) of candidate solutions (phenotypes) to the problem described in Section 2. The population is partially randomly initialised (Section 4.2.4). Then, its individuals are evaluated by applying a fitness function over them. From this population, some individuals are selected and, then, recombined (crossover). Subsequently, the offspring may suffer mutations in some genes. Afterwards, some of these individuals replace others from the population according to the replacement scheme. Every generation includes all previous actions. Finally, an LS mechanism is applied over a percentage of the population each *n* generations. All these steps are carried out until a predefined time has been elapsed.

4.2.2 Encoding

The first stage when designing an MA is to define a problem representation (chromosome or genotype) to encode candidate solutions (phenotype) to the problem in a form that every computer can interpret. The "physical" expression of the genotype is called the phenotype. This means that a mapping between genotype and phenotype must be defined. There are multiple forms to encode candidate solutions which range from binary strings, arrays of integers or arrays of decimal numbers to strings of letters. Concretely, our solution consists of an integer representation. We just need an array of integers whose indexes represent the available agents, $A^a \subseteq A$, at a given instant, *t*, and the array contents refer to the profile, P_j, assigned to each agent a_i. Then, call types are routed to the agents, according to the profiles assigned.

Figure 2 shows a fictitious example of encoding for 10 customer calls (c_0-c_9) queued in 3 different CGs (cg_0-cg_2) depending on the nature of the calls, 5 agents (a_0-a_4) and 4 profiles (P_0-P_3), where P_0={s_0, s_1}, P_1={s_1}, P_2={s_2} and P_3={s_1, s_2}. Now, suppose that a_0~{P_0,P_1}, a_1~{P_0, P_2}, a_2~{P_1,P_3}, a_3~{P_2,P_3} and a_4~{P_0,P_1}. We have seen the potential profiles for every agent but only one profile can be assigned to each agent at a given instant *t*; therefore, a feasible solution would be Figure 2.

Index (agents) →	0	1	2	3	4
Content (profiles) →	P_0	P_2	P_1	P_2	P_0

Fig. 2. Example of encoding

4.2.3 Population

The population of our MA is a compilation of chromosomes encoded as hinted in Section 4.2.2. The population is the minimum unit of evolution since individuals are static elements by themselves. This evolution can be observed in the changes

produced in the genetic configuration over the time in each successive generation. The changes between two generations are usually small but these differences mount up with each generation, causing significant changes in the "original" population.

The size of the population often depends on the nature of the problem and typically contains tens or hundreds of possible candidate solutions. After an empirical study, we have considered a single population of 20 individuals.

4.2.4 Initialisation

Typically, the initial population is fed with randomly generated individuals who should potentially cover different possible configurations. In some cases, we can use other algorithms to initialise the population (e.g. LS) but, in most cases, this is not possible since computing times increase too much and real applications require short computing times. In our case, we propose to start from a random initial population, including the best solution found in the previous time-frame because the configuration of agents' profiles should not change too much over two successive time-frames.

4.2.5 Fitness Function

The fitness function is an evaluating mechanism which is defined over the chromosome to measure the quality of a given solution. This function often guides the search and decides which individuals must be selected for the next generation (in fact, it also depends on the replacement policy). The fitness function is intrinsically linked to the problem. Frequently, the hardest action when defining an EA is to identify the right fitness function since results strongly depend on it. Occasionally, it is hard (sometimes impossible) to characterise the fitness expression; in these cases, interactive genetic algorithms are used. In other cases, long evaluating times imply that an approximate function is needed. Our fitness function is inspired in the estimation of the *total service level* provided in [5] although we also consider the priority of each CG weighted as follows:

$$Total\_service\_level = \sum_{i=0}^{k} (Pr_i \times SL_i(\gamma_i, \alpha_i)) \times \mu \quad \{sl : \Re \times [0,1] \times [0,1] \rightarrow [0,1]\}$$ where k refers to the number of CGs, μ is a normalising factor $(1/\sum_{i=0}^{k} Pr_i)$, Pr_i is the priority of the CG-i and its service level is $SL_i(\gamma_i, m_i) = 1 - P(\text{Agents\_are\_busy}) \times e^{-(\gamma_i - m_i)\frac{\tau_i}{\beta}}$ given that $P(\text{Agents\_are\_busy}) = \left[1 + \frac{\gamma_i - m_i}{m_i} \sum_{\zeta=0}^{\gamma_i - 1} \frac{(\gamma_i - 1)...(\zeta + 1)}{m_i^{\gamma_i - \zeta - 1}}\right]^{-1}$ where γ_i is the load of CG-i ($n_i \times \tau_i$), m_i is the number of agents of CG-i (based on the profiles assigned in the chromosome), τ_i is the number of agents of CG-i and β is the duration of the time-frame expressed in seconds. (2)

Additionally, we handle some hard and soft constraints derived from the business rules given by our business units. In our case, these constraints are associated to tasks, agents, timing, actions or desired/undesired scenarios. Thus, the algorithm cannot violate hard constraints (e.g. we cannot change agents' profiles continuously due to

certain laws and regulations); although we allow certain movements which may imply the violation of some soft constraints (e.g. we should not take agents from CGs in which the service level is below a given threshold). Undoubtedly, this type of movements is penalised according to the degree of non-accomplishment of these constraints and their relevance. Relevance values (weights) can be assigned by defining levels of constraints. For each level of constraints, we can define a range of values for the weights and the gap between two levels follows a logarithmic function to soften differences among levels. The values for each level should be assigned proportionally. These premises lead us to maximise the following *fitness function*:

$$\text{f} = (\text{total\_service\_level} - \text{penalisations\_constraints}) \quad f : [0,1] \times [0,1] \rightarrow [-1,1] \tag{3}$$

where *penalisation_constraints* is the value obtained after applying our business rules (e.g. agents from CG-*i* should not move to CG-*j*).

Finally, we can speed-up the evaluations by introducing a *partial fitness function*. The first time, we need to employ (3) but the rest of the time; we just need to evaluate those groups affected by a mutation or, in the case of the LS, when generating a new neighbour. Hence, we only process the affected CGs in (2) and update their original values. With this information, we then recalculate (3).

4.2.6 Genetic Operators

In this section, we briefly comment the final configuration of the genetic operators as the reasoning of this selection is a very important matter and deserves to be presented in a separate study. The configuration of our operators is the following one:

Selection: Since the population needs to be bred each successive generation, we have chosen a binary tournament selection as described in [16].

Crossover: The following step is to produce a new generation from selected individuals. We consider that children will inherit the common points in their parents (potentially, the best genes) and randomly receive the rest of genes from them [17].

Mutation: This operator causes tiny changes in the genes of the chromosome to explicitly maintain diversity (actually there are much more mechanisms). We apply a perturbation over each gene of the chromosome with a probability of 0.03. This perturbation corresponds to changes of profiles in some agents (e.g. agent a_2 who had assigned the *profile* P_1 has now associated the *profile* P_3 due to a mutation).

Replacement policy: Finally, we decide which individuals are incorporated (or maybe reinserted) into the population of our steady-state algorithm. We consider elitism [18] to replace the worst individuals of the population by the best ones.

4.2.7 Local Search

LS is a metaheuristic for solving optimisation problems. An LS algorithm starts out from a candidate solution and, thus, iteratively moves to a neighbour solution, generating the neighbourhood. To carry out this action, a neighbourhood relation must be defined on the search space. In our case, we state that two candidate solutions are neighbours if only one gene differs in both chromosomes. Note that we propose a simple LS due to the lack of time of our production environment (300 seconds).

The following pseudo-code illustrates the LS algorithm which is applied each 50 generations over the 5-best individuals:

```
void Local_Search (Chromosome & candidate_solution)
  Chromosome best_solution = candidate_solution;
  Chromosome neighbour = candidate_solution;
  For (i=0; i<candidate_solution.size(); i++)
      Agent a = neighbour.getAgent(i);
      For (j=0; j<a.get_number_profiles(); j++)
            neighbour.change_profile(i,j); //profile j for agent i
            If (neighbour.fitness() > best_solution.fitness())  best_solution = neighbour;
      neighbour = best_solution;
  candidate_solution = neighbour;
```

5 Evaluation of Results

In this section, we analyse the results obtained by our algorithm (forecast module + search module), during a demanding working day (there was a commercial campaign during the day which has been measured). In this way, we have run the algorithm over a whole day with approximately *315.000* calls (up to *28.800* calls/hour and *2.450* simultaneous calls) under three double-core processors of a *Sun Fire E4900* server (one processor for the forecast module, another one for the interfaces and data pre-processing, and the last one for the search module) with *96GB RAM*. The mean number of agents in each time-frame is *2.100*, having *16* different skills for each agent on average (*minimum=1* and *maximum=108*). The total number of CGs is *820*. The mean processing times differ a lot, depending on the CG (from seconds to minutes). All data were taken from the MSCC of a real-world environment within a large multinational telephone operator.

Once the magnitude of our MSCC has been presented, we compare our approach with classical SBR [7], ED-SBR (an improvement of classic SBR [7]) and Koole's algorithm [12]. Figure 3 illustrates the real service level given by these techniques during a demanding working day. The graphs compile the real service levels for each CG, considering the relevance (weight) of each one. Since incoming traffic mainly arrives from 9 a.m. till 8 p.m.; therefore, we need more accurate results for this time-interval and, particularly, for the peaks which occur around 13 p.m. (see point *32* in Figure 3), 15 p.m. (see point *66* in Figure 3) and 19 p.m. (see point *100* in Figure 3) because, in these points, the load is much higher. Our approach clearly improves the results reached by other algorithms in these critical points (peaks). For the rest of points, we see that our algorithm usually better behaves than the rest of techniques. Classic SBR and ED-SBR sometimes offer a similar configuration of agents than our approach for some points and, consequently, the same service levels; but, on average, the service levels are clearly worse than ours. Only in few points, the service level of ED-SBR and SBR is higher than ours (e.g. around 11:45, point *17*). This happens because in these points, our predictions had a greater error. However, we can see that differences are tiny in these critical points and we present more stable results over the time. This corroborates that a middle-term time-frame is recommended as algorithms can reach nearly optimal solutions while short-term algorithms often collapse in local optimums. But, short-term algorithms present a high adaptability to changes that

long-term time-frame techniques cannot cope with. These long-term based techniques generally extract patterns from the historical and are only appropriate for stable environments. For this reason, our algorithm and SBR outperform Koole's approach which is designed for more stable MSCCs. Koole's algorithm finds very accurate solutions when the dynamism is more reduced such as classical staffing. Nevertheless, this is not the case of our environment and this kind of techniques cannot be efficiently applied to our MSCC.

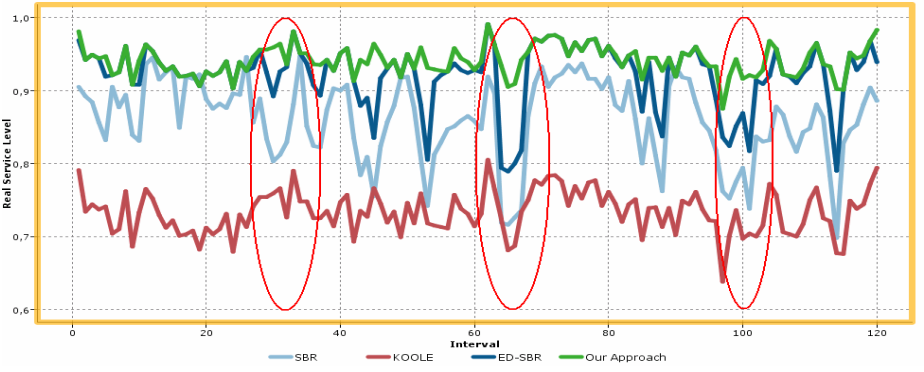


Fig. 3. Service level given by different techniques for a whole day

Table 2 compares the results obtained by all techniques presented in Figure 3. Table 2 presents the mean service level for 120 intervals, its standard deviation and the effectiveness, considering that our method represents the highest performance.

Table 2. Comparison of our approach and other relevant algorithms for 120 intervals

Algorithm	*Mean Service Level*	*Standard Deviation*	*Effectiveness*
Our Approach	0.941	0.020	100
ED-SBR	0.901	0.043	95.757
SBR	0.860	0.056	91.405
KOOLE	0.733	0.029	77.896

6 Conclusions and Future Work

In this paper, we have described a novel approach for workforce distribution in dynamic MSCCs which combines predictions and optimisations. We have seen that the traditional formulation of the problem cannot be satisfied by complex metaheuristics such as memetic algorithms, since these methods take a longer time to reach an optimal solution. However, we have illustrated that a reformulation of the problem enables us to enlarge the time-frame considered when planning in order to give more time to the algorithms. To handle this reformulation of the problem, we have proposed a neural network for predicting call arrivals and agents, and a memetic algorithm to carry out the assignment of incoming calls to agents. Afterwards, we have seen that our algorithm outperforms classical approaches to this problem in a demanding real-world environment. To conclude, we propose some guidelines for

future work. The following step will be to parallelise our algorithm and deeply study those factors which enable us to balance diversity (exploration) and intensity (exploitation). Additionally, we propose to do an analogous study, considering other metaheuristics (namely, SA and TS) and other multi-objective evolutionary approximations (such as SPEA-II and NSGA-II) given our problem reformulation.

Acknowledgements

This work has been partially supported by INTERLIGARE Institute for Innovation in Intelligence (I4), by Spanish Government Research Grants. CICYT TIN 2008-00508 and MEC Consolider Ingenio 2010 2007/2011 of the Spanish Council of Science and Technology. We would like to thank Severino F. Galan for his support.

References

1. Avramidis, A.N., Chan, W., Gendreau, M., L'Ecuyer, P., Pisacane, O.: Optimizing daily agent scheduling in a multiskill CC. European Journal of Operational Research (2009)
2. Brucker, P.: Scheduling algorithms, 2nd edn. Springer, Heidelberg (1998)
3. Chauvet, F., Proth, J.M., Soumare, A.: The simple and multiple job assignment problems. International Journal of Production Research 38(14), 3165–3179 (2000)
4. Bhulaii, S., Koole, G., Pot, A.: Simple Methods for Shift Scheduling in Multiskill Call Centers. M&SOM 10(3), 411–420 (2008)
5. Koole, G.: Call Center Mathematics: A scientific method for understanding and improving contact centers (2006), http://www.cs.vu.nl/~koole/ccmath/book.pdf
6. Whitt, W.: Staffing a call center with uncertain arrival rate and absenteeism. PO&M (2006)
7. Garnett, O., Mandelbaum, A.: An Introduction to Skills-Based Routing and its Operational Complexities, Teaching Note (2000)
8. Thompson, G.M.: Labor staffing and scheduling models for controlling service levels. Naval Res. Logist., 719–740 (1997)
9. Ingolfsson, A., Cabral, E., Wu, X.: Combining integer programming and the randomization method to schedule employees. TR, School of Business, University of Alberta (2007)
10. Land, A.H., Doig, A.G.: An automated method for solving discrete programming problems. Econometrica 28, 497–520 (1960)
11. Gomory, R.E.: Outline of an algorithm for integer solutions to linear programs. Bull. Amer. Math. Soc. 64, 275–278 (1958)
12. Bhulai, S., Koole, G., Pot, A.: Simple Methods for Shift Scheduling in Multiskill Call Centers. Published online in Articles in Advance (January 4, 2008)
13. Ahrens, J.H., Ulrich, D.: Computer Methods for Sampling from Gamma, Beta, Poisson and Binomial Distributions. Computing 12(3), 223–246
14. Koole, G., Pot, S., Talim, J.: Routing heuristics for multi-skill call centers. In: Proceedings of the Winter Simulation Conference, pp. 1813–1816 (2003)
15. Pacheco, J., Millán-Ruiz, D., Vélez, J.L.: Neural Networks for Forecasting in a Multi-skill Call Centre. In: EANN 2009, London, UK, August 27-29 (2009)
16. Prügel-Bennett, A.: Finite Population Effects for Ranking and Tournament Selection. Complex Systems 12(2), 183–205 (2000)
17. Pearson, D.W., Steele, N.C., Albrecht, R.F.: Artificial neural nets and genetic algorithms. In: Proceedings of the international conference, Roanne, France (2003)
18. Chakraborty, B., Chaudhuri, P.: On The Use of Genetic Algorithm with Elitism in Robust and Nonparametric Multivariate Analysis. Austrian Journal of statistics 32 (2003)

Geometric Generalization of the Nelder-Mead Algorithm

Alberto Moraglio and Colin G. Johnson

School of Computing, University of Kent, Canterbury, UK
{a.moraglio,c.g.johnson}@kent.ac.uk

Abstract. The Nelder-Mead Algorithm (NMA) is an almost half-century old method for numerical optimization, and it is a close relative of Particle Swarm Optimization (PSO) and Differential Evolution (DE). Geometric Particle Swarm Optimization (GPSO) and Geometric Differential Evolution (GDE) are recently introduced formal generalization of traditional PSO and DE that apply naturally to both continuous and combinatorial spaces. In this paper, we generalize NMA to combinatorial search spaces by naturally extending its geometric interpretation to these spaces, analogously as what was done for the traditional PSO and DE algorithms, obtaining the Geometric Nelder-Mead Algorithm (GNMA).

1 Introduction

The Nelder-Mead Algorithm published by Nelder and Mead in 1965 [9] is a numerical optimization method: despite its age, it is the method of choice for many practitioners. Contrasted with the majority of classic methods for numerical optimization, it only uses the values of the objective function without any derivative information. The search done by NMA is based on geometric operations (reflection, expansion, contraction and shrinking) on a current set of points, seen as the corners of a n-dimensional polygon (a simplex), to determine what points in space to evaluate next. The overall behaviour of the NMA expands or focuses the search adaptively on the basis of the topography of the fitness landscape.

Interestingly, the NMA can be seen as a form of (population-based) evolutionary algorithm with special selection and reproduction operators [13]. Also, there are similarities between the search operators employed by the NMA and those of DE [12] and PSO [2] that have led a number of authors to propose hybrid approaches (see for example [15] and [3]). As the original versions of DE and PSO, NMA requires the search space to be continuous and the points in space to be represented as vectors of real numbers. To the authors's best knowledge, there are no generalizations of the NMA to combinatorial spaces.

Both of the searches done by PSO and DE have natural geometric interpretations and both can be understood as the motion of points in space obtained by (different but related) linear combinations of their current and past positions to determine their new positions. Geometric Particle Swarm Optimization [5] and

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 190–201, 2010.

Geometric Differential Evolution [8] are recently devised formal generalizations of PSO and DE that, in principle, can be specified to any solution representation while retaining the original geometric interpretation of the dynamics of the points in space across representations. In particular, these formal algorithms can be applied to any search space endowed with a distance and associated with any solution representation to derive formally specific PSO and DE algorithms for the target space and for the target representation. Specific GPSOs were derived for different types of continuous spaces and for the Hamming space associated with binary strings [6], for spaces associated with permutations [7] and for spaces associated with genetic programs [14]. GDE was specialized to the space of binary strings endowed with the Hamming distance [8]. The derived algorithms performed satisfactorily in experimental results. This suggests that the generalization methodology employed is a promising one.

In the present paper we generalize the Nelder-Mead Algorithm to combinatorial spaces extending its geometric interpretation to these spaces, analogously to what was done for the traditional PSO and DE algorithms, derive the specific GNMA for the Hamming space associated with binary strings and present experimental results on standard benchmark problems.

2 Classic Nelder-Mead Algorithm

In this section, we describe the traditional NMA [9] (see Algorithm 1). The NMA uses $n + 1$ points in $\mathbf{R}^n$. These points form a type of n-dimensional polygon, a simplex, which has $n + 1$ points as vertices in $\mathbf{R}^n$. For example, the simplex is a triangle in $\mathbf{R}^2$ and a tetrahedron in $\mathbf{R}^3$. The initial simplex has to be non-degenerate, i.e., the points must not lie in the same hyperplane. This allows the NMA to search in all n dimensions. The method then performs a sequence of transformations of the simplex, which preserve non-degeneracy, aimed at decreasing the function values at its vertices. At each step, the transformation is determined by computing one or more test points and comparing their function values. In Figure 1, we illustrate the NMA transformations for the two-dimensional case, where the simplex S consists of three points.

The optimization process described by Algorithm 1 starts with creating a sample of $n + 1$ random points in the search space. Notice that apart from the creation of the initial simplex, all further steps are deterministic and do not involve random choices. In each loop iteration, the points in the simplex S are arranged in ascending order according to their corresponding objective values. Hence, the best solution candidate is $S[0]$ and the worst is $S[n]$. We then compute the center m of the n best points and then reflect the worst candidate solution $S[n]$ through this point, obtaining the new point r as also illustrated in Fig. 1(a). The reflection parameter α is usually set to 1. In the case that r is neither better than $S[0]$ nor as worse as $S[n]$, we directly replace $S[n]$ with it. If r is better than the best solution candidate $S[0]$, we expand the simplex further into this promising direction. As sketched in Fig. 1(b), we obtain the point e with the expansion parameter γ set to 1. We now take the best of these two points to

Algorithm 1. Nelder-Mead Algorithm

Input: f: the objective function to minimize
Input: $n+1$: number of points in the simplex
Input: α, ρ, γ, σ: reflection, expansion, contraction and shrink coefficients
Output: x^*: the best solution found

$S \leftarrow createPop(n+1)$
while stop criterion not met **do**
  $S \leftarrow sortPop(S, f)$
  // Center of mass: determine the center of mass of the n best points
  $m \leftarrow \frac{1}{n} \sum_{i=0,n-1} S[i]$
  // Reflection: reflect the worst point over m
  $r \leftarrow m + \alpha(m - S[n])$
  if $f(S[0]) < f(r) < f(S[n])$ **then**
    $S[n] \leftarrow r$
  else
    if $f(r) \leq f(S[0])$ **then**
      // Expansion: try to search farther in this direction
      $e \leftarrow r + \gamma(r - m)$
      if $f(e) < f(r)$ **then**
        $S[n] \leftarrow e$
      else
        $S[n] \leftarrow r$
      end if
    else
      $b \leftarrow true$
      if $f(r) \geq f(S[n-1])$ **then**
        // Contraction: a test point between r and m
        $c \leftarrow \rho r + (1-\rho)m$
        if $f(c) < f(r)$ **then**
          $S[n] \leftarrow c$
          $b \leftarrow false$
        end if
      end if
      if b = true **then**
        // Shrink towards the best solution candidate $S[0]$
        for i from n down to 1 **do**
          $S[i] \leftarrow S[0] + \sigma(S[i] - S[0])$
        end for
      end if
    end if
  end if
end while
return $S[0]$

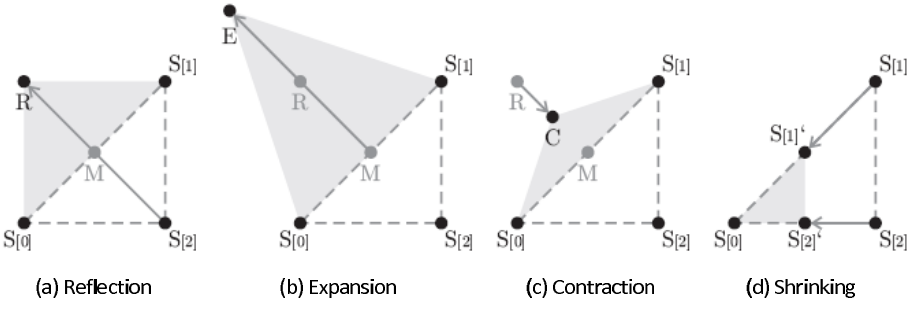


Fig. 1. One step of the NMA in R^2 (figure modified from [16])

replace $S[n]$. If r is no better than $S[n]$, the simplex is contracted by creating a point c somewhere in between r and m. In Fig. 1(c), the contraction parameter ρ was set to $1/2$. We substitute $S[n]$ with c only if c is better than r. When everything else fails, we shrink the whole simplex by moving all points (except $S[0]$) into the direction of the current optimum $S[0]$. The shrinking parameter σ normally has the value $1/2$, as is the case in the example outlined in Fig. 1(d).

3 Geometric Nelder-Mead Algorithm

In this section, we derive the general Geometric Nelder-Mead Algorithm (Algorithm 2) from the classic Nelder-Mead Algorithm (Algorithm 1). The generalization is obtained using a methodology to generalize search algorithms for continuous spaces to combinatorial spaces [8] based on the geometric framework introduced by Moraglio [4], sketched in the following.

1. Given a search algorithm defined on continuous spaces, one has to recast the definition of the search operators expressing them explicitly in terms of Euclidean distance between parents and offspring.
2. Then one has to substitute the Euclidean distance with a generic metric, obtaining a formal search algorithm generalizing the original algorithm based on the continuous space.
3. Next, one can consider a (discrete) representation and a distance associated with it (a combinatorial space) and use it in the definition of the formal search algorithm to obtain a specific instance of the algorithm for this space.
4. Finally, one can use this geometric and declarative description of the search operator to derive its operational definition in terms of manipulation of the specific underlying representation.

As mentioned in the introduction, this methodology was used to generalize PSO and DE to any metric space obtaining GPSO and GDE and then to derive the specific search operators for a number of specific representations and distances.

Following the methodology outlined above, in the following we generalize the classic Nelder-Mead Algorithm to general metric spaces. To do this, we recast the search operations described in the previous section (reflection, expansion, contraction and shrinking) as functions of the distance of the underlying search space, thereby obtaining their abstract geometric definitions. Then, in Section 4, we derive the specific GNMA for the Hamming space associated with binary strings by plugging this distance in the abstract definition of the search operators.

3.1 Geometric Generalization of the Nelder-Mead Algorithm

Using the notion of convex combination CX, extension ray ER and center of mass CM we can generalize all search operators of the classical Nelder-Mead Algorithm from the Euclidean case to generic metric spaces because, as we will see in the following sections, these are geometric elements well-defined on any metric space.

Algorithm 2. Formal Nelder-Mead Algorithm

Input: f: the objective function to minimize
Input: $n+1$: number of points in the simplex
Input: α, ρ, γ, σ: reflection, expansion, contraction and shrink coefficients
Output: x^*: the best solution candidate found

$S \leftarrow createPop(n+1)$
while stop criterion not met **do**
  $S \leftarrow sortPop(S, f)$
  // Center of mass: determine the center of mass of the n best points
  $m \leftarrow CM(S[0], S[1], ..., S[n-1])$
  // Reflection: reflect the worst point over m
  $r \leftarrow ER(S[n], m)$ with weights $(\frac{\alpha}{1+\alpha}, \frac{1}{1+\alpha})$
  if $f(S[0]) < f(r) < f(S[n])$ **then**
    $S[n] \leftarrow r$
  else
    if $f(r) \leq f(S[0])$ **then**
      // Expansion: try to search farther in this direction
      $e \leftarrow ER(m, r)$ with weights $(\frac{1}{\gamma}, \frac{\gamma-1}{\gamma})$
      if $f(e) < f(r)$ **then**
        $S[n] \leftarrow e$
      else
        $S[n] \leftarrow r$
      end if
    else
      $b \leftarrow true$
      if $f(r) \geq f(S[n-1])$ **then**
        // Contraction: a test point between r and m
        $c \leftarrow CX(r, m)$ with weights $(\rho, 1-\rho)$
        if $f(c) < f(r)$ **then**
          $S[n] \leftarrow c$
          $b \leftarrow false$
        end if
      end if
      if b = true **then**
        // Shrink towards the best solution candidate $S[0]$
        for i from n down to 1 **do**
          $S[i] \leftarrow CX(S[0], S[i])$ with weights $(1-\sigma, \sigma)$
        end for
      end if
    end if
  end if
end while
return $S[0]$

The graphical description of the search operations of NMA (Fig. 1) leads directly to their geometric interpretation in terms of convex combination and extension ray, as follows. The reflection of the worst point $S[n]$ over M can be seen as picking a point beyond M on the extension ray originating in $S[n]$ and passing through M. The expansion operation can be seen as picking a point beyond R on the extension ray originating in M and passing through R. The contraction operation can be seen as picking a point in the segment between R and M. The shrink of all points $S[i]$ towards the best in the population $S[0]$ can be seen as replacing each point $S[i]$ with a point in the segment between $S[i]$ and $S[0]$.

In the following, we rewrite the algebraic definitions of the search operations of NMA to determine the weights of the corresponding convex combination or extension ray combination.

The definition of the reflection operation is $r = m + \alpha(m - S[n])$ (see Algorithm 1, line 12) and it can be rewritten as $m = \frac{\alpha}{1+\alpha}S[n] + \frac{1}{1+\alpha}r$. Since the coefficients of $S[n]$ and r are positive and sum up to 1 (for $\alpha \in [0, 1]$), this equation says that m is the convex combination of $S[n]$ and r with those coefficients. However, since r is the unknown and $S[n]$ and m are given, we can determine r as the inverse operation of the convex combination above, which is the extension ray combination with origin in $S[n]$ passing through m and keeping the same weights $(\frac{\alpha}{1+\alpha}, \frac{1}{1+\alpha})$ of the convex combination.

The definition of the expansion operation is $e = r + \gamma(r - m)$ (see Algorithm 1, line 18) and it can be rewritten as $r = \frac{1}{\gamma}m + \frac{\gamma-1}{\gamma}e$, which for $\gamma > 1$ is a convex combination of m and e returning r. Analogously as the reflection operation, since e is unknown and m and r are given, we can determine e by the extension ray combination with origin in m passing through r with weights $(\frac{1}{\gamma}, \frac{\gamma-1}{\gamma})$.

The definition of the contraction operation is $c = \rho r + (1 - \rho)m$ (see Algorithm 1, line 28), which for $\rho \in [0, 1]$ is a convex combination of r and m with weights $(\rho, 1 - \rho)$ returning c.

The definition of the shrink operation for a point $S[i]$ is $S[i]' = S[0] + \sigma(S[i] - S[0])$ (where $S[i]'$ denotes $S[i]$ at the next time step) (see Algorithm 1, line 37). This can be rewritten as $S[i]' = (1 - \sigma)S[0] + \sigma S[i]$, which for $\sigma \in [0, 1]$ is a convex combination of $S[0]$ and $S[i]$ with weights $(1 - \sigma, \sigma)$ returning $S[i]'$.

By replacing in Algorithm 1 the original operations defined on the Euclidean space with their generalized definitions we obtain the definition of a Formal Nelder-Mead Algorithm valid for any metric space (see Algorithm 2).

3.2 Convex Combination, Extension Ray and Center of Mass

Center of mass, segments and extension rays in the Euclidean space and their weighted extensions can be expressed in terms of distances, hence, these geometric objects can be naturally generalized to generic metric spaces by replacing the Euclidean distance with a generic metric.

Let (S, d) be a metric space. A (metric) segment is a set of the form $[x; y] = \{z \in S | d(x, z) + d(z, y) = d(x, y)\}$ where $x, y \in S$. The notion of convex combination in metric spaces was introduced in the GPSO framework [5]. The convex combination $C = CX((A, W_A), (B, W_B))$ of two points A and B with weights W_A and W_B (positive and summing up to one) in a metric space endowed with distance function d returns the set of points C in the segment $[A; B]$ such that $d(A, C)/d(A, B) = W_B$ and $d(B, C)/d(A, B) = W_A$ (the weights of the points A and B are inversely proportional to their distances to C). When specified to Euclidean spaces, this notion of convex combination coincides with the traditional notion of convex combination of real vectors.

The extension ray $ER(A, B)$ in the Euclidean plane is a semi-line originating in A and passing through B (note that $ER(A, B) \neq ER(B, A)$). The notion of extension ray in metric spaces was introduced in the GDE framework [8]. The weighted extension ray ER is defined as the inverse operation of the weighted convex combination CX, as follows. The weighted extension ray

Algorithm 3. Binary Convex Combination Operator

```
inputs: binary strings A and B and weights W_A and W_B (weights must be positive and sum up to 1)
for all position i in the strings do
    if random(0,1) ≤ W_A then
        set C(i) to A(i)
    else
        set C(i) to B(i)
    end if
end for
return string C as offspring
```

$ER((A, w_{ab}), (B, w_{bc}))$ of the points A (origin) and B (through) and weights w_{ab} and w_{bc} returns those points C such that their convex combination with A with weights w_{bc} and w_{ab}, $CX((A, w_{ab}), (C, w_{bc}))$, returns the point B.

The notion of center of mass can be generalized to generic metric spaces, as follows. The center of mass CM of a set of points $p_1, ..., p_n$ in a metric space (S, d) is the point $p \in S$ that minimizes its average distance to that set of points, i.e. $CM(p_1, ..., p_n) = argmin_{p \in S} \frac{\sum_{i=1,..,n} d(p_i, p)}{n}$.

4 Binary NMA

In this section, we present convex combination, extension ray and center of mass operators for the Hamming space on binary strings, and show formally that they meet their geometric specifications presented in the previous section. These specific operators can be plugged in the formal NMA (Algorithm 2) to obtain a specific GNMA for the Hamming space, the Binary GNMA.

4.1 Convex Combination and Extension Ray

The convex combination operator in metric spaces was introduced in the GPSO framework [5]. When specified to Euclidean spaces, this notion of convex combination coincides with the traditional notion of convex combination of real vectors. In the Euclidean space, the output point C of a convex combination $CX((A, W_A), (B, W_B))$ is uniquely determined, however this is not the case for all metric spaces. In particular, it does not hold for Hamming spaces. When CX is formally specified to Hamming spaces on binary strings, we obtain the recombination operator outlined in Algorithm 3 [5], which is a weighted form of uniform crossover. This algorithm returns offspring C in the Hamming segment between A and B such that $hd(A, C)/hd(B, C) = W_B/W_A$ in expectation. This differs from the Euclidean case where this ratio is guaranteed.

The notion of extension ray in metric spaces was introduced in the GDE framework [8]. When specified to Euclidean spaces, this notion of extension ray coincides with the traditional notion. Analogously as for the convex combination case, in the Euclidean space, the output point C of an extension ray combination $ER((A, w_{ab}), (B, w_{bc}))$ is uniquely determined, however this is not the case for all metric spaces. In particular, it does not hold for Hamming spaces. When ER is

Algorithm 4. Binary Extension Ray Recombination

inputs: binary strings A (origin) and B (through) of length n and weights W_{AB} and W_{BC} (weights must be positive and sum up to 1)
set $HD(A,B)$ as Hamming distance between A and B
set $HD(B,C)$ as $HD(A,B) \cdot w_{AB}/w_{BC}$ (compute the distance between B and C using the weights)
set p as $HD(B,C)/(n - HD(A,B))$ (this is the probability of flipping bits away from A and B beyond B)
for all position i in the strings **do**
    set $C(i) = B(i)$
    if $B(i) = A(i)$ and random(0,1) $\leq p$ **then**
        set $C(i)$ to the complement of $B(i)$
    end if
end for
return string C as offspring

Algorithm 5. Binary Center of Mass Operator

inputs: binary strings $A[1], ..., A[n]$
for all position i in the strings **do**
    $C(i) = 1/n \cdot \sum_{j=1,n} A[j](i)$
    if $C(i) = 0.5$ **then**
        $C(i) = RandomInteger(0, 1)$
    else
        $C(i) = Round(C(i))$
    end if
end for
return string C as offspring

formally specified to Hamming spaces on binary strings, we obtain the recombination operator outlined in Algorithm 4 [8]. This algorithm implements the inverse operation of the convex combination CX reported above in that it returns offspring C such that the parent B is in the Hamming segment between A and C and that $hd(A,C)/hd(B,C) = w_{bc}/w_{ab}$ in expectation. This differs from the Euclidean case where this ratio is guaranteed.

4.2 Center of Mass

When specified to the Hamming space on binary strings the center of mass CM coincides with the multi-parental recombination that returns the offspring by taking position-wise the majority vote of the parents and breaking ties randomly (see Algorithm 5). We prove this in the following.

Theorem 1. *The binary string p returned by the binary center of mass operator CM (Algorithm 5) applied to parents $p_1, ..., p_n$ minimizes the average Hamming distance to its parents.*

Proof. From the definition of center of mass operator, we have to prove that p minimizes $\frac{\sum_{i=1...n} hd(p_i,p)}{n}$. Since n is constant in p, this is equivalent to prove that p minimizes $\sum_{i=1...n} hd(p_i,p)$. By expanding the definition of Hamming distance and exchanging the summations, we have $\sum_{k=1..m} \sum_{i=1...n} hd(p_i^k, p^k)$ where k is the position in the string and m the number of bits in the string. The last expression is a linear combination of the (boolean) variables p^k. Minimizing

the linear combination is equivalent to minimizing each term $\sum_{i=1\ldots n} hd(p_i^k, p^k)$ in p^k separately. Minimizing a term corresponds to finding the center of mass p^k of the strings p_i^k of a single bit size. Since p^k can take only two values we have only two cases: (i) when $p^k = 0$ the term $\sum_{i=1\ldots n} hd(p_i^k, p^k)$ is the number of bits in $\{p_i^k\}$ set to 1; (ii) when $p^k = 1$ the term $\sum_{i=1\ldots n} hd(p_i^k, p^k)$ is the number of bits in $\{p_i^k\}$ set to 0. So, $\sum_{i=1\ldots n} hd(p_i^k, p^k)$ is minimized by $p^k = 0$ when the number of bits in $\{p_i^k\}$ set to 1 is less than the number of bits set to 0; it is minimized by $p^k = 1$ when the number of bits in $\{p_i^k\}$ set to 0 is less than the number of bits set to 1. This is a way of describing majority voting for strings of a single bit size. This reasoning applies to any position k in the string independently, so we conclude that the majority voting returns the p that minimizes $\frac{\sum_{i=1\ldots n} hd(p_i,p)}{n}$.

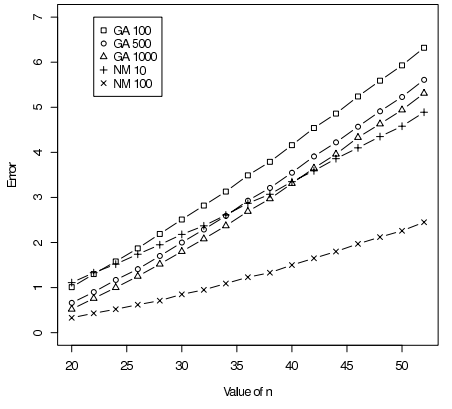
Unlike for the Euclidean case in which the simplex is maintained non-degenerate throughout the search process, so guaranteeing that any dimension is actually being searched, this does not hold true for the case of the Hamming space. To counteract the degeneracy of the simplex, in the experiments we will use a randomized version of the CM operator which uses the frequency of the ones and zeros at each position in the parents as the probability of generating a one or a zero in the offspring at that position. The expected offspring of the randomized operator is the one obtained with majority voting but the variance gives a greater chance to the search of staying open in all dimensions.

5 Experiments: Results and Discussion

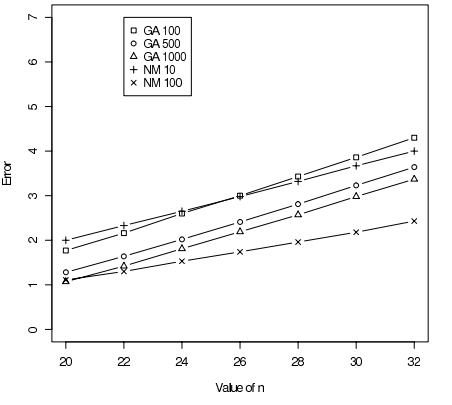
Experiments have been carried out using the well-known NK-Landscape problem [1], which provides a tunable set of rugged, epistatic landscapes over a space of binary strings. In our experiments we use landscapes of size $n = 20, \ldots, 52$ for $k = 2, \ldots, 6$ (higher values of n have not been provided for the higher values of k due to the difficulty of calculating the optimum using exact methods). These examples are due to Pelikan [11], and can be downloaded from Pelikan's website [10].

Two algorithms have been used. The GNMA above (referred to in the diagrams as NM) with populations 10 and 100, and a genetic algorithm (referred to in the diagrams as GA) with population sizes 100, 500 and 1000. Note that it is not fair to compare algorithms directly on identical population sizes as the meaning of population size in the two algorithms is very different. The GA uses uniform crossover with probability 0.8, bitflip mutation with a probability of $1/n$, elitism and roulette-wheel selection (i.e. we have chosen standard values from the literature). Both algorithms are run until they converge to a value; in the case of the GA this is taken to mean that the best value has the same value as in the previous 4 generations.

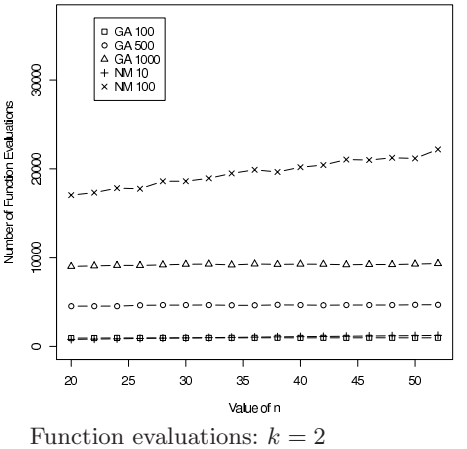
For each tuple of algorithm, population size n and k, the algorithm has been executed 10 times on each of 999 examples of NK-landscapes—therefore each point in the results graphs represents 9990 runs of that particular NK pair. Three features



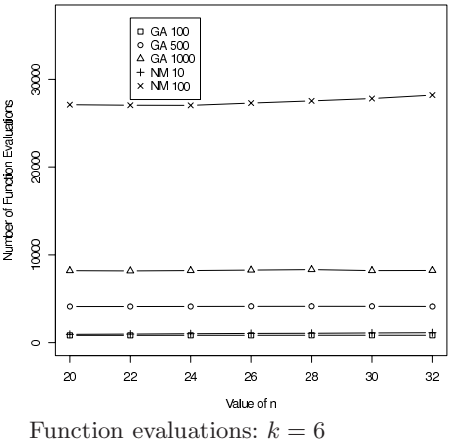
Error: $k = 2$



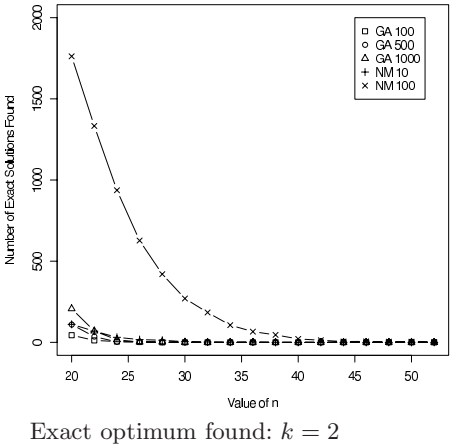
Error: $k = 6$



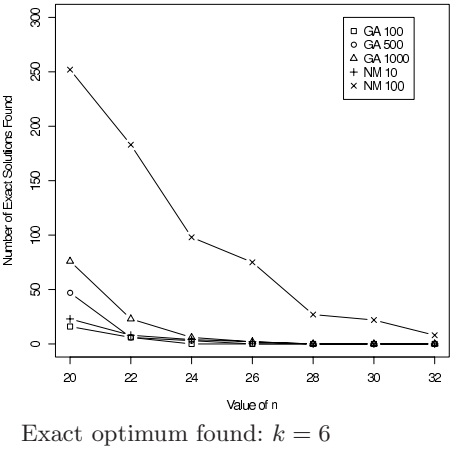
Function evaluations: $k = 2$



Function evaluations: $k = 6$



Exact optimum found: $k = 2$



Exact optimum found: $k = 6$

Fig. 2. Results for the five algorithms

have been measured: error on the objective function (measured against exactly-calculated optima computed using branch-and-bound [11]), number of function evaluations, and number of times the global optimum was found exactly.

Results are presented in Figure 2. The first set of results shows error measures. The error is presented as absolute error in arbitary units (i.e., *not* as a percentage or proportion of the optimum). The next pair of figures shows the number of function evaluations taken to convergence (of the population, not necessarily to convergence to optimum). The final set of results shows the number of runs (out of 9990) in which the exact optimum was found. Results for $k = 3, 4, 5$ are intermediate between $k = 2$ and $k = 6$, and are not reported due to space limitations.

For all problems, the NM-100 algorithm is the best (or equal-best) algorithm, in particular performing considerably better than the other algorithms on the problems with larger values for n, indicating good scaling performance. Clearly the NM-100 algorithm is using the most function evaluations of any of the algorithms; however, the current version of the GNMA continues until the whole population has converged on a single individual, so a better stopping condition might provide a better measure for this. Interestingly, the NM-100 algorithm achieves a considerably larger number of exact hits precisely on the global optimal value.

The experiments above, that have compared GNMA with a simple GA, are intended to show that the GNMA is a promising proof-of-concept rather than showing its competitiveness to state-of-the-art algorithms. As an important piece of future work, we will examine the performance of GNMA more thoroughly and compare it more broadly with a number of competitive algorithms including GPSO and GDE on a larger set of benchmark problems.

6 Conclusions

In this paper, we have generalized the Nelder-Mead Algorithm from continuous to generic combinatorial spaces by extending the geometric interpretation of the classic NMA to general metric spaces. The algorithm obtained (GNMA) can then be formally specified for specific spaces and specific representations. We have illustrated this by deriving the specific GNMA for the Hamming space associated with binary strings. We have tested the binary GNMA on a standard benchmark and compare it with a simple genetic algorithm.

In future work, we will specify the formal GNMA for search spaces associated with permutations and test it on hard combinatorial optimization problems. The Nelder-Mead Algorithm is similar to other classical derivation-free methods for continuous optimization that make use of geometric constructions of points to determine the next candidate solution (e.g., Controlled Random Search). We will use the same technique to generalize these algorithms to general metric spaces.

References

1. Kauffman, S.: Origins of order: self-organization and selection in evolution. Oxford University Press, Oxford (1993)
2. Kennedy, J., Eberhart, R.C.: Swarm Intelligence. Morgan Kaufmann, San Francisco (2001)
3. Luo, C., Yu, B.: Low dimensional simplex evolution: a hybrid heuristic for global optimization. In: Eighth International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, vol. 2, pp. 470–474 (2007)
4. Moraglio, A.: Towards a geometric unification of evolutionary algorithms. PhD thesis, University of Essex (2007)
5. Moraglio, A., Chio, C.D., Poli, R.: Geometric particle swarm optimization. In: Ebner, M., O'Neill, M., Ekárt, A., Vanneschi, L., Esparcia-Alcázar, A.I. (eds.) EuroGP 2007. LNCS, vol. 4445, pp. 125–136. Springer, Heidelberg (2007)
6. Moraglio, A., Chio, C.D., Togelius, J., Poli, R.: Geometric particle swarm optimization. Journal of Artificial Evolution and Applications (2008) Article ID 143624
7. Moraglio, A., Togelius, J.: Geometric pso for the sudoku puzzle. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 118–125 (2007)
8. Moraglio, A., Togelius, J.: Geometric differential evolution. In: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, pp. 1705–1712 (2009)
9. Nelder, J.A., Mead, R.A.: A simplex method for function minimization. Computer Journal 7, 308–313 (1965)
10. Pelikan, M.: NK landscape generator and instances provided online
11. Pelikan, M.: Analysis of estimation of distribution algorithms and genetic algorithms on nk-landscapes. In: Proceedings of the 2008 Genetic and Evolutionary Computation Conference (GECCO 2008), pp. 1033–1040. ACM Press, New York (2008)
12. Price, K.V., Storm, R.M., Lampinen, J.A.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Heidelberg (2005)
13. Takahama, T., Sakai, S.: Constrained optimization by applying the α-constrained method to the nonlinear simplex method with mutations. IEEE Transactions on Evolutionary Computation 9(5), 437–451 (2005)
14. Togelius, J., Nardi, R.D., Moraglio, A.: Geometric pso + gp = particle swarm programming. In: Proceedings of the Congress on Evolutionary Comptutation, CEC (2008)
15. Wang, F., Qiu, Y.: Empirical study of hybrid particle swarm optimizers with the simplex method operator. In: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications, pp. 308–313 (2005)
16. Weise, T.: Global Optimization Algorithms - Theory and Application. on-line ebook (2009)

Guided Ejection Search for the Pickup and Delivery Problem with Time Windows

Yuichi Nagata and Shigenobu Kobayashi

Interdisciplinary Graduate School of Science and Engineering,
Tokyo Institute of Technology,
4259 Nagatsuta Midori-ku Yokohama, Kanagawa 226-8502, Japan
nagata@fe.dis.titech.ac.jp, kobayasi@dis.titech.ac.jp

Abstract. This paper presents an efficient route minimization heuristic for the pickup and delivery problem with time windows (PDPTW) based on guided ejection search (GES). GES is a recently proposed metaheuristc framework and was first applied to the vehicle routing problem with time windows. The existence of the pickup and delivery constraint makes the feasible solution space tightly constrained and then makes the design of effective metaheuristics more difficult. We demonstrate that GES can be successfully applied to such an complicated problem. Experimental results on the Li and Lim's benchmarks demonstrate that the proposed GES algorithm outperforms existing algorithms and is able to improve 105 best-known solutions out of 298 instances.

Keywords: vehicle routing, pickup and delivery problem, ejection chain, guided local search, metaheuristics.

1 Introduction

The pickup and delivery problem with time windows (PDPTW) can be described as the problem of designing least cost routing plan to satisfy a set of transportation requests by a given identical vehicle fleet. Each request consists of delivering goods from a predefined location (pickup customer) to another one (delivery customer). The routing plan must be designed in such a way that all vehicles start and end at the depot, the amount of goods must not exceed the capacity of the vehicle (capacity constraint), each customer must be serviced within a given time interval (time window constraint), and for each request the corresponding pickup customer must be visited before the corresponding delivery customer by the same vehicle (pickup and delivery constraint). In standard benchmarks, a hierarchical objective of minimizing the number of routes (primary objective) and the total travel distance (secondary objective) is frequently used.

Given the hierarchical objective, recent metaheuristics for the PDPTW use a two-stage approach where the number of routes is minimized in the first stage and the total travel distance is then minimized in the second stage [3, 1, 11]. The two-stage approach allows us to independently develop algorithms for the route minimization and for the distance minimization. In this paper, we focus

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 202–213, 2010.

on the route minimization for the PDPTW because efficient algorithms for the route minimization and for the distance minimization would be different from each other.

In early work on the metaheuristics for the PDPTW, tabu search and simulated annealing algorithms were proposed for minimizing the number of routes [9] [1] where new neighborhood structures suited for the PDPTW were proposed. The current state-of-the-art algorithm by Ropke and Pisinger [11] is based on large neighborhood search (LNS) and it has shown good results for both the route and distance minimization. In the LNS algorithm [11], a move is defined by a fairly large change in a solution where up to 30–40 % of all requests are removed and re-inserted in a single move. For an extensive review of the metaheuristics as well as exact methods for the PDPTW, the reader is referred to Parragh *et al.* [10].

Recently, we proposed a powerful route minimization heuristic for the vehicle routing problem with time windows (VRPTW) [7]. This heuristic has dominated all of the previous route minimization heuristics applied to the VRPTW. Then we proposed a generalized metaheuristc framework based on this heuristic, which is called guided ejection search (GES), for solving a wide variety of combinational optimization problems and GES was successfully applied to the job shop scheduling problem [8]. In this paper we develop a route minimization heuristic for the PDPTW based on GES. The existence of the pickup and delivery constraint makes the feasible solution space tightly constrained and the design of an effective algorithm more difficult compared with the VRPTW case. Therefore, it is worthwhile to develop an effective GES algorithm for the PDPTW in order to investigate the effectiveness and robustness of GES to such an complicated problem.

The GES algorithm was tested on the standard benchmark problems of Li and Lim [4]. Computational results showed that the proposed algorithm is robust and highly competitive. It improved 105 best-known solutions out of 298 benchmark instances. The remainder of this paper is arranged as follows. First, the problem definition and notation are described in Section 2. The problem solving methodology based on GES is 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 Notation

The PDPTW is defined on a complete directed graph $G = (V, E)$ with a set of vertices $V = \{0, 1, \ldots, N\}$ and a set of edges $E = \{(v, w)|v, w \in V(v \neq w)\}$. Node 0 represents the depot and the set of nodes $\{1, \ldots, N\}$ represents the customers. Let $H = \{0, 1, \ldots, N/2\}$ be a set of the requests, and for each request $h \in H$, let p_h and d_h be the corresponding pickup and delivery customers, respectively. Let $P = \{p_h|h = 1, \ldots, N/2\}$ be a set of the pickup customers and $D = \{d_h|h = 1, \ldots, N/2\}$ a set of the delivery customers. Here, we assume that $P \cap D = \emptyset$ and $P \cup D = V \backslash \{0\}$. With each node v $(\in V)$ are associated a

demand q_v (with $q_0 = 0$), a non-negative service time s_v (with $s_0 = 0$), and a time window $[e_v, l_v]$. For each request h, the demand of the pickup customer, q_{p_h}, must be positive and the demand of the delivery customer is defined by $q_{d_h} = -q_{p_h}$. Each edge (v, w) has the non-negative travel distance d_{vw} and travel time c_{vw}. The capacity of the identical vehicles is given by Q.

Given a route r, let $< v_0, v_1, \ldots, v_n, v_{n+1} >$ be a sequence of the customers in this route where v_0 and v_{n+1} represent the depot and n refers to the number of customers in this route. The travel distance of the route is defined by $\sum_{i=0}^{n} d_{v_i v_{i+1}}$. A route is called feasible if the time window, capacity, and pickup and delivery constraints are all satisfied. These constraints are defined as follows. The earliest departure time at the depot, a_{v_0}, the earliest start time of service at a customer v_i, a_{v_i} $(i = 1, \ldots, n)$, and the earliest arrival time to the depot, $a_{v_{n+1}}$, are defined recursively as follows:

$$\begin{aligned} &a_{v_0} = e_0, \\ &a_{v_i} = \max\{a_{v_{i-1}} + s_{v_{i-1}} + c_{v_{i-1}v_i},\ e_{v_i}\}\ (i = 1, \ldots, n+1). \end{aligned} \tag{1}$$

The route satisfies the time window constraint if

$$a_{v_i} \leq l_{v_i}\ (i = 0, \ldots, n+1). \tag{2}$$

Let $Q_{v_i} (= \sum_{s=1}^{i} q_{v_s})$ $(i = 0, \ldots, n+1)$ denote the amount of goods in a vehicle at a location v_i. The route satisfies the capacity constraint if

$$Q_{v_i} \leq Q\ (i = 0, \ldots, n+1). \tag{3}$$

The route satisfies the pickup and delivery constraint if each pickup (delivery) customer in the route is visited before (after) the corresponding delivery (pickup) customer (the two customers must belong to the same route).

A feasible solution σ is defined as a set of feasible routes such that all customer are visited exactly once. In addition, we define a *partial* solution as a set of routes that pass through a subset of all customers exactly once. A partial solution is called feasible if it consists of feasible routes, and infeasible otherwise.

In standard benchmarks, the objective of the PDPTW consists of finding a feasible solution σ that minimizes the number of routes m (primary objective) and, in case of ties, minimizes the total distance traveled (secondary objective).

3 Problem Solving Methodology

This section first presents the GES algorithm for the PDPTW followed by the description of the difference from the previous work. Core parts of the GES algorithm are then described in more detail.

3.1 The GES Framework for the PDPTW

The GES algorithm (Procedure DELETE-ROUTE(σ)) is shown in Algorithm 1. The route minimization procedure starts with an initial solution where each

request (a pair of pickup and delivery customers) is served individually by a separate route. Procedure DELETE-ROUTE(σ) is then repeatedly applied to an incumbent solution σ to reduce the number of routes one by one until the termination condition is met.

Procedure DELETE-ROUTE(σ) is started by selecting and removing a route randomly from the current solution σ (line 1). Thus, σ is a feasible partial solution. An ejection pool (*EP*) [5] is then initialized with the set of the requests in the removed route (line 2). Here, the EP is a list that stores a set of temporarily unserved requests, currently missing from σ.

In each iteration (lines 5-14), a request h_{in} is selected from the EP with the last-in first-out (LIFO) strategy and is removed from the EP (line 5). The selected request h_{in} is then inserted into σ without violating the capacity, time window, and pickup and delivery constraints, if possible. More formally, the insertion of h_{in} is determined as follows. Let $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting $p_{h_{in}}$ and $d_{h_{in}}$ into σ in all possible ways. If there is a possible feasible insertion (*i.e.*, $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma) \neq \emptyset$), the insertion is executed by randomly selecting a solution from $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ and update the incumbent solution (lines 6-7).

If the request h_{in} cannot be inserted into σ, we remove (eject) requests from a route in σ in order to make room to insert h_{in} without violating the constraints. Let $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ be the set of feasible partial solutions that are obtained by inserting h_{in} into σ in all possible ways, and for each insertion, ejecting at most k_{max} requests from the resulting infeasible route in all possible ways. Note that we allow ejection of h_{in} itself and $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ always includes σ itself. The insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and the ejecting requests, denoted by $\{h^{(1)}_{out}, \ldots, h^{(k)}_{out}\}$ ($k \leq k_{max}$), are determined by selecting a solution from $\mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ so that the sum of penalty counters of the ejecting requests, $P_{sum} = p[h^{(1)}_{out}] + \ldots + p[h^{(k)}_{out}]$, is minimized (line 11). Here, the penalty counter $p[h]$ ($h \in \{1, \ldots, N/2\}$) refers to how many times an attempt to insert request h has failed during the current DELETE-ROUTE procedure (lines 3 and 10). This criterion is motivated to usually eject fewer requests at a time (*e.g.*, $k = 1$). In addition, this criterion is also motivated to avoid *cycling*. If a few requests conflict with each other due to the constraints, cycling will occur (*i.e.*, insertions and ejections of requests are repeated within a few conflicting requests) and the penalty counter of the conflicting requests will increase. Therefore, GES can escape from a cycling by ejecting a relatively large number of requests at a time that would not be so hard for the subsequent re-insertion when *cycling* occurs.

Each time after one or more requests are ejected from σ, the resulting feasible partial solution σ is perturbed by procedure PERTURB(σ) to diversify the search (line 13). Here, random local search moves are iterated inside the partial solution subject to the constraint that σ is a feasible partial solution. The detailed procedure is presented in Section 3.2.

The basic GES framework for the PDPTW is almost the same as that for the VRPTW. The main difference is that we employ the request as the fundamental component for the insertion and ejection whereas the customer was employed in

Algorithm 1. Procedure DELETE-ROUTE(σ)

1: Select and remove a route randomly from σ;
2: Initialize EP with the requests in the removed route;
3: Initialize all penalty counters $p[h] := 1$ $(h = 1, \ldots, N/2)$;
4: **while** $EP \neq \emptyset$ or termination condition is not met **do**
5: Select and remove request h_{in} from EP with the LIFO strategy;
6: **if** $\mathcal{N}^{fe}_{insert}(h_{in}, \sigma) \neq \emptyset$ **then**
7: Select $\sigma' \in \mathcal{N}^{fe}_{insert}(h_{in}, \sigma)$ randomly; Update $\sigma := \sigma'$;
8: **end if**
9: **if** h_{in} cannot be inserted in σ **then**
10: Set $p[h_{in}] := p[h_{in}] + 1$;
11: Select $\sigma' \in \mathcal{N}^{fe}_{EJ}(h_{in}, \sigma)$ such that $P_{sum} = p[h^{(1)}_{out}] + \ldots + p[h^{(k)}_{out}]$ is minimized; Update $\sigma := \sigma'$;
12: Add the ejected requests $\{h^{(1)}_{out}, \ldots, h^{(k)}_{out}\}$ to EP;
13: $\sigma :=$ PERTURB(σ);
14: **end if**
15: **end while**
16: **if** $EP \neq \emptyset$ **then** Restore σ to the input state;
17: **return** σ;

the previous work for the VRPTW. This choice is natural because the pickup and delivery constraint is imposed in the PDPTW. In the GES algorithm, the procedure for finding the best insertion-ejection combination that minimize P_{sum} (line 11) requires an efficient algorithm. As for the VRPTW, all possible insertion positions for a customer to be inserted, and for each insertion all possible ejections consisting of at most k_{max} customers were considered for finding the best inserting-ejection combination. Here, the parameter k_{max} was introduced in order to reduce the number of all possible insertion-ejection combinations. However, the number of all possible insertion-ejection combinations is usually very large (even if k_{max} is two) and it is therefore impractical to test all of them. So, we developed an efficient algorithm for finding the best insertion-ejection combination in the VRPTW case [7]. As described above, we employ the request as the fundamental component in the GES algorithm for the PDPTW. However, it makes more difficult to find the best insertion-ejection combination because the number of all possible insertion positions (for $p_{h_{in}}$ and $d_{h_{in}}$) is much greater than that in the VRPTW case. In Section 3.3, we present an efficient algorithm for finding the best insertion-ejection combination in the PDPTW case.

Another important feature of GES is the perturbation procedure (procedure PERTURB(σ)). As we will show in the experiments, this procedure significantly affects the performance. Ropke and Pisinger [11] suggested that very large moves (*e.g.*, up to 30–40 % of all requests are rearranged in a single iteration) are required to move a solution from one promising area of solution space to another, when faced with tightly constrained problems such as the PDPTW, even when embedded in metaheuristics. We believe that the perturbation procedure can efficiently move an incumbent solution σ to other promising area by a certain

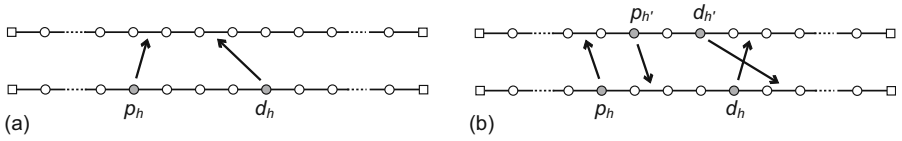


Fig. 1. The neighborhoods. (a) The pair relocation neighborhood is defined as a set of feasible solutions (feasible partial solutions in GES) that are obtained from the current solution by ejecting a request and re-inserting it in other possible ways (the request can be inserted into the same route). (b) The extended pair exchange neighborhood is defined as a set of feasible (partial) solutions that are obtained from the current solution by ejecting two requests from two different routes and inserting them into the two routes in all possible ways (each request must be inserted into another route).

number of iterations of small moves because an incumbent solution σ would not be so tightly constrained when requests are temporarily ejected. In this paper, we first employ the simple pair relocation neighborhood (See Fig. 1), which has been frequently used in previous metaheuristic algorithms for the PDPTW [9, 1]. However, we have confirmed that possible feasible moves by the pair relocation neighborhood is still limited and the incumbent solution σ cannot be fully moved even though some requests are temporarily ejected. So, we introduce an extended pair exchange neighborhood (See Fig. 1) to enhance the perturbation procedure. To the best of our knowledge, this neighborhood has not been used in the previous works.

3.2 The Perturbation Procedure

Let $\mathcal{N}^{fe}_{relocate}(h, \sigma)$ be the set of feasible partial solutions defined by the relocation moves of request h. Let $\mathcal{N}^{fe}_{exchange}(h, \sigma)$ be the set of feasible partial solutions defined by the extended exchange moves of request h (request h and other one are exchanged). In the perturbation procedure, the input solution σ is perturbed by the iteration of random moves for a given number of times, I_{rand}. At each iteration, a request h currently served in σ is randomly selected and a move is executed by randomly selecting a solution from $\mathcal{N}^{fe}_{relocate}(h, \sigma)$ or $\mathcal{N}^{fe}_{exchange}(h, \sigma)$, which are selected with a given probability. Here, the probability of selecting the extended pair exchange neighborhood is given by p_{ex}. Note that for a selected h if no feasible move is found in the selected neighborhood, no move is executed.

3.3 Algorithm for Finding the Best Insertion-Ejection Combination

Given a request h_{in} to be inserted, the procedure for finding the best insertion-ejection combination (insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and ejecting requests) that minimizes P_{sum} proceeds as follows. All insertion positions in σ are considered for $p_{h_{in}}$. For each insertion of $p_{h_{in}}$, a sub procedure is performed where all possible insertion positions for $d_{h_{in}}$ (inserting $d_{h_{in}}$ between $p_{h_{in}}$ and the end of the route (depot)) and ejections of at most k_{max} requests from the resulting infeasible routes are tested; check the feasibility of the resulting routes and record

the insertion positions for $p_{h_{in}}$ and $d_{h_{in}}$ and the ejected requests if a resulting route is feasible and P_{sum} is less than P_{best} (the minimum value of P_{sum} found so far).

We propose an efficient algorithm for the sub procedure. For each insertion of $p_{h_{in}}$, let $< 0, 1, \ldots, n, n+1 >$ be a sequence of the customers in the resulting infeasible route (we call it *original* route) where the customers are labeled according to the order in which they appear in the original route to simplify the notations (0 and $n+1$ refer to the depot). Although for any insertion-ejection combination, the feasibility of the resulting route can be checked in $O(n)$ time by a naive calculation, the feasibility check can be possible in constant time by testing insertion-ejection combinations in a particular order. Let us represent the possible ejections of requests as the corresponding customers. The combinations of all possible insertion positions for $d_{h_{in}}$ and ejections of at most k_{max} requests ($2k_{max}$ customers) can be denoted as $\{i(1), \ldots, i(j_d), \ldots, i(j)\}$ ($1 \leq i(1) < \ldots < i(j_d) < \ldots < i(j) \leq n$), $1 \leq j_d \leq j \leq 2k_{max}+1$, $1 \leq j_d \leq 2k_{max}+1$) (See Fig. 2). Here, the insertion position for $d_{h_{in}}$ is denoted as $i(j_d)$, meaning that $d_{h_{in}}$ is inserted just after $i(j_d)$ in the original route. The requests to be ejected are denoted as a set of the corresponding customers $\{i(1), \ldots, i(j_d-1), i(j_d+1) \ldots, i(j)\}$ where j refers to the number of temporarily ejected customers plus one. The basic idea is to search the possible insertion-ejection combinations denoted as $\{i(1), \ldots, i(j_d), \ldots, i(j)\}$ in the lexicographic order (we call it *lexicographic search*) (See Fig. 2), making it possible to check the feasibility of the resulting routes in $O(1)$ time for each route. Note that the lexicographic search is performed for each j_d ($1 \leq j_d \leq 2k_{max}+1$).

For each j_d ($1 \leq j_d \leq 2k_{max}+1$), the lexicographic search is performed as follows. Before starting the lexicographic search, we calculate the latest possible arrival times [6], z_i ($i = 1, \ldots, n+1$), for the original route (*i.e.*, if the vehicle arrives at customer i no later than time z_i, the time window constraints of i and the subsequent customers in the route are satisfied). Let a_i^{new} and Q_i^{new} denote, respectively, a_i and Q_i at location i in a route obtained through the lexicographic search. These values are dynamically updated. For example, when $i(j)$ is ejected (See Fig. 3 (a)), $a_{i(j)+1}^{new}$ can be updated in $O(1)$ time according to Eq. (1) because a_p^{new} (p is the predecessor of $i(j)$ in the current route) has already been updated. In the same way, when $i(j)$ is incremented (See Fig. 3 (b)), $a_{i(j)-1}^{new}$ can be updated in $O(1)$ time. In the same way, $a_{i(j_d)+1}^{new}$ (or $a_{i(j_d)}^{new}$) can be updated in $O(1)$ time when $d_{h_{in}}$ is inserted just after $i(j_d)$ (or $i(j_d)$ is incremented) (See Fig. 3). Q_i^{new} can also be updated in the same manner. When customer $i(j)$ is ejected, the resulting route is feasible in terms of the time window and capacity constraints if both $a_{i(j)+1}^{new} \leq z_{i(j)+1}$ and $Q_{i(j)+1}^{new} \leq Q$ hold, and the pickup and delivery constraint can be easily checked. Therefore, the feasibility of the resulting route can be checked in $O(1)$ time. Similarly, when $p_{h_{in}}$ is inserted just after $i(j_d)$, the feasibility of the resulting route can be checked in $O(1)$ time.

In fact, most of the lexicographic search can be pruned if one of the following conditions (i)–(v) is met and the efficiency is greatly improved.

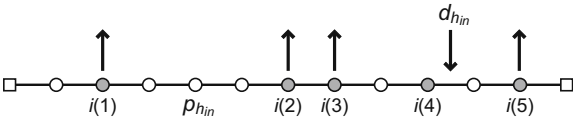


Fig. 2. An example of an insertion-ejection combination: $\{i(1), i(2), i(3), i(4), i(5)\} = \{2, 6, 7, 9, 11\}$ $(j_d = 4, j = 5)$. If $k_{max} = 2$, the lexicographic search proceeds in the following order: $\{1\}$, $\{1, 2\}$, $\{1, 2, 3\}$, $\{1, 2, 3, 4\}$, $\{1, 2, 3, 4, 5\}$, $\{1, 2, 3, 4, 6\}$ ..., $\{1, 2, 3, 4, n\}$, $\{1, 3\}$, $\{1, 3, 4\}$, $\{1, 3, 4, 5\}$, $\{1, 3, 4, 5, 6\}$, $\{1, 3, 4, 5, 7\}$, ..., $\{1, n-3, n-2, n-1, n\}$, $\{2\}$, $\{2, 3\}$, ...

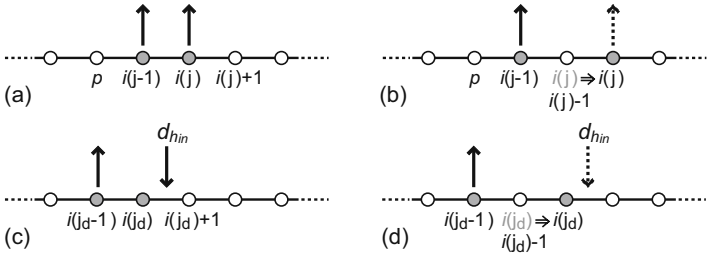


Fig. 3. Examples for the lexicographic search: (a) $i(j)$ is ejected ($i(j-1)$ is already ejected), (b) $i(j)$ is incremented ($i(j)$ is not yet ejected), (c) $d_{h_{in}}$ is inserted just after $i(j_d)$, (d) $i(j_d)$ is incremented ($d_{h_{in}}$ is not yet inserted).

- After $i(j)$ $(j \neq j_d)$ is ejected,
  - (i) P_{sum} (it is updated each time a pickup customer is ejected) $\geq P_{best}$
  - (ii) $i(j) \in P$ and the number of temporarily ejected pickup customers is greater than k_{max}
  - (iii) $i(j) \in D$ and the paired pickup customer is not temporarily ejected
- After $i(j)$ $(j \neq j_d)$ is incremented,
  - (iv) $l_{i(j)-1} < a^{new}_{i(j)-1}$ or $Q < Q^{new}_{i(j)-1}$
  - (v) $i(j) - 1 \in D$ and the paired pickup customer is temporarily ejected

In the cases (i)-(iii), the lower-level lexicographic search ($\{i(1), \ldots, i(j), *, *, \ldots\}$) can be pruned because (i) these insertion-ejection combinations never improve P_{best}, or (ii)(iii) the pickup and delivery constraint is never satisfied. In the cases (iv) and (v), the lower-level lexicographic search ($\{i(1), \ldots, i(j-1), v, *, *, \ldots\}$ $(v > i(j)-1)$) can be pruned because (iv) the time window or capacity constraint is violated already at customer $i(j)-1$, or (v) the pickup and delivery constrained is never satisfied. In addition, the lexicographic search can be further pruned in the similar manner when $d_{h_{in}}$ is inserted just after $i(j_d)$ or $i(j_d)$ is incremented (conditions are easily derived).

4 Experimental Results

The proposed GES algorithm was implemented in C++ and was executed on an AMD Opteron 2.6GHz (4GB memory) computer. Several computational

experiments have been conducted to analyze the behavior of the proposed GES algorithm. We also present results along with a comparison to the state-of-the-art metaheuristics for the PDPTW.

4.1 Benchmark Set

The GES algorithm was tested on the benchmark set constructed by Li and Lim [4]. The data set and the best-known solutions are available at `http://www.sintef.no/projectweb/top`. We used this benchmark set because recent state-of-the-art metaheuristics for the PDPTW are compared on this benchmark set. The benchmark set consists of five sets of 200, 400, 600, 800 and 1000 customers, with 60 instances in each set [1], resulting in 298 instances. For each size, the 60 instances are divided into six groups: R1, R2, RC1, RC2, C1, and C2, each consisting of 10 instances. The C1 and C2 classes have customers located in clusters and in the R1 and R2 classes the customers are at random positions. The RC1 and RC2 classes contain a mix of both random and clustered customers. The C2, R2 and RC2 classes (Class 2) have longer scheduling horizons and larger capacities than the C1, R1 and RC1 classes (Class 1), meaning that each vehicle can service a larger number of customers in the Class 2 instances.

4.2 Analysis of GES

First, we focus on the perturbation procedure in order to analyze the importance of this procedure where several parameter values of I_{rand} and p_{ex} are tested. Next, we investigate the effect of parameter k_{max} because this parameter would have a great impact on the behavior of GES. We test 12 different GES configurations, which are depicted in Table 1, where the following parameter values are combined: $k_{max} = 0, 1, 2, 3$, $I_{rand} = 0, 10, 100, 1000$, and $p_{ex} = 0, 0.5$. The GES algorithm with each configuration was applied to the benchmarks five times with the time limit of 600 seconds for each run.

Due to space limitation, Table 1 reports the results for only the 200 and 600-customer instances where the results are averaged over Class 1 and Class 2 instances (each consisting of 30 instances), respectively. For each problem class, the number of vehicles (m), the number of iterations to reach the best solution in each run ($iter$), and the computation time in seconds to reach the best solution in each run ($time$) are reported. Here, an iteration is defined as the sequence of procedures to insert a request h_{in} (lines 5-14) in Algorithm 1.

First, let us focus on the first setting ($k_{max} = 2$, $p_{ex} = 0$, $I_{rand} = 0, 10, 100, 1000$). We can see that m tends to improve (decrease) with increasing the value of I_{rand} from 0 to 100, but the improvement may be saturated around $I_{rand} = 10$ on 200-customer instances. Moreover, $iter$ also tends to decrease with increasing the value of I_{rand} from 0 to 100 even though $iter$, in general, tends to increase to reach higher quality solutions. However, the computational effort for the perturbation procedure is not negligible when I_{rand} is large. Therefore, the solution

[1] Two instances in the 1000-customer benchmark set are not available.

Table 1. The results of GES with the different configurations on 200 and 600-customer instances. The values of m calculated from the best-known solution are listed in the first column.

200-customer	$k_{max}=2$, $p_{ex}=0$				$k_{max}=2$, $p_{ex}=0.5$				$I_{rand}=100$, $p_{ex}=0.5$			
	I_{rand}	m	*iter*	*time*	I_{rand}	m	*iter*	*time*	k_{max}	m	*iter*	*time*
Class1 (15.60)	0	15.81	52418	20.0	0	15.77	34279	12.6	0	15.68	16763	58.3
	10	15.49	14610	6.3	10	15.47	5190	3.6	1	15.49	3357	12.7
	100	15.49	7721	7.1	100	15.47	1418	4.7	2	15.47	1418	4.7
	1000	15.50	1956	10.5	1000	15.47	840	22.6	3	15.46	1480	5.0
Class2 (4.60)	0	4.99	4954	9.0	0	4.97	8160	9.3	0	4.70	6449	45.6
	10	4.58	5865	12.9	10	4.57	5135	18.4	1	4.59	2029	17.2
	100	4.58	4142	14.7	100	4.57	2262	21.7	2	4.57	2262	21.7
	1000	4.57	1913	14.8	1000	4.61	751	36.5	3	4.58	2474	24.9
600-customer	$k_{max}=2$, $p_{ex}=0$				$k_{max}=2$, $p_{ex}=0.5$				$I_{rand}=100$, $p_{ex}=0.5$			
	I_{rand}	m	*iter*	*time*	I_{rand}	m	*iter*	*time*	k_{max}	m	*iter*	*time*
Class1 (43.1)	0	44.39	63750	86.0	0	44.36	61457	83.0	0	43.66	20530	200.8
	10	42.77	61603	106.7	10	42.52	38748	102.0	1	42.54	12463	121.1
	100	42.69	32858	118.7	100	42.45	12950	124.6	2	42.45	12950	124.6
	1000	42.90	8697	161.6	1000	42.76	3555	249.0	3	42.50	12251	110.8
Class2 (12.36)	0	14.43	16461	69.6	0	14.43	15928	62.5	0	12.42	6344	144.9
	10	12.45	13896	131.5	10	12.40	10741	115.1	1	12.33	5383	131.2
	100	12.38	7883	82.8	100	12.27	5026	122.2	2	12.27	5026	122.2
	1000	12.33	3826	101.1	1000	12.46	1898	230.0	3	12.29	4974	133.8

quality with $I_{rand}=1000$ sometimes inferior to those with $I_{rand}=10$ and 100 because of the termination condition given by the fixed time limit (600 seconds).

Next, we focus on the second setting ($k_{max}=2$, $p_{ex}=0.5$, $I_{rand}=0, 10, 100$, 1000). Here, the extended pair exchange neighborhood and pair relocation neighborhood are randomly selected in the perturbation procedure while only the pair relocation neighborhood is used in the first setting. The results show the same tendency as those in the first setting. But it should be noted that both of m and *iter* tend to be better than those in the first setting, indicating that the introduction of the extended pair exchange neighborhood enhances the ability of the perturbation procedure. However, *time* was not improved (compared with the first setting) even though *iter* was decreased because the computational effort for applying the extended pair exchange neighborhood is fairly greater than that for the pair relocation neighborhood. We can conclude from the results of the two settings that the more the incumbent solution is perturbed in the perturbation procedure, the better the solution quality is. If the extended pair exchange neighborhood can be applied in a more deliberate way, the performance of the proposed GES algorithm will be improved.

Finally, let us focus on the third setting ($I_{rand}=100$, $p_{ex}=0.5$, and $k_{max}=0, 1, 2, 3$). The solution quality (m) tends to improve with increasing the value of k_{max}, but the improvement is saturated at $k_{max}=2$ whereas $k_{max}=5$ seemed to be a good value in the VRPTW case.

4.3 Comparisons with Other Algorithms

We compare the GES algorithm with state-of-the-art route minimization heuristics for the PDPTW. According to the analysis conducted in the previous section,

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

	Best	BH (5 runs)		RP (10 runs)		GES (5 runs)			
N	known	Best	Time	Best	Time	Best	Average	Time	#new
200	606	614	300	606	(264)	601	601.2	600	5
400	1157	1188	600	1158	(881)	1139	1140.0	600	18
600	1664	1718	600	1679	(2221)	1636	1641.8	600	27
800	2175	2245	900	2208	(3918)	2135	2147.4	600	30
1000	2644	2759	900	2652	(5370)	2613	2624.8	600	25
total	8266	8524		8303		8124	8155.2		105
Computer		Athlon 1.2GHz		Penti.IV 1.5GHz		Opteron 2.6GHz			

the parameters of the GES algorithm were set as follows: $k_{max} = 2$, $p_{ex} = 0.5$, and $I_{rand} = 100$. The GES algorithm was applied five times to all of the benchmark instances where the time limit was set to 600 seconds for each run. We selected two algorithms for comparisons that have shown the best results from the literature: BH (Bent and Hentenryck [1] [2]) and RP (Ropke and Pisinger [11]). BH and RP heuristics are both based on the two-stage approach where BH and RP heuristics, respectively, use simulated annealing and adaptive large neighborhood search for the route minimization phase. BH and RP heuristics were executed five and ten times, respectively, on the Li and Lim's benchmarks. We compare the results in terms of the number of routes.

Table 2 shows the results for each problem size. The solution quality is evaluated by the cumulative number of vehicles where the columns "Best" and "Average" show the best and average results over the number of runs. The column "Best known" shows the cumulative number of vehicles calculated from the best-known solutions. The column "Time" shows the average computation time in seconds for a run spent on each instance. Here, the computation time for the RP heuristic presented in the table includes both route- and distance- minimization phases. The column "#new" shows the number of new -best solutions obtained by the five runs of the GES algorithm. At the bottom of the table, specifications on the computers used in the experiments are shown.

As can be seen from the table, the GES algorithm outperforms the compared heuristics in terms of the cumulative number of vehicles in all problem sizes although the GES algorithm may be assigned longer computation time than the compared heuristics. However, these results are very good because our result are clearly better than the best-known solutions and 105 new best-known solutions were found by the five runs of the GES algorithm.

5 Conclusions

In the paper we extended a GES algorithm, which was first developed for the VRPTW, to the PDPTW. We showed that the perturbation procedure has a great impact on both the solution quality and computation time, and developed an effective GES algorithm for the PDPTW by giving a careful attention to

the perturbation procedure. We believe that this feature comes from the tightly constrained solution space of the PDPTW because the GES algorithm applied to the VRPTW was not so sensitive to the structure of the perturbation procedure. In addition, we presented an efficient algorithm for finding the best insertion-ejection combination from numerous candidates in terms of the given criterion. In the future work, a more innovative criterion for selecting a insertion-ejection move would also be possible and the proposed efficient algorithm will also be available.

References

1. Bent, R., Hentenryck, P.V.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. Computers and Operations Research 33, 875–893 (2006)
2. Bent, R., Hentenryck, P.V.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows, Appendix, http://www.cs.brown.edu/people/rbent/pickup-appendix.ps
3. Lau, H., Liang, Z.: Pickup and Delivery with Time Windows: Algorithms and Test Case Generation. In: Proc. of the 13th IEEE International Conference on Tools with Artificial Intelligence, pp. 333–340 (2001)
4. Li, H., Lim, A.: A Metaheuristic for the Pickup and Delivery Problem with Time Windows. In: Proc. of the 13th IEEE International Conference on Tools with Artificial Intelligence, pp. 160–170 (2001)
5. 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, 443–457 (2007)
6. Kindervater, G.A.P., Savelsbergh, M.W.P.: Vehicle routing: handling edge exchanges. In: Local Search in Combinatorial Optimization, pp. 337–360. Wiley, Chichester (1997)
7. Nagata, Y., Bräysy, O.: A Powerful Route Minimization Heuristic for the Vehicle Routing Problem with Time Windows. Operations Research Letters 37, 333–338 (2009)
8. Nagata, Y., Tojo, S.: Guided Ejection Search for the Job Shop Scheduling Problem. In: Cotta, C., Cowling, P. (eds.) EvoCOP 2009. LNCS, vol. 5482, pp. 168–179. Springer, Heidelberg (2009)
9. Nanry, W.P., Barnes, J.W.: Solving the pickup and delivery problem with time windows using reactive tabu search. Transportation research. Part B 34, 107–121 (2000)
10. Parragh, S.N., Doerner, K.F., Hartl, R.F.: A survey on pickup and delivery problems, Part II: Transportation between pickup and delivery locations. Journal für Betriebswirtschaft 58, 81–117 (2008)
11. Ropke, S., Pisinger, D.: An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. Transportation Science 40, 455–472 (2006)

An Evolutionary Algorithm Guided by Preferences Elicited According to the ELECTRE TRI Method Principles

Eunice Oliveira[1] and Carlos Henggeler Antunes[2]

[1] School of Technology and Management, Polytechnic Institute of Leiria, Morro do Lena, Ap. 4163, 3411-901 Leiria, Portugal and R&D Unit INESC Coimbra
[2] Dept. of Electrical Engineering and Computers, University of Coimbra, Polo II, 3030 Coimbra, Portugal and R&D Unit INESC Coimbra
eunice@ipleiria.pt, ch@deec.uc.pt

Abstract. The resolution of a multi-objective optimization problem involves not just a search and computation phase, capable of providing a representative sample of the Pareto-optimal front, but also a decision support process to aid the Decision Maker (DM) to progress in the learning of the trade-offs at stake in different regions of the search space. This is accomplished by integrating in the search process the DM's preferences to guide the search and limit both the cognitive effort, in assessing Pareto-optimal solutions with distinct characteristics, and the computational effort, by reducing the scope of the search according to the preferences expressed by the DM. The introduction of meaningful preference expression parameters used in the ELECTRE TRI method for sorting problems in the framework of an evolutionary algorithm is proposed. Illustrative results in an operational planning problem in electricity networks are reported.

Keywords: Evolutionary algorithm, ELECTRE TRI, handling preferences, adaptive algorithms.

1 Introduction

Most real-world optimization problems need to cope with multiple aspects of evaluation of the merit of alternative solutions. Even though models generally recognize the multiple concerns at stake, these are sometimes "amalgamated" in a single aggregate objective function or even considered as constraints by specifying some lower/upper bounds. However, mathematical models become more adequate vis-à-vis the real-world problem to be tackled if multiple, generally conflicting and incommensurate, objective functions are explicitly considered. Therefore, a single optimal solution no longer exists but a set of Pareto-optimal (non-dominated) solutions must be computed. These solutions are characterized by the fact that an improvement in any objective function can only be obtained with the degradation of at least another objective, therefore representing a compromise between the different axes of evaluation made operational through the objective functions.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 214–225, 2010.

In the last years, multi-objective Evolutionary Algorithms (EAs) became a prevalent tool to characterize the Pareto-optimal front to multi-objective optimization problems, especially those non-linear and/or of combinatorial nature. EAs' ability to deal with a set (a population) of possible solutions in a single simulation run makes them quite suited to multi-objective optimization problems [1], [2]. Additionally, EAs are less susceptible to the shape or continuity of the Pareto-optimal front than classical (mathematical programming) optimization methods.

In most EAs, the design of genetic operators (selection, crossover and mutation) and the associate parameters (e.g., crossover and mutation probability) used in the evolutionary process are predefined. However, the efficiency of an operator may change during the search process (that is, an operator may offer better results in the beginning of the process and others in the final) or a given operator can be more useful for a solution than for another depending on the evaluation in each objective function. Thus the performance of EAs can be improved using an adaptive approach to the actual performance of operators, possibly changing the values of some parameters and using different operators during the evolutionary process.

Beyond the search and computation process, the resolution of a multi-objective optimization problem involves also a decision aiding process. In fact presenting the DM with a (generally huge) set of Pareto-optimal solutions after the end of a whole run does not provide great assistance in improving his/her decision making capabilities having in mind the choice of a final solution for practical implementation or even the selection of a reduced sub-set of solutions for further screening. That is, in most cases a significant computational effort is incurred, which in turns imposes also a considerable cognitive effort on the DM to discriminate among Pareto-optimal solutions (which cannot be distinguished based on the non-dominance relation only) due to the conflicting nature of the objective functions. Therefore, it is necessary for actual decision support purposes to empower EAs with techniques aimed at capturing and incorporating the DM's preferences into the evolutionary process.

The incorporation of the DM's preferences may be done before (a priori), during (progressive) or after (a posteriori) the optimization process [3], [4]. A priori approaches usually transform the multi-objective optimization problem into a single objective problem by aggregating the multiple objective functions into a scalar function, which includes some information about the DM's preferences (e.g., utility/value function based approaches). In the second case, the search is guided using the DM's preferences, which can be changed during the computation process according to the learning (about the own preferences and the problem characteristics) taking place by analyzing the solutions computed. In the a posteriori approach the Pareto-optimal front is initially determined and then some method is applied to deal with the solutions (in a choice, ranking or sorting problem perspective). Some research dealing with preference handling in evolutionary multi-objective optimization is reported in [5].

The main idea underlying our proposal is to combine the advantages of the adaptive EAs and the incorporation of the DM's preferences in the search process. The introduction of meaningful preference expression parameters used in the ELECTRE TRI method for sorting problems is proposed to be included in the operational framework of an EA. These preferences are represented through technical

devices such as weights, as well as indifference, preference and veto thresholds. The ELECTRE TRI principles were embedded into an EA to guide the search to more convenient directions according to the preference information expressed. At the end of the process that method is used to classify the solutions obtained by the EA into ordered categories of merit. Illustrative results in an operational planning problem in electricity networks are presented.

The paper is organized as follows. The next section gives a brief presentation of the ELECTRE TRI method and in section 3 it is described how the ELECTRE TRI principles are embedded into the operational framework of the EA. Section 4 presents an actual problem of reactive power compensation in electrical distribution networks. The application of the methodological approach herein proposed to provide decision support in that problem is presented in section 5. Some illustrative results are briefly analyzed in section 6. Finally, some conclusions are drawn and some ideas to future work are outlined.

2 The ELECTRE TRI Method

The ELECTRE ("ELimination Et Choix Traduisant la REalité") family of multi-criteria methods developed by Roy and his co-workers [6] is based on the construction and the exploitation of an outranking relation in face of the problem to be tackled (choice, ranking or sorting). The term outranking in this context means "is at least as good as" as a synonym for "not worse than" [7]. The choice problem refers to identify the best alternative or a limited set of the best alternatives (since incomparability is allowed). The ranking problem deals with constructing a rank ordering of the alternatives from the best to the worst ones. The sorting problem consists in partitioning the alternatives being evaluated into a pre-defined set of ordered categories. The major difference between these formulations concerns the judgment of the alternatives [8], [9]. The choice and ranking approaches are based on relative judgments and consequently the evaluation depends entirely on the set of alternatives considered. The sorting approach considers an absolute judgment, in the sense that the pair-wise comparisons are made between the alternative to sort and a set of reference profiles.

The ELECTRE TRI method is devoted to the sorting problem, i.e., each alternative in $A=\{a_1,a_2,\ldots,a_m\}$ must be assigned into pre-defined categories (classes) according to its evaluation by a set of k quantitative and/or qualitative criteria $g_1,g_2,\ldots,g_k$. Each category, C^1, C^2, …, C^n, $n \geq 2$, is defined using two reference alternatives (profiles), b_i and b_{i+1}, $i = 0,\ldots,n\text{-}1$, which limit the categories (Fig. 1). We assume, without any loss of generality, that C^n is the best category and C^1 the worst category. The assignment of each alternative a_j, $j=1,\ldots,m$, to a category is done by comparing its performances in each criterion with the reference profiles. The outranking relation between an alternative and the reference profiles is decided by comparing a credibility index, involving the differences in performances and the criterion weights, with a cutting-level λ ($\lambda \in [0.5,1]$) which defines the "majority requirement".

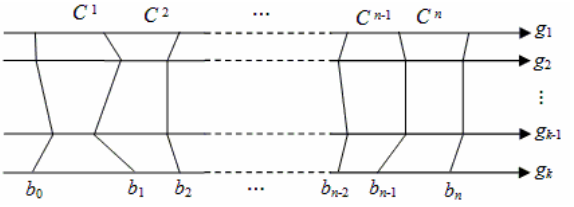


Fig. 1. Definition of categories through reference profiles

The ELECTRE TRI method requires some parameters that bear the DM's preferences, such as preference (p_l), indifference (q_l) and veto (v_l) thresholds, l=1,…,k, a set of weights (assigned to the criteria) and the cutting-level λ. Indifference and preference thresholds characterize the acceptance of imprecision in the judgment. When the difference between the individual performances of two alternatives in a given criterion g_l is less than a specified amount q_l, the alternatives are deemed indifferent concerning this criterion g_l. Moreover, the transition from indifference to preference is gradual, changing linearly from q_l to p_l. The veto thresholds prevent an alternative having a good performance in one or more criteria but having a very bad performance in another criterion to be assigned to the best category, or they force this alternative to be assigned to a low preference category independently of having very good performance in all other criteria, thus allowing for the introduction of some non-compensatory aspects in the decision. The weights in ELECTRE TRI are scale-independent; that is, they are not linked to the scales in which each criterion is measured. In this framework the weights are not technical devices for translating the performances in the criteria into a common value measure (as in weighted-sum approaches) but weights play the role of true importance coefficients (the voting power) of each criterion. The cutting-level λ is a real value in the interval [0.5,1] that indicates the level of exigency (the majority requirement of criterion "coalitions") to enforce the assignment of a given alternative to a category.

The outranking relation (aSb) between an alternative a and a reference profile b is established in the following stages:

1. Criterion concordance indexes $c_j(a,b)$ evaluation;
2. Global concordance indexes $C(a,b)$ evaluation;
3. Discordance indexes $d_j(a,b)$ evaluation;
4. Credibility degree σ (a,b) evaluation;
5. Establishing the outranking relation.

The last step in ELECTRE TRI consists in the assignment of each alternative into one of the pre-defined ordered categories. This can be done using a pessimistic or an optimistic procedure. In the pessimistic procedure (the procedure used in our work) each action a_j, j=1,…,m, is assigned to the highest category for which a_j outranks the category's lower bound.

For further details about ELECTRE TRI see [8].

These characteristics led us to use the principles of ELECTRE TRI to be combined with an EA in order to enrich the non-dominance relation through meaningful preference information. Therefore, the evolutionary process is guided using this

preference information and solutions (alternatives) are sorted into ordered categories. This is a more realistic process than attempting to elicit at the outset a disputable "true" analytical form of the DM's preferences to be then optimized in a rigid way. However, the use of ELECTRE TRI principles within an evolutionary approach imposes the specification of the additional parameters required. This can be indeed an extra burden, but this process can be easily automated in a meaningful manner. For example, the thresholds can be predetermined as percentages of the values ranges in each category (e.g., 2% and 10% for the indifference and preference thresholds, respectively).

3 ELECTRE TRI within an Evolutionary Algorithm

The approach presented in this section has been designed to combine the advantages of adaptive EAs with the incorporation of the DM's preferences in the evolutionary process. With this aim ELECTRE TRI principles are embedded in an EA to provide information about the performance of each individual (alternative solution) according to the multiple objective functions (the criteria). Using this information the EA can perform a more adequate choice of the mutation operator from a portfolio of different mutation strategies specifically designed for the problem at hand. Consequently the search can be guided to more convenient directions, favoring the best categories.

The sequence of the operators (selection, crossover and mutation) is as usual but the use of the mutation operator is guided. The criterion discordance indexes of ELECTRE TRI are evaluated (according to expressions (1)) before applying one of the mutation operators to an individual.

$$\begin{aligned} d_j(a_i,b_k) = 1 \quad &\Leftrightarrow \quad v_l < g_l(b_k) - g_l(a_i) \\ 0 < d_j(a_i,b_k) < 1 \quad &\Leftrightarrow \quad p_l < g_l(b_k) - g_l(a_i) \le v_l \\ d_j(a_i,b_k) = 0 \quad &\Leftrightarrow \quad g_l(b_k) - g_l(a_i) \le p_l \end{aligned} \tag{1}$$

These values provide information about how far an objective function (criterion) is from the respective reference profile, and consequently it is possible to select a more adequate mutation operator attempting to improve the objective function with the worst performance. That is, mutation operators that are likely to improve this objective function have a greater probability of being chosen (although the process remains essentially a random one).

It is important to mention that the criterion discordance indexes are based on a comparison with the pre-defined reference profiles and use the preference and the veto thresholds defined by the DM. Therefore, the search in the evolutionary process is indeed guided by the preference information established by the DM and made operational by means of the ELECTRE TRI parameters.

At the end of the evolutionary process, the whole ELECTRE TRI method is applied to assign the Pareto-optimal solutions to one of the pre-defined categories. If the DM would like a different or a more detailed sorting of the solutions, it is possible to re-apply the procedure using new parameters of the ELECTRE TRI method (or even using another member of the ELECTRE family devoted to the ranking or the choice problems).

A concern with this approach is the need to balance favorable moves that, in principle, generate solutions to be assigned to the best categories without losing the diversity of the Pareto-optimal front, i.e. solutions are assigned to all or the wider range possible of categories.

One advantage of using just the criterion discordance indexes (and not the whole ELECTRE TRI method) in each iteration (generation) to guide the evolutionary search process is to reduce the running time. Other important aspect is that with this approach some of the parameters of the ELECTRE TRI method (such as the objective function weights, the indifference threshold and the cutting-level λ) can be defined at the end of the evolutionary process when more knowledge has been gathered by the DM about the achievable range of the objective functions as well as the trade-offs involved.

A possible drawback of this approach is that it is necessary to evaluate the performance of each offspring before determining the criterion discordance indexes. However, this aspect is in part overcome by the fact that the convergence to the Pareto-optimal front is faster when using the guided search as described above.

4 A Reactive Power Compensation Problem

The compensation of reactive power is an important issue in electric power systems. This is generally achieved by the allocation of shunt capacitors (sources of reactive power) in adequate nodes of the electrical distribution network. The compensation of reactive power ensures a more efficient delivery of active power (which is converted into "useful" energy, such as light or heat, …) to loads, thus releasing system capacity, reducing system losses, and improving system power factor and bus voltage profile. The merit of solutions (determining the size of the capacitors and the network nodes in which they are located) must be assessed using operational, economical and quality of service aspects. Therefore, multiple objective programming models have been used to provide decision support in this problem considering different sets of, generally incommensurate and conflicting, objective functions to be optimized [10]. In our model we consider the minimization of resistive losses, capacitor installation cost and maximum deviation to the nominal voltage at each network node. A similar mathematical model with two objective functions can be seen in [11].

In this work a Portuguese radial electrical distribution network with 94 nodes is considered. This network has some challenging features due to its extension in a rural region and a poor voltage profile. The voltage profile without compensation does not respect the quality requirements (the node's voltage magnitude must be within ±10% of the nominal value).

The study is done for peak load conditions, in which the active power losses are 320.44 kW and the number of nodes not respecting the voltage lower bounds is 66 (in 94 nodes). That is, the network is not working according to regulations in peak load conditions and therefore capacitors must be installed for reactive power compensation and voltage profile improving purposes. This means that the zero cost solution is not feasible.

The capacitors to be installed are characterized by their capacity and the acquisition cost (Table 1).

Table 1. Capacitor dimension and acquisition cost

	Maximum Capacity (KVAr)	Cost (Euro)
C1	50	2035
C2	100	2903
C3	140	4545
C4	200	4875
C5	240	5716
C6	300	6578
C7	360	7337
C8	400	9395

5 Resolution of the Reactive Power Compensation Problem

The methodology described in section 3 has been applied to the network presented in the previous section to find a set of non-dominated solutions to this problem of reactive power compensation and to assign these solutions to predefined categories according to the DM's preference information.

A solution is encoded as an array with 94 positions, each of them corresponding to a network node (Fig. 2). A "0" means that no capacitor is installed in the corresponding node and a non-zero value (belonging to the interval [1, 8], see table 1) refers to the type of the capacitors therein installed.

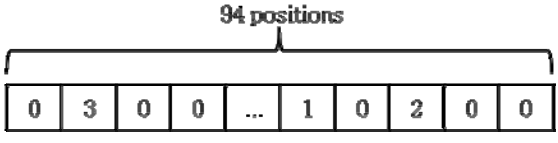


Fig. 2. Solution encoding

Initially a random population is generated retaining just the feasible individuals; that is, the solutions that satisfy the voltage requirements at the nodes.

At each iteration parents are selected using a tournament technique. Other selection techniques, such as roulette wheel, were tested, but with worse overall results. A 2-point crossover operator is applied and the offspring generated are evaluated. This evaluation leads to the criterion discordance indexes, using the preference information on reference profiles as well as the preference and veto thresholds. The objective function with the worst performance, when compared with the reference profiles, determines then the choice of the mutation operator to be applied from within a portfolio tailored for the actual characteristics of the problem at hand. Mutation operators that are likely improve the objective function with worst performance have larger probability to be chosen than others.

Six types of mutation operator have been implemented specifically for this problem (in the sense that a physical meaning in the electrical network can be

associated with each of them, thus also conveying information about the actual problem):

1. Reducing the capacity of a capacitor previously installed in a given node to the immediately lower size (Fig. 3 (a));
2. Increasing the capacity of capacitor previously installed in a given node to the immediately upper size (Fig. 3 (b));
3. Removing the capacitor previously installed in a given node (Fig. 3 (c));
4. Installing a new capacitor in an uncompensated node (Fig. 3 (d));
5. Relocating the capacitor previously installed in a given node to an adjacent node (Fig. 3 (e) and (f)).

Some of these moves clearly determine the sense of variation of some objective functions; for instance, by removing a capacitor the cost objective function improves. However, it is important to emphasize that when the size of a capacitor is increased or the number of capacitors in the network is increased the cost objective function will be degraded, but it is not possible to guarantee that the losses objective function improve (because this depends on several characteristics of the network and on the capacitors already installed).

After the mutation process takes place only the feasible solutions compete with their progenitors to pass to the next generation. All these solutions (parents and feasible offspring population) are sorted into non-dominance ranking levels. The *n* solutions belonging to the highest level ranks pass to the next generation, where *n* is the dimension of the population.

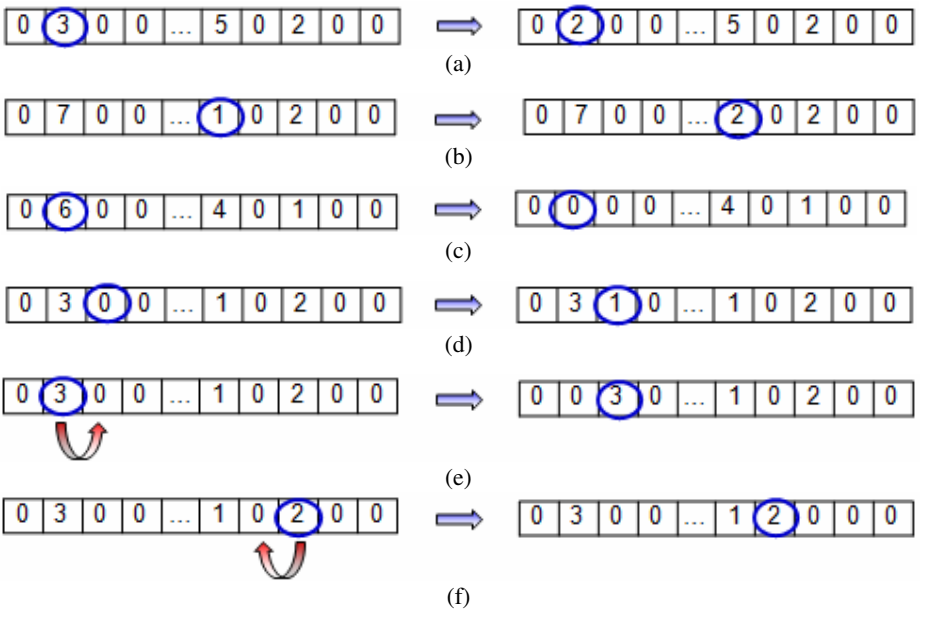


Fig. 3. Examples of the mutation operators

This process is repeated until the stop condition (for instance, a maximum number of iterations) is satisfied. The whole ELECTRE TRI method is then applied to the Pareto-optimal front obtained to assign each solution to a category.

6 Analysis of the Results

The growth of the number of the objective functions implies that the number of solutions in Pareto-optimal front also increases, leading a more difficult task to identify a solution that may be recommended for practical implementation. The use of ELECTRE TRI principles guiding the search and sorting the Pareto-optimal solutions (at the end of the search process) is aimed at easing the DM's tasks since his/her preferences have been taken into account throughout the evolutionary process. The classification of the solutions into ordered classes offer the DM the possibility of analyzing just a (desirably) small sub-set of "most preferred" solutions, in particular, the ones within the best performance category.

Fig. 4 presents an example of a Pareto-optimal front obtained using this approach. 500 iterations were performed and the front has 400 individuals, 10 of them belonging to class 5 (the class with best performance according to the profiles defined by the DM).

The ELECTRE TRI parameters considered are presented in Tables 2 and 3.

Table 2. Reference profiles

Resistive losses	Cost	Maximum deviation
250	40 000	0.02
280	60 000	0.04
300	100 000	0.06
350	130 000	0.08

Table 3. Preference, indifference and veto thresholds

	Resistive losses	Cost	Maximum deviation
Indifference threshold	10	10 000	0.01
Preference threshold	30	30 000	0.03
Veto threshold	85	85 000	0.07

In this example, all the objective functions had the same "importance" (equal weights) and the level of exigency was the minimum possible - a simple majority of λ=0.5.

As emphasized above the weights in ELECTRE TRI are true importance coefficients associated with the objective functions. This is an important issue for the classification of the solutions. If maximum voltage deviation had been considered the most important objective function, the weights should reflect this and the solutions in

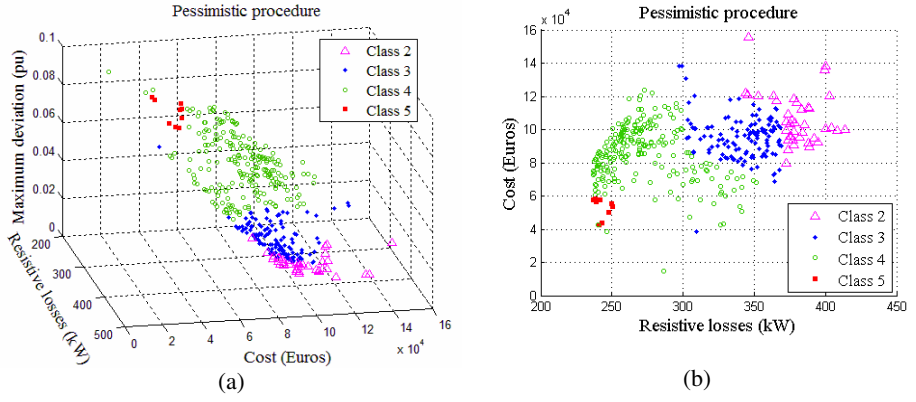


Fig. 4. Pareto-optimal front after the final sorting using ELECTRE TRI. (a) A 3D perspective. (b) A 2D projection.

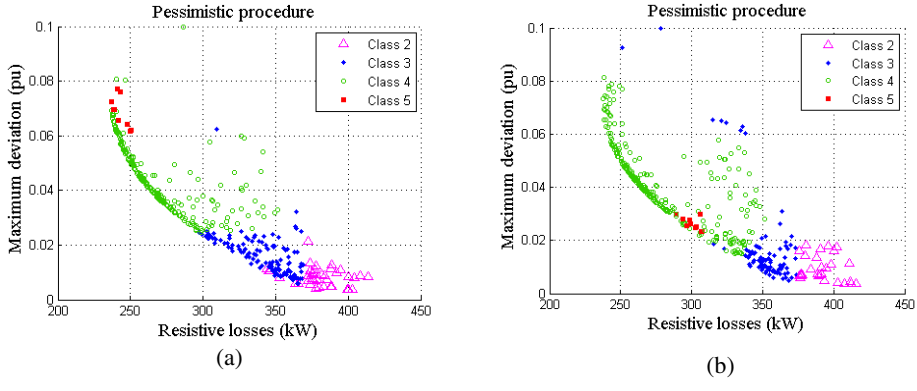
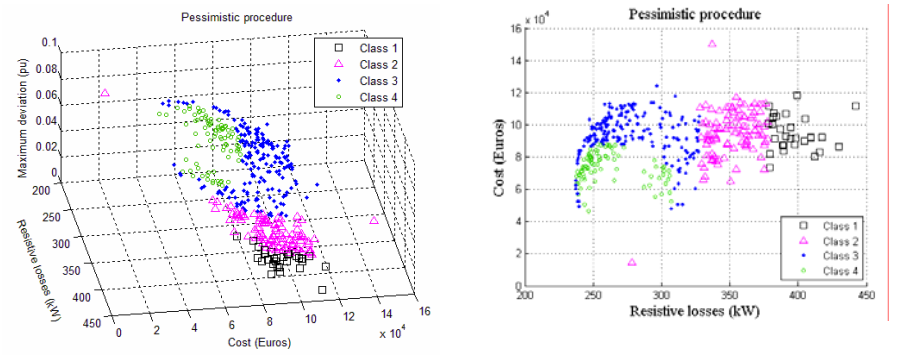


Fig. 5. Pareto-optimal front after the classification with ELECTRE TRI: (a) Equal weights for all objective functions. (b) Resistive losses and cost objective function weights = 25% and maximum deviation objective function weight = 50%.

the Pareto-optimal front would have been classified in a different manner. In Fig.5 it is possible to observe that solutions classified into the best category have a smaller maximum voltage deviation when the weight of this function is increased.

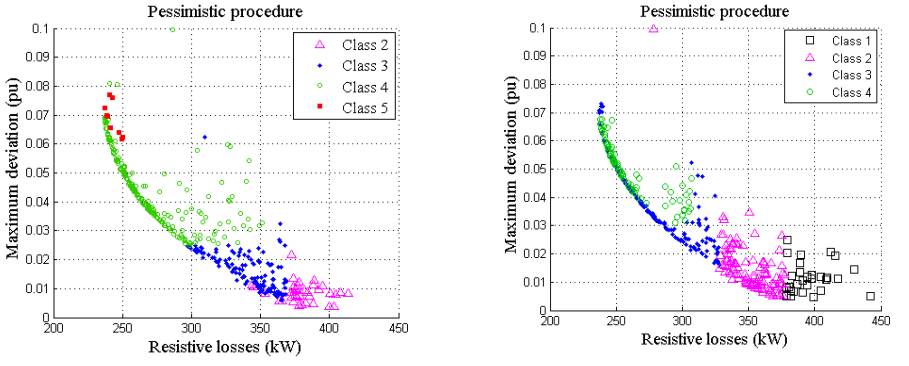
The sorting can also reflect the level of exigency. If the cutting-level λ is increased from 0.5 to 0.7, it is expected that fewer solutions belong to the best performance class due to the increase of the exigency level (see Fig. 4 and 6).

Another aspect of the ELECTRE TRI method that could modify the evolutionary process it is the veto threshold. The use of this parameter influences the selection of the mutation operator as well as the final sorting of the solutions. In this last stage, the veto threshold prevents that solutions with a good performance in one or more objective functions but with a very bad evaluation in other objectives do not be assigned to the best class. Fig. 7 presents an example of two simulations, in which the parameters are the same with the exception of the veto threshold associated with the



(a) (b)

Fig. 6. Pareto-optimal front after the sorting with ELECTRE TRI with cutting-level λ=0.7. (a) A 3D perspective. (b) A 2D projection.



(a) (b)

Fig. 7. Pareto-optimal front after the sorting with ELECTRE TRI. (a) Veto threshold equal to 0.07. (b) Veto threshold equal to 0.04.

minimization of maximum voltage deviation. The introduction of the veto threshold prevents solutions being assigned to the best category, Fig. 7(b), because solutions classified in category 5 in Fig. 7(a) have bad performance in the maximum voltage deviation objective with respect to the veto threshold value defined in Fig. 7(b).

7 Conclusions and Future Work

In this work an adaptive EA has been proposed and its implementation has been described, in which the principles of the ELECTRE TRI method are embedded into the evolutionary process to guide the search to more convenient regions of the space according to the preferences expressed by the DM.

At the end of the evolutionary process, the Pareto-optimal front obtained is classified into ordered categories of merit, which allow the DM to restrict the choice of practical solutions to the problem to a smaller set of solutions.

Illustrative examples have been presented using a real-world reactive power compensation problem, which is a relevant issue in electrical distribution networks.

Work is underway to include other information about DM's preferences into the adaptive process: making the probability of each mutation operator to be influenced by the objective function (criterion) weights, establishing a dependence between the solution classification and the selection of the individuals that pass to the next generation, reinforcing the role of the veto threshold to influence the evolutionary process.

Acknowledgments. This research has been partially funded by the Portuguese Foundation for Science and Technology/FEDER under Project Grant PTDC/ENR/ 64971/2006.

References

1. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester (2001)
2. Coello, C.A.C., Veldhuizen, D.V., Lamont, G.B.: Evolutionary Algorithms for Solving Multiobjective Problems. Kluwer Academic Publishers, New York (2002)
3. Veldhuizen, D.A.V., Lamont, G.B.: Multiobjective evolutionary algorithms: analyzing the state-of the-art. Evol. Comput. 8(2), 125–148 (2000)
4. Horn, J.: Multicriterion decision making. In: Bäck, T., Fogel, D., Michalewicz, Z. (eds.) Handbook of Evolutionary Computation, vol. 1, pp. F1.9:1–F1.9:15.5. Oxford Univ. Press, Oxford (1997)
5. Coello, C.A.C.: Handling Preferences in Evolutionary Multiobjective Optimization: A Survey. In: Proc. of the 2000 Congress on Evo. Computation, pp. 30–37 (2000)
6. Roy, B.: Multicriteria Methodology for Decision Aiding. Springer, Dordrecht (1996)
7. Roy, B.: The outranking approach and the foundation of ELECTRE methods. Theory and Decision 31, 49–73 (1991)
8. Mousseau, V., Slowinski, R., Zielniewicz, P.: A user-oriented implementation of the ELECTRE-TRI method integrating preference elicitation support. Comp. and Operations Research 27, 757–777 (2000)
9. Zopounidis, C., Doumpos, M.: Multicriteria classification and sorting methods: A literature review. European Journal of Operational Research 138, 229–246 (2002)
10. Zhang, W., Li, F., Tolbert, L.M.: Review of Reactive Power Planning: Objectives, Constraints, and Algorithms. IEEE Transctions on Power Systems 22, 2177–2186 (2007)
11. Antunes, C.H., Barrico, C., Gomes, A., Pires, D.F., Martins, A.G.: An evolutionary algorithm for reactive power compensation in radial distribution networks. Applied Energy 86, 977–984 (2009)

Multilevel Variable Neighborhood Search for Periodic Routing Problems*

Sandro Pirkwieser and Günther R. Raidl

Institute of Computer Graphics and Algorithms,
Vienna University of Technology, Vienna, Austria
{pirkwieser,raidl}@ads.tuwien.ac.at

Abstract. In this work we present the extension of a variable neighborhood search (VNS) with the multilevel refinement strategy for periodic routing problems. The underlying VNS was recently proposed and performs already well on these problems. We apply a path based coarsening scheme by building fixed (route) segments of customers accounting for the periodicity. Starting at the coarsest level the problem is iteratively refined until the original problem is reached again. This refinement is smoothly integrated into the VNS. Further a suitable solution-based recoarsening is proposed. Results on available benchmark test data as well as on newly generated larger instances show the advantage of the multilevel VNS compared to the standard VNS, yielding better results in usually less CPU time. This new approach is especially appealing for large instances.

1 Introduction

Periodic routing problems are a generalized variant of the classical routing problem where customers must be served several times in a given planning period instead of only once on a single day. Applications exist in many real-world scenarios as in courier services, grocery distribution, waste collection, or for various sorts of suppliers. The *Periodic Vehicle Routing Problem* (PVRP) is defined on a directed graph $G = (V, A)$, where $V = \{v_0, v_1, \dots v_n\}$ is the vertex set and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set. A planning horizon of t days, referred to by $T = \{1, \dots, t\}$, is considered. Vertex v_0 represents the depot at which are based m vehicles with positive capacities Q_k and maximum route durations D_k, $k = 1, \dots, m$. Each vertex of $V \setminus \{v_0\}$ corresponds to a city or a customer and has a nonnegative demand q_i, a nonnegative service duration d_i, a service frequency f_i, and a non-empty set $C_i \subseteq \{T' \mid T' \subseteq T,\ |T'| = f_i\}$ of allowed combinations of visit days. Each arc $(i, j) \in A$ has associated a nonnegative travel time (costs) c_{ij}. The PVRP consists of selecting one visit combination per customer and designing (at most) m vehicle routes on each day on G such that

* This work is supported by the Austrian Science Fund (FWF) under contract number P20342-N13.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 226–238, 2010.

1. each route starts and ends at the depot,
2. each customer i belongs to f_i routes over the planning horizon corresponding to a feasible visit combination in C_i,
3. the total demand of the customers on a route followed by vehicle k does not exceed Q_k, and its total duration does not exceed D_k, and
4. the total travel times (costs) over all routes is minimized.

We further consider a special case of the PVRP, the *Periodic Traveling Salesman Problem* (PTSP), where $m = 1$ and the vehicle capacity as well as the route duration are unconstrained.

As all available benchmark instances for both problems assume a homogeneous vehicle fleet, and vehicle capacities and maximum route durations are supposed to be independent of the day, we also restrict ourselves to this situation.

Multilevel refinement strategies [1,2] successively coarsen an initial problem instance, yielding a sequence of instances of decreasing size, representing the problem on different abstraction levels. The smallest instance is then solved by some auxiliary technique (e.g. a metaheuristic), and the obtained solution is *extended* to a solution of the previous level. This solution is eventually improved (e.g. again by some metaheuristic) and the solution extension continued until a solution for the original problem is obtained. Optionally, the whole process is iterated; hereby, a *recoarsening* is performed exploiting the last obtained solution in order to derive an eventually better hierarchy of abstraction levels. Multilevel refinement strategies have been successfully applied to several combinatorial optimization problems, including graph partitioning, the traveling salesman problem, and also the classical capacitated vehicle routing problem. When applied sensible, they seem to be able to often improve the scalability of metaheuristics, either improving running times or final solution qualities.

The subject of this work therefore is to study the extension of a variable neighborhood search (VNS) metaheuristic that was already successfully applied to periodic routing problems by a multilevel refinement strategy in order to improve the performance of the VNS especially on larger instances.

In the following section we review related work. The underlying VNS and the multilevel extension are presented in Sections 3 and 4, respectively. Experimental results, in which also new larger benchmark instances are used, are discussed in Section 5. Finally, conclusions and an outlook on future work is given in Section 6.

2 Related Work

From a problem oriented view recent successful approaches for one or both of the described problems are [3], [4], [5], and [6]. Cordeau et al. [3] describe a Tabu search for periodic and multi-depot routing problems, achieving for all previously known benchmark instances new best results at that time. The algorithm is based on rather simple standard neighborhoods that move single customers to different routes or change single visit combinations. New random test instances have further been introduced in this work. A well performing construction type algorithm with an embedded improvement procedure for the PTSP is presented

by Bertazzi et al. [4]. Alegre et al. [5] apply a Scatter search heuristic tailored especially to solve PVRP instances having a long planning horizon. Nevertheless they also obtained improved results for many standard benchmark instances. Most recently Hemmelmayr et al. [6] tackled the PVRP and PTSP with a sophisticated VNS that again yielded many new best results. As neighborhoods for diversification they utilized moving or exchanging route segments of different maximal size as well as changing several visit combinations. As improvement procedure they applied the well-known 2-opt and a restricted 3-opt. Similar to [3] they also allow infeasible solutions during the search but additionally apply an acceptance criterion similar to Simulated Annealing [7]. Francis et al. [8] gave a recent survey on periodic routing problems considering several variants of it.

A VNS based on the one from [6] has further been described for the PVRP with time windows in [9]. The concept of multilevel refinement including an overview on applications is covered in [1,2]. We point out the work by Rodney et al. [10] which introduces a multilevel refinement strategy for the capacitated vehicle routing problem. They coarsen the problem by building paths (segments) of customers through fixing the corresponding edges and consider such a path as an atomic unit in the following. Due to the capacity restrictions they propose to balance these paths according to the accumulated demand and systematically allow a gradually decreasing violation of the limiting constraint. Further the multilevel strategy has been previously applied to the traveling salesman problem [11,12] in a similar way. Yet we are not aware of an approach that was designed to handle the periodicity as well as to accomplishing such a smooth integration into a metaheuristic. Further, to our knowledge VNS has not been combined with multilevel refinement so far.

3 Underlying Variable Neighborhood Search

Variable neighborhood search (VNS) [13] in general utilizes random moves in a series of neighborhoods of growing size for diversification, referred to as shaking, and usually an embedded local search component for intensification. The core of the VNS is parameter free (after deciding about the neighborhoods), making it relatively easy to use. In the last decade this metaheuristic has been successfully applied to a wide range of combinatorial optimization problems. In the following, we give a rather short overview on our underlying VNS as most parts of it have already been described in more detail in [6,9].

To smooth the search space and help escaping local optima, the VNS relaxes the restrictions on vehicle load and route duration and adds penalties corresponding to the excess of these constraints to the cost function; similar to [9] a constant penalty factor of 1000 turned out to work reasonably well for the benchmark instances from literature and is therefore used, whereas an adaptive penalty was applied in [6]. Initially a possible visit combination is selected for each customer at random. Afterwards we apply the Clarke and Wright savings algorithm [14] in case of multiple routes. If we end up with too many routes, the customers of those routes holding the least customers are relocated in a greedy

way (for details we refer to [6]). By doing so the constructed routes might exceed maximal vehicle load or allowed tour duration. For instances with a single vehicle, i.e. especially for all PTSP instances, we make use of best insertion.

In the shaking phase we utilize three different neighborhood structures with increasing perturbation size per type: (i) randomly changing up to six visit combinations with greedy insertion for the new visit days, whereas for the PVRP we also allow reassigning the same visit combination, (ii) moving a random segment of up to three customers of a route to another one on the same day, and (iii) exchanging two random segments of up to three customers between two routes on the same day. We thus have a total of 12 shaking neighborhoods (i.e. $k_{\max} = 12$) which are always considered in a fixed order. These settings reflect previous experience and showed a good performance in preliminary tests. Contrary to [6] only segments of up to three instead of six customers are exchanged, we recognized no performance gain when doing so.

For intensification we apply the well-known 3-opt intra-route exchange procedure in a first improvement fashion for the PVRP and 2-opt for PTSP (restricting the latter to segments of length 10 in case of $n > 300$), only considering routes changed during shaking. Both are applied as long as an improvement is achieved, i.e. until a local optimum is reached. Additionally each new PVRP incumbent solution is subject to a 2-opt* inter-route exchange heuristic [15]. Hereby for each pair of routes of the same day all possible exchanges of the routes' end segments are considered. In case of the PTSP we additionally apply 3-opt on all routes. This is an addition to the work of [6].

To enhance the overall VNS performance Hemmelmayr et al. [6] propose to not only accept better solutions, but sometimes also solutions having a worse objective value. This is done in a systematic way using the Metropolis criterion like in simulated annealing [7]. A linear cooling scheme is used in a way such that the acceptance rate of worse solutions is nearly zero in the last iterations.

4 Multilevel Variable Neighborhood Search

In this section we extend the described VNS with ideas from the multilevel refinement strategy, hence we call it the *Multilevel VNS* (MLVNS). The basic idea of multilevel refinement [1,2] is to suitably coarsen the problem which has two effects: (i) to concentrate more on the costly parts of the problem during optimization, and (ii) at the same time to (temporarily) reduce its size, making it more tractable at the coarser levels. The problem is then (approximately) solved at the highest level and the obtained solution refined in an iterative way until a solution to the original problem is reached.

Contrary to most existing multilevel refinement approaches, our method

- does not automatically obtain one specific feasible solution at the coarsest level to start with, which is due to the periodicity in our case, and
- does not have several subsequent (and even independent) applications of the same method on different levels, but smoothly integrates the transitions from

the most abstract level to the original problem in the VNS; i.e. many levels with small changes (in fact always only one) are used.

In the following we propose an initial coarsening process operating at a given problem instance, as well as a recoarsening based on an incumbent solution.

4.1 Initial Problem Coarsening

Two common coarsening schemes on graphs are based on either merging nodes (into "super-nodes") or building segments (paths). Since we have to determine sequences of customer visits it is rather natural to follow the latter scheme here, keeping in line with previous work [11,10]. We opt for an exact coarsening, i.e. the objective value of a coarsened solution always equals the one of the corresponding fully refined solution. Basically during coarsening the nodes of the graph (i.e. single customers) are merged to segments via fixing a specific edge between them. One such coarsening step of merging two segments into a longer segment and the corresponding reversed refinement step are shown in Figure 1. Note that a single customer can be regarded a segment with equal start and end point, too. The accumulated cost, demand, and service duration of the newly created segment must be set accordingly.

Due to the periodicity, building customer sequences on a daily basis as for the single day of the classical vehicle routing problem would not make much sense. On the one hand this would require to choose a visit day combination per customer in advance (a bad choice would potentially result in a bad coarsening) and on the other hand such segments would most likely not allow to change its visit day combination without the need of breaking it apart, thus creating only short-lived segments being inconsistent with the general idea of the multilevel refinement strategy. Therefore it is necessary to apply a coarsening process respecting the periodicity and building segments spanning the whole planning horizon. This is achieved by incorporating the customers' sets of available visit day combinations—the service frequencies alone might not be sufficient—and allow customers and subsequently segments to be merged only if their sets are equal.

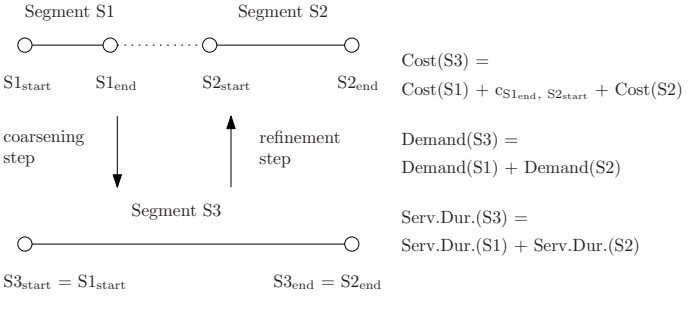


Fig. 1. Coarsen: merge two segments into one, refine: split a segment into two

At this point several initial problem coarsenings could be obtained by using different cost criteria as well as processing sequences. For example, in [10] per level they randomly choose an unmatched segment and fix an edge between it and the nearest unmatched segment according to the savings measure, whereas the actual costs of connecting the segments' free end nodes only considering segments in a defined surrounding for matching are used in [11]. We investigated several possibilities and after preliminary tests came up with these findings/settings:

- In general we observed the tendency that the greedier the selection for matching is, the better are the results on average; hence we always select the cheapest matching among all possible ones.
- Regarding the size (length) of the segments, i.e. how many customers are contained in the segment, we used two options:
  - *Setting I*: Enforce no limit at all, thus coarsen in a pure greedy fashion.
  - *Setting II*: Start with an allowed maximal length of two and gradually increase this limit by one whenever no more matching could be found. The maximum number of occurrences of one set of visit day combinations is used as an upper bound for this maximal length.
- We settled on using the actual costs of connecting the segments' free end nodes, thereby trying all four possible ways.
- Optionally we set a limit for the connection costs by multiplying the average inter-customer travel costs by a given factor δ_c, hence having an instance independent measure
- Optionally we respect given limits on vehicle capacity and route duration when building the segments.

A small schematic coarsening example for both settings I and II without limiting the connection costs is shown in Figure 2. The common practice of coarsening with a random factor yielded (so far) clearly worse results.

Having the coarsest problem, we generate the initial solution as described in Section 3 (there is no need to change this procedures). Then we interweave the iterative refinement with the VNS' execution. First, we select a fraction of the total VNS iterations in which this initial coarsening/refinement phase takes place. Then these iterations are divided by the number of present refinements (plus one to also execute some iterations on the fully refined problem) to determine the iterations per (mini-)level.

Refinement is exactly the reversed process as coarsening (in a "last in–first out" fashion), thus splitting the usually most costly matchings first and keeping the good ones accordingly longer. Hence in a refinement step (refer to Figure 1) we have to locate the segment in a route at all days of the actual visit day combination and split it in two segments again, thereby preserving the direction.

At the end of this phase we may either continue with the fully refined problem or apply a recoarsening which will be the subject of the next section.

4.2 Solution-Based Recoarsening

Solution-based recoarsening is a very common practice to extend the concept of multilevel refinement beyond the initial coarsening/refinement phase. For this,

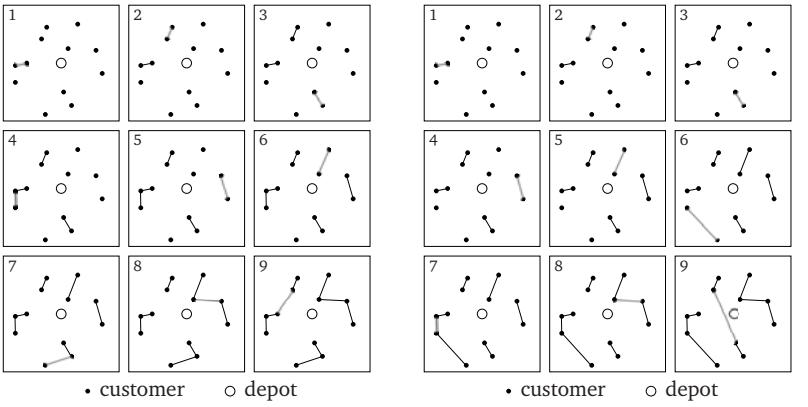


Fig. 2. Exemplary (deterministic) initial problem coarsenings for customers having the same visit day combinations, iteratively building longer (route) segments

the problem is recoarsened on the basis of a current incumbent solution in such a way as to preserve its structure, hence neglecting/destroying no obtained information. Despite this the coarsening principle stays the same. This obviously leads to considerably less degrees of freedom during the coarsening and in our case automatically to rather short segments.

This time we restrict our attention to adjacent segments in the currents solution's routes, whereas again, they must have equal sets of visit day combinations. Further, such a segment pair must be adjacent on all visit days, only a whole reversed occurrence (s.t. the same end points are connected) is acceptable. As a cost criterion we directly use the present connection costs.

Regarding the recoarsening procedure we basically adhere again to the greedy approach as used in the initial problem based coarsening, but try to prevent obtaining the same recoarsening in case we use the same solution multiple times: Using a parameter x, we always select one of the x cheapest possible matchings per step (or level) at random. Initially, x is set to one, but when we encounter the same solution to be used for recoarsening again, x is incremented. Contrary, if a different solution than in the previous iteration is used we reset x to one and turn the procedure into a pure greedy selection again. This extension further reduces the risk of getting stuck in local optima.

For recoarsening we have so far fixed the connection cost limit to $\delta_c/2$ of the average inter-customer travel costs (in case the factor δ_c is used at all). Since we utilize a feasible solution for guiding the recoarsening a violation of the vehicle capacity or route duration is impossible and must not be checked. Finally, the subsequent refinement phase is handled in the same way as the initial refinement in the preceding section. Therefore we also choose a fraction of the VNS' iterations to be allotted to a recoarsening/refinement phase.

4.3 Handling Segments in the VNS

Due to incorporating the multilevel refinement strategy into the VNS there are a few more things to consider when dealing with segments of merged customer:

- As already mentioned the solution construction heuristics need not to be changed, the same applies to moving or exchanging route segments and all local search procedures.
- After each refinement step we immediately apply the intra-route local search in use on the routes that contain refined segments.
- Although the segments have different start and end points, i.e. are asymmetric, when changing visit combinations during shaking we always reinsert them in the direction at the time of creation instead of the one in the previous route. We observed no gain doing otherwise nor when trying both directions.

5 Experimental Results

Currently our experiments are aimed at investigating the different performance of the standard VNS and the multilevel VNS, hence we are not yet hunting for new best solutions. Nevertheless, we already did find a few ones but we will not elaborate on this. The algorithms have been implemented in C++, compiled with GCC 4.3 and executed on a single core of a 2.83 GHz Intel Core2 Quad Q9550 with 8 GB RAM.

For each chosen setting we performed 20 runs per instance with a limit of 10^6 iterations, i.e. solution evaluations. Preliminary tests on all considered instances suggested to ignore the given limits on vehicle capacity and tour duration during coarsening (which would only affect PVRP instances), probably because the VNS was built to cope with infeasibility. If not stated otherwise 20% of all iterations are devoted to the initial coarsening/refinement phase and recoarsening is applied four times also devoting 20% of the iterations each. The two remaining options we varied are the type of problem coarsening (setting I or II) and the cost limit factor δ_c; these will be denoted by $[\text{I,II}]_{\delta_c}$. We experienced that usually it is better to enforce a merging cost limit.

Due to space limitations results on available "old" and "new" benchmark test data[1] both for the PVRP and the PTSP are presented in concise form in Table 1 (see [3] for further details and origins of these instances). Here we state the percentage gap to the so far best known solutions (%-gap BKS) as well as to the average results of the VNS described by Hemmelmayr et al. [6] using the same iteration limit of 10^6 (%-gap HDH), whereas all these values are given in [6]. The corresponding standard deviations are written in parentheses. For the MLVNS we further state the amount of CPU time spent given in percentage of the VNS' CPU time (%-time VNS). We chose following settings for the initial temperature

[1] Available on `http://neumann.hec.ca/chairedistributique/data/`

Table 1. Summarized average results on available benchmark test data

test data		instances	VNS		MLVNS			
			%-gap BKS	%-gap HDH	setting	%-gap BKS	%-gap HDH	%-time VNS
PVRP	old	32	2.67 (3.63)	-0.30 (1.01)	$II_{0.33}$	2.27 (3.18)	-0.67 (1.43)	105.0
	new	10	2.47 (1.58)	-0.80 (0.86)	$II_{0.5}$	2.45 (1.52)	-0.82 (0.88)	87.4
PTSP	old	23	0.80 (1.03)	0.01 (0.32)	$II_{0.5}$	0.32 (0.41)	-0.47 (0.74)	90.0
	new	10	0.28 (0.16)	-0.01 (0.03)	$II_{0.5}$	0.22 (0.14)	-0.08 (0.07)	79.0

Table 2. New larger PVRP and PTSP (setting $m = 1$ and ignoring D and Q) instances using the generation method introduced in [3], and our best found solutions' values

Id	n	m	t	D	Q	service frequencies					best found solutions	
						f_1	f_2	f_3	f_4	f_6	PVRP	PTSP
pr11	336	14	4	480	185	112	112		112		11611.19	6618.53
pr12	384	16	4	475	195	128	128		128		11428.81	7062.96
pr13	432	18	4	450	185	144	144		144		10686.30	6757.86
pr14	480	20	4	475	185	160	160		160		13733.77	7740.23
pr15	528	22	4	470	190	176	176		176		15430.77	8377.39
pr16	576	24	4	455	185	192	192		192		13465.62	7640.57
pr17	360	15	6	445	165	90	90	90		90	16166.45	9459.08
pr18	432	18	6	450	170	108	108	108		108	19891.65	11329.53
pr19	504	21	6	440	160	126	126	126		126	23825.45	11344.27
pr20	576	24	6	450	165	144	144	144		144	24285.09	11660.07

utilized in the Metropolis criterion: 10 for PVRP_new, 7 for PVRP_old (except 70 for instances p27–p32 having large average travel costs), as well as also 7 for PTSP_old and PTSP_new. 1000 iterations are applied on each temperature level. As can be seen the MLVNS generally performs better than the standard VNS, achieving especially on the old data sets a significant improvement. Contrary, on the new data sets no clear improvement can be noticed. Yet for all but PVRP_old there is a CPU time reduction of 15% on average. For these data sets the coarsening setting II gave consistently better results, whereas especially for PVRP/PTSP_new differences are negligible.

Since the multilevel refinement strategy is in general especially appealing for large(r) instances, but the available test data lack them, we created some on our own by applying the generation method described in [3][2]. They can be regarded a continuation of the instances introduced in this latter work, except that we evenly distributed the visit frequencies among the customers for all instances; more details, including our best found solutions' values, are given in Table 2.

When doing preliminary tests we recognized that the VNS' acceptance decision using the Metropolis criterion in some way weakens the potential gain of the multilevel extension. Hence we also performed tests with only accepting improved solutions, whose results are given in Table 3 and 5 for the PVRP and the PTSP, respectively. The corresponding results using the default acceptance

[2] These instances are available on `http://www.ads.tuwien.ac.at/sandro/routing/`

Table 3. Average results of standard and multilevel VNS on new larger PVRP instances only accepting improved solutions

Id	VNS			MLVNS $I_{0.5}$				
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	%-gap	%-time
pr11	12182.87	162.55	43.9	**12091.85**	161.61	34.2	-0.75	77.9
pr12	12115.62	94.96	48.6	**11957.43**	100.46	37.5	-1.31	77.2
pr13	11490.63	97.32	57.1	**11286.32**	157.02	44.2	-1.78	77.4
pr14	14553.84	90.33	69.6	**14351.15**	126.11	52.0	-1.39	74.7
pr15	16542.60	124.83	87.1	**16028.83**	59.82	61.2	-3.11	70.3
pr16	14764.22	119.59	107.9	**14109.85**	173.60	73.2	-4.43	67.8
pr17	17127.64	109.75	47.1	**16794.86**	105.07	38.3	-1.94	81.3
pr18	20978.79	104.37	60.8	**20582.41**	115.18	48.3	-1.89	79.4
pr19	25191.24	257.97	85.3	**24811.23**	324.06	62.8	-1.51	73.6
pr20	25803.80	140.15	116.1	**25310.04**	153.18	83.4	-1.91	71.8
avg.							-2.00	75.2

Table 4. Average results of standard and multilevel VNS variants on new larger PVRP instances using the acceptance decision with the Metropolis criterion

Id	VNS			MLVNS $I_{0.5}$				
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	%-gap	%-time
pr11	12018.84	205.47	42.1	**11815.43**	127.86	33.9	-1.69	80.5
pr12	11627.45	69.89	46.0	11639.32	87.91	37.1	0.10	80.7
pr13	10859.17	48.62	54.7	**10808.92**	86.46	43.7	-0.46	79.9
pr14	13895.99	47.52	67.3	13884.29	77.81	52.7	-0.08	78.3
pr15	15661.02	67.56	84.2	**15595.55**	58.51	62.5	-0.42	74.2
pr16	13808.28	73.71	103.7	**13714.56**	118.58	75.2	-0.68	72.5
pr17	16378.20	106.68	47.5	16340.21	86.95	40.1	-0.23	84.4
pr18	20107.76	107.32	62.3	**20043.43**	102.61	51.6	-0.32	82.8
pr19	24411.55	121.89	90.3	**24207.79**	291.51	67.1	-0.83	74.3
pr20	24686.82	87.57	117.7	**24540.46**	107.78	87.9	-0.59	74.7
avg.							-0.52	78.2

decision are given in Table 4 and 6, using an initial temperature of 7 for the PVRP and 10 for the PTSP instances again with a temperature decrease every 1000 iterations. For both variants and problems we tested coarsening setting I and II with a cost limit factor δ_c of 0.5 and 0.33, as well as devoting all iterations to the initial coarsening/refinement phase and doing no recoarsening at all, whereas we always state the results of the setting yielding the best solutions on average. The tables show average travel costs (avg.), corresponding standard deviations (sdv.), CPU times in seconds (t[s]), as well as the percentage gaps to the VNS (%-gap) and the times in percent of the VNS (%-time) for the MLVNS. In Tables 3–6 average values of the MLVNS are printed bold whenever a statistically significant improvement compared to the VNS has been achieved, according to a Wilcoxon rank sum test with an error level of 5%. For these new larger instances the greedy coarsening setting I turned out to achieve consistently better results than setting II. Comparing Table 3 and 4, as well as Table 5 and 6 it can be clearly seen that the MLVNS yields a higher relative improvement when only accepting improved solutions, nevertheless also when using the default acceptance decision the improvement is notable, especially in case of the PTSP. Interestingly, for the PTSP using no recoarsening and

Table 5. Average results of standard and multilevel VNS variants on new larger PTSP instances only accepting improved solutions

Id	VNS			MLVNS $I_{0.5}$ (no recoarsening)				
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	%-gap	%-time
pr11	6856.76	28.09	188.6	**6775.87**	25.99	87.7	-1.18	46.5
pr12	7314.39	32.13	241.0	**7178.50**	29.23	109.1	-1.86	45.3
pr13	6969.09	30.63	320.7	**6861.12**	28.31	140.5	-1.55	43.8
pr14	8055.41	38.49	397.9	**7860.15**	33.72	175.9	-2.42	44.2
pr15	8678.21	33.31	503.1	**8525.48**	20.00	193.9	-1.76	38.5
pr16	7974.06	40.39	631.3	**7741.01**	25.78	246.8	-2.92	39.1
pr17	9854.09	36.76	205.9	**9639.06**	49.01	100.6	-2.18	48.9
pr18	11705.94	57.41	295.6	**11536.59**	41.65	141.6	-1.45	47.9
pr19	11795.08	61.93	424.5	**11591.77**	39.54	180.5	-1.72	42.5
pr20	12101.01	61.23	578.1	**11878.36**	31.65	244.1	-1.84	42.2
avg.							-1.89	43.9

Table 6. Average results of standard and multilevel VNS variants on new larger PTSP instances using the acceptance decision with the Metropolis criterion

Id	VNS			MLVNS $I_{0.5}$ (no recoarsening)				
	avg.	sdv.	t[s]	avg.	sdv.	t[s]	%-gap	%-time
pr11	6736.19	32.29	182.1	**6670.51**	21.08	87.1	-0.98	47.8
pr12	7177.48	24.15	232.3	**7110.25**	25.11	106.5	-0.94	45.8
pr13	6881.71	30.23	295.1	**6789.76**	17.15	137.6	-1.34	46.6
pr14	7883.37	38.56	370.7	**7792.40**	25.88	171.2	-1.15	46.2
pr15	8518.06	35.33	471.4	**8424.37**	31.92	190.2	-1.10	40.3
pr16	7827.16	27.17	575.9	**7672.95**	22.59	238.4	-1.97	41.4
pr17	9596.32	30.20	215.9	**9534.96**	24.31	103.0	-0.64	47.7
pr18	11443.24	38.55	305.2	**11370.88**	29.46	143.1	-0.63	46.9
pr19	11551.89	39.00	421.8	**11435.78**	35.91	184.7	-1.01	43.8
pr20	11806.66	38.64	569.5	**11706.07**	28.03	257.8	-0.85	45.3
avg.							-1.06	45.2

extending the initial problem coarsening/refinement phase to all iterations turned out to be usually better (which only holds for the new larger instances, we checked it for the available test data, too). This has the additional positive effect that the required runtime is more than halved. Contrary, for the PVRP the decrease in runtime is again about 25%. In summary, for almost all instances and both acceptance decisions the MLVNS yields on average significantly better results than the VNS.

6 Conclusions

We extended a recently proposed leading variable neighborhood search (VNS) with the multilevel refinement strategy to a multilevel VNS (MLVNS) for better solving periodic routing problems. A path based coarsening scheme is used that builds fixed (route) segments of customers accounting for the periodicity. The refinement process, i.e. starting at the coarsest level and iteratively refining until the original problem is reached again, is smoothly integrated into the VNS. Furthermore a suitable solution-based recoarsening is proposed that respects

the structure of a given solution during coarsening. We presented results on available benchmark test data as well as on newly generated larger instances that show the advantage of the multilevel VNS compared to the standard VNS, often yielding significantly better results in usually less CPU time. In general the performance gain on the PTSP instances is higher. This new approach is especially appealing for large instances.

In the future we want to test with longer runs, also for the available test data, to further analyze the performance of the MLVNS and hopefully to find some (more) new best solutions. It might further be interesting to create even larger instances having different characteristics. We also think about alternative ways of interweaving the refinement with the VNS, such as refining whenever the VNS gets stuck for a while; similar thoughts apply to the recoarsening. Also the interrelation of the multilevel extension and the Metropolis criterion is worth to investigate further. Finally, this promising multilevel refinement strategy for periodic routing problems could be applied to other underlying algorithms as well.

References

1. Walshaw, C.: Multilevel refinement for combinatorial optimisation problems. Annals of Operations Research 131, 325–372 (2004)
2. Walshaw, C.: Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In: Blum, C., et al. (eds.) Hybrid Metaheuristics: An Emerging Approach to Optimization. SCI, vol. 114, pp. 261–289. Springer, Heidelberg (2008)
3. Cordeau, J.F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks 30(2), 105–119 (1997)
4. Bertazzi, L., Paletta, G., Speranza, M.G.: An improved heuristic for the period traveling salesman problem. Computers & Operations Research 31(8), 1215–1222 (2004)
5. Alegre, J., Laguna, M., Pacheco, J.: Optimizing the periodic pick-up of raw materials for a manufacturer of auto parts. European Journal of Operational Research 179(3), 736–746 (2007)
6. Hemmelmayr, V.C., Doerner, K.F., Hartl, R.F.: A variable neighborhood search heuristic for periodic routing problems. European Journal of Operational Research 195(3), 791–802 (2009)
7. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220(4598), 671–680 (1983)
8. Francis, P.M., Smilowitz, K.R., Tzur, M.: The period vehicle routing problem and its extensions. In: Golden, B., et al. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 73–102. Springer, US (2008)
9. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In: Prodhon, C., et al. (eds.) Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France (2008)
10. Rodney, D., Soper, A., Walshaw, C.: Multilevel refinement strategies for the capacity vehicle routing problem. International Journal of Information Technology & Intelligent Computing 2(3) (2007)

11. Walshaw, C.: A multilevel approach to the travelling salesman problem. Operations Research 50(5), 862–877 (2002)
12. Bouhmala, N.: Combining local search with the multilevel paradigm for the traveling salesman problem. In: Blum, C., et al. (eds.) Proc. 1st Intl Workshop on Hybrid Metaheuristics, pp. 51–58 (2004)
13. Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 145–184. Kluwer Academic Publishers, Boston (2003)
14. Clarke, G., Wright, J.W.: Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12(4), 568–581 (1964)
15. Potvin, J.Y., Rousseau, J.M.: An exchange heuristic for routeing problems with time windows. Journal of the Operational Research Society 46, 1433–1446 (1995)

Enhancing Genetic Algorithms by a Trie-Based Complete Solution Archive

Günther R. Raidl and Bin Hu

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

Abstract. Genetic algorithms (GAs) share a common weakness with most other metaheuristics: Candidate solutions are in general revisited multiple times, lowering diversity and wasting precious CPU time. We propose a complete solution archive based on a special binary trie structure for GAs with binary representations that efficiently stores all evaluated solutions during the heuristic search. Solutions that would later be revisited are detected and effectively transformed into similar yet unconsidered candidate solutions. The archive's relevant insert, find, and transform operations all run in time $O(l)$ where l is the length of the solution representation. From a theoretical point of view, the archive turns the GA into a complete algorithm with a clear termination condition and bounded run time. Computational results are presented for Royal Road functions and NK landscapes, indicating the practical advantages.

Keywords: genetic algorithms, solution archive, revisits, tries.

1 Introduction

Genetic algorithms (GAs) [7] are population-based metaheuristics for solving difficult optimization problems. This popular class of evolutionary algorithms often is able to find good approximate solutions within a huge search space in relatively short computation times. However, a drawback of GAs is that they usually do not keep track of the search history, and already evaluated solutions are often revisited. This in particular holds when the used selection pressure is rather high, the population size only moderate, or the variation operators do not introduce much innovation. In the extreme case, the population's diversity drops strongly and the GA gets stuck by creating almost only duplicates of a small set of leading candidate solutions, called *super-individuals*. In such a situation of premature convergence, it becomes very obvious that the heuristic search is not performing well anymore, and something must be changed in the GA's setup. However, also in scenarios which are believed to be rather well-configured, solutions are sometimes revisited and evaluated multiple times. Especially when

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 239–251, 2010.

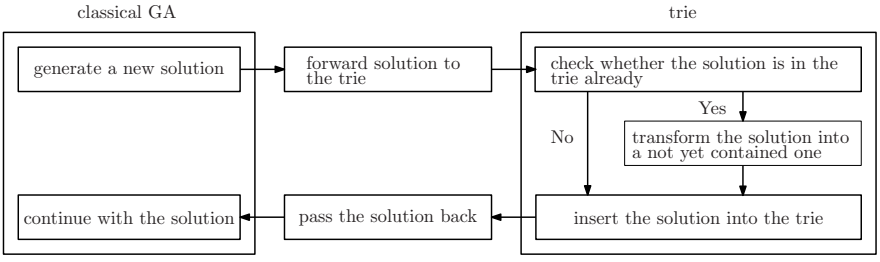


Fig. 1. Cooperation between GA and trie

the evaluation function is costly in terms of running time, such reconsiderations of the same candidate solutions may degrade performance substantially; re-evaluation in a general sense is a clear waste of precious computation time[1].

Various kinds of *population management strategies* have been suggested in literature to counteract premature convergence and to ensure a reasonable diversity in the population. They also reduce the number of revisits but are in general not able to completely avoid them. Rejecting new candidate solutions that are already contained in the current population of a steady-state GA often improves the situation and increases performance; it is therefore considered a good default strategy by many researchers. See e.g. [15] for a study in this respect. Nevertheless, revisits are obviously not entirely avoided in this way.

We consider a *complete solution archive* based on a memory-efficient *trie* data structure for GAs with binary solution representations in order to (a) efficiently detect already evaluated candidate solutions and to (b) efficiently transform them by typically small adaptations into yet unconsidered candidate solutions. Figure 1 illustrates how this archive is attached to the GA. In principle, this archive turns the GA into a *complete* optimization approach, which from a theoretical point of view is guaranteed to find an optimal solution in bounded time.

The computational overhead introduced by the archive is relatively low: The essential operations of inserting a new candidate solution, checking whether or not a new candidate solution is already contained in the archive, and transforming an already contained solution can all be performed in time $O(l)$, where l is the number of bits the binary solution representation has. Thus, the asymptotic time complexity of each iteration of the GA is not increased.

2 Related Work

Roland [17], for example, has shown that diversity loss through duplicates is a serious weakness in the steady-state GA model. He also proposed a simple way of removing duplicates by using a hash table to store all solutions currently in the population [16]. Going even further, Mauldin [12] compares newly generated

[1] With the exception of dynamic or stochastic scenarios where multiple evaluations might be intended to acquire updated or more reliable objective values.

solutions to all members of the population by their Hamming distance in order to select solutions for removal, hereby maintaining even higher diversity. Many similar approaches can be found in literature, however, they only take into account solutions of the current population and do not maintain a memory for the search history.

Looking more generally onto the field of heuristic search techniques, the popular *tabu search* (TS) [5] metaheuristic actually is based on the concept of maintaining a memory, usually called *tabu list*, that keeps track of the search progress in order to avoid cycling. Different kinds of memories are used, but typically only attributes of recently performed moves are recorded and used to forbid moves into parts of the search space. Usually, these tabu lists also have restricted length, and an appropriate choice of this length parameter often is a non-trivial task. Too long tabu lists may restrict the search too strongly, while too short lists will not effectively avoid cycles. The trend goes towards adaptive parameter control mechanisms, as e.g. in *reactive tabu search* [2]. Of course all these attribute-based TS approaches are not guaranteed to entirely avoid revisits.

Only very few TS approaches exist where entire solutions are directly or indirectly recorded in a memory and precisely those moves that would yield revisits are forbidden during the remaining search. The *reverse elimination method* [5] is one such example to realize what may be called *strict tabu search*. It is, however, relatively slow as at iteration n it requires a computation of order $O(n)$ to check whether or not a move is allowed. Therefore, Battiti and Tecchiolli [2] suggest to use classical *hashing methods*, see e.g. [1], or a *digital tree* [9] instead, by which the essential insertion and find operations can be performed in (expected) time $O(l)$, thus only depending on the size of the binary representation.

Battiti and Tecchiolli [2], however, further argue that such strict TS approaches might not work well in general as they often converge very slowly for problems where the local optima are surrounded by large *basins of attractions*, i.e., by large sets of candidate solutions that converge to the local optimum when performing local search. This slow convergence is related to a slow "basin filling" effect. Well-tuned attribute-based approaches in which larger parts of the search space are temporarily disallowed are therefore typically more effective. In addition, global optima might even become unreachable because of the creation of barriers consisting of already visited solutions. We believe that these negative aspects also require careful attention in the context of an archive-enhanced GA. However, the possible implications are obviously substantially less critical, as in contrast to TS a GA also includes other mechanisms for diverting the search and jumping over barriers (i.e., recombination and mutation).

Focusing again on evolutionary algorithms, solution archives have been used in general for several different purposes. Sometimes, elite solutions are explicitly stored in order to re-use them later in some way for improving search performance, see e.g. [4]. In particular in evolutionary algorithms for multiobjective optimization, explicit solution archives are frequently used for storing non-dominated pareto-optimal solutions [3]. The idea of *caching* objective values of visited solutions in order to avoid re-evaluations when they are revisited is quite natural in

particular when the evaluation function is costly. For example Louis and Li [11] suggest to store solutions in a binary tree in such cases. Among others, Kratica [10] and Ronald [16] described similar caching approaches using hash tables. Depending on the time effort of the objective function and the frequency of revisits, the total computation time can be lowered substantially. However, revisits are not prevented (or rejected) in these methods.

Aiming at completely avoiding revisits in a GA targeted towards continuous optimization problems, Yuen and Chow [19] proposed to use an archive based on a k-d tree data structure for storing all visited solution. When encountering a revisit, the corresponding solution is mutated in a special way in order to always derive a new, yet unvisited solution. This approach actually comes closest to the concept we pursue in the current work. Differences are, however, that we concentrate on a discrete binary search space and our trie-based approach is extended by certain features (e.g., randomization) in order to avoid a strong biasing when modifying already visited solutions.

More detailed, preliminary results of the current work can be found in the master thesis of Sramko [18], which has been supervised by the first author of the current work. In a follow-up master thesis by Zaubzer [20], the trie-based archive has been applied to a memetic algorithm for the multidimensional knapsack problem. Here, the knapsack constraints make the situation more complicated. Substantial problem-specific extensions have thus been considered for the archive in order to only produce candidate solutions on the boundary of the feasible region. Depending on the test instances and configuration, some benefits could be observed when using this archive. However, due to the strong repair and local improvement procedures embedded in this memetic algorithm, general advantages turned out to be rather small. Nevertheless, also this work gives a clear indication that the basic idea of enhancing a GA by a trie-based complete solution archive might be promising for many kinds of problems.

3 Trie-Based Solution Archive

We propose a solution archive for GAs based on a *binary trie*. This archive is, in principle, more generally applicable to metaheuristics for problems in which solutions are encoded as binary strings.

Tries are a class of data structures that are typically used to efficiently store a possibly very large set of strings [6]. Applications lie e.g. in language dictionaries for spell checking and translation or the indexing of documents for allowing a fast search. Different variants of tries exist, but they all have in common that the computational complexity of the essential insert and find operations only depends linearly on the string-length of the respective key, i.e., it is in $O(l)$. In comparison, balanced binary search trees would require $O(l \log n)$ time for these operations and typically also require significantly more memory. Hash tables are w.r.t. insert and find in the expected case similarly efficient as binary tries, i.e., they also only require $O(l)$ time, but they do not allow an efficient implementation of a *transform* operation that modifies an already stored solution

into a similar new one. It shall also be remarked that *digital trees* [9] exhibit strong similarities to binary tries.

3.1 Basic Trie Structure

The basic version of our binary trie is simple. It is a binary tree T of maximum height l. Each trie node t_i at level $i = 1, \ldots, l$ has identical structure: It consists of two entries $t_i.next[0]$ and $t_i.next[1]$ which are either pointers referring to successor nodes at the next level or are set to the flags *complete* or *empty*. The root node t_1 refers to the whole binary space $\{0,1\}^l$ and each other trie node t_i, $i > 1$, to a well-defined part of it: Considering binary vectors (i.e., candidate solutions) $x = (x_1, x_2, \ldots, x_l) \in \{0,1\}^l$, entries $t_i.next[0]$ and $t_i.next[1]$ of a node t_i always partition the corresponding space into the two subspaces containing those vectors with $x_i = 0$ and $x_i = 1$, respectively. Thus, the i-th bit of a vector x decides whether to go "left" or "right" at level i. An entry of *empty* means that none of the vectors lying in the corresponding subspace is yet contained in the trie, while an entry *complete* indicates the case that all corresponding vectors are contained (i.e., these solutions have already been visited by the GA). Figure 2 shows an exemplary trie containing the solution 010 and all solutions with $x_1 = 1$ ('x' denotes the *complete*-flag and '/' the *empty*-flag).

To check whether or not a solution x is stored in T, the search algorithm steps down the trie beginning at the root and follows $t_i.next[x_i]$ in each level i. If a *complete*-flag is encountered during this process, x is contained in T; if *empty* is encountered, x has not yet been inserted.

Adding new, not yet contained solutions works similarly, but new trie nodes must eventually be created when encountering an *empty* flag, and after considering the last bit a corresponding *complete*-flag is stored.

An important principle to keep the trie small is to prune a subtrie when all solutions corresponding to it have been added, i.e., if $t_i.next[0] = t_i.next[1] =$ *complete*, the respective pointer to t_i in the previous level is replaced by a *complete*-flag; see Fig. 3, where 011 has additionally been inserted. The special situation that the root pointer becomes *complete* thus indicates that all solutions of the whole search space have been added, and the GA can be terminated returning its best found solution as (proven) optimum.

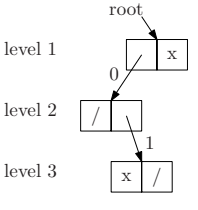


Fig. 2. Trie T containing solutions 010 and all with $x_1 = 1$

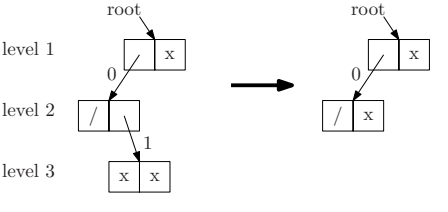


Fig. 3. Pruning the subtrie containing solutions 010 and 011

3.2 Avoiding Revisits by Transforming Solutions

One of the most important features of our solution archive is the ability to transform an already contained solution x, which would lead to a revisit in the GA, into a typically similar but yet unconsidered solution x'. By "similar" we mean that the Hamming distance between x and x' is low. Considering the example of Fig. 2, if the GA's variation operators yield $x = 010$ again, this solution can be modified to $x' = 011$, which is then inserted in T leading to the situation shown in Fig. 3.

More generally, the basic idea of the transformation is to go back to some previous node at the search path of x whose alternate branch $p.next[1 - x_i] \neq complete$, i.e., contains at least one yet unconsidered solution. Here, at this deviation position, we set $x_i = 1 - x_i$ and go down this alternate subtrie following the remaining bits of x whenever possible, i.e., unless we encounter a *complete*-flag in which case we choose again the alternate branch that must contain at least one unconsidered solution. We distinguish two algorithm variants w.r.t. the selection of the deviation position:

Deepest Position Transformation (DPT): In this basic variant the last (deepest possible) deviation position is always chosen. While this method probably is the most straight-forward one, it has the disadvantage of a strong biasing towards modifying bits at higher positions while keeping bits at lower positions unchanged. In fact, only when large portions of the search space have already been covered, bits at lower positions will be considered.

Random Position Transformation (RPT): To substantially reduce the above mentioned biasing, a random choice from all feasible deviation positions is made in the RPT algorithm variant. Otherwise, this method behaves in the same way as DPT.

A more detailed pseudo-code covering both transformation variants is given in Algorithm 1. We first search for x, then go back to the deviation point and go down once more in order to insert the transformed solution. The algorithm can be implemented in time $O(l)$.

3.3 Randomized Trie Structure

Using the basic trie structure where trie nodes on level i are always bounded to the bits on the i-th position has a significant weakness. Especially when using DPT for handling duplicates, we already observed that there is a strong bias towards some genes being changed much more frequently than others. In particular, bits at higher positions are always tried to be modified first. This bias typically results in repeated, rigid patterns when visualizing visited solutions in the search space. In general, when an intensive search around an incumbent solution has been performed and the trie is already moderately filled, the transform operation might need to flip not just the single bit at the deviation point but more bits in the course of finding a yet unconsidered solution, resulting in larger Hamming distances. As DPT considers the positions always in the strictly same

Algorithm 1. Transform Solution

```
Input: solution x = (x_1, ..., x_l); algorithm variant var ∈ {DPT, RPT}
p = root; devpoints = ()
// search for x storing possible deviation positions in devpoints
i = 1
while i ≤ l and p ≠ complete do
    if p.next[1 − x_i] ≠ complete then
        devpoints = devpoints ∪ (i, p)
    p = p.next[x_i]
    i = i + 1
if var = DPT then
    (i, p) = last entry of devpoints   // go back to last feasible deviation position
else
    (i, p) = random entry from devpoints   // go back to a random dev. position
x_i = 1 − x_i                          // actually deviate by flipping bit i
while i ≤ l do
    if p.next[x_i] == complete then
        x_i = 1 − x_i
    if p.next[x_i] == empty then
        p.next[x_i]=new trie node (empty, empty)
    q = p
    p = p.next[x_i]
    i = i + 1
q.next[x_i] = complete                 // insert transformed x in T
prune trie nodes of type (complete, complete) bottom up
return x
```

order, this effect is significantly amplified, and transformed solutions with larger distances, i.e., less similar solutions, are created earlier and/or more frequently. This behavior can be compared with the well-known effect of *primary clustering* in hash tables when using linear probing as collision handling strategy.

The described RPT variant reduces these weaknesses substantially, but it is still not able to avoid a biasing entirely. As an alternative or additional improvement, we consider the randomization of the trie structure itself. The idea is to use individual, in general different bit orders on different search paths of the trie. Trie nodes at depth i are no longer related to the bit at position i, i.e., x_i, but a deterministic pseudo random (or hash) function is used to calculate the specific bit position j related to a given trie node t_i in dependence of the whole path from the root to t_i. Fig. 4 illustrates this randomized trie variant. One must be careful in the choice of the underlying pseudo random function. For example, classical Park-Miller random number generators are unsuitable as there are correlations between input and output data. We used the "pseudo data-encryption standard" algorithm *ran4* [14].

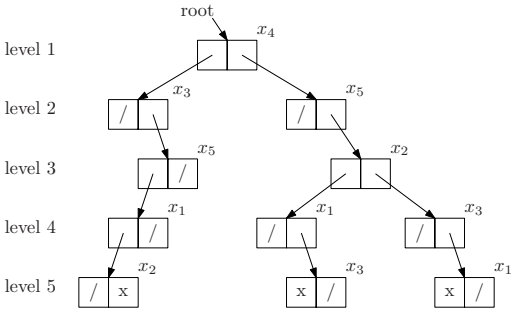


Fig. 4. Solutions 01100, 10011, 01111 in a trie with randomized structure. Nodes are labeled by their related bits of the solution vector.

4 Computational Results

We present test results on two classes of standard benchmark problems from the binary domain: The Royal Road functions and NK fitness landscapes.

A more-or-less standard steady-state GA was used, which derives in each iteration one new solution by performing tournament selection with replacement, always applying single point crossover, and mutating each bit with probability $1/l$. This new solution always replaces the population's worst solution. When the archive is attached, classical mutation is turned off and replaced by the transform operation of the archive in the case already visited solutions are obtained from crossover. Initial solutions were randomly created, the population size was 100, and the tournament selection group size 10. We compare the GA-variants and trie configurations summarized in Table 1.

In case the solution archive is not used (variant std), a classical duplicate elimination strategy as described in [16] is applied in the GA, i.e., a newly derived solution is only accepted in the population if it is different to all other solutions therein and discarded otherwise. The necessary duplicate-checks are efficiently performed by means of a hash table. Tests without any duplicate elimination and without the suggested archive yielded consistently clearly worse results, and we therefore do not include them here. All experiments were performed on a Pentium 4, 2.8 GHz PC with 1GB RAM.

Table 1. Considered GA variants

algorithm	trie	transformation	trie structure
std	not used	–	–
tdb	used	deepest position	basic
trb	used	random position	basic
tdr	used	deepest position	randomized
trr	used	random position	randomized

4.1 Results on Royal Road Functions

Royal Road functions were specifically designed for evaluating GAs with their crossover operators by Mitchell et al. [13]. They are defined for binary strings $x \in \{0,1\}^l$ on a set of hierarchically created schemas $S = \{s_1, s_2, \ldots, s_n\}$, and the objective is to maximize $\sum_{s \in S} o(s)\sigma_s(x)$, where $o(s)$ is the order of schema s (i.e., the number of defined bits) and $\sigma_s(x)$ is the binary indicator-function yielding 1 if x matches s and 0 otherwise. For details on S we refer to [13]; let b be the order of the smallest basic building blocks in S and $r = l/b$ be the number of them.

Table 2 lists results on differently parameterized Royal Road functions. All runs are terminated after 1000 iterations. Average final solution values (avg) and corresponding standard deviations (sd) of 100 independent runs are printed for each test case. Best values are marked bold.

Table 3 additionally lists the following total results over all considered Royal Road functions: Average objective values, average elapsed CPU-times when the best solutions were identified, the number of solution transformations (i.e., avoided revisits), and for each pair of GA variants the error probability of a one-sided Wilcoxon rank sum test for the assumption that one GA-variant performs better than the other.

We can observe that the GA without the trie archive performs worst in general. Among the different trie-variants, performance differences are rather small, but using the random deviation transformation and the randomized trie structure leads to noticeable improvements.

Table 2. Results on Royal Road functions

instance		std		tdb		trb		tdr		trr	
b	r	avg	sd	avg	sd	avg	sd	avg	sd	avg	sd
3	4	**36.00**	0.00	**36.00**	0.00	**36.00**	0.00	**36.00**	0.00	**36.00**	0.00
4	4	**48.00**	0.00	**48.00**	0.00	**48.00**	0.00	**48.00**	0.00	**48.00**	0.00
5	4	**60.00**	0.00	**60.00**	0.00	**60.00**	0.00	59.30	4.95	**60.00**	0.00
6	4	62.88	18.71	64.32	16.58	68.52	11.94	68.28	12.94	**68.64**	11.51
3	8	**96.00**	0.00	**96.00**	0.00	**96.00**	0.00	**96.00**	0.00	**96.00**	0.00
4	8	124.40	14.39	126.80	8.49	124.40	14.39	125.60	11.88	**128.00**	0.00
5	8	103.50	45.41	115.40	43.22	115.20	42.89	110.60	42.64	**125.30**	44.01
6	8	73.80	35.16	**92.64**	36.95	81.72	51.38	77.76	38.06	81.84	44.88
3	16	206.28	45.43	215.76	41.31	217.68	40.12	**226.98**	32.60	219.54	38.92
4	16	148.08	43.99	160.08	55.48	166.16	63.70	**168.00**	66.07	153.44	55.26
5	16	**106.50**	37.95	100.00	38.69	104.50	38.61	93.00	32.09	102.30	41.46
6	16	70.44	30.52	79.44	41.81	74.16	29.34	74.52	29.50	**82.68**	35.01

Table 3. Royal Road functions: Averages over all instances and Wilcoxon rank sum tests for each pair of GA-variants

alg	avg	time	transforms	.vs std	.vs tdb	.vs trb	.vs tdr	.vs trr
std	92.96	0.02s	–	–	0.9953	0.9942	0.9677	0.9996
tdb	99.54	0.03s	312	0.0047	–	0.3854	0.2901	0.6849
trb	99.36	0.05s	297	0.0058	0.6150	–	0.3609	0.7099
tdr	98.67	0.03s	307	0.0323	0.7102	0.6394	–	0.9122
trr	100.15	0.05s	301	0.0004	0.3154	0.2904	0.0879	–

4.2 Results on NK Landscapes

Introduced by Kauffman [8], NK landscapes serve as another popular benchmark suit for GAs with binary representations. The goal is to maximize the function

$$F(x) = \frac{1}{N} \sum_{i=1}^{N} f_i(x_{i_1}, x_{i_2}, \ldots, x_{i_K})$$

with $x \in \{0,1\}^N$. Each subfunction f_i takes gene x_i and K neighboring genes $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$ into account and returns a value in $[0,1]$ according to a random value table. Hence there are N tables of size 2^{K+1} from which the values are read out. With increasing K, the coupling between particular genes rises and the problem becomes more complex. Two variants exist for choosing the neighboring genes $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$: the *adjacent neighbors* version, where these genes are the closest ones to x_i, and the *random neighbors* version, where they are selected randomly distributed among all $x_1, \ldots, x_N$. We examine the NP-hard latter one.

Parameter N was set to 20, 50, 100 and 300 and K to 1, 2, 5, 6, 7, 8, 9 and 10. For each combination, we performed 50 independent runs and each run was terminated after 10s. Since the final solution values for different N but the same K are relatively similar, we decided to present here only accumulated results grouped by K due to space reasons. Table 4 shows these average final solution values and corresponding standard deviations for the standard GA and the four trie-enhanced variants. Though the average objective values here are close, the advantage of the trie is very obvious. In particular, the GA variant using the random deviation transformation together with the randomized trie structure was able to generate best average results most of the time. This becomes even more

Table 4. Results on NK landscapes (random neighbors) over $N \in \{20, 50, 100, 300\}$, 10s CPU-time limit

	std		tdb		trb		tdr		trr	
K	*avg*	*sd*	*avg*	*sd*	*avg*	*sd*	*avg*	*sd*	*avg*	*sd*
1	0.7090	0.0285	0.7089	0.0288	0.7086	0.0288	**0.7092**	0.0286	0.7089	0.0288
2	0.7351	0.0220	0.7354	0.0217	0.7364	0.0219	0.7361	0.0217	**0.7365**	0.0216
5	0.7603	0.0222	0.7623	0.0222	0.7611	0.0219	0.7609	0.0219	**0.7633**	0.0211
6	0.7628	0.0232	0.7631	0.0239	0.7641	0.0227	0.7645	0.0226	**0.7649**	0.0225
7	0.7595	0.0223	0.7583	0.0217	0.7592	0.0211	**0.7607**	0.0216	0.7600	0.0219
8	0.7567	0.0241	0.7583	0.0249	0.7582	0.0244	0.7597	0.0232	**0.7608**	0.0245
9	0.7545	0.0219	0.7526	0.0239	0.7557	0.0212	0.7560	0.0244	**0.7571**	0.0228
10	0.7505	0.0231	0.7522	0.0247	0.7507	0.0269	**0.7543**	0.0248	0.7528	0.0267

Table 5. NK landscapes (random neighbors): Averages over all instances and Wilcoxon rank sum tests for each pair of GA-variants, 10s absolute time limit

alg	*avg*	*sd*	time	transforms	.vs std	.vs tdb	.vs trb	.vs tdr	.vs trr
std	0.7485	0.0290	3.47s	–	–	0.6851	0.9420	1.0000	1.0000
tdb	0.7489	0.0295	3.53s	52215	0.3149	–	0.8358	0.9999	1.0000
trb	0.7492	0.0294	3.69s	49602	0.0580	0.1643	–	0.9994	1.0000
tdr	0.7502	0.0294	3.48s	54003	0.0000	0.0001	0.0006	–	0.8472
trr	0.7506	0.0298	3.79s	50718	0.0000	0.0000	0.0000	0.1529	–

evident in Table 5 where average values over all K together with pairwise error probabilities of Wilcoxon rank sum tests are presented analogously to Table 3.

5 Conclusions and Future Work

In this paper we suggested the use of a complete solution archive based on a binary trie data structure for genetic algorithms using classical binary string representations. The archive stores all solutions visited during the optimization process in a relatively compact way and provides the essential insert and find operations in time $O(l)$, i.e., independently of the number of already visited solutions. In contrast to classical objective value caching strategies, the archive further provides a new transform operation, which changes already visited candidate solutions effectively into in general similar but yet unvisited solutions. This procedure also runs in time $O(l)$. Randomized variants of this transformation procedure and the trie structure itself were proposed in order to minimize a biasing towards genes in certain positions being changed more frequently than others. From a theoretical point-of-view, a nice property is that the solution archive turns the GA into a complete optimization approach with a well-defined termination condition and bounded runtime.

Tests were performed on Royal Road functions and NK fitness landscapes. Although differences are not too large, we could observe that the use of the archive in general led to significantly better results and only moderate runtime increases. Especially the randomization of the transform operation and the trie structure also proved to be advantageous. More generally, we consider the proposed solution archive particularly promising for problems with expensive objective functions and relatively small search spaces, where solutions would otherwise be frequently revisited. There, the additional computational effort introduced by the archive becomes neglectable and the advantages can be expected to increase.

Further investigations and tests on more types of problems are necessary. In future work, we intend to study the application of trie-based solution archives in particular on hybrid GAs including e.g. local search or repair components for more complex optimization problems. This also includes problems where solutions are represented in other ways than binary strings. For them, the trie must be adapted appropriately. Constraints of a problem might further impose additional challenges on the trie and in particular its solution transform operator. For the multidimensional knapsack problem, some positive preliminary results have already been obtained [20]. Furthermore, we are currently working on archive-extended approaches for generalized network design problems where two dual representations are used at the same time. We also believe that it is promising to consider such trie-based solution archives for other metaheuristics besides GAs.

Last but not least, a particularly interesting aspect is the possibility to additionally calculate lower bounds (when considering a minimization problem) for individual subtries and to prune them (i.e., mark as completed) when this lower bound exceeds the objective value of the so far best solution. In this way, our approach becomes related to the well-known concept of branch-and-bound.

Acknowledgements

We thank Andrej Sramko, who helped in the implementation of the described concepts and did the testing as part of his master thesis [18]. This work is further supported by the Austrian Science Fund (FWF) under contract nr. P20342-N13.

References

1. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley, Reading (1985)
2. Battiti, R., Tecchiolli, G.: The reactive tabu search. ORSA Journal on Computing 6, 126–140 (1994)
3. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley, Chichester (2008)
4. Gantovnik, V.B., Anderson-Cook, C.M., Grdal, Z., Watson, L.T.: A genetic algorithm with memory for mixed discrete-continuous design optimization. Computers and Structures 81, 2003–2009 (2003)
5. Glover, F.: Tabu search - part II. ORSA Journal on Computing 2(1), 4–32 (1990)
6. Gusfield, D.: Algorithms on Strings, Trees, and Sequences. Cambridge University Press, Cambridge (1997)
7. Holland, J.: Adaptation In Natural and Artificial Systems. University of Michigan Press, Ann Arbor (1975)
8. Kauffman, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford (1993)
9. Knuth, D.E.: The Art of Computer Programming Vol. III: Sorting and Searching. Addison-Wesley, Reading (1973)
10. Kratica, J.: Improving performances of the genetic algorithm by caching. Computers and Artificial Intelligence 18(3), 271–283 (1999)
11. Louis, S.J., Li, G.: Combining robot control strategies using genetic algorithms with memory. In: Angeline, P.J., McDonnell, J.R., Reynolds, R.G., Eberhart, R. (eds.) EP 1997. LNCS, vol. 1213, pp. 431–442. Springer, Heidelberg (1997)
12. Mauldin, M.L.: Maintaining diversity in genetic search. In: AAAI, pp. 247–250 (1984)
13. Mitchell, M., Forrest, S., Holland, J.H.: The royal road for genetic algorithms: Fitness landscapes and GA performance. In: Varela, F.J., Bourgine, P. (eds.) Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, pp. 245–254. MIT Press, Cambridge (1992)
14. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes in C: The Art of Scientific Computing, 2nd edn. Cambridge University Press, Cambridge (1992)
15. Raidl, G.R., Gottlieb, J.: On the importance of phenotypic duplicate elimination in decoder-based evolutionary algorithms. In: Brave, S., Wu, A.S. (eds.) Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference, Orlando, FL, pp. 204–211 (1999)
16. Ronald, S.: Complex systems: Mechanism of adaption. In: Stonier, R., Yu, X.H. (eds.) Complex Systems: Mechanism of Adaptation, pp. 133–140. IOS Press, Amsterdam (1994)
17. Ronald, S.: Duplicate genotypes in a genetic algorithm. In: Fogel, D.B., Schwefel, H.P., Bck, T., Yao, X. (eds.) IEEE World Congress on Computational Intelligence (WCCI 1998), pp. 793–798 (1998)

18. Sramko, A.: Enhancing a genetic algorithm by a complete solution archive based on a trie data structure. Master's thesis, Vienna University of Technology, Institute of Computer Graphic s and Algorithms, Vienna, Austria (February 2009)
19. Yuen, S.Y., Chow, C.K.: A non-revisiting genetic algorithm. In: IEEE Congress on Evolutionary Computation (CEC 2007), pp. 4583–4590. IEEE Press, Los Alamitos (2007)
20. Zaubzer, S.: A complete archive genetic algorithm for the multidimensional knapsack problem. Master's thesis, Vienna University of Technology, Institute of Computer Graphics and Algorithms, Vienna, Austria (May 2008)

A New Primal-Dual Genetic Algorithm: Case Study for the Winner Determination Problem

Madalina Raschip and Cornelius Croitoru

"Al.I.Cuza" University of Iasi, Romania
{mionita,croitoru}@info.uaic.ro

Abstract. This paper presents a new evolutionary computing strategy which uses the linear programming duality information to help the search for optimum solutions of hard optimization problems. The algorithm is restarted several times when it is stuck into a local optima. At each restart, the appropriate dual space is constructed. A new population of primal individuals is generated using the information from dual solutions and is considered for next iterations. The pursued goal was not to find the best algorithm for solving winner determination problem, but to prove the method's viability using the problem as a case study. Experiments on realistic instances were performed.

1 Introduction

Evolutionary Algorithms (EAs) are powerful heuristics for optimization, based on the natural evolution paradigms [1]. Fitter individuals will result in future generations by natural selection and adaptation processes. Evolutionary Algorithms are used to tackle hard optimization problems for which classical methods are inappropriate. One of the major problem is that the algorithms could converge to suboptimal solutions. Maintaining diversity to avoid premature convergence toward local optima is an important goal inside EAs.

This paper describes a new scheme based on evolutionary computing strategies and linear programming (LP) duality. The new approach is completely different from the notion of duality in evolutionary algorithms, which is usually connected with diploidy and dominance from nature. In this context, a primal-dual genetic algorithm [2] operates on pairs of chromosomes which are primal-dual in a combinatorial way, governed by the Hamming distance.

Including techniques from operations research in metaheuristics is a popular approach leading to an efficient behavior [3]. The solutions of the relaxed problem indicate in which areas of the search space good solutions might lie. The LP-relaxed solutions are used to create promising initial solutions, to repair infeasible solutions and to guide local improvement [3]. Dual variables of the relaxed problem could also be exploited. This is the main goal of this paper. A related idea was considered in the paper [4], where a GA for the Multi-constrained Knapsack problem uses the shadow prices of the LP-relaxed solution, but inside a repairing heuristic. Ratios based on the shadow prices give the likeliness

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 252–263, 2010.

of the items to be included in a solution. In [5] a primal-dual variable neighborhood search for the simple plant location problem is presented. A primal feasible solution is obtained by a variable neighborhood decomposition search. A dual solution, resulted by exploiting the complementary slackness conditions and improved then locally is transformed into an exact solution by the sliding simplex algorithm. The dual solution is used to derive a good lower bound and to strengthen next a Branch and Bound algorithm for solving the problem.

The primal-dual method [6] from the linear programming theory was initially proposed for solving linear programs. The method has found widespread use because it leads to a general methodology for the design of approximation algorithms for NP-hard problems [7]. In the primal-dual method, a feasible primal solution is searched that satisfies the complementary slackness conditions, given a feasible dual solution. If one cannot be found, there is a way to modify the dual solution to increase the dual objective value (considering the dual objective is a maximization one). The primal-dual method for approximation algorithms considers the dual of a linear programming relaxation of the integer program. In this case, the method usually leads to dual-ascent algorithms in which dual variables are never decreased (although there are some exceptions). The method yields a solution that is within a factor of α of optimal.

Our new approach uses the duality information in order to escape from local optima. A series of returns in primal after consulting the corresponding dual space take place. The new primal solutions constructed at each return uses the dual information. The construction of the dual, respectively primal solutions for each restart obeys the complementary slackness conditions. To maintain the progress, i.e. increasing the objective value, the best primal solution from previous iteration is kept in the next one.

The method is tested for one of the major problems raised by combinatorial auctions, the winner determination problem (WDP). The problem is to select a set of winning bids that maximize the auctioneer's revenue. Exact algorithms have been proposed to solve the problem, especially based on Branch-and-Bound procedure [8], [9] and linear programming [10]. Optimal solutions have been obtained using CPLEX solver [11]. Approximative methods were also developed: a stochastic local search method [12], a hybrid simulated annealing [13] and a memetic algorithm [14]. The winner determination problem can be modeled as multidimensional knapsack problem (MDKP). Evolutionary algorithms have proved their efficiency for solving the MDKP, thus many of the methods used for MDKP can be used for WDP [4].

The paper is organized as follows. In the next section, the new general approach is presented. In section 3 the case study on the combinatorial auctions problem is discussed. The experimental settings and results are reported next. We conclude the work in section 5.

2 The Primal-Dual Genetic Approach

The new method is suited for hard linear programming problems. We know that for a given linear program, a dual linear program could be easily constructed.

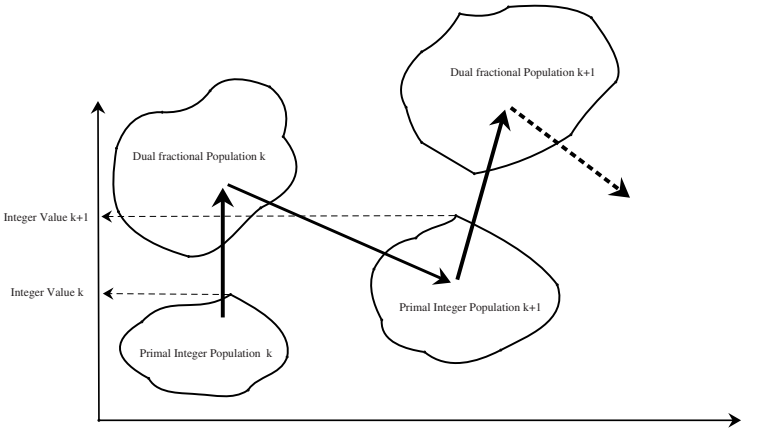


Fig. 1. The new primal-dual genetic schema

The basic idea is to help the genetic search with the dual information from the linear relaxation of the integer linear programming formulation. The steps of the procedure are described next. First, a primal genetic algorithm for solving the integer programming problem is started. Since the problem presents restrictions, the GA must use techniques from constrained optimization field for modeling and solving. When the algorithm gets trapped into a local optimum, the search is stopped.

From the last primal population, a set of dual relaxed solutions is constructed. The dual objective function is a lower bound on the value of the optimum integer solution. The dual solutions are improved using a specific algorithm.

The information gathered by the dual solutions is transmitted to the primal algorithm. The algorithm constructs a new population of primal individuals from dual solutions and restarts the search with a new improved initial configuration. The best solution from the previous iteration is copied into the new population. Starting with a better population, the chances of finding a better solution grow.

Algorithm 1. PrimalDualGA()

```
initialize primal population
while optima not found or stopping condition not met do
  while not trapped in local optima do
    selection
    apply operators
    local optimization (using best dual relaxed solution)
  end while
  construct dual relaxed solutions from primal population
  improve dual relaxed solutions
  initialize primal population from dual relaxed solutions
end while
```

A local optimization step, which makes use of the dual information takes place on all primal individuals at each iteration. The best dual solution could be used to improve the primal solutions.

The algorithm repeats these steps until a good enough solution for the problem is found. The graphical representation and the pseudocode for the scheme are given next.

3 A Case Study: Primal-Dual GA for Winner Determination

3.1 Combinatorial Auctions

Combinatorial auctions [15] allow bids on combinations of items. They present difficulties in terms of preference elicitation, how the bidder describes their preferences concisely, and winner determination. The problem we consider is the WDP, i.e. how to label the bids as winning or losing in order to maximize the auctioneer's revenue. The constraint is that each item can be allocated to at most one bidder. The problem is computationally difficult (NP-complete and inapproximable) [16].

The model. The problem is usually formalized as a packing problem. Consider there are m items for sale and a set of bidders. The identity of the bidder does not matter for the allocation algorithm, so consider there are n bids $\{B_i, i = 1, n\}$. Each bid B_i is given by a pair (S_i, p_i) where S_i is the subset of items and p_i specifies the bid for that subset, i.e. the price the bidder is willing to pay for getting this bundle of goods. The problem is to find a feasible allocation with maximum social welfare. It can be easily formalized as an integer programming problem.

$$\begin{aligned} max \ & \sum_{i=1}^{n} x_i p_i \\ & s.t. \sum_{i|j \in S_i} x_i \leq 1, \forall j = 1..m \\ & x_i \in \{0, 1\}, \forall i = 1..n \end{aligned}$$

We assume that the bids are given in OR* language (OR with phantom items, i.e. dummy items) [8]. In an optimum solution, if $x_i = 1$ then B_i is a winner bid.

We consider the following LP relaxation of WDP:

$$\begin{aligned} max \ & \sum_{i=1}^{n} x_i p_i \\ & s.t. \sum_{i|j \in S_i} x_i \leq 1, \forall j = 1..m \\ & x_i \geq 0, \forall i = 1..n \end{aligned}$$

In this formulation, the variables x_i have real values.

The dual of the LP relaxation is:

$$\begin{aligned} min \ & \sum_{j=1}^{m} y_j \\ & s.t. \sum_{j \in S_i} y_j \geq p_i, \forall i = 1..n \\ & y_j \geq 0, \forall j = 1..m \end{aligned}$$

The problem is to find some individual item prices to minimize the total price over all items under some restrictions (the sum of the item prices from each bid must be greater than the bid price). The dual variables, often called shadow prices, serve as approximations of how valuable the goods are.

LP duality. Linear Programming is worth using for finding the optimal allocation in (multi-unit) combinatorial auctions [11], [17]. Approximation algorithms may benefit from the construction of dual information. Individual item prices, which correspond to a dual solution of the relaxed problem could be useful to bidders in multiple round auctions. They can be used as approximate marginal values that enable bidders to bid more efficiently in subsequent rounds [18].

Let x^* and y^* be the optimal solutions of the primal, respectively dual problem. The following comes from the LP-duality theory:

- Strong-duality
  x^* and y^* are a pair of optimal solutions if and only if they are feasible and $\sum_{i=1}^{n} x_i^* p_i = \sum_{j=1}^{m} y_j^*$.
- Complementary slackness
  For each i such that $x_i^* > 0$, the i'th constraint of the dual problem is saturated, i.e. $\sum_{j \in S_i} y_j^* = p_i$.
  For each j such that $y_j^* > 0$, the j'th constraint of the primal problem is saturated, i.e. $\sum_{i \mid j \in S_i} x_i^* = 1$.

3.2 The Primal-Dual Genetic Approach for WDP

Next, we detail the algorithms for the primal and the dual problem, and also for the transition steps between them.

Primal algorithm. Because the considered problem is a constrained one, we must include into the genetic algorithm a method to handle the constraints.

We have used an order based representation: a solution is encoded by a permutation of length n, the number of bids from the combinatorial auction. The permutation is decoded using a first-fit heuristic. Initially, all x_i are equal with 0, and then set to 1 in the order they appear in permutation: if the current bid together with the previous set bids satisfies the restrictions, it is assigned as winning (the corresponding x_i is set to 1). The representation assures that any chromosome will express a feasible solution. A disadvantage is that the same

solution can be encoded by multiple permutations, the search space becomes larger this way. Phenotypic duplicate elimination is beneficial here [19].

The fitness function is equal with the auctioneer's revenue, i.e. the sum of prices of the selected bids. A generational model was considered. Fitter solutions survive by the selection mechanism; the fitness proportionate selection was used by the algorithm. The genetic operators used are: the uniform order based crossover and inversion. In the uniform order based crossover [20] each position of the first parent is transferred to the child with some probability. The remaining gaps are completed from the second parent in the order they appear there. The inversion operator reverses a part of the solution string.

Next a local optimization step is applied on the chromosomes of the population. At each step, the method selects greedily an unsatisfied bid to include in solution. The bid with the highest value of the shadow surplus [9], [4] is taken. The shadow surplus of a bid i is equal with $\frac{p_i}{\sum_{j \in S_i} y_j}$. The dual prices of the best dual individual from the previous step are considered. The bid is placed on the position of the first bid in conflict with. The assignment of bids to solution is renewed. If the value of the new chromosome is better, the algorithm continues; otherwise the method stops.

The algorithm uses the elitism mechanism: it keeps the best solution in population.

The next step consists in constructing the corresponding dual solutions from local optima solutions of the primal problem.

The construction of a dual solution from a primal one. In the paper [21] a technique to construct a dual solution for the LP relaxation of the generalized set packing problem is specified. Each element can have multiple copies and there is an upper bound on the number of copies in the packing. The primal solution is obtained using a greedy algorithm. We present next the method adapted for our model.

Let $\mathcal{S} = \{B_i = (S_i, p_i), i = 1, \ldots, n\}$ be the set of bids and $\mathcal{P}$ a primal feasible solution ($S_i \in \mathcal{P}$ if and only if $x_i = 1$). The item j is *saturated* w.r.t. $\mathcal{P}$ if there is a set $S_k \in \mathcal{S} \setminus \mathcal{P}$ with $j \in S_k$ such that $|S|_{S \in \mathcal{P}, j \in S} > 0$. Denote by $SAT_{\mathcal{P}} = \{j \in \{1, \ldots, m\} | j \textit{ is saturated w.r.t. } \mathcal{P}\}$ the set of saturated items. Observe that for each $S_k \in \mathcal{S} \setminus \mathcal{P}$, there is a $j \in SAT_{\mathcal{P}}$ called a *witness*: when S_k was considered to be added at the (partial) solution $\mathcal{P}$, element j violates the restriction. For each set $S_k \in \mathcal{S} \setminus \mathcal{P}$, we keep in $SAT_{\mathcal{P}}$ only one (arbitrary) witness.

The feasible dual solution is represented by $\sqrt{m}y$, where y is a convex combination of two fractional solutions, y^1 and y^2. The main idea is to use the second solution as a back-up: whenever on some set $S_k \in \mathcal{S}$, y^1 is not high enough to satisfy the primal restriction, y^2 is sufficiently high.

$$y_j^1 = \frac{\sigma}{m}, \ \sigma = \sum_{S_k \in \mathcal{P}} p_k, \quad \forall j$$

y^2 is defined in a sequence of steps. Initially, for every item j, $y_j^2 = 0$. For each bid B_k which was selected in solution, $y_j^2 = y_j^2 + \Delta_j^{S_k}$, for all $j \in \mathcal{P}(S_k)$.

$$\mathcal{P}(S_k) = \begin{cases} S_k \cap SAT_{\mathcal{P}}, & \text{if } S_k \cap SAT_{\mathcal{P}} \neq \emptyset \\ S_k, & \text{otherwise} \end{cases}$$

$$\Delta_j^{S_k} = \frac{p_k}{|\mathcal{P}(S_k)|}, \ for\ j \in \mathcal{P}(S_k).$$

Notice that, for $j \in S_k \setminus SAT_{\mathcal{P}}$ the value of y_j^2 is not updated and remains zero.

By weak duality, $\sqrt{m}\sum_{j=1}^{m} y_j$ is an upperbound on the value of the optimal integral solution [21].

The dual heuristic. The next step in the approach is to improve the dual solutions. We have used a simple local optimization procedure to improve the dual solutions constructed from primal optima solutions. All items which did not appear in any bids have zero prices. The price for each item is decreased with a value such that the restrictions also stand. The value is equal with a percent from the smallest difference of the sum of item prices and the bid price, divided by the number of items (computed only for the bids in which the item appears).

By the weak duality theorem, the dual solution of the LP relaxation gives an upper-bound to the WDP.

From the optimized dual set of solutions the method construct an initial set of primal individuals. They will form the initial population of the primal problem.

The construction of a primal solution from a dual one. After returning from the dual algorithm, we construct the initial set of primal individuals based on the dual set of solutions. The normalized shadow surplus bid ordering heuristic was used for this purpose. The differences between the sum of item prices and the bid price (eventually divided by log of sum of item prices) are ordered. This order represents a feasible primal solution.

The steps are repeated for a number of times (or until the primal and dual bests have the same value). The implementation of the method is available online, at the following address `http://profs.info.uaic.ro/~mionita/pdga-ca/`.

4 Experiments

To measure the performance of the algorithm, test problems were generated using the CATS [22]. Each generator models a realistic scenario. Problems from each of the main distributions were generated: arbitrary, matching, paths, regions and scheduling.

Two categories of experiments were made: first, instances with a variable number of bids and items were generated, and second, instances with a larger number of bids were used. For the first category, the number of items ranges from 40 to 400 and the number of bids ranges from 50 to 2000. In the second category, only the paths distribution was considered with 10000 and 20000 bids (and 256 items). Ten problem instances of each distribution were generated.

For small instances we used the mixed integer linear programming solver [23] to compute the exact solution. For problems where the solver could not give a result, we have used the approximation algorithm from literature [24] (ALPH). The algorithms was run with the approximation error parameter ϵ equal with 1%.

The new approach, the primal-dual genetic algorithm (PDGA) was compared with the ALPH algorithm and with the stochastic local search algorithm, Casanova from [12]. Two other genetic algorithms were used for comparison:

- a simple genetic algorithm (sGA) with the same settings as PDGA algorithm, without any restarting, and
- a genetic algorithm (rrGA) similar to PDGA, except that when restarting a new random population of individuals is generated and the local optimization step is excluded.

All genetic algorithms were executed ten times for each problem instance. The genetic algorithms work with a population of 50 individuals. They are restarted if the best value does not change after 75 iterations or after the termination of 250 iterations. The crossover probability is equal with 0.6 and the gene mutation probability is 0.2.

The experimental results are given in Table 1 and Table 2. The experiments consider five restarts of the rrGA and PDGA algorithms. The ALPH algorithm was run with the parameter ϵ equal with 20% (the same value as in the experiments from [24]). For the Casanova algorithm we have considered the walk probability equal with 0.5 and the novelty probability 0.15. The maximum number of steps from Casanova was equal with the product of the number of individuals and the number of iterations from the genetic algorithms. To compare the results we used first the gap from optimum (the difference between optimum value and the value of the objective function for the solution found, divided by optimum value) as measure.

Table 1. The average gap (in percents) for CATS instances with 'varsize' bids and for paths instances with bigger number of bids

Distribution	ALPH	Casanova	sGA	rrGA	PDGA
arbitrary	2.3	10.5	10.2	7.3	7.9
matching	0.3	4.8	23.9	21.2	4.8
paths	0.3	11.7	9.7	8.3	7
regions	-1	11.7	7	4.5	4.6
scheduling	0.2	2.2	2	1	0.7
paths (10000,256)	0.3	20.1	12.7	11.1	10.6
paths (20000,256)	0.2	21.2	11	10.2	8.8

The best solutions are provided by the ALPH algorithm (note that ALPH is an algorithm specially constructed for WDP). The Primal-Dual Genetic Algorithm outperforms obviously the simple sGA on all instances. The PDGA provides better results than rrGA for almost all distributions (exception is the arbitrary data set); the results for the regions distribution for the two algorithms are very close to each other. For the match distribution, the differences in values between the PDGA algorithm and the simple versions sGA and rrGA are bigger.

The same relation is found in the experiments with greater number of bids: ALPH remains the best algorithm and the PDGA improves the simple versions of genetic algorithms. The CATS instances with bigger number of bids prove to be not more difficult than the smaller instances.

The PDGA algorithm gives better results than Casanova for all distributions. For larger instances, the solution quality of PDGA is clearly superior to the one returned by Casanova.

The mean and the standard deviation for ten instances of the matching distribution for all genetic algorithms are listed in Table 2. The best values are the ones of the new method. The small values of the standard deviation of the PDGA show that the algorithm is stable. The algorithm finds good solutions more consistently.

Table 2. The mean and the standard deviation for ten instances of the matching data set

Instance	optimal	sGA	rrGA	PDGA
		mean (stdev)	mean (stdev)	mean (stdev)
1	556.8	429.82 (5.42)	433.07 (2.22)	525.06 (3.08)
2	369.33	309.56 (7.58)	315.54 (2.44)	359.76 (1.7)
3	281.59	190.04 (5.82)	201.68 (3.22)	266.59 (2.74)
4	485.66	359.9 (7.21)	367.67 (5.32)	455.62 (1.67)
5	223.35	189.5 (4.11)	193.18 (1.87)	216.7 (1.71)
6	156.55	123.87 (5.33)	136.78 (3.23)	156.51 (0.06)
7	421.78	343.03 (5.29)	349.21 (4.57)	406.83 (3.41)
8	483.02	336.04 (8.59)	350.16 (7.61)	446.92 (4.61)
9	468.72	331.19 (5.9)	341.82 (4.29)	427.09 (3.29)
10	293.82	211.31 (8.7)	220.17 (3.48)	278.11 (4.24)

Another measure of efficiency was also considered; the average number of fitness evaluations for rrGA and PDGA algorithms is represented in Figure 2. Except for the paths distribution, the PDGA algorithm uses a smaller number of evaluations than the simple version rrGA.

We examine next the impact of the number of restarts in the proposed algorithm. As we can expect, both algorithms rrGA and PDGA give better solutions when the number of restarts increases. This is a consequence of the natural progress of the genetic algorithm. When restarting, the new population receives the previous best individual. The best value remains the same or increases. The Figure 3 a) shows the results for the rrGA and PDGA algorithms on the paths data set where the number of restarts ranges from 0 to 20. The results are better for the PDGA algorithm because the PDGA takes advantage also of the dual information provided by the dual solutions. A balance between the solution quality and the time resources must be considered.

Next we show how many times the best solution increases on restarts, as a percent from the total number of restarts. Figure 3 b) shows the results for

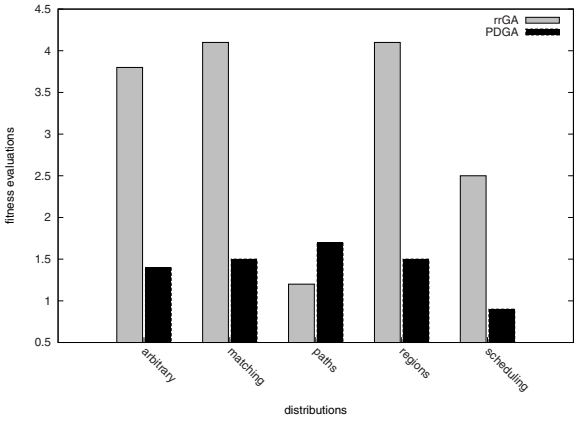
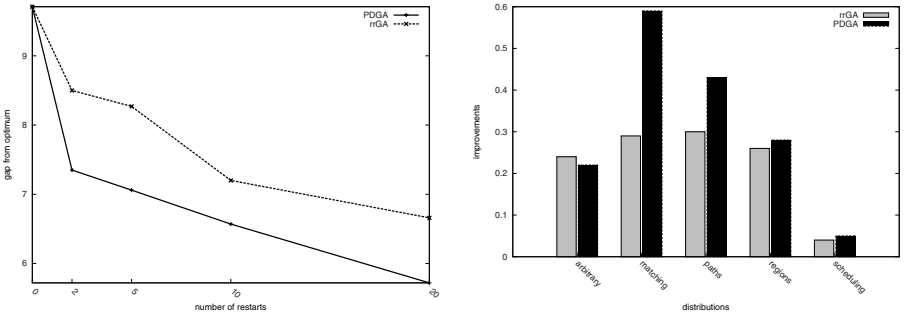


Fig. 2. The fitness evaluations for CATS instances with 'varsize' bids (the values are of 10^6 order)



a) b)

Fig. 3. a)A comparison between the PDGA and rrGA regarding the number of restarts b)The percent of improvements for CATS instances with 'varsize' bids

distributions with a variable number of bids; the algorithms are restarted five times. Less than half of the times a restart increases the best value. Only the matching distribution surpasses the 0.5 value.

From the experimental results we can conclude that the PDGA method takes benefits from dual solutions. The scope was not to develop the best method for solving winner determination problem, but to prove the effectiveness of the PDGA method.

5 Conclusion

A new hybrid approach based on evolutionary computation and LP-duality for solving hard integer linear programming problems is proposed. The information

transmitted by the dual solutions helps the primal genetic algorithm to get out from local optima. The approach is applied for solving a hard real-world problem from the combinatorial auction field, the winner determination. The method is evaluated on several realistic instances and compared with more simple genetic algorithms, and with algorithms among the best ones from literature. The obtained results are encouraging and show the feasibility of the proposed method. Future experiments on different test sets from the same problem and for other combinatorial optimization problems are needed.

References

1. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Kluwer Academic Publishers, Boston (1989)
2. Yang, S.: PDGA: the Primal-Dual Genetic Algorithm. In: Abraham, A., Koppen, M., Franke, K. (eds.) Design and Application of Hybrid Intelligent Systems, pp. 214–223. IOS Press, Amsterdam (2003)
3. Raidl, G., Puchinger, J.: Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In: Blum, C., Blesa Aguilera, M.J., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics, An Emerging Approach to Optimization, Studies in Computational Intelligence, vol.114, pp. 31–62 (2008)
4. Pfeiffer, J., Rothlauf, F.: Analysis of Greedy Heuristics and Weight-Coded EAs for Multidimensional Knapsack Problems and Multi-Unit Combinatorial Auctions. In: Proceedings of the 9th annual Conference on Genetic and Evolutionary Computation, p.1529 (2007)
5. Hansen, P., Brimberg, J., Mladenović, N., Urosević, D.: Primal-dual variable neighbourhood search for the simple plant location problem. INFORMS Journal on Computing 19(4), 552–564 (2007)
6. Dantzig, G.B., Ford, L.R., Fulkerson, D.R.: A primal-dual algorithm for linear programs. In: Kuhn, H.W., Tucker, A.W. (eds.) Linear Inequalities and Related Systems, pp. 171–181. Princeton University Press, Princeton (1956)
7. Williamson, D.P.: The Primal-Dual Method for Approximation Algorithms. Mathematical Programming, Series B 91(3), 447–478 (2002)
8. Fujishima, Y., Leyton-Brown, K., Shoham, Y.: Taming the computational complexity of combinatorial auctions: optimal and approximate approaches. In: Sixteenth international joint conference on artificial intelligence, pp. 48–53 (1999)
9. Sandholm, T., Suri, S., Gilpin, A., Levine, D.: CABoB: a fast optimal algorithm for combinatorial auctions. In: Proceedings of the international joint conferences on artificial intelligence, pp. 1102–1108 (2001)
10. Nisan, N.: Bidding and Allocation in Combinatorial Auctions. In: Proceedings of ACM conference on electronic commerce EC, pp. 1–12 (2000)
11. Andersson, A., Tenhunen, M., Ygge, F.: Integer programming for combinatorial auction winner determination. In: 4th Internationl Conference on Multiagent Systems (2000)
12. Hoos, H.H., Boutilier, C.: Solving combinatorial auctions using stochastic local search. In: Proceedings of the 17th national conference on artificial intelligence, pp. 22–29 (2000)
13. Guo, Y., Lim, A., Rodrigues, B., Zhu, Y.: Heuristics for a bidding problem. Computers and Operations Research 33(8), 2179–2188 (2006)

14. Boughaci, D., Benhamou, B., Drias, H.: A memetic algorithm for the optimal winner determination problem. Soft Computing 13(8-9), 905–917 (2009)
15. Vohra, R., de Vries, S.: Combinatorial auctions: A survey. INFORMS Journals of Computing 15(3), 284–309 (2003)
16. Rothkopf, M.H., Pekec, A., Harstad, R.M.: Computationally manageable combinatorial auctions. Management Science 44(8), 1131–1147 (1998)
17. Gonen, R., Lehmann, D.: Linear Programming helps solving large multi-unit combinatorial auctions. In: Proceedings of the Electronic Market Design Workshop (2001)
18. DeMartini, C., Kwasnica, A.M., Ledyard, O., Porter, D.: A New and Improved Design for Multi-Object Iterative Auctions. Management Science 51(3), 419–434 (2005)
19. Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In: Proceedings of the 2000 ACM symposium on Applied computing, vol. (1), pp. 408–414 (2000)
20. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold (1991)
21. Krysta, P.: Greedy Approximation via Duality for Packing, Combinatorial Auctions and Routing. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 615–627. Springer, Heidelberg (2005)
22. Leyton-Brown, K., Pearson, M., Shoham, Y.: Towards a Universal Test Suite for Combinatorial Auction Algorithms. In: Proceedings of the 2nd ACM conference on Electronic commerce, pp. 66–76 (2000)
23. Berkelaar, M.: lp_solve - version 5.5. Eindhoven University of Technology, http://sourceforge.net/projects/lpsolve/
24. Zurel, E., Nisan, N.: An Efficient Approximate Allocation Algorithm for Combinatorial Auctions. In: Proceedings of the 3rd ACM conference on Electronic commerce, pp. 125–136 (2001)

Local Search Algorithms on Graphics Processing Units. A Case Study: The Permutation Perceptron Problem

Thé Van Luong, Nouredine Melab, and El-Ghazali Talbi

INRIA Dolphin Project / Opac LIFL CNRS
40 avenue Halley, 59650 Villeneuve d'Ascq Cedex, France
The-Van.Luong@inria.fr, {Nouredine.Melab,El-Ghazali.Talbi}@lifl.fr

Abstract. Optimization problems are more and more complex and their resource requirements are ever increasing. Although metaheuristics allow to significantly reduce the computational complexity of the search process, the latter remains time-consuming for many problems in diverse domains of application. As a result, the use of GPU has been recently revealed as an efficient way to speed up the search. In this paper, we provide a new methodology to design and implement efficiently local search methods on GPU. The work has been experimented on the permuted perceptron problem and the experimental results show that the approach is very efficient especially for large problem instances.

Keywords: GPU-based metaheuristics, local search algorithms on GPU.

1 Introduction

Nowadays, optimization problems become increasingly large and complex, forcing the use of parallel computing for their efficient and effective resolution. Indeed, although near-optimal algorithms such as local search (LS) methods allow to reduce the temporal complexity of their resolution, they are unsatisfactory to tackle large problems. Therefore, parallel computing has recently undergone a significant evolution with the emergence of new high performance computing environments including accelerators such as GPUs.

Recently, the use of graphics processors has been extended to general application domains such as computational science [1]. Indeed, GPUs are very efficient at manipulating computer graphics, and their parallel structure makes them more efficient than general-purpose CPUs for a range of complex algorithms. This is why it would be very interesting to exploit this huge capacity of computing to implement parallel metaheuristics. However, there only exists few research works related to evolutionary algorithms on GPU [2,3,4]. Indeed, the design and implementation of parallel optimization methods raise several issues related to the characteristics of these methods and those of the new hardware execution environments at the same time.

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 264–275, 2010.

Several scientific challenges mainly related to the hierarchical memory management on GPU have to be considered: the efficient distribution of data processing between CPU and GPU, the optimization of data transfer between the different memories, the capacity constraints of these memories, etc. The main objective of this paper is to deal with such issues for the re-design of parallel LS models to allow solving of large scale optimization problems on GPU architectures. We propose a new general methodology for building efficient parallel LS methods on GPU. This methodology is based on a three-level decomposition of the GPU hierarchy allowing a clear separation between generic and problem-dependent LS features.

To be validated the work has been experimented on the permuted perceptron problem (PPP) introduced by Pointcheval [5]. The problem is a cryptographic identification scheme based on NP-complete problems, which seems to be well suited for resource constrained devices such as smart cards. The proposed work has been experimented using three GPU configurations with different performance capabilities in terms of threads that can be created simultaneously.

The remainder of the paper is organized as follows: In Section 2, the characteristics of the GPU architecture are described according to the three-level decomposition. Section 3 presents generic concepts for designing parallel LS methods on GPU (high-level). In Section 4, efficient mappings between state-of-the-art LS structures and NVIDIA CUDA model are performed (intermediate-level). A depth look on memory management in CUDA adapted to LS heuristics is depicted in Section 5 (low-level). Section 6 reports the performance results obtained for the PPP mentioned above. Finally, a discussion and some conclusions of this work are drawn in Section 7.

2 Graphics Processing Units and Three-Level Decomposition

Driven by the demand for high-definition 3D graphics, GPUs have evolved into a highly parallel, multithreaded and manycore environment. Since more transistors are devoted to data processing rather than data caching and flow control, GPU is specialized for compute-intensive and highly parallel computation. A complete review of GPU architecture can be found in [6].

The adaptation of LS algorithms on GPU requires to take into account at the same time the characteristics and underlined issues of the GPU architecture and the LS parallel models. In this section, we propose a three-level decomposition of the GPU adapted to the popular parallel iteration-level model [7] (generation and evaluation of the neighborhood in parallel) allowing a clear separation of the GPU memory hierarchical management concepts (Fig. 1). The different aspects of the three-level decomposition model will be discussed throughout the next sections.

In the high-level layer, task distribution is clearly defined: the CPU manages the whole sequential LS process and the GPU is dedicated to the parallel evaluation of solutions at the other levels. The intermediate-level layer focuses on

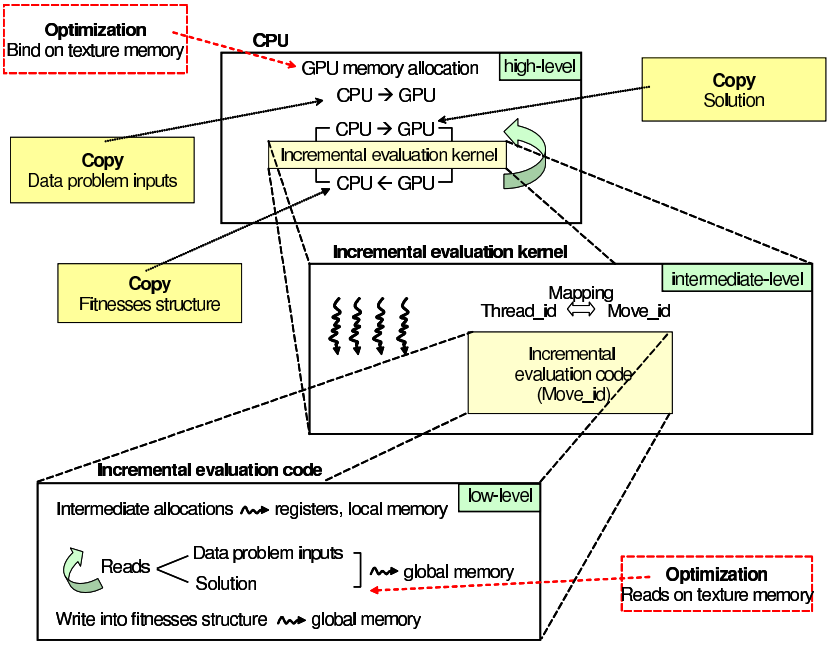


Fig. 1. Three-level decomposition

the generation and partitioning of the LS neighborhood on GPU. Afterwards, GPU memory management of the evaluation function computation is done at low-level.

2.1 High-Level Layer: General GPU Model

This level describes common GPU concepts which are language-independent. In general-purpose computing on graphics processing units, the CPU is considered as a host and the GPU is used as a device coprocessor. This way, each GPU has its own memory and processing elements that are separate from the host computer. Data must be transferred between the memory space of the host and the memory of GPU during the execution of programs. In LS algorithms, the types of data which are manipulated are the data inputs of the tackled problem and the solution representation.

Each processor device on GPU supports the single program multiple data (SPMD) model, i.e. multiple autonomous processors simultaneously execute the same program on different data. For achieving this, the concept of kernel is defined. The kernel is a function callable from the host and executed on the specified device simultaneously by several processors in parallel. Regarding the iteration-level parallel model, generation and evaluation of neighboring candidates are done in parallel. Therefore, a kernel on GPU can be associated with these two steps.

Memory transfer from the CPU to the device memory is a synchronous operation which is time consuming. In the case of LS methods, memory copying operations from CPU to GPU are essentially the solution duplication operations which generate the neighborhood. Afterwards, the kernel representing the generation and evaluation of the neighborhood is processed at both intermediate-level and low-level. Regarding transfers from GPU to CPU, the results of the evaluation function (fitnesses) of each candidate solution of the neighborhood are stored in an array structure.

2.2 Intermediate-Level Layer: CUDA Threading Model

The intermediate-level layer focuses on the neighborhood generation on GPU. This kernel handling is dependent of the general-purpose language. CUDA was chosen because the toolkit introduces a model of threads which provides an easy abstraction for SIMD architecture [8]. A thread on GPU can be seen as an element of the data to be processed and changing the context between two threads is not a costly operation. Therefore, GPU threads management is clearly identified as the main task of the generation step of LS neighborhood.

Regarding their spatial organization, threads are organized within so called thread blocks. A kernel is executed by multiple equally threaded blocks. Blocks can be organized into a one-dimensional or two-dimensional grid of thread blocks, and threads inside a block are grouped in a similar way. All the threads belonging to the same thread block will be assigned as a group to a single multiprocessor. Thus, a unique *id* can be deduced for each thread to perform computation on different data. Regarding LS algorithms, a move which represents a particular neighbor candidate solution can also be associated with a unique *id*. However, according to the solution representation of the problem, finding a corresponding *id* for each move is not straightforward.

2.3 Low-Level Layer: Kernel Memory Management

The low-level layer focuses on the specific part of the evaluation function. As stated before, each GPU thread executes the same kernel i.e. each candidate solution of the neighborhood executes the same evaluation function. From a hardware point of view, since multiprocessors are used according to the SPMD model, threads share the same code and have access to different memory areas.

Communication between the CPU host and its device is done through the global memory. For LS algorithms, more exactly for the evaluation function, the global memory stores the data input of problems and their solution representation. Since this memory is not cached and its access is slow, one needs to minimize accesses to global memory (read/write operations). Graphics cards provide also read-only texture memory to accelerate operations such as 2D mapping. In the case of LS algorithms, binding texture on global memory can provide an alternative optimization. Registers among streaming processors are partitioned among the threads running on it, they constitute fast access memory. In the evaluation function kernel code, each declared variable is automatically put into registers.

Local memory is a memory abstraction and is not an actual hardware component. Complex structures such as declared array will reside in local memory.

The memory management in the low-level layer is problem-specific. A clear understanding of the characteristics described above is required to provide an efficient implementation of the evaluation function. According to the SPMD model, the same code is executed by all the neighbors in parallel and the resulting fitnesses must be stored into the fitnesses structure (global memory) previously mentioned.

3 Design of Parallel Local Search Algorithms on GPU

In this section, the focus is on the re-design of the iteration-level parallel model. This model fits well with the high-level layer since parallel LS concepts are generic. Designing parallel LS model is a great challenge as nowadays there is no generic GPU-based LS algorithms to the best of our knowledge.

3.1 A General Model for LS Algorithms

According to the SPMD model, multiple autonomous processors simultaneously execute the same program at independent points. Therefore, the mapping at the high-level layer between the LS iteration-level parallel model and the GPU model becomes quiet natural.

First, the CPU sends the number of expected running threads to the GPU, then candidate neighbors are generated and evaluated on GPU (at intermediate-level and low-level), and finally newly evaluated solutions are returned back to the host. This model can be seen as a cooperative model where the GPU is used as a coprocessor in a synchronous manner. The resource-consuming part i.e. the generation and evaluation kernel, is calculated by the GPU and the rest is handled by the CPU.

3.2 The Proposed GPU-Based Algorithm

Adapting traditional LS methods to GPU is not a straightforward task because hierarchical memory management on GPU has to be handled. We propose (see algorithm 1) a methodology to adapt LS methods on GPU in a generic way.

First of all, at initialization stage, memory allocations on GPU are made: data inputs and candidate solution of the problem must be allocated (lines 4 and 5). Since GPUs require massive computations with predictable memory accesses, a structure has to be allocated for storing all the neighborhood fitnesses at different addresses (line 6). Additional solution structures which are problem-dependent can also be allocated (line 7). Second, all the allocated structures have to be copied on the GPU (lines 8 to 10). Since problem data inputs are a read-only structure, their associated memory is copied only once during all the execution. Third, comes the parallel iteration-level, in which each neighboring solution is generated (intermediate-level), evaluated (low-level) and copied into

Algorithm 1. Local Search Template on GPU

```
 1: Choose an initial solution
 2: Evaluate the solution
 3: Specific LS initializations
 4: Allocate problem data inputs on GPU device memory
 5: Allocate a solution on GPU device memory
 6: Allocate a neighborhood fitnesses structure on GPU device memory
 7: Allocate additional solution structures on GPU device memory
 8: Copy problem data inputs on GPU device memory
 9: Copy the solution on GPU device memory
10: Copy additional solution structures on GPU device memory
11: repeat
12:    for each generated neighbor in parallel on GPU do
13:       Incremental evaluation of the candidate solution
14:       Insert the resulting fitness into the neighborhood fitnesses structure
15:    end for
16:    Copy neighborhood fitnesses structure on CPU host memory
17:    Specific LS solution selection strategy on the neighborhood fitnesses structure
18:    Specific LS post-treatment
19:    Copy the chosen solution on GPU device memory
20:    Copy additional solution structures on GPU device memory
21: until a stopping criterion satisfied
```

the neighborhood fitnesses structure (from lines 12 to 15). Fourth, since the order in which candidate neighbors are evaluated is undefined, the neighborhood fitnesses structure has to be copied to the host CPU (line 16). Then a specific LS solution selection strategy is applied to this structure (line 17) on CPU. Finally, after a new candidate has been selected, this latter and its additional structures are copied to the GPU (lines 19 and 20). The process is repeated until a stopping criterion is satisfied.

4 Efficient Mappings of Local Search Structures on GPU

The neighborhood structures play a crucial role in the performance of LS methods and are problem-dependent. In this section, a focus is made on the neighborhood generation in the intermediate-level layer.

The challenging issue of this level is to find efficient mappings between a thread *id* and a particular neighbor. Indeed, on the one hand, the thread *id* is represented by a single index. On the other hand, the move representation of a neighbor varies according to the neighborhood. In the following, we provide a methodology to deal with different structures of the literature.

4.1 Binary Representation

In binary representation, a solution is coded as a vector of bits. The neighborhood representation for binary problems is based on the Hamming distance where a

given solution is obtained by flipping one bit of the solution (for a Hamming distance of one).

A mapping between LS neighborhood encoding and GPU threads is quiet trivial. Indeed, on the one hand, for a binary vector of size n, the size of the neighborhood is exactly n. On the other hand, threads are provided with a unique *id*. That way, the kernel associated to the generation and evaluation steps is launched with n threads (each neighbor is associated to a single thread), and the size of the neighborhood fitnesses structure allocated on GPU is n. As a result, a $\mathbb{N} \rightarrow \mathbb{N}$ mapping is straightforward.

4.2 Discrete Vector Representation

Discrete vector representation is an extension of binary encoding using a given alphabet Σ. In this representation, each variable takes its value over the alphabet Σ. Assume that the cardinality of the alphabet Σ is k, the size of the neighborhood is $(k-1) \times n$ for a discrete vector of size n.

Let *id* be the identity of the thread corresponding to a given candidate solution of the neighborhood. Compared to the initial solution which allowed to generate the neighborhood, $id/(k-1)$ represents the position which differs from the initial solution and $id\%(k-1)$ is the available value from the ordered alphabet Σ (both using zero-index based numbering).

As a consequence, a $\mathbb{N} \rightarrow \mathbb{N}$ mapping is possible. $(k-1) \times n$ threads execute the generation and evaluation kernel, and a neighborhood fitnesses structure of size $(k-1) \times n$ has to be provided.

4.3 Permutation Representation

Building a neighborhood by pairwise exchange operations is a standard way for permutation problems. For a permutation of size n, the size of the neighborhood is $\frac{n \times (n-1)}{2}$.

Unlike the previous representations, for permutation encoding a mapping between a neighbor and a GPU thread is not straightforward. Indeed, on the one hand, a neighbor is composed by two element indexes (a swap in a permutation). On the other hand, threads are identified by a unique *id*. As a result, a $\mathbb{N} \rightarrow \mathbb{N} \times \mathbb{N}$ mapping has to be considered to transform one index into two ones. In a similar way, a $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ mapping is required to transform two indexes into one.

Proposition 1. ***Two-to-one index transformation***
Given i and j the indexes of two elements to be exchanged in the permutation representation, the corresponding index $f(i,j)$ in the neighborhood representation is equal to $i \times (n-1) + (j-1) - \frac{i \times (i+1)}{2}$, where n is the permutation size.

Proposition 2. ***One-to-two index transformation***
Given $f(i,j)$ the index of the element in the neighborhood representation, the corresponding index i is equal to $n - 2 - \lfloor \frac{\sqrt{8 \times (m - f(i,j) - 1) + 1} - 1}{2} \rfloor$ and j is equal

to $f(i,j) - i \times (n-1) + \frac{i \times (i+1)}{2} + 1$ *in the permutation representation, where* n *is the permutation size and* m *the neighborhood size.*

The proofs of these two index transformations can be found in [9]. The generation and evaluation kernel is executed by $\frac{n \times (n-1)}{2}$ threads, and the size of the neighborhood fitnesses structure is $\frac{n \times (n-1)}{2}$. Notice that for binary problem encodings, the mapping of a neighborhood based on a Hamming distance of two can be done in a similar manner.

5 Memory Management of Local Search Algorithms on GPU

Task repartition between CPU and GPU and efficient thread mappings in parallel LS heuristics have been proposed on both high-level and intermediate-level layers. In this section, the focus is on the memory management in the low-level layer. Understanding the GPU memory organization and issues is useful to provide an efficient implementation of parallel LS heuristics.

5.1 Memory Coalescing Issues

In CUDA, each block of threads is split into SIMD groups of threads called *warps*. At any clock cycle, each processor of the multiprocessor selects a half-warp (16 threads) that is ready to execute the same instruction on different data. Global memory is conceptually organized into a sequence of 128-byte segments. The number of memory transactions performed for a half-warp will be the number of segments having the same addresses than those used by that half-warp. Fig. 2 illustrates an example of the low-level layer for a simple vector addition.

For more efficiency, global memory accesses must be coalesced, which means that a memory request performed by consecutive threads in a half-warp is associated with precisely one segment. The requirement is that threads of the same

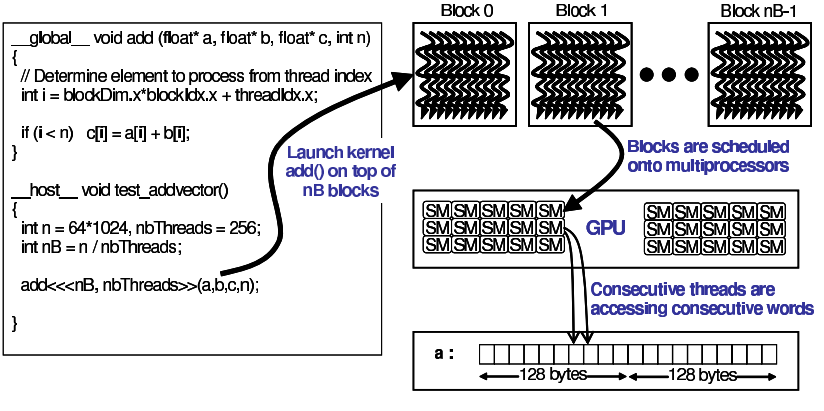


Fig. 2. An example of kernel execution for vector addition

warp must read global memory in an ordered pattern. If per-thread memory accesses for a single half-warp constitute a contiguous range of addresses, accesses will be coalesced into a single memory transaction. In the example of vector addition, memory accesses to the vectors a and b are fully coalesced, since threads with consecutive thread indices access contiguous words.

Otherwise, accessing scattered locations results in memory divergence and requires the processor to perform one memory transaction per thread. The performance penalty for non-coalesced memory accesses varies according to the size of the data structure. Regarding LS evaluation kernels, coalescing is difficult when global memory access has a data-dependent unstructured pattern. As a result, non-coalesced memory accesses imply many memory transactions and it can lead to a significant performance decrease for LS methods.

Notice that in the latest cards (200-series), due to the relaxation of the coalescing rules, applications developed in CUDA get better global memory performance.

5.2 Texture Memory

Optimizing the performance of CUDA applications often involves optimizing data accesses which includes the appropriate use of the various CUDA memory spaces. The use of texture memory is a solution for reducing memory transactions due to non-coalesced accesses. Texture memory provides a surprising aggregation of capabilities including the ability to cache global memory. Indeed, each texture unit has some internal memory that buffers data from global memory. Therefore, texture memory can be seen as a relaxed mechanism for the thread processors to access global memory because the coalescing requirements do not apply to texture memory accesses. The use of texture memory is well adapted for LS algorithms for the following reasons:

- Data accesses are frequent in the computation of LS evaluation methods. Then, using texture memory can provide a high performance improvement by reducing the number of memory transactions.
- Texture memory is a read-only memory i.e. no writing operations can be performed on it. This memory is adapted to LS algorithms since the problem data and the solution representation are also read-only values.
- Minimizing the number of times that data goes through cache can increase the efficiency of algorithms. In most of optimization problems, problem inputs do not often require a large amount of allocated space memory. As a consequence, these structures can take advantage of the 8KB cache per multiprocessor of texture units.
- Cached texture data is laid out to give best performance for 1D/2D access patterns. The best performance will be achieved when the threads of a warp read locations that are close together from a spatial locality perspective. Since optimization problem inputs are generally 2D matrices or 1D solution vectors, LS structures can be bound to texture memory.

6 Application to the Permuted Perceptron Problem

An ϵ-vector is a vector with all entries being either +1 or -1. Similarly an ϵ-matrix is a matrix in which all entries are either +1 or -1. The PPP is defined as follows according to [5]:

Definition 1. *Given an ϵ-matrix A of size $m \times n$ and a multiset S of non-negative integers of size m, find an ϵ-vector V of size n such that $\{\{(AV)_j / j = \{1, \ldots, m\}\}\} = S$.*

As the iteration-level parallel model does not change the semantics of the sequential algorithm, the effectiveness in terms of quality of solutions is not addressed here. Only execution times and acceleration factors are reported. The objective is to evaluate the impact of a GPU-based implementation in terms of efficiency.

A generic tabu search has been implemented on GPU using a binary encoding. The adaptation to GPU of the tabu search is straightforward according to the proposed GPU algorithm in the high-level layer (see Algorithm 1 in Section 3.2). First, the specific LS pre-treatment on line 3 is the tabu list initialization. Second, the replacement strategy (line 17) is performed by the best admissible neighbor according to its availability in the tabu list. Finally, the specific post-treatment (line 18) represents the tabu list update.

Experiments have been implemented on top of three different configurations. The three GPU cards have a different number of multiprocessors (respectively 4, 16 and 30), which determines the number of active threads being executed. The number of global iterations of the tabu search is 10000 and 10 runs were performed for each instance. Time measurement is reported in seconds, and for both GPU implementation and GPU version using texture memory (GPU_{tex}), acceleration factors compared to a standalone CPU are designated using subindexes. Standard deviation (not represented here) is close to zero.

Experimental results for a Hamming neighborhood of distance one are depicted in Table 1 (m-n instances). From $m = 601$ and $n = 617$, the standard GPU version starts to provide better results (from $\times 1.1$ to $\times 2.2$). Regarding the GPU version using texture memory, from $m = 301$ and $n = 317$, it starts to be

Table 1. Time measurements for the 1-Hamming distance neighborhood

Instance	Core 2X 2Ghz 8600M GT 4 multi-proc		Core 4X 2.4Ghz 8800 GTX 16 multi-proc		Xeon 8X 3Ghz GTX 280 30 multi-proc	
	GPU	GPU_{Tex}	GPU	GPU_{Tex}	GPU	GPU_{Tex}
101-117	$8.9_{\times 0.4}$	$6.6_{\times 0.5}$	$4.8_{\times 0.5}$	$4.3_{\times 0.6}$	$4.9_{\times 0.4}$	$4.2_{\times 0.5}$
301-317	$34_{\times 0.7}$	$18_{\times \mathbf{1.4}}$	$16_{\times \mathbf{1.1}}$	$13_{\times \mathbf{1.5}}$	$12_{\times \mathbf{1.4}}$	$11_{\times \mathbf{1.6}}$
601-617	$169_{\times \mathbf{1.1}}$	$98_{\times \mathbf{1.9}}$	$96_{\times \mathbf{1.4}}$	$77_{\times \mathbf{1.7}}$	$47_{\times \mathbf{2.2}}$	$43_{\times \mathbf{2.4}}$
801-817	$248_{\times \mathbf{1.5}}$	$122_{\times \mathbf{3.1}}$	$125_{\times \mathbf{2.1}}$	$100_{\times \mathbf{2.7}}$	$55_{\times \mathbf{3.6}}$	$50_{\times \mathbf{4.0}}$
1001-1017	$348_{\times \mathbf{1.7}}$	$146_{\times \mathbf{4.1}}$	$145_{\times \mathbf{3.0}}$	$107_{\times \mathbf{4.0}}$	$63_{\times \mathbf{5.3}}$	$58_{\times \mathbf{5.8}}$
1301-1317	$573_{\times \mathbf{2.1}}$	$288_{\times \mathbf{4.1}}$	$228_{\times \mathbf{3.4}}$	$180_{\times \mathbf{4.3}}$	$93_{\times \mathbf{7.4}}$	$85_{\times \mathbf{8.0}}$

faster than CPU version for both configurations (from $\times 1.4$ to $\times 1.6$). The speed-up grows with the problem size increase (up to $\times 8$ for $m = 1301$, $n = 1317$). The acceleration factor for this implementation is significant but not impressive. This can be explained by the fact that since the neighborhood is relatively small (n threads), the number of threads per block is not enough to fully cover the memory access latency.

To validate this point, a neighborhood based on a Hamming distance of two on top of GPU has been implemented. Incremental evaluation is performed by a larger number of threads ($\frac{n \times (n-1)}{2}$ threads). The obtained results from experiments are reported in Table 2. Due to misaligned accesses to global memories (ϵ-matrix and ϵ-vector) of this new neighborhood, non-coalescing memory reduces the performance of the GPU implementation on G80 series. Binding texture on global memory allows to overcome the problem. Indeed, for the first instance ($m = 101$, $n = 117$), acceleration factors of the texture version are already important (from $\times 4$ to $\times 19$). As long as the instance size increases, the acceleration factor grows accordingly (from $\times 4$ to $\times 8.1$ for the first configuration). Since a large number of multiprocessors are available on both 8800 and GTX 280, efficient speed-ups can be obtained (from $\times 13$ to $\times 42.6$). As a consequence, parallelization on top of GPU provides an efficient way for handling large neighborhoods.

Table 2. Time measurements for the 2-Hamming distance neighborhood

Instance	**Core 2X 2Ghz 8600M GT 4 multi-proc**		**Core 4X 2.4Ghz 8800 GTX 16 multi-proc**		**Xeon 8X 3Ghz GTX 280 30 multi-proc**	
	GPU	GPU_{Tex}	GPU	GPU_{Tex}	GPU	GPU_{Tex}
101-117	$13_{\times 0.7}$	$2.5_{\times \mathbf{4.0}}$	$4.1_{\times \mathbf{1.8}}$	$0.6_{\times \mathbf{13.0}}$	$0.6_{\times \mathbf{12.8}}$	$0.4_{\times \mathbf{19.0}}$
301-317	$251_{\times 0.9}$	$58_{\times \mathbf{4.9}}$	$61_{\times \mathbf{2.8}}$	$10_{\times \mathbf{16.0}}$	$9.5_{\times \mathbf{17.8}}$	$6.2_{\times \mathbf{27.4}}$
601-617	$1881_{\times \mathbf{1.7}}$	$512_{\times \mathbf{6.3}}$	$355_{\times \mathbf{7.1}}$	$88_{\times \mathbf{28.5}}$	$67_{\times \mathbf{30.5}}$	$51_{\times \mathbf{40.2}}$
801-817	$4396_{\times \mathbf{2.0}}$	$1245_{\times \mathbf{6.9}}$	$815_{\times \mathbf{8.5}}$	$210_{\times \mathbf{32.9}}$	$152_{\times \mathbf{35.4}}$	$128_{\times \mathbf{42.2}}$
1001-1017	$8474_{\times \mathbf{2.1}}$	$2502_{\times \mathbf{7.0}}$	$1469_{\times \mathbf{9.8}}$	$416_{\times \mathbf{34.7}}$	$291_{\times \mathbf{38.1}}$	$262_{\times \mathbf{42.2}}$
1301-1317	$17910_{\times \mathbf{2.2}}$	$4903_{\times \mathbf{8.1}}$	$3050_{\times \mathbf{10.9}}$	$912_{\times \mathbf{36.4}}$	$647_{\times \mathbf{38.7}}$	$587_{\times \mathbf{42.6}}$

7 Discussion and Conclusion

High-performance computing based on the use of GPUs is recently revealed to be a good way to accelerate computational applications. However, the exploitation of parallel models is not trivial and many issues related to GPU memory hierarchical management of this architecture have to be considered. To the best of our knowledge, GPU-based parallel LS approaches have never been deeply investigated.

In this paper, efficient mapping of the iteration-level parallel model on the GPU has been proposed according to a three-level decomposition of the GPU hierarchy. In the high-level layer, the CPU manages the whole LS process and

let the GPU be used as a coprocessor dedicated to intensive calculations. Efficient mappings between neighborhood candidate solutions and GPU threads were made in the intermediate-level layer. Memory management is handled at the low-level layer. Code optimization based on texture memory is applied to the evaluation function kernel. The re-design of the parallel LS iteration-level model on GPU fits well for deterministic LS methods such as tabu search and iterated local search. Indeed, for problem instances with a large neighborhood set, the reported speed-ups provide promising results on the PPP (up to $\times 40$ with texture memory) compared to traditional CPUs. A next perspective is to apply our approach on other problems using different representations.

The approach presented in this paper might be easily extended to the variable neighborhood search heuristic, in which the same parallel exploration is applied for various neighborhoods. However, few other LS algorithms only partially explore neighborhoods and take the first improving local neighbor that is detected. Applied to LS algorithm such as simulated annealing, this model needs to be re-thought.

References

1. Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J.W., Skadron, K.: A performance study of general-purpose applications on graphics processors using cuda. J. Parallel Distributed Computing 68(10), 1370–1380 (2008)
2. Chitty, D.M.: A data parallel approach to genetic programming using programmable graphics hardware. In: GECCO, pp. 1566–1573 (2007)
3. Fok, K.L., Wong, T.T., Wong, M.L.: Evolutionary computing on consumer graphics hardware. IEEE Intelligent Systems 22(2), 69–78 (2007)
4. Banzhaf, W., Harding, S.: Accelerating evolutionary computation with graphics processing units. In: GECCO (Companion), pp. 3237–3286 (2009)
5. Pointcheval, D.: A new identification scheme based on the perceptrons problem. In: Guillou, L.C., Quisquater, J.-J. (eds.) EUROCRYPT 1995. LNCS, vol. 921, pp. 319–328. Springer, Heidelberg (1995)
6. Ryoo, S., Rodrigues, C.I., Stone, S.S., Stratton, J.A., Ueng, S.Z., Baghsorkhi, S.S., Wen-mei, W.H.: Program optimization carving for gpu computing. J. Parallel Distribributed Computing 68(10), 1389–1401 (2008)
7. Talbi, E.G.: Metaheuristics: From design to implementation. Wiley, Chichester (2009)
8. NVIDIA: CUDA Programming Guide Version 2.1 (2009)
9. Luong, T.V., Melab, N., Talbi, E.G.: Parallel Local Search on GPU. Research Report RR-6915, INRIA (2009)

Efficient Cycle Search for the Minimum Routing Cost Spanning Tree Problem

Steffen Wolf[1] and Peter Merz[2]

[1] University of Kaiserslautern
P.O. Box 3049, 67653 Kaiserslautern, Germany
wolf@informatik.uni-kl.de
[2] University of Applied Sciences and Arts, Hannover
Ricklinger Stadtweg 120, 30459 Hannover, Germany
peter.merz@fh-hannover.de

Abstract. The Minimum Routing Cost Spanning Tree problem is an optimization problem that strongly benefits from local search. Well-established approaches are the Ahuja-Murty local search and a weaker subtree search used in an evolutionary framework. We present a new and efficient cycle search that has a lower time complexity but achieves the same results as the strong but slow Ahuja-Murty local search. Moreover, we show that an evolutionary framework using this cycle search outperforms previous approaches regarding both quality and time. Our approach is able to find (near-)optimal solutions in all runs for all tested instances.

1 Introduction

In many applications, such as Peer-to-Peer Topology Construction, Desktop Grids, or telecommunication networks, the problem arises to find a common spanning tree that minimizes the routing cost for all pairs of nodes in the network. This problem is called the Minimum Routing Cost Spanning Tree (MRCST). It is a special case of the Optimum Communication Spanning Tree (OCST), which was introduced by Hu in [1]. Both MRCST and OCST are NP-hard graph optimization problems [2,3].

In this paper, we want to concentrate on the MRCST and present a new and efficient local search based on a cycle search. We will show that this CycleLS, when used in an Evolutionary Local Search (ELS) framework, takes less time and finds better solutions than the subtree local search we presented in [4]. We will also compare CycleLS to a well-established local search for the OCST, Ahuja-Murty Local Search (AMLS) [5], and show that CycleLS performs slightly better and requires much shorter time than an adaptation of AMLS for the MRCST.

1.1 Problem Formulation

The problem can be formulated as follows: Given a weighted graph $G = (V, E, d)$ and a demand matrix r, find a spanning tree $T \subseteq E$ that minimizes the routing cost:

P. Cowling and P. Merz (Eds.): EvoCOP 2010, LNCS 6022, pp. 276–287, 2010.

$$C(T) = \sum_{u \in V} \sum_{v \in V} d_T(u, v) \cdot r(u, v) \tag{1}$$

The function $d_T(u, v)$ denotes the length of the path from u to v in T, i. e. the sum of the distances $d_{i,j}$ of all edges of this path.

In the MRCST, the demand is a constant $r \equiv 1$, while in the OCST it is a full demand matrix $r : V \times V \to \mathbb{R}$. If we use a directed tree rooted at an arbitrary node $v^* \in V$, equation (1) for the MRCST can be reformulated to show the weights on the edges [4]:

$$C(T) = \sum_{v \in V \setminus \{v^*\}} C_T(v) \tag{2}$$

$$\text{with } C_T(v) = 2 \cdot s_T(v) \cdot (n - s_T(v)) \cdot d\,(v, p_T(v)) \tag{3}$$

Here, $n = |V|$ denotes the size of the tree, $s_T(v)$ denotes the size of the subtree rooted at v, $p_T(v)$ denotes the parent of v in the directed tree T, and $(v, p_T(v))$ denotes the edge between v and its parent. The cost $C_T(v)$ of this edge is calculated by weighing the edge's length with the number of connections that traverse it. Since each node needs to communicate to every other node ($r \equiv 1$), the number of connections traversing an edge is twice the product of the number of nodes on one side of the edge and the number of nodes on the other side of the edge. The root node v^* does not contribute to the cost, it has no parent.

Using formulation (2), we can calculate the routing cost of a tree T in linear time, compared to a quadratical time for formulation (1).

1.2 Related Work

Wu and Chao showed that the Shortest Path Tree (SPT) rooted at the median node is a 2-approximation of the MRCST [6]. The median node is the node that has the minimal distance to all other nodes in the network when distance is measured via shortest paths only. This node can be found in polynomial time by constructing all SPTs, and picking the shortest one.

Based on this 2-approximation Wu *et al*. proposed a Polynomial Time Approximation Scheme to find a $(1 + \epsilon)$-approximation in $\mathcal{O}(n^{2\lceil \frac{2}{\epsilon} \rceil - 2})$ time [7]. They also show how the general case can be reduced to the metric case by replacing all edges that violate the triangle inequality with their corresponding shortest path. A solution for this metric closure graph can be transformed back to a solution for the original graph without increasing its cost.

In [4], we presented an Evolutionary Local Search (ELS) based on subtree moves, which can find near-optimal solutions in relatively short time. We also presented a set of test instances based on real world Internet data, which has since been used in other research concerning a variety of network topology optimization problems. It is this ELS that we want to improve upon.

In [8], Singh presented a perturbation based local search using a randomized variant of the AMLS shown in Section 2.4. However, he failed to make use of the reduced complexity Ahuja and Murty presented in [5], so his local search is weaker, but has the same high time complexity as the AMLS.

```
s ← Initialization()
s ← LocalSearch(s)
mut ← n                                          set mutation rate to problem size
for iter = 1 ... κ do
    s* ← s
    for i = 1 ... λ do
        if mut = n then
            s_temp ← Initialization()
        else
            s_temp ← Mutate(s, mut)
        s_temp ← LocalSearch(s_temp)
        s* ← min{s*, s_temp}
    if s* < s then
        s ← s*
    else
        mut ← round(mut · α)                     decrease mutation rate
return s
```

Fig. 1. The Evolutionary Local Search (ELS) Framework

2 Evolutionary Local Search

In this section we introduce our new and efficient CycleLS, and give a short description of the Evolutionary Local Search (ELS) framework, which was already used in [4]. The ELS works by starting from a valid tree and applying small changes, called moves, to the current tree. In order to escape from local optima, a mutation step is applied. The ELS framework is similar to Iterated Local Search (ILS) [9] or Memetic Algorithms (MAs) [10]. Its structure is shown in Fig. 1. It differs from a pure ILS by using more than one offspring in each iteration. It also differs from usual MAs, because it does not use recombination.

2.1 Initialization

The first tree in the ELS is produced by connecting all nodes in random order to a random node that is already part of the tree. By starting from a random tree, we give away the opportunity to use an approximation as a starting point, and therefore cannot guarantee an approximation ratio for the ELS. However, random trees can be generated faster than any approximation, and they give higher diversification, allowing exploration of different parts of the solution space.

2.2 Mutation

In order to adapt the mutation rate to the current state of the search, we use the following scheme from [4]: We use a higher mutation rate in the beginning of a run and decrease the mutation rate for each non-improving step. The ELS stays at the highest mutation rate as long as this gives an improvement. Thus it

can fall back to random restart. When the search reaches better solutions, and random restart fails to find further improvements, the ELS gradually shifts from exploration toward exploitation using the mutation rate adaptation scheme.

The actual mutation is done by moving random subtrees to random new positions. However, in the beginning of the search, when the mutation rate demands to move n subtrees to new positions, the ELS simply generates a new random tree instead. The mutation rate is reduced by 20 % in each non-improving iteration ($\alpha = 0.8$ in Fig. 1), and to prevent the ELS from being stuck once the mutation rate drops too low, the lowest mutation rate is set to 2.

2.3 Population

The ELS uses a population of only one individual. In each generation, it produces λ offspring using mutation and local search, out of which the best individual replaces the original if there was an improvement. Thus, when only $\lambda = 1$ offspring is produced, the ELS behaves similarly to an ILS with $\kappa + 1$ iterations, but still adapts the mutation rate to the current search progress. With $\kappa = 1$, the ELS equals random restart with $\lambda + 1$ iterations. Thus by varying the parameters κ and λ, we can observe how well a specific local search is suited for random restart, ILS, and ELS.

The ELS can also be expanded to a full Memetic Algorithm (MA) by using a larger population and a suitable recombination.

2.4 Local Search

Since the MRCST is a special type of an OCST, each local search for the OCST can be used for the MRCST. The most prominent local search for the OCST is the Ahuja-Murty Local Search (AMLS) [5]. But any local search for trees can be used as well. In this section, we present our CycleLS, and compare it to the AMLS and ST-2opt from [4].

Ahuja-Murty Search. Ahuja and Murty originally presented a construction heuristic as well as a local search in [5]. Both algorithms have the same structure, as they calculate the cost for an edge that is inserted in the tree. In the AMLS, the tree is split in two parts by removing an edge e. Both parts, called S and $\overline{S}$ are reconnected using the least expensive edge $\{i, j\}$, $i \in S$, $j \in \overline{S}$. Two sets of variables are calculated: w_i and h_i. The total demand from node i to the other part of the tree is stored in w_i, while h_i gives the cost for routing the demands of the whole tree part to node i:

$$w_i = \begin{cases} \sum_{j \in \overline{S}} r(i,j) & i \in S \\ \sum_{j \in S} r(i,j) & i \in \overline{S} \end{cases} \qquad h_i = \begin{cases} \sum_{j \in S} w_j \cdot d_T(i,j) & i \in S \\ \sum_{j \in \overline{S}} w_j \cdot d_T(i,j) & i \in \overline{S} \end{cases} \tag{4}$$

Here, $d_T(i, j)$ is the length of the path from i to j in the corresponding part of the tree. Using these values, the increase in cost can be computed as the sum of

the routing cost to i and j in their respective parts of the tree plus the cost for routing the demands over the new edge $\{i, j\}$:

$$\alpha_{ij} = h_i + h_j + d_{ij} \cdot \sum_{x \in S} w_x \tag{5}$$

By removing the initial edge, the cost is reduced according to the same calculations. The total gain g of exchanging edge e with $\{i, j\}$ is simply:

$$g = C(T) - C(T \cup \{\{i, j\}\} \setminus \{e\}) = \alpha_e - \alpha_{ij} \tag{6}$$

Since in the MRCST we have equal demands $r \equiv 1$, the calculation can be simplified. The demand only depends on the size of the two parts of the tree:

$$w_i = \begin{cases} |\overline{S}| & i \in S \\ |S| & i \in \overline{S} \end{cases} \qquad h_i = \begin{cases} w_i \cdot \sum_{j \in S} d_T(i, j) & i \in S \\ w_i \cdot \sum_{j \in \overline{S}} d_T(i, j) & i \in \overline{S} \end{cases} \tag{7}$$

$$\alpha_{ij} = h_i + h_j + d_{ij} \cdot w_i \cdot (n - w_i) \tag{8}$$

The first step in this calculation lies in determining the two parts of the tree and determining the path lengths. All path lengths in the tree can be calculated in $\mathcal{O}(n^2)$ time. They can be stored, as they do not change until the tree is changed. Determining the side each node is on can be done in $\mathcal{O}(n)$ time. This has to be done for every edge that is to be removed. Calculating all h_i values can be done in $\mathcal{O}(n^2)$ time, and again they have to be recalculated for every edge to be removed. So calculating the gain for every new edge can be done in $\mathcal{O}(n^2)$ time, and finding the best pair of edges to be exchanged can be done in $\mathcal{O}(n^3)$ time. The time complexity of the AMLS is the same for MRCST and OCST.

Cycle Search. During the AMLS, the tree is split first, before reconnecting the parts by inserting an appropriate edge. This can be reversed: Inserting an edge in a spanning tree creates a cycle. To repair the tree, the cycle has to be broken by removing an edge from the cycle. The resulting graph is again a spanning tree.

In the proposed CycleLS, each of the $\mathcal{O}(n^2)$ edges is tried to be inserted. The insertion creates a cycle of at most n edges. If the tree is balanced, the expected cycle length is in $\mathcal{O}(\log n)$. For the following explanation, we enumerate the nodes in the cycle: $(1, 2, 3, \ldots, m, 1)$, and call the inserted edge $(1, m)$.

Determining the gain of removing an edge from the cycle can be done in $\mathcal{O}(1)$ time, using a set of auxiliary variables. First, the number of paths crossing the cut $(i, i+1)$ can be determined as $2 \cdot \left(\sum_{x=1}^{i} c_x\right) \cdot \left(\sum_{x=i+1}^{m} c_x\right)$. Here, c_x gives the number of nodes that node x connects to the cycle. The factor 2 is needed because paths exist for both directions. See Fig. 2 for an example with $m = 5$ nodes in the cycle. In this example, $c_1 = c_3 = c_5 = 1$, $c_2 = 2$, and $c_4 = 3$. The cut at edge (2,3) affects $2 \cdot (1+2) \cdot (1+3+1)$ paths. Calculating the gain on the

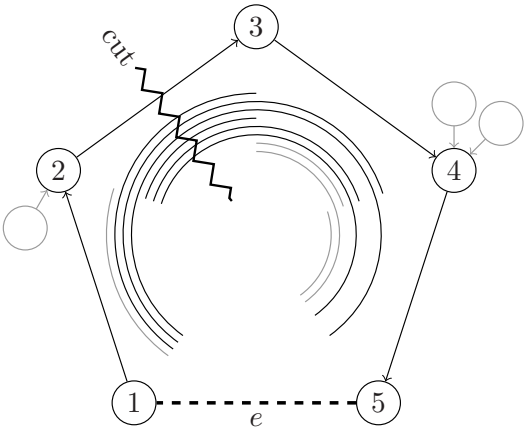


Fig. 2. Calculating the gain for CycleLS. By inserting edge $e = (1,5)$, a cycle (1,2,3,4,5,1) is formed. Cutting an edge, e. g. (2,3), affects some of the paths between those nodes, here: $\{1,2\}\times\{3,4,5\}$. After the cut, these paths are routed over edge e, taking the opposite direction in the cycle, but the remaining paths are left untouched.

ChangeRoot(m) *updates all $s_T(i)$, $\mathcal{O}(n)$, done in outer loop*

$\ell \leftarrow d_{m,1}$
for $i = 1 \dots m-1$ **do**
  $\ell \leftarrow \ell + d_{i,i+1}$ *calculate the length of the cycle*

$c_1 \leftarrow s_T(1)$
for $i = 2 \dots m$ **do**
  $c_i \leftarrow s_T(i) - s_T(i-1)$ *calculate the number of nodes behind i*

$r_{\text{after}} \leftarrow n$ *stores the sum of demands after the cut*
$\ell_{\text{after}} \leftarrow 0$ *stores the sum of half-weighted path lengths after the cut*
for $i = 1 \dots m-1$ **do**
  $r_{\text{after}} \leftarrow r_{\text{after}} - c_i$
  $\ell_{\text{after}} \leftarrow \ell_{\text{after}} + r_{\text{after}} \cdot d_{i,i+1}$

$r_{\text{before}} \leftarrow 0$ *stores the sum of demands before the cut*
$\ell_{\text{before}} \leftarrow 0$ *stores the sum of half-weighted path lengths before the cut*
for $i = 1 \dots m-1$ **do**
  $r_{\text{before}} \leftarrow r_{\text{before}} + c_i$
  $r_{\text{after}} \leftarrow n - r_{\text{before}}$
  $\ell_{\text{before}} \leftarrow \ell_{\text{before}} + r_{\text{before}} \cdot d_{i,i+1}$
  $\ell_{\text{after}} \leftarrow \ell_{\text{after}} - r_{\text{after}} \cdot d_{i,i+1}$
  $p_{i,i+1} \leftarrow \ell_{\text{before}} \cdot r_{\text{after}} + r_{\text{before}} \cdot \ell_{\text{after}}$ *calculate the sum of weighted path lengths*
  $g_{i,i+1} \leftarrow 2 \cdot (2 \cdot p_{i,i+1} - r_{\text{before}} \cdot r_{\text{after}} \cdot \ell)$ *calculate the gain of a cut at $(i, i+1)$*

Fig. 3. Calculating the gain of removing an edge from the cycle. The cycle is induced by edge $e = (1, m)$ and consists of the nodes $1 \dots m$, in this order. The gain for removing edge $(i, i+1)$ is stored in $g_{i,i+1}$. All operations can be performed by iterating from node 1 to m using the parent pointers. Changing the root to m and calculating the sizes of the subtrees $s_T(i)$ is supposed to be done in an outer loop.

cut edge can be done by multiplying this value with the length of the cut edge $d_{i,i+1}$. Using running sums, this calculation can be done in $\mathcal{O}(1)$.

However, this is not the total gain of this exchange, as the paths extend before and after the cut edge. Since these paths have different lengths and are used by different numbers of nodes c_x, the calculation seems expensive at first glance. Nevertheless, it can still be achieved in $\mathcal{O}(1)$ using running sums of path lengths, as Fig. 3 shows.

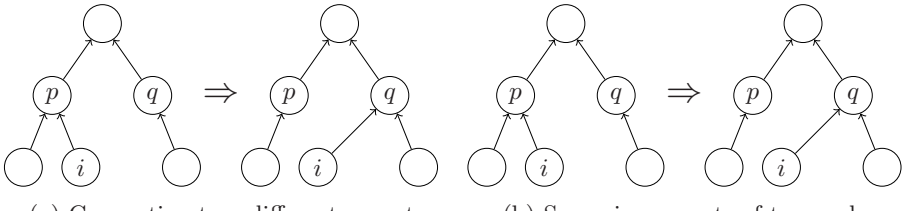
After calculating the weighted sum of path lengths $p_{i,i+1}$ over the cut $(i, i+1)$, the gain can be obtained by replacing these paths (adds $p_{i,i+1}$ to the gain) with paths in the opposite direction of the cycle (substracts $r_{\text{before}} \cdot r_{\text{after}} \cdot \ell - p_{i,i+1}$ from the gain). A factor of 2 is needed to account for the fact that each path is used in both directions.

Using this algorithm, the best exchange can be found in $\mathcal{O}(n^3)$ time, which is the same time complexity as the AMLS, but the expected time is in $\mathcal{O}(n^2 \cdot \log n)$. Since both AMLS and CycleLS pick the best pair of edges to exchange, both lead to the same local optima. However, it remains to be seen in the experiments which local search is faster.

Subtree Search. The subtree searches we presented in [4] are based on an attempt to further reduce the complexity of the cycle search. Instead of considering all edges from the cycle, the subtree searches only consider the edge adjacent to the newly inserted edge. Thus, a node is disconnected from its parent and reconnected to another part of the tree, taking its subtree along.

We will use the strongest subtree search from [4] and call it ST-2opt. It uses two types of moves shown in Fig. 4. In move (a), a subtree is moved to a different part of the tree. We improved upon the original formulation in [4] by allowing node i to connect to a node q in its own subtree. To avoid creating cycles, we have all parent pointers point in the direction of q. In move (b), two subtrees are swapped. The gain g of these moves can be calculated easily when using formulation (2). For both moves, only those $C_T(v)$ have to be recalculated that do change. Affected are all nodes for which the number of children or the parent edge changes, i. e. all nodes on the path P in the tree T from i to the new parent q (or from i to j for move (b)):

$$g = C(T) - C(T') = \sum_{v \in P} (C_T(v) - C_{T'}(v)) \tag{9}$$



(a) Connecting to a different parent (b) Swapping parents of two nodes

Fig. 4. Two types of subtree moves

The number of updates is expected to be in $\mathcal{O}(\log n)$, but in the worst case, if the tree degenerates to a single path, it is linear in n. If we assume an expected path length of $\mathcal{O}(\log n)$, the time to search the complete neighbourhood becomes $\mathcal{O}(n^2 \log n)$, which is the same as the CycleLS. Since CycleLS already covers move (a), the subtree search seems weaker. But CycleLS does not cover move (b), which can be seen as two moves of type (a) applied consecutively. This difference can give ST-2opt an advantage.

3 Experiments

In order to evaluate the ELS and compare the proposed CycleLS to the established AMLS and ST-2opt, we use the same real world problem instances as presented in [4]. These instances are based on delay measurements in the PlanetLab [11], a platform connecting computers around the world to form a global research network for the development of new network services. In [12], the interdomain connectivity of PlanetLab is analyzed, showing that triangle inequality violations are frequent even in such a small and controlled environment as the PlanetLab. The Round Trip Time (RTT) measurements used in [12] are accredited to Jeremy Stribling, who provided RTT measurements between PlanetLab hosts taken every 15 minutes from February 2003 to December 2005 [13]. Our instances are taken from the first measurement for each month of the year 2005 (denoted by `mm-2005`), using the minimal RTTs rounded to milliseconds as communication costs for the links. The measurements are flawed, not all hosts in PlanetLab were reachable all the time, in some cases measurements for single edges are missing for no obvious reasons. After eliminating hosts that did not report any measurements or that reported only a few measurements, the reduced distance matrices consist of 70 to 419 nodes.

Table 1 gives an overview over the PlanetLab instances. The percentage of missing links varies between 3.2 % and 7.8 %. Triangle inequality violations are very frequent, between 64.2 % and 86.0 % of all links do not represent the shortest path between the corresponding nodes. Column APSP (All-Pairs-Shortest-Path) shows the total communication cost when only shortest paths are used for communication. This is the best communication cost achievable by any topology and therefore a valid lower bound for the MRCST. Column Direct gives the communication cost when only direct links are used. Because of the triangle inequality violations this value is higher. Comparing this column to the APSP shows the degree of triangle inequality violations in the networks. Using shortest paths instead of direct links helps reducing the communication cost by at least 12.0 % (`04-2005`) and up to 68.9 % (`12-2005`) in these networks.

Also shown are the costs for the SPT rooted at the median, which gives an upper bound and a 2-approximation to the MRCST. This upper bound is already better than the direct connection, with the exception of only three instances. The last column in Table 1 gives the best known solution for the MRCST. In only one of the considered instances this gives a slightly worse cost than using direct connections (`04-2005`).

Table 1. Overview of the PlanetLab instances. For each instance, the network size, the percentage of missing links (missing), the percentage of links violating the triangle inequality (TIV), and the communication costs using either shortest paths for all connections (APSP), direct connections (Direct), the SPT rooted at the median node (SPT), or the best known solution for the MRCST (Best) are shown.

Instance	Size	missing	TIV	APSP [ms]	Direct [ms]	SPT [ms]	Best [ms]
01-2005	127	3.2 %	79.8 %	2 447 260	3 868 930	3 025 120	2 743 556
02-2005	321	6.0 %	83.5 %	15 868 464	18 045 792	19 607 936	17 567 152
03-2005	324	7.8 %	84.3 %	17 164 734	19 551 610	21 117 374	19 288 320
04-2005	70	3.7 %	64.2 %	663 016	725 532	787 856	751 404
05-2005	374	6.3 %	86.0 %	17 324 082	25 198 386	19 949 036	19 175 890
06-2005	365	7.1 %	85.3 %	18 262 984	24 982 820	22 217 312	20 312 884
07-2005	380	3.8 %	85.1 %	24 867 734	35 449 904	29 288 762	27 731 218
08-2005	402	3.8 %	84.0 %	27 640 136	36 000 372	32 209 226	30 540 984
09-2005	419	5.9 %	85.5 %	23 195 568	36 620 526	27 204 468	25 712 960
10-2005	414	5.8 %	83.8 %	28 348 840	40 139 262	33 639 378	31 195 732
11-2005	407	5.5 %	83.2 %	23 694 130	31 144 230	29 322 480	26 797 284
12-2005	414	6.0 %	83.4 %	20 349 436	61 751 346	25 985 050	22 723 454

3.1 Results

We performed several experiments to evaluate the effectiveness of the different local searches as well as the evolutionary framework. All algorithms were implemented in C++, CPU times reported in this section refer to a Xeon E5420 (2.5 GHz, running Linux), and results are averaged over 30 runs for each instance.

In a first set of experiments, we varied the number of generations κ, the number of offspring λ, and the local search used in the ELS. The results for these experiments are shown in Table 2. To allow for quick comparison, we print the same number of digits for the values in each column. In the following paragraphs, we will discuss certain parts of this table in more detail.

Iterated Local Search (ILS). When setting the number of offspring to $\lambda = 1$, the ELS behaves like an ILS. All local searches reached solution quality of less than 1 % gap after the first iteration ($\kappa = 1$), and further improve this quality during the following iterations. The subtree search ST-2opt is weaker and slightly slower than CycleLS. Interestingly, our proposed CycleLS is not only faster than the AMLS, but also finds better solutions with about half the gap to the best known solutions, thus making it the best choice for finding best solutions.

During ILS mode, no local search is able to find all best known solutions in all runs even with $\kappa = 1000$ iterations. CycleLS finds the best known solution in about $3/4$ of the runs, ST-2opt finds them in about $7/12$ of the runs, and AMLS finds them in almost $1/2$ of the runs, with a slightly better gap than ST-2opt.

Random Restart. Using only $\kappa = 1$ iteration, the ELS produces the same results as random restart with $\lambda+1$ iterations. For the considered local searches,

Table 2. Results for the ELS. For the various local searches, numbers of iterations κ, and numbers of offspring λ, the CPU times and gaps to the best known solutions are shown. These values are averaged over all 12 instances and 30 runs each, totalling 360 runs for each parameter set. The columns *Best* give the number of runs that found the best known solution.

		ST-2opt			CycleLS			AMLS		
κ	λ	Time [s]	Gap [%]	Best	Time [s]	Gap [%]	Best	Time [s]	Gap [%]	Best
1	1	0.3	0.97014	19	0.2	0.52725	55	128.5	0.51195	56
	10	1.9	0.18579	60	1.4	0.04615	162	739.2	0.08945	108
	100	17.8	0.00894	199	12.9	0.00050	307	6731.1	0.00325	213
10	1	1.5	0.32805	107	1.1	0.11449	187	425.6	0.16368	149
	10	14.0	0.01813	312	10.3	0.00066	351	4191.5	0.00312	291
	100	133.7	0.00007	358	99.2	0.00000	360	39296.8	0.00007	356
100	1	4.4	0.25586	159	3.6	0.08862	244	542.0	0.15439	165
	10	43.1	0.01499	328	34.7	0.00063	353	5456.0	0.00215	304
	100	420.4	0.00007	359	341.1	0.00000	360	50889.9	0.00004	358
1000	1	29.3	0.17630	213	24.9	0.05947	277	1154.5	0.14654	179
	10	290.4	0.01135	336	247.8	0.00062	354	11922.3	0.00196	320
	100	2914.7	0.00007	359	2477.1	0.00000	360	114978.7	0.00004	358

random restart takes longer, but produces better results than the ILS (comparing $(\kappa, \lambda) = (1, 100)$ with $(100, 1)$). However, random restart still fails to find the best known solutions in more than one third of the runs for ST-2opt and AMLS, and about one sixth of the runs for CycleLS.

Evolutionary Local Search (ELS). When using the full ELS with $\kappa > 1$ and $\lambda > 1$, we can see that increasing the number of offspring λ significantly increases the average solution quality. All local searches with $\kappa = 1000$ generations and $\lambda = 100$ offspring per generation find the best solutions very frequently. The best local search is CycleLS, which found all best solutions in all 360 runs in at most $\kappa = 5$ iterations, taking an average of 18 s and at most 146 s. AMLS failed to find the best known solutions in two runs, and the weaker ST-2opt failed in only one run but produced a higher gap.

A statistical analysis with 95 % confidence shows no significant difference between the quality of these local searches with $\kappa \geq 10$ and $\lambda = 100$, where almost all runs found the best solutions. In all other settings, ST-2opt was significantly worse than CycleLS and AMLS, except for $\kappa = 1000$ where the difference to AMLS is not significant. CycleLS was also significantly better than AMLS with $\kappa = 1$ and $\lambda \geq 10$, or with $\kappa \geq 100$ and $\lambda = 1$.

The disadvantage of all considered local searches is their high complexity ($\mathcal{O}(n^2 \log n)$). With $\kappa = 1000$ iterations and $\lambda = 100$ offspring, their running times can already be measured in hours. When considering larger networks, randomized variants (N_{fast} from [4] or a randomized CycleLS) with complexity $\mathcal{O}(n \log n)$ should be preferred. Although they tend to converge to lower-quality

Table 3. Properties of the local optima. For each local search neighbourhood, the minimal, average, and maximal Fitness Distance Correlation (FDC) coefficient for the considered PlanetLab instances is shown.

	Excess [%]			FDC			Distance		
LS	min	avg	max	min	avg	max	min	avg	max
ST-2opt	0.9	1.8	2.4	0.40	0.63	0.87	17	53	75
CycleLS	0.4	1.1	1.8	0.53	0.72	0.92	12	45	74
AMLS	0.4	1.0	1.6	0.38	0.73	0.93	13	46	69

solutions, they already provide reasonably good results when given less time than one iteration of their full counterparts.

3.2 Fitness Distance Analysis

To evaluate the potential of the local searches, we analyzed the properties of the local optima. For each local search, we searched for 1000 local optima, starting from 1000 random solutions, and measured the percentage excess over the best solution and the distance in the solution space to the best solutions (i. e. the number of edges not found in any of the best solutions). From these values we calculated the Fitness Distance Correlation (FDC) coefficient of the local optima for each problem instance. The results are displayed in Table 3, giving the minimal, maximal, and average values for all 12 PlanetLab instances.

When comparing the average excess of the local searches, the same conclusions can be drawn as from the ELS results. Using ST-2opt leads to local optima with higher gaps to the best known solutions. CycleLS and AMLS produce roughly the same results with about half the gap of ST-2opt, but no local search can find the global optimum in all runs without an ELS.

The distance to the best solutions in the solution space indicates how many changes have to be applied to the local optima in order to reach a best solution. It is bounded by the number of edges $n - 1$ in the solution tree. In the considered PlanetLab instances, the size varies between 69 and 418, with an average of 333.75. The values for the average distance for all local searches show that the local optima are still far from the best solutions in the solution space. On average, between every sixth and every seventh edge has to be changed to reach the best known solutions when starting from these local optima.

If the FDC correlation coefficient is positive, shorter distances correlates with better solutions. For all considered local searches and problem instances, the FDC coefficient is positive, indicating that an evolutionary algorithm can exploit the structure of the search space.

The FDC coefficients as well as the FDC scatter plots indicate that independent of the used local search the fitness landscapes appear to be correlated, and a recombination might be effective. However, as the ELS already finds the best known solutions in almost all runs using these local searches, recombination seems unnecessary.

4 Conclusion

We presented a new local search CycleLS for the MRCST. It has lower time complexity than the AMLS from [5], and is stronger than ST-2opt from [4] which has the same time complexity. In experiments conducted on real world instances, we showed that CycleLS outperforms both competitors considering time and solution quality. When used in an ELS with $\kappa = 5$ and $\lambda = 100$, CycleLS found the best known solutions of all considered instances in every run.

References

1. Hu, T.C.: Optimum Communication Spanning Trees. SIAM Journal of Computing 3(3), 188–195 (1974)
2. Johnson, D.S., Lenstra, J.K., Rinnooy Kan, A.H.G.: The Complexity of the Network Design Problem. Networks 8, 279–285 (1978)
3. Garey, M.R., Johnson, D.S.: Computers and Intractibility: A guide to the theory of NP-completeness. W. H. Freeman and Co., San Francisco (1979)
4. 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)
5. Ahuja, R.K., Murty, V.V.S.: Exact and Heuristic Algorithms for the Optimum Communication Spanning Tree Problem. Transportation Science 21(3), 163–170 (1987)
6. Wu, B.Y., Chao, K.M.: Spanning Trees and Optimization Problems. In: Discrete Mathematics and its Applications. Chapman & Hall/CRC, Boca Raton (2004)
7. Wu, B.Y., Lancia, G., Bafna, V., Chao, K.M., Ravi, R., Tang, C.Y.: A Polynomial-Time Approximation Scheme for Minimum Routing Cost Spanning Trees. SIAM Journal of Computing 29(3), 761–778 (1999)
8. Singh, A.: A New Heuristic for the Minimum Routing Cost Spanning Tree Problem. In: International Conference on Information Technology (ICIT 2008), pp. 9–13. IEEE Computer Society, Los Alamitos (2008)
9. 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)
10. Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Caltech Concurrent Computation Program, C3P Report 826, California Institute of Technology, Pasadena, USA (1989)
11. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: PlanetLab: An Overlay Testbed for Broad-Coverage Services. ACM SIGCOMM Computer Communication Review 33(3), 3–12 (2003)
12. Banerjee, S., Griffin, T.G., Pias, M.: The Interdomain Connectivity of PlanetLab Nodes. In: Barakat, C., Pratt, I. (eds.) PAM 2004. LNCS, vol. 3015, pp. 73–82. Springer, Heidelberg (2004)
13. Stribling, J.: PlanetLab All-Pairs-Pings (2003–2005), http://pdos.csail.mit.edu/~strib/pl_app/

Author Index