María J. Blesa
Christian Blum
Paola Festa
Andrea Roli
Michael Sampels (Eds.)

LNCS 7919

Hybrid Metaheuristics

8th International Workshop, HM 2013
Ischia, Italy, May 2013
Proceedings

Springer

Lecture Notes in Computer Science 7919

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

María J. Blesa Christian Blum Paola Festa
Andrea Roli Michael Sampels (Eds.)

Hybrid Metaheuristics

8th International Workshop, HM 2013
Ischia, Italy, May 23-25, 2013
Proceedings

Springer

Volume Editors

María J. Blesa
Universitat Politècnica de Catalunya, 08034 Barcelona, Spain
E-mail: mjblesa@lsi.upc.edu

Christian Blum
University of the Basque Country, 20018 San Sebastian, Spain
E-mail: christian.blum@ehu.es

Paola Festa
University of Naples Federico II, 80126 Naples, Italy
E-mail: paola.festa@unina.it

Andrea Roli
Università di Bologna, 47521 Cesena, Italy
E-mail: andrea.roli@unibo.it

Michael Sampels
Université Libre de Bruxelles, 1050 Bruxelles, Belgium
E-mail: msampels@ulb.ac.be

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38515-5 e-ISBN 978-3-642-38516-2
DOI 10.1007/978-3-642-38516-2
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013938275

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

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

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

Printed on acid-free paper

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

Preface

The research field of hybrid metaheuristics is today established as a reference field in the areas of optimization and problem solving. Hybrid metaheuristics have a strong impact on applications because they provide efficient and powerful problem-solving techniques for optimization problems in science and industry. The interdisciplinary research community with scientists from various backgrounds provides a fertile environment where innovative techniques are presented and discussed.

The International Workshop on Hybrid Metaheuristics (HM) pursues the direction of combining application-oriented and fundamental research. This is demonstrated by the papers in the proceedings of this eighth edition of HM. The contributions selected for this volume represent an important sample of current research in hybrid metaheuristics. It is worth emphasizing that the selected papers cover both theoretical and experimental results, including new paradigmatic hybrid solvers, and automatic design approaches as well as applications to logistics and public transport.

As is the tradition of the workshop, special care was exercised in the review process: out of 30 submissions received, 16 papers were selected on the basis of the reviews by the Program Committee members and evaluations by the Program Chairs. Reviews were in great depth: reviewers provided authors with constructive suggestions for improvement. Special thanks are extended to the Program Committee members who devoted their time and effort.

We believe that this selection of papers will be of interest not only to researchers working in the area of optimization and problem solving. We hope that, by showing the success of ideas generated from connections between different research fields, we can stimulate the cross-fertilization between computer science, mathematics, engineering, biology, operations research, economics, and others.

April 2013

María J. Blesa
Christian Blum
Paola Festa
Andrea Roli
Michael Sampels

Organization

General Chair

Paola Festa	Università degli Studi di Napoli Federico II, Italy

Program Chairs

María J. Blesa	Universitat Politècnica de Catalunya, Barcelona, Spain
Christian Blum	IKERBASQUE, Basque Foundation for Science, and University of the Basque Country, San Sebastian, Spain
Andrea Roli	Università di Bologna, Italy
Michael Sampels	Université Libre de Bruxelles, Belgium

Program Committee

Massimo Benerecetti	Università degli Studi di Napoli Federico II, Italy
Mauro Birattari	Université Libre de Bruxelles, Belgium
Marco Chiarandini	Syddansk Universitet, Denmark
Luca Di Gaspero	Università di Udine, Italy
Karl F. Doerner	Universität Wien, Austria
Andreas T. Ernst	CSIRO, Australia
Antonio J. Fernández	Universidad de Málaga, Spain
Oliver Kramer	Universität Oldenburg, Germany
Andrea Lodi	Università di Bologna, Italy
Vittorio Maniezzo	Università di Bologna, Italy
Rafael Martí	Universitat de València, Spain
Daniel Merkle	Syddansk Universitet, Denmark
Bernd Meyer	Monash University, Australia
Panos M. Pardalos	University of Florida, USA
Günther R. Raidl	Technische Universität Wien, Austria
Helena Ramalhinho Lourenço	Universitat Pompeu Fabra, Barcelona, Spain
Andreas Reinholz	Deutsches Zentrum für Luft- und Raumfahrt, Germany
Mauricio G.C. Resende	AT&T Labs Research, USA
Celso C. Ribeiro	Universidade Federal Fluminense, Niterói, Brazil
Andrea Schaerf	Università di Udine, Italy

Marc Sevaux	Université Européenne de Bretagne, France
Patrick Siarry	Université Paris-Est Créteil, France
Kenneth Sörensen	Universiteit Antwerpen, Belgium
Thomas Stützle	Université Libre de Bruxelles, Belgium
Éric Taillard	Haute École Spécialisée de Suisse Occidentale, Switzerland
El-Ghazali Talbi	Université de Lille, France
Stefan Voß	Universität Hamburg, Germany

Local Organization

Daniele Ferone	Università degli Studi di Napoli Federico II, Italy
Paola Festa (Chair)	Università degli Studi di Napoli Federico II, Italy
Demetrio Laganà	Università della Calabria, Italy

Sponsoring Institutions

- Dipartimento di Matematica e Applicazioni "R. Caccioppoli", Università degli Studi di Napoli Federico II, Italy
- Università degli Studi di Napoli Federico II, Italy

Table of Contents

A Pre-processing Aware RINS Based MIP Heuristic

Thiago M. Gomes, Haroldo G. Santos, and Marcone J.F. Souza

Departamento de Computação, Universidade Federal de Ouro Preto (UFOP)
Ouro Preto, MG, Brasil

Abstract. This paper proposes an adaptation of the RINS MIP heuristic which explicitly explores pre-processing techniques. The method systematically searches for the ideal number of fixations to produce sub-problems of controlled size. These problems are explored in a Variable Neighborhood Descent fashion until a stopping criterion is met. Preliminary experiments implemented upon the open source MIP solver COIN-OR CBC are presented.

1 Introduction

One of the most important techniques for solving complex optimization problems is Mixed Integer Programming (MIP). A MIP problem involves a set of variables, a set of constraints on these variables, a set of integrality constraints and a linear objective function to optimize.

MIPs are typically solved by branch-and-bound or branch-and-cut techniques. These approaches explore a tree of relaxations of the original MIP, in which each node in the tree is divided into two disjointed sets by imposing limiting restrictions upon an integer variable. Even tough MIP solvers can be applied to a variety of problems, their performance in producing good feasible solutions and strong dual bounds greatly differs for different applications and their respective formulations. Thus, operations research practitioners often consider the use of specifically tailored heuristics such as Tabu Search [1], Genetic Algorithms [2], [3], Reconnect paths [4], among others, which objective is only in the production and improvement of feasible solutions. In recent years, the application of MIP solvers in different domains has motivated the development of many MIP heuristics, such as Feasibility Pump [12][13], Local Branching [14] and RINS[11].

Among the solvers which have their source code available, we can highlight the COIN-OR CBC [16] [17]. It is a solver of linear and integer programs. The package includes features such as pre-processing, cutting planes, heuristics and branching strategies. Although it was initially designed to be used as a library, it includes an independent solver callable by the command line. It is possible to use the file formats .lp and .mps. It can also run in parallel to take advantage of multi-core computers.

In this paper we propose modifications in MIP heuristic Relaxation Induced Neighborhood Search (RINS) proposed by [11] in the literature. The idea is to

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 1–11, 2013.

search for an ideal number of variables to be fixed in a specific problem. If this number is too small, the search space may be too large to be searched efficienly. On the other hand, a large number of fixations may restrict the search space so much that no improved solution will be found. All our implementations were coded using the COIN-OR CBC libraries.

The remainder of this paper is divided as follows. Section 2 presents the literature review considering integer programming problems and metaheuristics. Section 3 describes the proposed adaptive heuristics RINS(pRins) and the methodology used. In section 4, the test instances are described, while in Section 5 the experiments and computational results are presented. The last section concludes the paper and suggests possible improvements.

2 Literature Review

The heuristics were originated in Operations Research and Artificial Intelligence communities. At first, combinations of heuristics were not explored, since they worked well separately. The motivation for hybridization emerged aiming at exploring the benefits of the synergy between the methods. However, it is not trivial to find good combinations [5].

[5] presents a survey on hybridization of metaheuristics with other optimization methods for solving problems of combinatorial nature. The authors emphasize the importance of hybrid methods combining features of diversification and intensification when searching for a solution. Among the methods of integer programming, we highlight those based on Lagrangian relaxation, as well as iterative heuristic (LPA, and IIRH IRH), which fix and solve the subproblems encountered at every stage.

For [6], the reason for using heuristics is that they must help MIP finding good solutions earlier, thus avoiding search in regions of low quality in the tree, and at the same time, they must enhance the search for promising regions. In his paper it is presented a 0-1 heuristic , stand-alone implementation, in other words, that is independent of branch-and-bound. It is built around a merit function by measuring the completeness of the solution. The method involves four steps: gradient-based pivoting, pivoting poll, cuts convexity/intersection and exploration of the tree of variables blocks.

While solving a problem in a Linear Programming based branch-and-bound algorithm, there are usually two solutions available, one that meets the requirements of completeness of variables, but is not optimal, and another one that is a fractional solution which doesn't meet the requirements of variables completeness, however has a better value. The Relaxation Induced Neighborhood Search(RINS) method, proposed by [11], is based upon this assumption and fixes the variables that are equal in both solutions, as they meet both criteria. Then a cut is added, based on the current value of the objective function and the subproblem is solved with the remaining variables. The method has characteristics which are similar to Path Relinking [4], since it is also connecting two solutions.

In [7], the authors apply two tree search techniques to the Traveling Salesman Problem: Local Branching (LB) and Sliced Neighborhood Search(SNS). While LB produces intensification in the region of the incumbent solution exploring nearby space, the SNS technique works diversifying the search. The SNS technique improves the incumbent solution by the random exploration of distant spaces in the neighborhood. It explores the space considering the disparities between the incumbent solution and the neighbors. At each iteration, a piece of the neighborhood is explored by choosing a set of variables or using short time limits. The combined result of the two techniques has proven itself to be satisfactory, finding better quality solutions.

[8] presents the heuristic Distance Induced Neighborhood Search (DINS). The idea of this heuristic is to use a metric distance between the linear relaxation solution and the current integer solution, exploring the nodes generated by the search. The DINS method incorporates hard-fixating and soft-fixating, rounding variables according to the metric defined. It follows the intuition that good solutions are close to the relaxed solution. The method also considers a criterion to avoid excessive fixations, in case many integer variables are fixed.

[15] describes an evolutionary approach to improve solutions to mixed integer programming (MIP) models. Evolutionary algorithms adopt a natural-selection analogy, exploring concepts such as population, combination, mutation, and selection to explore a diverse space of possible solutions to combinatorial optimization problems while, at the same time, retaining desirable properties from known solutions. The proposed method maintains a fixed-size pool of the P best distinct solutions found so far, to perform the operations of combination and mutation. For mutation, a solution is chosen, and then a percentage of variables are fixed. The resulting sub-MIP is solved, the best solution is added to the pool and the percentage of fixation is updated. The combination chooses a pair of solutions, fix variables whose values agree in all of the chosen solutions. Then the sub-MIP is solved and adds the best solution found to the solution pool. The experiments showed satisfactory results.

The heuristic Relaxation Enforced Neighborhood Search (RENS) presented in [9], works with large neighborhood search. The method builds a sub-problem considering the viable rounding of some fractional point - usually the optimal LP relaxation of the original MIP. The current solution is the starting point and neighborhoods are defined by using fixations and adding restrictions. The RENS idea is fixing variables with integer value in the relaxed solution and searching the remaining solutions, rounding to the nearest integer.

In [10] a beverage production plant is modeled using mixed-integer programming, and it involves lot sizing, decisions planning and dependent time machine preparation. It proposes a fixation and relaxation heuristic to explore the problem, since CPLEX does not satisfactorily resolve instances of the problem. In this procedure, the set of integer variables is divided into disjunctive sets. At each iteration, the variables of one of the sets are fixed, while the others are relaxed, and the resulting submodel is resolved. The sets were divided considering the production periods.

3 The pRINS MIP Heuristic

The heuristic developed in this work is based on the method RINS. In this method, the following steps are taken at each node in the branch-and-cut tree:

1. fix the variables with the same value of the incumbent solution and the relaxed solution;
2. set a cut based on the value of the objective function in the current incumbent solution;
3. solve the sub-MIP with the remaining variables.

The RINS sets all the variables that are equal in both solutions. Depending on how many fractional variables appear on the fractional solution, the number of the fixes in the resulting neighborhood can be too small or too large. The proposed method explores existing pre-processing techniques to quickly generate sub-problems of controled size. These sub-problems are solved in a VND(Variable Neighborhood Descend) [18].

Algorithm 1. pRINS

Input: $Sol_f, Sol_i, mip, Req_{size}, N_{atpmax}, Dif_p$
Output: Sol^*
$Sol^* \leftarrow Sol_i$;
X_p receives the sort of the variables according to their priority for fixing considering the integer and fractional solution (Sol_i, Sol_f);
repeat
  $fix_{size} \leftarrow$ buildSizes$(X_i, X_p, N_{atpmax}, Req_{size}, Dif_p, Lim_{Rel})$;
  $mip'' \leftarrow$ createProblem$(X_i, X_p, fix_{size}, mip)$;
  if *mip'' relaxation cost indicates possible improvement* **then**
    $Sol_{rins} \leftarrow$ solve mip'';
    if *found better solution* **then**
      $Sol^* \leftarrow Sol_{rins}$;
      Recalculate the vector of priorities (X_p) considering Sol^*, Sol_f ;
    else
      Increase the required size Req_{size};
    end
  else
    Increase the required size Req_{size};
  end
until *the required size is smaller than the total number of variables and time limit not exceeded* ;
return Sol^*;

The method pRINS(pre-processing aware RINS), algorithm 1, establishes a stopping criterion when applying this adapted heuristic. Given an integer solution (Sol_i), fractional solution (Sol_f), a required size to created

problem (Req_{size}), maximum number of attempts for building a subproblem with the desired size(N_{atpmax}), acceptable percentage difference between desired and founded size (Dif_p) and the original problem MIP (mip). Initially an ordered array indicating the priorities of variables for fixation is created. To define them, these vector variables are ordered according to the difference in magnitude between the value in the integer solution and fractional solution. The variables whose difference in (Sol^*) and (Sol_f) is zero are the first ones of this vector and the others are placed in this array in ascending order of difference. Then the method buildSizes, defined in algorithm 2, is used to determine the number of fixations (fix_{size}) necessary to achieve the required size problem. It is created a new problem(mip'') which is pre-processed, considering the number of fixations determined previously. If the relaxation cost does not indicates possible improvement, then the size of the problem increases. Otherwise, the problem is solved, if the best solution is found, the integer solution is updated, priorities recalculated, trying to solve the new problem. If the solution is worse, the size of the problem increases (Req_{size}) in percentage to consider a larger search space.

Algorithm 2. buildSizes

Input: $X_i, X_p, N_{atpmax}, N_{des}, Dif_p, Lim_{Rel}$
Output: N_{fix}
$N_a \leftarrow 0$;
$L_l \leftarrow 0$;
$L_u \leftarrow maxSize_{last}$;
$N_{fix} \leftarrow (L_l + L_u)/2$;
$Dif_{max} \leftarrow 0$;
repeat
  $N_a + +$;
  $mip'' \leftarrow$ createProblem(X_i, X_p, N_{fix}, mip);
  $Prob_{size} \leftarrow$ nvars(mip'');
  if $|N_{des} - Prob_{size}| \leq N_{des} * Dif_p$ **then**
    return N_{fix};
  end
  if $Prob_{size} == 0$ **or** $Prob_{size} < N_{des}$ **or** $Lim_{Rel} \geq 0.99 * cost(S_i)$ **then**
    $L_u \leftarrow N_{fix}$;
    $L_l \leftarrow N_{fix}/2$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  else
    $L_l \leftarrow N_{fix}$;
    $N_{fix} \leftarrow (L_l + L_u)/2$;
  end
until $N_a < N_{atpmax}$ **or** $L_l < L_u$;
return N_{fix};

The method buildSizes, is used to determine the ideal number of fixations. Given the desired size of the problem (N_{des}), which in the beginning is the size

of a mip that is small enough to be quickly solved, the total number of variables (size(X_i)), maximum number of retries (N_{atpmax}), the vector of priorities fixation (X_p), vector of integer variables (X_i), the MIPs are created and the number of fixations is found. To reach the desired size, the number of fixations is changed according to a binary search until the number of attempts is exhausted, the binary search ends or size is achieved. Initially, the upper limit of fixation is the total number of variables in the problem. This upper limit is changed at the end of each binary search, being upgraded to the number of fixations used in the last sub-problem solved. This control is made through the variable ($maxSize_{last}$). This method calls another one (createProblem, defined in algorithm 3) which checks the feasibility of the resulting subproblem. After the problem is built, nvars function returns the number of free variables of the problem. The search ends when the number of free variables($Prob_{size}$) approaches the desired size(N_{des}), taking a percentage of tolerance(Dif_p). Otherwise, the search terminates if the maximum number of attempts(N_{atpmax}) is reached, or the upper limit(L_u) is less than or equal to lower limit(L_l).

Algorithm 3. createProblem

Input: X_i, X_p, N_{fix}, mip
Output: $mip^{'}$
$mip' \leftarrow fix(mip, N_{fix}, X_p, X_i)$;
$mip'' \leftarrow preprocessed(mip')$;
if $mip''notfeasible$ **then**
  $mip'' \leftarrow$ null ;
end
return mip'';

The method createProblem, in algorithm 3, will build mip problems, considering a number of variables to be fixed (N_{fix}), the vector of priorities (X_p). The function $fix(mip, N_{fix}, X_p, X_i)$ will create a new problem, from the fixing of (N_{fix}) first variables of vector (X_p), through the definition of upper and lower limits of each one by the value in X_i. Next, the created problem is preprocessed, generating another (mip''). If the mip (mip'') is viable, the method returns the size of the new problem. However if unfeasible, returns 0 as the size.

4 Characterization of Instances

The models (instances) to be used in this work are all related to binary problems. They were obtained from two groups:

- 25 MIPLIB library problems `http://miplib.zib.de/` [20].
- 12 nurse scheduling problems used in the International Nurse Rostering Competition 2010 [19]

A detailed description of the instances, containing the number of binary variables, the number of constraints and number of non-zero values in constraints, is available in table 1.

Table 1. Nurse Scheduling and MIPLIB Library Instances

instance	binary variables	constraints	non-zero
long01	51,695	17,241	1,011,556
long-hidden01	61,950	28,370	1,064,380
long-hint01	61,550	27,480	1,061,430
long-late01	61,750	27,875	1,062,795
medium01	29,605	8,668	621,829
medium-hidden01	36,690	16,070	635,220
medium-hint01	34,050	14,062	622,800
medium-late01	34,050	14,062	622,800
sprint01	3,522	10,230	204,000
sprint-hidden01	10,308	3,332	202,420
sprint-hint01	11,630	5,032	208,410
sprint-late01	11,630	5,032	208,410
air04	8,904	823	72,965
bley-xl1	5,831	175,620	869,391
cov1075	120	637	14,280
eil33-2	4,516	32	44,243
eilB101	2,818	100	24,120
iis-100-0-cov	100	3,831	22,986
iis-bupa-cov	345	4,803	38,392
iss-pima-cov	768	7,201	71,941
macrophage	2,260	3,164	9,492
mine-166-5	830	8,429	19,412
mine-90-10	900	6,270	15,407
n3div36	22,120	4,484	340,740
n3seq24	119,856	6,044	3,232,340
neos-1109824	1,520	28,979	89,528
neos-1337307	2,840	5,687	30,799
neos18	3,312	11,402	24,614
netdiversion	129,180	119,589	615,282
ns1688347	2,685	4,191	66,908
opm2-z7-s2	2,023	31,798	79,762
reblock67	670	2,523	7,495
rmine6	1,096	7,078	18,084
sp98ic	10,894	825	316,317
tanglegram1	34,759	68,342	205,026
tanglegram2	4,714	8,980	26,940
vpphard	51,471	47,280	372,305

We can see the diversity of binary problems that will be used, with this number of constraints and variables quite different.

5 Experiments and Results

The proposed heuristic in this work reads a series of test instances (initially applied to binary problems, as described in the previous section), the fractional solution and an initial integer feasible solution to the problem. This initial solution was generated using Feasibility Pump and is informed as initial solution for all tested MIP heuristics.

For most of the practical operations research applications, solution methods are only useful if they are able to produce satisfatory solutions in short periods of time. Thus, we imposed a time limit of 300 seconds. The initial size parameter of the problem to be solved in these tests was defined as 100 (the number of free variables). Another parameter is the percentage of increase in the size of the problem(in this case, it was used 50). The tests were run using the following hardware configuration: Intel (R) Core (TM) i7 CPU, 1.90GHz, 6 GB RAM.

Preliminary results described below, consider the implementation of the method RINS adapted as proposed in this work. The instances used in the tests are described in Table 1.

We compare our results with the standalone CBC solver as well, with an implementation method RINS in original form. Both CBC and RINS use the same initial solution that our method uses.

In Table 2 the values found for each instances are described, considering the implementations used. For each instance, the best value found is highlighted in bold. Comparing the values obtained, we can notice that the proposed method achieves better results on 14 instances when compared to CBC and is better in 18 instances when compared to the original form of RINS. In 15 instances, the method obtains the same value achieved when the problem is solved using the CBC, while the result is the same in 15 cases when compared to the original RINS.

Tables 3 and 4 present the results considering the number of wins, draws and defeats comparing the implementations tested. In all groups of instances, the proposed method has achieved good results, reaching better solutions than or equal to the comparison method. For example, the group of MIPLib instance, the pRINS obtains 76% of equal or better values when compared to CBC, and 88% compared to RINS.

Table 5 shows the results for each tested implementation considering the sum and the average of gaps (percentage difference between the obtained value and the best known value). The proposed method was the one that was closest, on average, to the best values. The metric gap solution was calculated according to the following expression: $\min(100, (z - best) \div best \times 100)$.

Table 2. Results

Instances	pRINS		CBC		RINS		Best
	z	gap %	z	gap %	z	gap %	z
long01	**161**	0.0	**161**	0.0	**161**	0.0	161
long-hidden01	2,745	100.0	2,747	100.0	**2,606**	100.0	346
long-hint01	**74**	0.0	**74**	0.0	**74**	0.0	74
long-late01	**2,895**	100.0	**2,895**	100.0	**2,895**	100.0	235
medium01	**374**	0.0	383	2.4	383	2.4	374
medium-hidden01	**1,287**	100.0	**1,287**	100.0	**1,287**	100.0	111
medium-hint01	**245**	0.0	255	4.0	250	2.0	245
medium-late01	**779**	100.0	970	100.0	966	100.0	157
sprint01	91	22.9	**74**	0.0	96	29.7	74
sprint-hidden01	92	100.0	**80**	100.0	99	100.0	32
sprint-hint01	**297**	0.0	400	34.6	496	67.0	297
sprint-late01	**74**	0.0	81	9.4	84	13.5	74
air04	**56,137**	0.0	56,138	0.0	60,093	7.0	56,137
bley-xl1	230	21.0	230	21.0	**215**	13.1	190
cov1075	**20**	0.0	**20**	0.0	**20**	0.0	20
eil33-2	1,011	8.2	**1,000.24**	7.0	1,858	98.9	934
eilB101	2,093	72.1	**1,529.88**	25.7	2,298	88.9	1216
iis-100-0-cov	30	3.4	**29**	0.0	32	10.3	29
iis-bupa-cov	**37**	2.7	38	5.5	38	5.5	36
iss-pima-cov	34	3.0	34	3.0	**33**	0.0	33
macrophage	**447**	19.5	809	100.0	596	59.3	374
mine-166-5	**-3.98740e+08**	29.6	**-3.98740e+08**	29.6	**-3.98740e+08**	29.6	-5.66396e+08
mine-90-10	**-3.98740e+08**	49.1	**-3.98740e+08**	49.1	**-3.98740e+08**	49.1	-7.84302e+08
n3div36	**136,000**	3.9	148,600	13.6	149,800	14.5	130,800
n3seq24	73,200	40.2	**62,800**	20.3	69,400	32.9	52,200
neos-1109824	467	23.5	**380**	0.5	760	100.0	378
neos-1337307	-201,447	0.4	**-201,466**	0.4	-201,447	0.4	-202319
neos18	**16**	0.0	**16**	0.0	**16**	0.0	16
netdiversion	**370**	52.8	451	86.3	451	86.3	242
ns1688347	**35**	29.6	**35**	29.6	**35**	29.6	27
opm2-z7-s2	-1,317	87.1	**-9,365**	8.9	-1,317	87.1	-10,280
reblock67	**-2.19704+e07**	36.5	**-2.19704+e07**	36.5	**-2.19704+e07**	36.5	-3.46306+e07
rmine6	**-449.25**	1.7	**-449.25**	1.7	**-449.25**	1.7	-457.18
sp98ic	**4.50307+e08**	0.2	4.87071+e08	8.4	4.68947+e08	4.4	4.49145+e08
tanglegram1	**5,494**	6.0	**5,494**	6.0	**5,494**	6.0	5182
tanglegram2	**443**	0.0	**443**	0.0	**443**	0.0	443
vpphard	**16**	68.7	22	100.0	22	100.0	5

Table 3. Number of victories and defeats for each group of instances

Group	Best Value		
	pRINS	Equal	CBC
MIPLIB library	8	11	6
nurse scheduling	6	4	2

Table 4. Number of victories and defeats for each group of instances

Group	Best Value		
	pRINS	Equal	RINS
MIPLIB library	11	11	3
nurse scheduling	7	4	1

Table 5. Sum and average of gaps

	p-RINS	CBC	RINS
Sum	1,082.10	1,103.52	1,475.70
Average	29.24	29.82	39.88

6 Final Remarks

Even tough this work is still being developed, encouraging results were obtained for the proposed RINS variant, denoted here as pRINS. Adjustments are being made in the algorithm to speed up the execution of multiple pre-processing phases, which can produce further speedups. Our computational results show that pRINS is already better or equal to other methods (using only CBC or original RINS) in most cases, considering the production good feasible solutions in a restricted period of time.

References

[1] Glover, F.: Tabu Search and adaptive memory programming - advances, applications and challenges. In: Interfaces in Computer Sciences and Operations Research, pp. 1–75 (1996)

[2] Reeves, C.R.: Genetic Algorithms Modern Heuristic Techniques for Combinatorial Problems. Advanced Topics in Computer Science Series, ch. 4, pp. 151–196. Blackwell Scientific Publications (1993)

[3] Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley (1989)

[4] Glover, F.: Future paths for Integer Programming and links to Artificial Intelligence. COR 13(5), 533–549 (1986)
[5] Blum, C., Puchinger, J., Raidl, G., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing 11, 4135–4151 (2011)
[6] Eckstein, J., Nediak, M.: Pivot, Cut, and Dive: a heuristic for 0-1 mixed integer programming. Journal Heuristics 13, 471–503 (2007)
[7] Parisini, F., Milano, M.: Improving CP-based Local Branching via Sliced Neighborhood Search. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 887–892 (2011)
[8] Ghosh, S.: DINS, a MIP Improvement Heuristic. In: Fischetti, M., Williamson, D.P. (eds.) IPCO 2007. LNCS, vol. 4513, pp. 310–323. Springer, Heidelberg (2007)
[9] Berthold, T.: Rens: The Relaxation Enforced Neighborhood Search (2009)
[10] Ferreira, D., Morabito, R., Rangel, S.: Relax and fix heuristics to solve one-stage one-machine lot-scheduling models for small-scale soft drink plants. Computers and Operations Research 37, 684–691 (2009)
[11] Danna, E., Rothberg, E., Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions 102, 71–90 (2005)
[12] Fischetti, M., Bertacco, L., Lodi, A.: A feasibility pump heuristic for general mixed-integer problems. Technical report, Università di Bologna D.E.I.S. Operations Research (2005)
[13] Fischetti, M., Glover, F., Lodi, A.: The feasibility pump. Mathematical Programming 104(1), 9104 (2005)
[14] Fischetti, M., Lodi, A.: Local branching. Mathematics Programming, ser. B 98, 23–47 (2003)
[15] Rothberg, E.: An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions. INFORMS Journal on Computing 19(4), 534–541 (2007)
[16] Forrest, J., Lougee-Heimer, R.: INFORMS Tutorials in Operations Research. CBC User Guide, pp. 257–277 (2005)
[17] Lougee-Heimer, R.: The Common Optimization Interface for Operations Research: Promoting open-source software in the operations research community. IBM Journal of Research and Development 47(1), 57–66 (2003)
[18] Mladenovic, N., Hansen, P.: Variable Neighborhood Search. Computers and Operations Research 24, 1097–1100 (1997)
[19] Haspeslagh, S., De Causmaecker, P., Stolevik, M., Schaerf, A.: First international nurse rostering competition 2010. CODeS, Department of Computer Science. KULeuven Campus Kortrijk, Belgium (2010)
[20] Koch, T., Achterberg, T., Andersen, E., Bastert, O., Berthold, T., Bixby, R., Danna, E., Gamrath, G., Gleixner, A., Heinz, S., Lodi, A., Mittelmann, H., Ralphs, T., Salvagnin, D., Steffy, D., Wolter, K.: MIPLIB 2010. Mathematical Programming Computation. Mathematics and Statistics 3, 103–163 (2011), http://dx.doi.org/10.1007/s12532-011-0025-9

A Hybrid Simulated Annealing Algorithm for Location of Cross-Docking Centers in a Supply Chain

S.M. Mousavi[1], R. Tavakkoli-Moghaddam[1], A. Siadat[2], and B. Vahdani[1]

[1] Dep. of Industrial Engineering, College of Engineering, University of Tehran, and National Elite Foundation Tehran, Iran
{sm.mousavi,tavakoli,b.vahdani}@ut.ac.ir
[2] LCFC, Arts et Métier Paris Tech, Metz, France
ali.siadat@ensam.eu

Abstract. In this paper, cross-docking centers are designed at a strategic level for distribution planning in a supply chain. To make the strategic decision, a zero-one programming (ZOP) model is presented for the location to determine the minimum number of cross-docks among a set of location centers so that each retailer demand should be met. Then, a hybrid simulated annealing (HSA) algorithm embedded with tabu search (TS) is proposed to solve the presented model. A number of test problems in small and large sizes are examined to illustrate the performance of the proposed HSA algorithm in terms of the solution quality and computational time. Moreover, its efficiency is compared with the classical SA and TS algorithms in detail. Finally, computational results demonstrate that the proposed HSA algorithm outperforms the two classical algorithms and converges fast to high-quality solutions.

Keywords: Supply chain, Location, Cross-docking centers, Hybrid simulated annealing, Tabu search.

1 Introduction

Cross-docking warehousing systems involve holding no inventory in the warehousing center but simply moving them through the warehousing center when it arrives from suppliers, orders are disaggregated and sent to different retailers based on their orders [1]. In supply chain management, the location of cross-docking centers has an important role for the strategic planning. In fact, the cross-docking location problem includes a set of service facilities to serve a set of retailers which have to be located according to one or several objective functions by considering the interaction between retailers' demands and cross-docking centers.

In the last decade, there were some studies considering the location of cross-docking center problems. Ratliff et al. [2] addressed a load-driven network problem in a railroad context in order to determine the number and location of the mixing centers by minimizing the total delay in the automotive delivery system. Donaldson et al. [3] studied on schedule-driven transportation planning in the design of cross-docking distribution networks. Jayaraman and Ross [4] introduced a two-stage network

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 12–21, 2013.

planning that have determined the location of cross-docks and distribution centers in a supply chain network, and then the proposed model solved by a simulated annealing (SA) algorithm under various scenarios. Sung and Song [5] presented an integer programming model to provide the optimal location of cross-docking and the allocation of vehicles in the context of service network by minimizing the cost of locating cross-docking and the cost of allocating vehicles. Then, the model is solved by a tabu search (TS) algorithm. Makui et al. [1] addressed a cross-docking location allocation problem by considering on the classical capacitated facility location presented by Klose et al. [6] and the analytic hierarchy process (AHP) method. Ross and Jayaraman [7] continued the previous study [4] and provided an evaluation of the meta-heuristics solution. Lim et al. [8] extended the traditional transshipment problem that involved a number of supply, transshipment and demand nodes. Ma et al. [9] addressed the shipment consolidation and transportation problem in cross-docking systems. In this study, the setup cost and time window constraint were taken into consideration.

In this paper, the location problem is formulated for multi-period multi-cross-dock to determine the minimum number of cross-docks among a set of location centers by assigning suppliers to cross-docks and assigning cross-docks to retailers so that three types of costs are minimized. These costs include a fixed cost for cross-docks, operating cost for cross-docks to handle product, and transportation cost for moving the product from suppliers to cross-docks and from cross-docks to retailers. By considering the uncertainties in this cross-docking warehousing system, and the diversification of demands as well as the needed quantities in the pickup and delivery nodes, this paper aims to present a new multi-period location model at strategic level by introducing a zero-one programming (ZOP) method in order to show the results to be close to real-life applications in the long-term planning.

The outline of this paper is as follows. The location problem is defined and then concerned formulation of the presented model is demonstrated in Section 2. In Section 3, the proposed HSA algorithm is presented as the problem-solving approach. Computational experiments are provided in Section 4. Finally, conclusions are provided in Section 5.

2 Problem Description and Formulation

The problem considered in this paper is depicted in Fig. 1. In this figure, a white cycle represents a supplier (i.e., pickup node), and a black cycle represents a retailer (i.e., delivery node). The limitations are as follows. Each demand of a pickup or delivery node should be satisfy in each period by one of the potential cross-docking centers. The total quantity of pickup (supply quantities) should equal the quantity to be delivered (total demands). The number of cross-docks should be less than R.

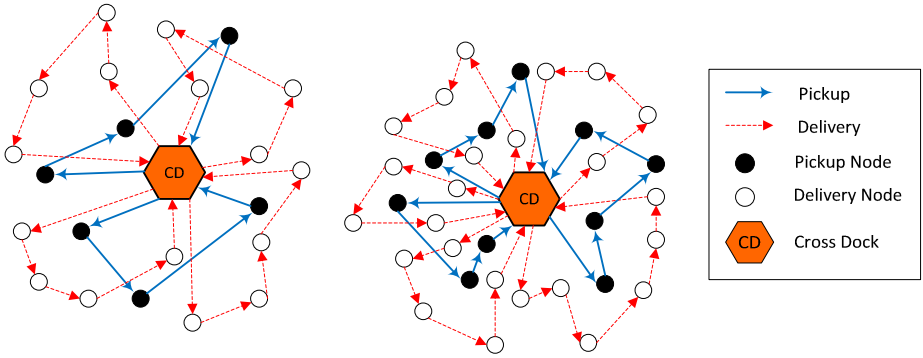


Fig. 1. Proposed network for multiple cross-docking centers

2.1 Proposed Model

The following notations are used in the formulation of the ZOP model for the location problem of multiple cross-docking centers.

Sets and input parameters:

P Set of suppliers in the pickup process
O Set of cross-dock centers
D Set of retailers in the delivery process
T Set of periods
c_{ipt} Cost to transport product from supplier i to cross-dock center p in period t
$c_{pi't}$ Cost to transport product from cross-dock center p to retailer i' in period t
OC_{pt} Operating cost at cross-dock center p in period t
F_p Fixed cost to open cross-dock p
CA_p Capacity of cross-dock center p to handle product
$D_{i't}$ Demand of retailer i' in period t
S_{it} Quantity of product from supplier i in period t
R Maximum number of cross-dock centers to be opened

Decision variables:

$$x_{ipt} = \begin{cases} 1 & \text{if supplier } i \text{ is assigned to cross} - \text{dock } p \text{ for product in period } t, \\ 0 & \text{otherwise,} \end{cases}$$

$$y_{pi't} = \begin{cases} 1 & \text{if cross} - \text{dock } p \text{ is assigned to retailer } i' \text{ for product in period } t, \\ 0 & \text{otherwise,} \end{cases}$$

$$z_p = \begin{cases} 1 & \text{if cross} - \text{dock } p \text{ is open,} \\ 0 & \text{otherwise,} \end{cases}$$

In terms of the above notations, the location problem with multiple cross-docking centers is formulated as follows:

$$\text{Min } Z = \sum_{t=1}^{T}\sum_{p=1}^{O}(F_p + OC_{pt})\, z_p + \sum_{i=1}^{n}\sum_{p=1}^{O}\sum_{t=1}^{T} c_{ipt}x_{ipt} + \sum_{p=1}^{O}\sum_{i'=1}^{m}\sum_{t=1}^{T} c_{pi't}y_{pi't} \quad (1)$$

s.t.

$$\sum_{p=1}^{O} y_{pi't} = 1 \qquad \forall(i' = 1,2,\dots,m) \text{ and } \forall(t = 1,2,\dots,T) \quad (2)$$

$$\sum_{p=1}^{O} x_{ipt} = 1 \qquad \forall(i = 1,2,\dots,n) \text{ and } \forall(t = 1,2,\dots,T) \quad (3)$$

$$\sum_{p=1}^{O} z_p \le R \quad (4)$$

$$\sum_{i=1}^{n}\sum_{t=1}^{T} S_{it}\, x_{ipt} \le CA_p z_p \quad \forall(p = 1,2,\dots,O) \quad (5)$$

$$\sum_{i'=1}^{m}\sum_{t=1}^{T} D_{i't}\, y_{pi't} \le CA_p z_p \quad \forall(p = 1,2,\dots,O) \quad (6)$$

$$x_{ipt}, y_{pi't}, z_p \in \{0,1\} \quad \forall(i = 1,2,\dots,n), \forall(p = 1,2,\dots,O), \forall(i' = 1,2,\dots,m), \text{and } \forall(t = 1,2,\dots,T) \quad (7)$$

The objective function (1) minimizes the fixed cost to open cross-docking centers, the operating cost for cross-docking centers to handle product, and costs to transport a product from suppliers to cross-docking centers in the pickup process as well as costs to supply the product from cross-docking centers to satisfy the demand of retailers in the delivery process. Constraint (2) ensures that the demand of each retailer in the delivery process during each period is only satisfied by one of the open cross-docking centers. Constraint (3) ensures that the quantity of product from each supplier during each period is only satisfied by one of the open cross-docking centers. Constraint (4) limits the number of cross-docks that can be located. Constraint (5) and (6) ensure that the quantity or demand of product from each supplier or retailer during all periods should be equal/ less than capacities of open cross-docking centers in the pickup process. Also, these constraints ensure that transporting product from suppliers to cross-docking center, and from cross-docking center to retailers in the pickup and delivery processes can be performed only when the corresponding cross-docking center is open. Finally, constraint (7) enforces the binary restrictions on the decision variables.

3 Proposed Hybrid Meta-heuristic Algorithm

The proposed hybrid meta-heuristic algorithm is presented for the location problem based on the hybridization of two famous algorithms, namely SA and TS. The proposed hybrid simulated annealing (HSA) algorithm has a number of advantages, including stochastic feature avoiding cycling and tabu list to escape from local optima. These characteristics limit the search from a previously visited solution and improve the performance of classical SA properly. The SA is regarded as a random

search optimization algorithm. It was first introduced by Metropolis et al. [10] and popularized by Kirkpatrick et al. [11]. The algorithm works based on the annealing process that is applied to the metallurgical industry. In addition, the TS is regarded as a local search algorithm applied to combinatorial optimization problems. It was first introduced by Glover [12]. The algorithm is able to escape the local optima occurred during the search via the list of prohibited neighboring solutions, called tabu list. Both algorithms (i.e., SA and TS) have been used in a wide variety of conventional and practical optimization problems in real-life applications [e.g., 7, 12-14].

The steps of the proposed HSA algorithm with a tabu list taken from TS are described for the optimal location problem. The search is conducted for least-cost solutions by a control parameter, called temperature and the cooling schedule that determines the number of iterations (i.e., epochs) for the algorithm. A randomly generated initial configuration is first regarded in the proposed HSA algorithm that denotes the cross-docking centers to be opened, the suppliers and retailers assigned to the cross-docking centers. Then, the total cost is calculated by the objective function in the proposed ZOP model.

Step 1: Initialization. Initial and final values are taken into account for the control parameter temperature, known as T_0 and T_f, respectively, and i is the number of a particular iteration and N is the total number of iterations. An initial cross-docking center solution is randomly obtained by allocating adequate supply of suppliers and demand flows of retailers between cross-docking centers and delivery nodes in the cross-docking distribution network. It leads to an initial feasible solution that involves the product flows. The objective function value for the solution can be regarded as the objective function value for the best configuration obtained (BS), current configuration $OBF(x_c)$ and the newest configuration $OBF(x_a)$. All counters are set to 1.

Step 2: Check feasibilities. The algorithm investigates product flow assignments for cross-docking centers to assure the capacity of cross-docking, fixed costs and number of potential cross-docking centers. Furthermore, the quantity of the product and demand of retailers should be considered to be met. If the configuration is not feasible, we return to Step 1.

Step 3: Provide a feasible neighboring solution. Once the location problem has been initialized, the objective function value is calculated and feasibility is considered. Then, the current feasible configuration of cross-docking warehousing system is updated by choosing a supplier and reassigning the amount of product between a cross-docking center and supplier. Additionally, this method can be utilized for retailers. It is executed by randomly choosing a supplier and a retailer to perturb. Its flow is randomly allocated to another combination of pickup/cross-docking center /delivery nodes. All feasibilities must be investigated once again. Finally, the value of objective function is obtained for the neighboring solution $OBF(x_a)$.

Step 4: Assess current solution with neighboring solution. If the objective function value for the neighboring solution is higher than the current solution (i.e., $OBF(x_a) > OBF(x_c)$), proceed to Step 5. Otherwise, if the objective function value for the newest configuration enhances the current solution (i.e., $OBF(x_a) < OBF(x_c)$), the

neighboring solution can be regarded as the current solution. Then, this solution is compared to the best solution obtained (BS). If the objective function value for the newest configuration is lower than the best one determined so far (i.e., $OBF(x_a) < BS$), then replace the best solution with this neighboring solution. Proceed to Step 8.

Step 5: Investigate Metropolis condition. The difference between the neighboring solution and the current solution is calculated, $\Delta\ cost = OBF(x_a) - OBF(x_c)$. Then, the Metropolis criterion is employed to obtain the probability, in which the relatively inferior neighboring solution can be accepted, $P(A)$. This probability is calculated by [7]:

$$P(A) = \exp \quad (\Delta\ cost/T_i), \tag{8}$$

where T_i is the present temperature. Then, a random number is determined from the interval (0, 1). If the random number is lower than $P(A)$, then the neighboring solution is substituted for the current solution. Proceed to Step 8.

Step 6: Tabu list. The tabu list can investigate for each step of the algorithm whether the obtained solution is latterly visited or not. Hence, this leads to the restriction of the algorithm regarding revisiting the pre-visited solutions. This characteristic of the proposed HSA algorithm decreases the CPU time of algorithm to achieve reasonable solutions.

Step 7: Aspiration. Aspiration is linked to the TS. It attempts to restrict the search of the algorithm from being trapped at a solution, which is surrounded by tabu neighbors. If an obtained solution has a neighborhood of the tabu solutions, the solution via the value of objective function higher than the aspiration is selected for further exploring.

Step 8: Increase counters. Memory and variables are updated. The counters can be incremented by one. If the iteration counter value is lower than or equal to the maximum iterations for the temperature level, then return to Step 3. Otherwise, go to Step 9.

Step 9: Adjust temperature. Temperature is adapted in iteration i using the cooling schedule:

$$T_i = \frac{1}{2}(T_0 - T_f)\left(1 - \tanh\left(\frac{10i}{N} - 5\right)\right) + T_f. \tag{9}$$

If the new value of T_i is higher than or equal to the stopping value (T_f), then iteration counters are restarted from one and return to Step 3. Otherwise, the procedure stops.

4 Computational Results

Computational experiments in this section are reported to verify and examine the performance of the proposed HSA algorithm for solving the location problem of cross-docking centers in the cross-docking warehousing system. For this purpose, ten

test problems in a supply chain environment with varying sizes generated at random in small and large-scale cases. Hence, five test problems are solved in small sizes by the branch-and-bound method using the GAMS software for the presented model. Sizes of the test problems are given in Table 1. All parameters are given in Table 2. Some parameters are generated randomly in uniform distributions. It is noteworthy that the SA, TS and proposed HSA are tuned for the test problems based on the authors' experience and related literature. Also, the problem-solving approach by the proposed HSA algorithm is coded in the MATLAB®. All small and large-sized test problems are run by using the Intel Dual Core, 2.8 GHz compiler and 2 GB of RAM.

Table 1. Sizes of small-sized problems

Problem no.	No. of suppliers (m)	No. of potential cross-docking centers (O)	No. of customers (n)
1	4	3	5
2	8	4	7
3	10	5	9
4	11	6	10
5	12	7	12

Table 2. Sources of random generations for the presented location model

Parameters	Problem 1	Problem 2	Problem 3	Problem 4	Problem 5
n	5	7	9	10	12
m	4	8	10	11	12
O	3	4	5	6	7
$D_{i'}$	~Uniform (10,40)	~Uniform (5,45)	~Uniform (5,50)	~Uniform (5,55)	~Uniform (5,60)
CA_p	~Uniform (600,1000)	~Uniform (500,1100)	~Uniform (400,1200)	~Uniform (400,1400)	~Uniform (500,1500)
S_i	~Uniform (10,30)	~Uniform (5,40)	~Uniform (5,45)	~Uniform (5,50)	~Uniform (5,55)
F_p	~Uniform (300,4000)	~Uniform (200, 5000)	~Uniform (200,6000)	~Uniform (100,6500)	~Uniform (50,7000)
c_{ip}	~Uniform (40,300)	~Uniform (50,500)	~Uniform (100,500)	~Uniform (40,550)	~Uniform (30,600)
$c_{pi'}$	~Uniform (25,300)	~Uniform (70,550)	~Uniform (80,520)	~Uniform (70,600)	~Uniform (80,650)
TC	~Uniform (5000,30000)	~Uniform (4000,35000)	~Uniform (3000,40000)	~Uniform (3000,45000)	~Uniform (3000,50000)

For five small-sized test problems, the reported results in these tables are calculated by Eq. (10), which denotes the gap between the optimal solutions and meta-heuristic solutions obtained by the proposed HSA algorithm.

$$\frac{obj_{meta-heuristic}-obj_{optimal\ solution}}{obj_{optimal\ solution}} \times 100. \tag{10}$$

Furthermore, the objective function values, CPU times and the gaps of the objective function values are reported in Table 3 for the given seven small-sized problems that are solved by the proposed HSA meta-heuristic algorithm, the classical SA and TS algorithms and branch-and-bound method using the GAMS software.

The comparison of the branch-and-bound method with the proposed hybrid HSA algorithm illustrates that the HSA can approximately obtain a near-optimal solution in less time than the branch-and-bound method. The average gap between the optimal and meta-heuristic solutions is 4.16% indicating the efficiency of the proposed HSA algorithm supplementary with a tabu list in the cross-docking warehousing system.

Table 3. Results in small-sized test problems for the location of cross docking center problem

No. of test problems	Branch-and-bound method		SA (300 iterations)			TS (300 iterations)			Proposed HSA (300 iterations)		
	Best solution	Time (s)	Best solution	Time (s)	Gap (%)	Best solution	Time (s)	Gap (%)	Best solution	Time (s)	Gap (%)
1	5996.8	2.8	6263.2	8.3	4.44	6388.5	9	6.53	6154.9	7.8	2.64
2	16401.0	7.0	17454.4	9.3	6.42	17105.3	10	4.29	16723.1	8.4	1.96
3	23053.8	12.7	24797.4	12.4	7.56	25789.3	11.4	11.87	24271.1	11.6	5.28
4	25339.2	14.3	27411.6	11.8	8.18	29577.1	12.7	16.72	27009.6	11.4	6.59
5	26935.2	17.9	29145.7	13.2	8.21	29728.6	14.3	10.37	28099.2	12	4.32
Average	19545.2	10.8	21014.4	11.1	6.96	21717.8	11.5	9.96	20451.6	10.2	4.16

In Table 4, the computational results are given in large-sized test problems for the presented ZOP model. The average time of the proposed hybrid HSA algorithm based on the combination of SA and TS for five large-sized test problems in 300 and 500 iterations are 280.7 and 409.1 seconds, respectively. The run time of the proposed HSA meta-heuristic algorithm is acceptable for solving these problems.

In addition, the convergence rate of the proposed hybrid HSA meta-heuristic algorithm is depicted in Fig. 2 for this test problem. Finally, the results illustrate that this algorithm for solving the location problem of cross-docking centers can perform well and converge fast to reasonable solutions.

Table 4. Results in large-sized test problems for the location of cross docking center problem

No. of problems	No. of suppliers	No. of cross-docking centers	No. of customers	SA (500 iterations)		TS (500 iterations)		Proposed HSA (300 iterations)		Proposed HSA (500 iterations)	
				Best solution	Time (s)	Best solution	Time (s)	Best solution	Time (s)	Best solution	Time (s)
1	20	10	25	34149.5	317.5	35173.9	343	33104.1	244.4	32541.9	307.7
2	30	12	30	39716.7	445.2	43291.2	441	39620	268.9	37123	434.2
3	30	15	35	37002.9	506.5	36632.7	545.3	35589.6	299.2	34247.5	447.5
4	40	18	40	60063.1	443.9	58861.8	457.1	51246	329.6	50832.5	432
5	50	20	45	62940.4	473.1	66716.8	484.8	57238	261.4	48847	424.1
Average	34	15	35	46774.5	437.2	48135.3	454.2	43359.5	280.7	40718.4	409.1

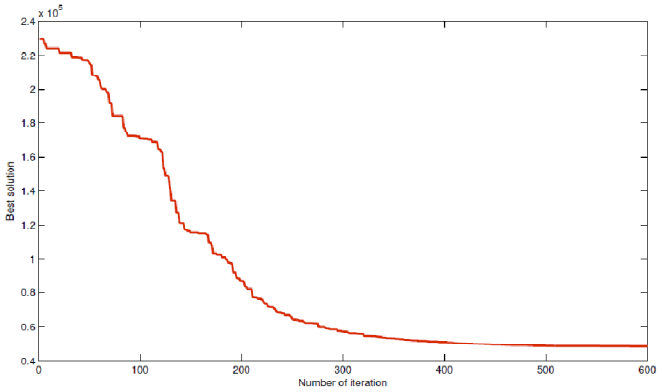


Fig. 2. Convergence rate for the fifth large-sized problem

5 Conclusion

This paper has introduced a zero-one programming (ZOP) model for the location problem of cross-docking centers in a supply chain. Three types of costs have been considered that minimize the fixed cost for cross-docks, the operating cost for cross-docks to handle product and the transportation cost for moving the product from suppliers to cross-docks and from cross-docks to retailers. Then, a hybrid simulated annealing (HSA) algorithm with a tabu list has been proposed to solve the presented ZOP model. In the presented HSA by the combination of simulated annealing (SA) and tabu search (TS), not only the number of solution revisits but also computational time to obtain a near-optimal solution has been considerably decreased. To validate the proposed hybrid HSA algorithm, different test problems have been solved in order to evaluate its performance and reliability in comparison with the classical SA. The computational results have demonstrated that the proposed HSA algorithm can be employed in situations, in which popular commercial solvers cannot solve the optimal location model for large-sized problems in real-life applications for cross-docking warehousing systems.

Acknowledgment. This work has been partially supported by the Center for International Scientific Studies & Collaboration (CISSC) and the French Embassy in Tehran. The authors are also grateful for the financial support from the Égide Program in France.

References

1. Makui, A., Haerian, L., Eftekhar, M.: Designing a multi-objective nonlinear cross-docking location allocation model using genetic algorithm. Journal of Industrial Engineering International 2(3), 27–42 (2006)
2. Ratliff, H.D., Vate, J.V., Zhang, M.: Network design for load-driven cross-docking systems. Technical report, Georgia Institute of Technology (1998), http://www.isye.gatech.edu/research/files/misc9914.pdf

3. Donaldson, H., Johnson, E.L., Ratliff, H.D., Zhang, M.: Schedule-driven crossdocking networks. Technical report, Georgia Institute of Technology (1999), http://www.isye.gatech.edu/apps/research-papers/papers/misc9904.pdf
4. Jayaraman, V., Ross, A.: A simulated annealing methodology to distribution network design and management. European Journal of Operational Research 144, 629–645 (2003)
5. Sung, C.S., Song, S.H.: Integrated service network design for a cross-docking supply chain network. Journal of the Operational Research Society 54(12), 1283–1295 (2003)
6. Klose, A., Drexl, A.: Facility location models for distribution system design. European Journal of Operational Research 162(1), 4–29 (2005)
7. Ross, A., Jayaraman, V.: An evaluation of new heuristics for the location of cross-docks distribution centers in supply chain network design. Computers & Industrial Engineering 55, 64–79 (2008)
8. Lim, A., Miao, Z., Rodrigues, B., Xu, Z.: Transshipment through crossdocks with inventory and time windows. Naval Research Logistics 52(8), 724–733 (2005)
9. Ma, H., Miao, Z., Lim, A., Rodrigues, B.: Cross docking distribution networks with setup cost and time window constraint. Omega 39, 64–72 (2011)
10. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machines. Journal of Chemical Physics 21, 1087–1092 (1953)
11. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
12. Glover, F.: Tabu search: a tutorial. Interfaces 20(4), 74–94 (1990)
13. Ghodratnama, A., Rabbani, M., Tavakkoli-Moghaddam, R., Baboli, A.: Solving a single-machine scheduling problem with maintenance, job deterioration and learning effect by simulated annealing. Journal of Manufacturing Systems 29, 1–9 (2010)
14. Liao, T.W., Egbelu, P.J., Chang, P.C.: Simultaneous dock assignment and sequencing of inbound trucks under a fixed outbound truck schedule in multi-door cross docking operations. International Journal of Production Economics 141, 212–229 (2013)

Intensification/Diversification in Decomposition Guided VNS

Samir Loudni, Mathieu Fontaine, and Patrice Boizumault

Université de Caen Basse-Normandie, UMR 6072 GREYC, F-14032 Caen, France
CNRS, UMR 6072 GREYC, F-14032 Caen, France

Abstract. Tree decomposition introduced by Robertson and Seymour aims to decompose a problem into clusters constituting an acyclic graph. In a previous paper, we have introduced DGVNS (Decomposition Guided VNS) which uses the graph of clusters to manage the exploration of large neighborhoods. In this paper, we go one step further by proposing three new strategies that exploit the graph of clusters enabling a better intensification and diversification in DGVNS. Experiments performed on random instances (GRAPH) and real life instances (RLFAP, SPOT5 and tagSNP) show the appropriateness and the efficiency of our proposals.

1 Introduction

Many real-life problems, such as frequency assignment [4], or the daily management of an earth observation satellite [3], are very large and exhibit a highly structured constraints graph. Exploiting such structural properties may lead these problems to be tractable.

Tree decomposition introduced by Robertson and Seymour [21] aims to decompose a problem into subproblems (called clusters) constituting an acyclic graph. Each cluster corresponds to a subset of variables that are strongly connected. As each subproblem is significantly smaller in size than the original one, it can be solved more efficiently. The interest for exploiting structural properties of a problem has been attested in various domains: for checking satisfiability in SAT [20], for solving CSP [6], in Bayesian or probabilistic networks [17], in relational databases [9], for constraint optimization [5,22,25]). All these proposals exploit tree decomposition for complete search methods.

For local search methods that use large neighborhoods, as *Large Neighborhood Search* (LNS) [23] or *Variable Neighborhood Search* (VNS) [18], the design of neighborhood structures is crucial, since they provide a way to intensify/diversify the search in order to explore promising regions of the search space. In a previous paper [8], we have introduced DGVNS (Decomposition Guided VNS) which uses the *graph of clusters* provided by a tree decomposition of the constraints graph to guide the exploration of large neighborhoods in VNS.

In this paper, we go one step further by proposing three new strategies (DGVNS-1, DGVNS-2 and DGVNS-3) that exploit the graph of clusters enabling a better intensification and diversification in DGVNS. They correspond to different

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 22–36, 2013.

schemes for moving between clusters according to the quality of the new solution found. Our main idea is to ensure a wider coverage of different parts of the problem associated to clusters of the tree decomposition, whilst focusing more intensively within promising regions. Experiments performed on random instances (GRAPH) and on real life instances (RLFAP, SPOT5 and tagSNP, see Section 5.2) show that achieving a better compromise of intensification and diversification, as performed by DGVNS-2, leads to better results compared to DGVNS-1 and DGVNS-3. Moreover, comparisons made with VNS/LDS+CP [16] and ID-Walk [19], one of the most performing local search methods on RLFAP instances (see Section 5.3) show that our approaches (i.e. DGVNS-1 and DGVNS-2) are very effective.

Section 2 introduces the context. Section 3 details our strategies for intensification and diversification in DGVNS. Section 4 presents the problem instances we used for our experiments. Section 5 is devoted to experimentations. Finally, we conclude and draw some perspectives.

2 Context and Definitions

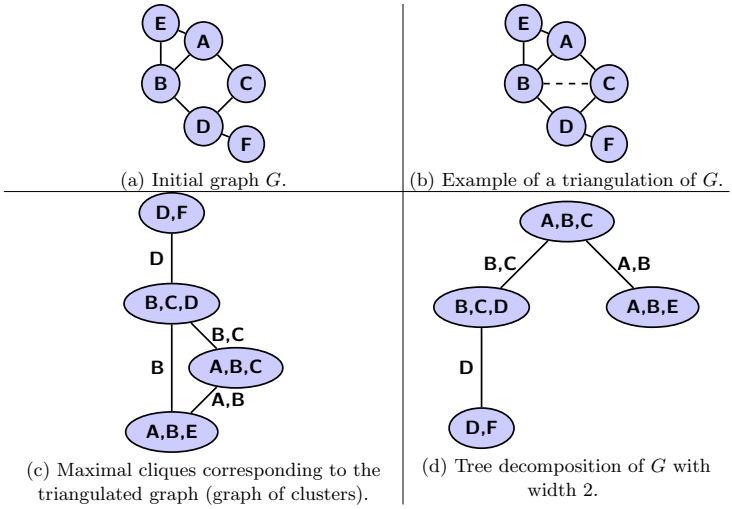
First, we recall the definition of a Cost Functions Network, the framework we have retained for modeling all problems considered for our experiments (see Section 4). Then, we present the MCS tree decomposition method that relies on the concept of graph triangulation. Finally, we detail the DGVNS method.

2.1 Cost Functions Network

A *Cost Functions Network* (CFN) [14] is a generic framework used to model and solve constrained optimization problems which allows to deal with over-constrained problems. It is defined as a pair (X, W) where $X = \{x_1, \ldots, x_n\}$ is a set of n variables (with a maximum domain size d) and W is a set of e cost functions. Each variable $x_i \in X$ has a finite domain D_i of values that can be assigned to it. A value a in D_i is denoted (x_i, a). For a set of variables $S \subseteq X$, D^S denotes the cartesian product of the domains of the variables in S. A *complete* assignment $t{=}(a_1, ..., a_n)$ is an assignment of all variables; on the contrary, it will be called a *partial* assignment. For a given complete assignment t, $t[S]$ denotes the projection of t over S. A cost function $w_S \in W$, with scope $S \subseteq X$, is a function $w_S : D^S \mapsto [0, k_\top]$ where, $k_\top$ is a maximum integer cost (finite or not) used to represent forbidden assignments (expressing hard constraints). Costs are combined using the bounded addition defined by $\alpha \oplus \beta = \min(k_\top, \alpha + \beta)$. Solving a CFN consists in finding a complete assignment t minimizing $\oplus_{w_S \in W} w_S(t[S])$.

2.2 Tree Decomposition

The constraints graph of a CFN is a graph $G{=}(X, E)$ with one vertex for each variable and one edge (u, v) for every cost function $w_S \in W$, such that $u, v \in S$.



(a) Initial graph G. (b) Example of a triangulation of G.

(c) Maximal cliques corresponding to the triangulated graph (graph of clusters).

(d) Tree decomposition of G with width 2.

Fig. 1. Steps for computing a tree decomposition of a graph G

Definition 1. *A tree decomposition [21] of G=(X,E) is a pair* (C_T, T) *where:*

- $T = (I, A)$ *is a tree with nodes set* I *and edges set* A,
- $C_T = \{C_i \mid i \in I\}$ *is a family of subsets of* X *(called clusters) such that:*
  - $\cup_{i \in I} C_i = X$,
  - $\forall (u, v) \in E$, $\exists C_i \in C_T$ *s.t.* $u, v \in C_i$,
  - $\forall i, j, k \in I$, *if* j *is on the path from* i *to* k *in* T, *then* $C_i \cap C_k \subseteq C_j$.

Definition 2. *The intersection of two clusters* C_i *and* C_j *is called a separator, and noted* $sep(C_i, C_j)$. *Two clusters* C_i *and* C_j *are adjacent if* $sep(C_i, C_j) \neq \emptyset$. *Variables belonging to one, and only one, cluster are called proper variables.*

Definition 3. *A graph of clusters for a tree decomposition* (C_T, T) *is an undirected graph* $G_T = (C_T, E_T)$ *that has a vertex for each cluster* $C_i \in C_T$, *and there is an edge* $(C_i, C_j) \in E_T$ *when* $sep(C_i, C_j) \neq \emptyset$. *The edges are labeled by the shared variables.*

Definition 4. *The width of a tree decomposition is defined as* $w^- = \max_{i \in I}(|C_i| - 1)$. *The treewidth* $tw(G)$ *of a graph* G *is defined as the smallest width of all possible tree decompositions of* G.

As finding an optimal tree decomposition is NP-hard [2], approximate tree decompositions using *triangulation* of a given graph are often exploited. Several effective heuristics that rely on the notion of graph *triangulation* have been proposed (see [13] for an introduction to triangulated graphs). Such heuristics provide upper bounds for the treewidth.

Computing a tree decomposition for a graph is equivalent to finding a triangulation of this graph, i.e. finding a suitable set of edges to add to the graph to obtain a chordal graph [13].

Algorithm 1. Pseudo-code of DGVNS

```
   function DGVNS(X, W, k_init, k_max, δ_max);
   begin
1      let G be the constraints graph of (X, W) ;
2      let (C_T, T) be a tree decomposition of G ;
       let C_T = {C_1, C_2, ..., C_p} ;
3      S ← genInitSol() ;
4      k ← k_init ;
5      i ← 1 ;
6      while (k < k_max) ∧ (notTimeOut) do
7          C_s ← CompleteCluster(C_i, k) ;
8          X_un ← Hneighborhood(C_s, N_{k,i}, S) ;
9          A ← S\{(x_i, a) | x_i ∈ X_un} ;
10         S' ← Rebuild(A, X_un, δ_max, f(S), S) ;
11         NeighbourhoodChangeDGVNS(S, S', k, i);
12     return S ;
   end
```

Fig. 1 depicts the three steps for computing a tree decomposition of a graph G (see Part a). First, triangulation is performed on G by adding edge BC (see Part b). Then, maximal cliques in the chordal graph are determined in order to build the graph of clusters (see Part c). Finally, tree decomposition is achieved (see Part d). In this paper, we have used the heuristic called *Maximum Cardinality Search* (MCS) [24]. This heuristic provides a good compromise between the width of the tree decomposition and the time required to compute it [13].

2.3 Decomposition Guided VNS (DGVNS)

DGVNS (Decomposition Guided VNS) [8] extends the Variable Neighborhood Decomposition Search (VNDS[1]) method [11], by exploiting the graph of clusters in order to guide the exploration of large neighborhoods. Neighborhoods are obtained by unfixing a part of the current solution according to a neighborhood heuristic. Then the exploration of the search space, related to the unfixed part of the current solution, is performed by a partial tree search LDS (*Limited Discrepancy Search*, [12]) with Constraint Propagation (CP).

Definition 5 (Neighborhood Structure $N_{k,i}$). *Let G be a constraint graph and G_T=($\mathcal{C}_T$,E_T) its associated graph of clusters. Let $C_i \in \mathcal{C}_T$ be a cluster of G_T and k the neighborhood dimension. $N_{k,i}$ denotes the set of all subsets of k variables from C_i.*

Figure 1 depicts the pseudo-code of DGVNS. It starts from a tree decomposition of G (line 2) and from an initial solution S which is randomly generated

[1] VNDS extends the basic VNS into a two-level VNS scheme based upon decomposition of the problem.

(line 3). To favor moves on regions that are closely linked, DGVNS uses neighborhood structures $N_{k,i}$ (see Definition 5). Indeed, the concept of cluster embodies this criterion, because of its size (smaller than the original problem), and by the strong connection of the variables it contains. Thus, the set of candidate variables C_s to be unassigned are selected from cluster C_i. If $(k > |C_i|)$, then we complete C_s by adding the clusters C_j adjacent to C_i in order to take into account the topology of the graph of clusters. This treatment is achieved by function CompleteCluster(C_i, k) (line 7). Moreover, thank to the strong connection of the selected variables, the rebuilding step will benefit from an effective pruning and a better computing of lower bounds. A subset of k variables X_{un} is randomly selected in C_s among conflicted ones[2] by the neighborhood heuristic Hneighborhood (line 8). Such a heuristic which is mainly based on random choices, allows to diversify the search. A partial assignment $\mathcal{A}$ is generated from the current solution S by unassigning the k selected variables; the $(n-k)$ non-selected variables keep their current value in S (line 9). Then, unassigned variables are rebuilt by a partial tree search (LDS) combined with Constraint Propagation (CP) (line 10). The search stops when the maximal dimension size allowed or the $TimeOut$ is reached (line 6). Function NeighborhoodChangeDGVNS determines the way to intensify and diversify the search according to the quality of the new solution found S' (line 11). It will be detailed in the next section.

3 Strategies for Intensification/Diversification in DGVNS

We propose three new strategies that exploit the graph of clusters enabling a better intensification and diversification in DGVNS. They correspond to different schemes for moving between clusters according to the quality of the new solution found. Our aim is to ensure a wider coverage of different parts of the problem, whilst focusing more intensively within promising regions. In the following, we denote by DGVNS-i, the DGVNS method using the function NeighborhoodChangeDGVNS-i.

First, we discuss the role of intensification and diversification for local search methods. Then, we detail our strategies for intensification/diversification.

3.1 Intensification versus Diversification

One of the most crucial points which greatly impacts the performance of local search methods is their capability to exhibit high intensification and high diversification. *The aim of intensification* is to focus more intensively within a relatively small region to converge towards a local optimum, while *the goal of diversification* is to sample a large number of different regions to ensure that the search space has been properly explored, and to locate the region containing the global optimum. In DGVNS, intensification is performed by rebuilding the partial solutions using LDS+CP and by resetting the size of the neighborhood to

[2] A variable is said to be conflicted if it occurs in at least one unsatisfied constraint.

(*a*)

Procedure NeighborhoodChangeDGVNS-1(S, S', k, i);

```
(1)  begin
(2)     if f(S') < f(S) then
(3)        S ← S';
(4)        k ← k_init;
(5)        i ← succ(i);
(6)     else
(7)        k ← k + 1;
(8)        i ← succ(i)
(9)     endif
(10)    return ;
(11) end
```

(*b*)

Procedure NeighborhoodChangeDGVNS-2(S, S', k, i);

```
(1)  begin
(2)     if f(S') < f(S) then
(3)        S ← S';
(4)        k ← k_init;
(5)     else
(6)        k ← k + 1;
(7)        i ← succ(i)
(8)     endif
(9)     return;
(10) end
```

(*c*)

Procedure NeighborhoodChangeDGVNS-3(S, S', k, i);

```
(1)  begin
(2)     if f(S') < f(S) then
(3)        S ← S';
(4)        k ← k_init;
(5)        i ← succ(i);
(6)     else
(7)        k ← k + 1;
(8)     endif
(9)     return;
(10) end
```

Fig. 2. Strategies for intensification/diversification in DGVNS

k_{init} at each improvement. This will accelerate the search for complete assignments in small neighborhoods. The diversification is ensured by enlarging the neighborhood dimension (i.e. moving from k to $(k+1)$).

However, as stated in [15], most local search methods handle diversification and intensification as two opposite objectives: as one gets more intensification, one can loose diversification. So, more coordination is required between these two main components.

3.2 DGVNS-1: Move Systematically to the Next Cluster

DGVNS-1 considers successively all the C_i. Fig. 2 (Part (a)) depicts the pseudo-code of the function NeighborhoodChangeDGVNS-1. Let p be the total number of clusters, *succ* a successor function[3], and $N_{k,i}$ the current neighborhood structure. If LDS+CP fails to find a solution of better quality S' in the neighborhood of S, DGVNS-1 looks for improvements in $N_{(k+1),succ(i)}$ (neighborhood structure where $(k+1)$ variables of C_s will be unassigned) (lines 7-8). First, diversification performed by moving from cluster C_i to cluster $C_{succ(i)}$ enables to favor the search to visit new parts of the search space and to try to find higher quality solutions

[3] if $i < p$ then $succ(i) = i + 1$ else $succ(p) = 1$.

that reside elsewhere. Second, when a local minimum is found in the current neighborhood, moving from k to $(k+1)$ will also provide some diversification by enlarging the neighborhood size.

However, when a solution of better quality S' is found by LDS+CP in the current neighborhood $N_{k,i}$, we reset k to k_{init} (line 4), and we consider the next cluster (line 5). Indeed, remaining in the same cluster makes it more difficult to improve: selecting a new cluster will enable to diversify the exploration around the new solution S'. Clearly DGVNS-1 favors a more "aggressive" diversification of the search space.

3.3 DGVNS-2: Move to the Next Cluster If No Improvement Is Made

The pseudo-code of function NeighborhoodChangeDGVNS-2 is depicted Fig. 2 (Part (b)). In this strategy, we move to cluster $C_{succ(i)}$ if no improvement of the current solution is made (lines 6-7). However, unlike to DGVNS-1, if LDS+CP finds a solution of better quality S' in the neighborhood of S (line 2), k is reset to k_{init} (line 4), and DGVNS-2 looks for new improvements in $N_{k_{init},i}$ (lines 3-4). This enables to intensify the exploration in the neighborhood of S' in subsequent iterations in the rebuilding step of DGVNS-2. First, remaining in the same cluster will enable to propagate, over variables of C_s, the consequences of the new reassignments of variables in S'. Second, resetting k to k_{init} will favor search to perform small moves within the neighborhood of S'.

Contrary to DGVNS-1, DGVNS-2 seems to favor a balance between intensification and diversification. Indeed, as long as no improvement is made, DGVNS-2 considers successively all the C_i (i.e. diversification effort), but as soon as a solution is improved, DGVNS-2 switches to an intensification scheme.

3.4 DGVNS-3: Move to the Next Cluster after Each Improvement

To evaluate the impact of our diversification process by considering $C_{succ(i)}$ as the next cluster, we propose a third strategy DGVNS-3 which consists in remaining in the same cluster as long as no improvement is performed (see the pseudo-code depicted in Fig. 2, Part (c)). DGVNS-3 restricts the diversification effort by only increasing the neighborhood dimension (line 7). As for DGVNS-2, when a solution of better quality S' is found, DGVNS-3 looks for improvements in $N_{k_{init},succ(i)}$ (lines 4-5). This strategy can be seen as a kind of compromise between the two previous ones.

4 Benchmark Problems

Experiments have been performed on instances of four different problems modeled as CFNs (see Section 2.1).

RLFAP Instances: The CELAR (Centre d'Electronique de l'Armement) has made available a set of instances for the Radio Link Frequency Assignment Problem (RLFAP) [4]. They consist in assigning a limited number of frequencies to a set of radio links defined between pairs of sites, in order to minimize interferences

due to the re-use of frequencies. We report experiments on the most difficult instances: Scen06, Scen07 and Scen08.

`GRAPH` **Instances:** The `GRAPH` generator (Generating Radio link frequency Assignment Problems Heuristically) has been developed by the CALMA project [26] in order to provide structured random instances close to `RLFAP` ones.

`SPOT5` **Instances:** The daily management of an earth observation satellite such as `SPOT5` consists in selecting a subset of candidate photographs to fit physical limitations and maximize their importance [3]. We report experiments on seven instances from those without hard capacity constraint.

`tagSNP` **Instances:** A Single Nucleotide Polymorphism (SNP) is a DNA sequence variation occurring when a single nucleotide - `A`, `T`, `C` or `G` - in the genome differs between members of a biological species or paired chromosomes in an individual [7]. SNPs act as biological markers that may help predict risk of developing particular diseases. The tagSNP problem consists in selecting a small subset of SNPs, called tagSNPs, that captures most of the genetic information. This problem is known to be very hard to solve, due to its close relation to the *set covering problem* (NP-Hard) [22]. We report experiments on twelve challenging instances derived from human chromosome-1-data[4] with r_0=0.5 (up to n=1,550 variables with maximum domain size d ranging from 30 to 266, and up to e=250,000 cost functions). Eight instances are medium-sized, while the four other instances are large ones.

5 Experiments

First, we compare the three strategies for managing intensification and diversification and discuss their impact (see Section 5.2). Then, we compare `DGVNS` with `VNS/LDS+CP`[5][16] and `ID-Walk` [19], one of the most performing local search methods on `RLFAP` instances (see Section 5.3). Finally, note that comparing CPU times for our approach with those for complete methods that exploit tree decompositions would be rather difficult. In fact, all reported CPU times include both finding an optimal solution and proving its optimality. These two tasks take generally about a few days [22]. Experiments we performed clearly demonstrate:

1. The relevance of exploiting the graph of clusters to achieve better intensification and diversification. Moreover, `DGVNS-2` gets better results on most of the instances, except for `tagSNP` instances where `DGVNS-1` is the best one.
2. The efficiency of `DGVNS-1` and `DGVNS-2` compared with both `VNS/LDS+CP` and `ID-Walk` on structured problems like `RLFAP`, `SPOT5` and `tagSNP`. For `GRAPH` instances, the results show the limits of `MCS` decomposition method (see Section 2.2) to reveal pertinent neighborhood structures.

[4] `http://www.costfunction.org/benchmark`

[5] For both `DGVNS` and `VNS/LDS+CP`, the rebuilding step is performed using `LDS+CP`, but `VNS/LDS+CP` uses neighborhood structures N_k of dimension k.

Instance	Method	Succ.	Time	Avg
Scen06	DGVNS-3	50/50	521	3,389
$n = 100$, $d = 44$, $e = 1,222$	DGVNS-2	**50/50**	146	3,389
$S^* = 3,389$	DGVNS-1	**50/50**	**112**	3,389
Scen07	DGVNS-3	4/50	2,752	347,583
$n = 200$, $d = 44$, $e = 2,665$	DGVNS-2	**46/50**	**901**	**344,012**
$S^* = 343,592$	DGVNS-1	40/50	317	345,614
Scen08	DGVNS-3	-	-	519 (500)
$n = 458$, $d = 44$, $e = 5,286$	DGVNS-2	**5/50**	**595**	277
$S^* = 262$	DGVNS-1	3/50	1,811	**275**
Graph06	DGVNS-3	50/50	654	4,123
$n = 200$, $e = 1,970$	DGVNS-2	**50/50**	413	**4,123**
$S^* = 4,123$	DGVNS-1	**50/50**	**367**	4,123
Graph11	DGVNS-3	1/50	3,507	6,637
$n = 340$, $e = 3,417$	DGVNS-2	**23/50**	**3,031**	3,480
$S^* = 3,080$	DGVNS-1	8/50	3,046	4,234
Graph13	DGVNS-3	-	-	31,180 (28,585)
$n = 458$, $e = 4,915$	DGVNS-2	-	-	**21,796** (18,323)
$S^* = 10,110$	DGVNS-1	-	-	22,489 (18,639)

Fig. 3. Comparing the three strategies on RLFAP and GRAPH instances

5.1 Experimental Protocol

Each instance has been solved by each method, with a discrepancy of 3 for LDS, which is the best value found on RLFAP instances (see [16]). k_{min} and k_{max} have been respectively set to 4 and n (the total number of variables) so that all variables of the problem will be covered, and $TimeOut$ fixed to 3,600 seconds. For tagSNP instances, $TimeOut$ was set to 2 hours (resp. 4 hours) for medium-sized (resp. large) instances. A set of 50 runs per instance has been performed on an AMD opteron with 2.1 GHz CPU and 256 GB of RAM. All search strategies have been implemented in C++ using the library toulbar2[6].

For each instance and each method, we report the number of successful runs to reach the optimum, "succ. runs/total runs", the average CPU time (in seconds) for the successful runs, the average cost over the 50 runs and the best cost (between brackets) for unsuccessful runs.

5.2 Comparing the Three Strategies

First, we compare the different DGVNS-i (i=1,2,3) on each of the four problems considered (see Section 4). Then, we summarize the results obtained.

RLFAP Instances. The impact of the neighborhood change strategy is very significant, particularly on the two challenging instances Scen07 and Scen08 (see Fig. 3) for which very large improvements are gained by DGVNS-2 compared to DGVNS-1. For Scen07, DGVNS-2 improves the success rate about 12% (from 80% to 92%) and obtains solutions with a *mean deviation* (percentage deviation from the optimum) of 0.12% above the optimum against 0.55% for DGVNS-1. For Scen08, the success rate is improved about 4% (from 6% to 10%) and DGVNS-2 is

[6] http://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/SoftCSP

Instance	Method	Succ.	Time	Avg
#408	DGVNS-3	50/50	1,697	6,228
$n = 200$, $d = 4$, $e = 2,232$	DGVNS-2	**50/50**	**81**	**6,228**
S^* =6,228	DGVNS-1	49/50	117	6,228
#412	DGVNS-3	-	-	32,384 (32,383)
$n = 300$, $d = 4$, $e = 4,348$	DGVNS-2	**48/50**	484	**32,381**
S^* =32,381	DGVNS-1	36/50	84	32,381
#414	DGVNS-3	-	-	38,486 (38,482)
$n = 364$, $d = 4$, $e = 10,108$	DGVNS-2	**45/50**	670	**38,478**
S^* =38,478	DGVNS-1	38/ 50	554	38,478
#505	DGVNS-3	36/50	3,094	21,253
$n = 240$, $d = 4$, $e = 2,242$	DGVNS-2	**50/50**	90	21,253
S^* =21,253	DGVNS-1	**50/50**	**63**	**21,253**
#507	DGVNS-3	-	-	27,396 (27,393)
$n = 311$, $d = 4$, $e = 5,732$	DGVNS-2	**45/50**	463	**27,390**
S^* =27,390	DGVNS-1	33/50	71	27,390
#509	DGVNS-3	-	-	36,454 (36,450)
$n = 348$, $d = 4$, $e = 8,624$	DGVNS-2	**47/50**	509	**36,446**
S^* =36,446	DGVNS-1	40/50	265	36,446

Fig. 4. Comparing the three strategies on SPOT5 instances

3 times faster than DGVNS-1. Let us note that, on this instance, DGVNS-1 gets best results on average, with a mean deviation above the optimum of 0.5% against 0.57% for DGVNS-2. For Scen06, results obtained by both methods are quite similar, but DGVNS-1 is slightly faster. Finally, DGVNS-3 is less effective, particularly on Scen08: it is not able to reach the optimum and the best found solution has a cost 500. This confirms the importance of strengthening the diversification effort by covering a large part of clusters as performed by DGVNS-1 and DGVNS-2.

SPOT5 Instances. This trend is confirmed by SPOT5 instances (see Fig. 4), where DGVNS-2 clearly outperforms DGVNS-1. On the four largest instances (#412, #414, #507 et #509), DGVNS-2 improves the success rate about 19% on average, but DGVNS-1 is faster. For instances #408 and #505, both methods reach the optimum with success rates of 100%. However, for instance #408, DGVNS-2 is 1.4 times faster, while for #505 DGVNS-1 is the best one. Once again, DGVNS-3 is less effective: it reaches the optimum for only 2 instances among 6.

GRAPH Instances. Once again, DGVNS-2 outperforms DGVNS-1 and DGVNS-3 (see Fig. 3), both in terms of success rates and CPU times, particularly on the two challenging instances Graph11 and Graph13. Indeed, for Graph11, the success rate for DGVNS-2 is 3 times as high as for DGVNS-1 (from 14% to 46%). The mean deviation above the optimum decreases from 29% to 12% (gain of 16%). For Graph13, even though no method reaches the optimum, DGVNS-2 obtains solutions with a mean deviation about 115%, against 135% for DGVNS-1. Furthermore, DGVNS-2 finds the best solution with cost 18,323. For comparison, the best solution found by DGVNS-1 has a cost 20,956.

tagSNP Instances. The results are in favor of DGVNS-1 (see Fig. 5). Indeed, for the two *medium-sized instances* #6835 and #8956, DGVNS-1 clearly outperforms

Instance	Method	Succ.	Time	Avg
#3792	DGVNS-3	14/50	4,509	6,360,029
$n = 528,\ d = 59,\ e = 12,084$	DGVNS-2	**50/50**	1,564	**6,359,805**
$S^* = 6359805$	DGVNS-1	**50/50**	**954**	**6,359,805**
#4449	DGVNS-3	9/50	3,371	5,094,261
$n = 464,\ d = 64,\ e = 12,540$	DGVNS-2	**50/50**	1,186	**5,094,256**
$S^* = 5,094,256$	DGVNS-1	**50/50**	**665**	**5,094,256**
#6835	DGVNS-3	3/50	6,746	4,571,344
$n = 496,\ d = 90,\ e = 18,003$	DGVNS-2	34/50	3,568	4,730,484
$S^* = 4571108$	DGVNS-1	**50/50**	**2,409**	**4,571,108**
#8956	DGVNS-3	-	-	6,966,493 (6,660,336)
$n = 486,\ d = 106,\ e = 20,832$	DGVNS-2	34/50	5,138	6,660,383
$S^* = 6,660,308$	DGVNS-1	**50/50**	**4,911**	**6,660,308**
#9319	DGVNS-3	-	-	6,477,395 (6,477,242)
$n = 562,\ d = 58,\ e = 14,811$	DGVNS-2	**50/50**	1,248	**6,477,229**
$S^* = 6,477,229$	DGVNS-1	**50/50**	**788**	**6,477,229**
#15757	DGVNS-3	50/50	2,041	2,278,611
$n = 342,\ d = 47,\ e = 5,091$	DGVNS-2	**50/50**	127	**2,278,611**
$S^* = 2,278,611$	DGVNS-1	**50/50**	**60**	**2,278,611**
#16421	DGVNS-3	8/50	5,552	3,437,125
$n = 404,\ d = 75,\ e = 12,138$	DGVNS-2	**50/50**	**2,001**	**3,436,849**
$S^* = 3,436,849$	DGVNS-1	**50/50**	2,673	3,436,849
#16706	DGVNS-3	-	-	2,632,319 (2,632,319)
$n = 438,\ d = 30,\ e = 6,321$	DGVNS-2	**50/50**	**127**	**2,632,310**
$S^* = 2,632,310$	DGVNS-1	49/50	153	2,632,310
#10442	DGVNS-3	-	-	22,491,631 (22,491,496)
$n = 908,\ d = 76,\ e = 28,554$	DGVNS-2	48 / 50	8,257	21,591,915
$S^* = 21,591,913$	DGVNS-1	**50/50**	**4,552**	**21,591,913**
#14226	DGVNS-3	-	-	29,256,033 (27,996,275)
$n = 1,058,\ d = 95,\ e = 36,801$	DGVNS-2	**50/50**	8,237	**25,665,437**
$S^* = 25,665,437$	DGVNS-1	46/50	7,606	25,688,751
#17034	DGVNS-3	-	-	41,016,388 (39,852,093)
$n = 1,142,\ d = 123,\ e = 47,967$	DGVNS-2	35/50	10,288	38,777,581
$S^* = 38,318,224$	DGVNS-1	**41/50**	**8,900**	**38,563,232**
#9150	DGVNS-3	-	-	51,510,783 (50,281,105)
$n = 1,352,\ d = 121,\ e = 44,217$	DGVNS-2	-	-	46,123,257 (44,697,387)
$S^* = 43,301,891$	DGVNS-1	-	-	**44,754,916 (43,302,028)**

Fig. 5. Comparing the three strategies on tagSNP instances

DGVNS-2 both in terms of success rates (gain of 32%) and CPU times. For the other instances (except for instances #16421 and #16706), both methods reach the optimum for each of the 50 runs, however DGVNS-1 is in average 2 times faster. For the two instances #16421 and #16706, DGVNS-2 obtains better CPU times. For *large-size instances* (except for instance #14226), DGVNS-1 obtains better results than DGVNS-2 on three instances.

Synthesis. These experiments show clearly the impact of the neighborhood change strategy on the performances of DGVNS. First, diversification performed by considering $C_{succ(i)}$ as the next cluster is necessary. Second, DGVNS-2 gets better success rates on most of the instances (except for tagSNP instances), but DGVNS-1 is faster. The results of DGVNS-2 on tagSNP instances can be explained by the size of the instances and the CPU times which are 2 to 4 times larger compared to RLFAP and SPOT5 instances. For these instances, remaining in the same cluster when a better solution is found requires much more time to find a

Instance	Method	Succ.	Time	Avg
Scen06	DGVNS-2	**50/50**	**146**	**3,389**
	VNS/LDS+CP	15/50	83	3,399
$S^* = 3,389$	ID-Walk	NA	840	3,447 (3,389)
Scen07	DGVNS-2	**46/50**	**901**	**344,012**
	VNS/LDS+CP	1/50	461	355,982
$S^* = 343,592$	ID-Walk	NA	360	373,334 (343,998)
Scen08	DGVNS-2	**5/50**	**595**	**277**
	VNS/LDS+CP	0/50	-	394 (357)
$S^* = 262$	ID-Walk	NA	3,000	291 (267)
Graph06	DGVNS-2	**50/50**	413	4,123
	VNS/LDS+CP	**50/50**	**218**	4,123
$S^* = 4,123$	ID-Walk	0/50	-	18,949 (13,001)
Graph11	DGVNS-2	23/50	3,031	4,234
	VNS/LDS+CP	**44/50**	**2,403**	**3,090**
$S^* = 3,080$	ID-Walk	0/50	-	41,604 (27,894)
Graph13	DGVNS-2	0/50	-	21,796 (18,323)
	VNS/LDS+CP	**3/50**	**3,477**	**14,522**
$S^* = 10110$	ID-Walk	0/50	-	58,590 (47,201)

Instance	Method	Succ.	Time	Avg
#408	DGVNS-2	**50/50**	**89**	**6,228**
	VNS/LDS+CP	26/50	149	6,228
$S^* = 6,228$	ID-Walk	**50/50**	**3**	**6,228**
#412	DGVNS-2	**48/50**	**484**	**32,381**
	VNS/LDS+CP	32/50	130	32,381
$S^* = 32,381$	ID-Walk	10/50	2102	3,238
#414	DGVNS-2	**45/50**	**670**	**38,478**
	VNS/LDS+CP	12/50	434	38,481
$S^* = 38,478$	ID-Walk	0/50	-	38,481
#505	DGVNS-2	**50/50**	**90**	**21,253**
	VNS/LDS+CP	41/50	143	21,253
$S^* = 21,253$	ID-Walk	**50/50**	**358**	**21,253**
#507	DGVNS-2	**45/50**	**463**	**27,390**
	VNS/LDS+CP	11/50	232	27,391
$S^* = 27,390$	ID-Walk	7/50	1,862	27,391
#509	DGVNS-2	**47/50**	**265**	**36,446**
	VNS/LDS+CP	12/50	598	36,448
$S^* = 36,446$	ID-Walk	0/50	-	36,450

Fig. 6. Comparing with other approaches on RLFAP, GRAPH and SPOT5 instances

new improvement, due to the important size of neighborhoods. This significantly reduces the speed improvement of the quality of solutions provided by DGVNS-2. This is not the case for DGVNS-1, which benefits from the systematic change of neighborhood structures for improving solutions faster. This helps DGVNS-1 to cut branches earlier in subsequent iterations of the rebuilding step.

5.3 Comparing with Other Approaches

First, we compare DGVNS-2 with VNS/LDS+CP and ID-Walk on each of the four problems considered. Then, we summarize the results obtained.

RLFAP Instances. First, DGVNS-2 clearly outperforms VNS/LDS+CP on RLFAP instances (see Fig. 6). DGVNS-2 reaches the optimum with success rates of 100%, 92% and 10% on Scen06, Scen07 and Scen08 respectively. VNS/LDS+CP gets successful runs only very few times on the first two instances, and is not able to find the optimum for Scen08: the best solution found has a cost 357.

Second, DGVNS-2 clearly outperforms ID-Walk[7], particularly on the two challenging instances Scen07 and Scen08 (see Fig. 6). For Scen07 (resp. Scen08), DGVNS-2 obtains solutions with a *mean deviation* (percentage deviation from the optimum) of 0.12% (resp. 5.7%) above the optimum, while ID-Walk only finds solutions whose average costs are respectively 8% and 11% above the optimum.

SPOT5 instances. This trend is confirmed by SPOT5 instances (see Fig. 6), where DGVNS-2 outperforms VNS/LDS+CP, both in terms of success rates (with a gain of 50% on average) and CPU times. Indeed, DGVNS-2 reaches the optimum for each run on two instances (#408 and #505). For the other instances, the success rate

[7] Results for RLFAP instances are taken from [19]. The number of successful runs and the CPU times are not available (NA in Fig. 6), Thus, we only report the time per trial, the average cost over 10 trials and the best cost found. For other instances (SPOT5, GRAPH and tagSNP) results were obtained using ID-Walk in the library INCOP [1], with the same experimental protocol as described above.

Instance	Method	Succ.	Time	Avg
#3792, $n = 528$, $d = 59$, $e = 12,084$	DGVNS-2	**50/50**	**954**	**6,359,805**
$S^* = 6,359,805$	VNS/LDS+CP	15/50	2,806	6,359,856
#4449, $n = 464$, $d = 64$, $e = 12,540$	DGVNS-2	**50/50**	**1,186**	**5,094,256**
$S^* = 5,094,256$	VNS/LDS+CP	48/50	2,616	5,094256
#6835, $n = 496$, $d = 90$, $e = 18,003$	DGVNS-2	34/50	3,568	4,730,484
$S^* = 4,571,108$	VNS/LDS+CP	**50/50**	**7,095**	**4,571,108**
#8956, $n = 486$, $d = 106$, $e = 20,832$	DGVNS-2	**34/50**	**5,138**	**6,660,308**
$S^* = 6,660,308$	VNS/LDS+CP	12/50	8,665	6,660327
#9319, $n = 562$, $d = 58$, $e = 14,811$	DGVNS-2	**50/50**	**1,248**	**6,477,229**
$S^* = 6,477,229$	VNS/LDS+CP	47/50	2,434	6,477,229
#15757, $n = 342$, $d = 47$, $e = 5,091$	DGVNS-2	**50/50**	**127**	**2,278,611**
$S^* = 2,278,611$	VNS/LDS+CP	50/50	229	2,278,611
#16421, $n = 404$, $d = 75$, $e = 12,138$	DGVNS-2	**50/50**	**2,001**	**3,436,849**
$S^* = 3,436,849$	VNS/LDS+CP	37/50	3,146	3,436,924
#16706, $n = 438$, $d = 30$, $e = 6,321$	DGVNS-2	**50/50**	**127**	**2,632,310**
$S^* = 2,632,310$	VNS/LDS+CP	50/50	629	2,632,310
#9150, $n = 13,52$, $d = 121$, $e = 44,217$	DGVNS-2	0/50	-	**44,754,916 (43,302,028)**
$S^* = 43,301,891$	VNS/LDS+CP	0/50	-	52,989,981 (51,677,673)
#10442, $n = 908$, $d = 76$, $e = 28,554$	DGVNS-2	**48/50**	**8,257**	**21,591,913**
$S^* = 21,591,913$	VNS/LDS+CP	0/50	-	22,778,811 (22,490,938)
#14226, $n = 1,058$, $d = 95$, $e = 36,801$	DGVNS-2	**50/50**	**8,237**	**25,688,437**
$S^* = 25,665,437$	VNS/LDS+CP	0/50	-	28,299,904 (26,830,579)
#17034, $n = 1142$, $d = 123$, $e = 47,967$	DGVNS-2	**35/50**	**10,288**	**38,777,581**
$S^* = 38,318,224$	VNS/LDS+CP	0/50	-	41,352,709 (39,850,974)

Fig. 7. Comparing DGVNS-2 and VNS/LDS+CP on tagSNP instances

is at least 90%. VNS/LDS+CP gets an average success rate of 67% on instances #408 and #505, while for the other instances (except for instance #412), the success rate is at most 24%.

Once again, DGVNS-2 clearly dominates ID-Walk (except for instance #408) (see Fig. 6), particularly on the two large instances #414 and #509 where ID-Walk does not reach the optimum. For these two instances, the best solution found has a cost 38,479 and 36,447 respectively.

tagSNP **Instances.** Fig. 7 compares DGVNS-2 with VNS/LDS+CP on tagSNP instances. As ID-Walk exhibited poor performances, they are not reported here. For *medium-sized instances* (except for instance #6835), DGVNS-2 clearly outperforms VNS/LDS+CP: the optimum is reached for each of the 50 runs. VNS/LDS+CP improves the success rates about 32% on instance #6835 and gets the same success rates on two instances #15757 and #16706, but DGVNS-2 is on average 3.4 times faster. For the two instances #3792 and #8956, VNS/LDS+CP gets successful runs only very few times (i.e. success rate of 30% and 24% respectively). For comparison, DGVNS-2 improves very significantly this success rate about 70% (from 30% to 100%) for instance #3792 and 44% (from 24% to 68%) for instance #8956. For the other instances, VNS/LDS+CP remains less competitive both in terms of success rates and CPU times.

For *large instances*, DGVNS-2 clearly dominates VNS/LDS+CP (see Fig. 7), particularly on the three instances #10442, #14226 and #17034, where VNS/LDS+CP is not able to find the optimum. For these three instances, the best costs found (between brackets) are about 4% above the optimum. For instance #9150, even though no method reaches the optimum, DGVNS-2 performs better than

VNS/LDS+CP. For this instance, the deviation (resp. mean deviation) above the optimum of the best (resp. average) cost found decreases from 19% to 0.0003% (resp. from 22% to 3.35%).

GRAPH **Instances.** DGVNS-2 clearly outperforms ID-Walk but is less competitive than VNS/LDS+CP (see Fig. 6). For Graph06, both methods perform similarly in terms of success rates and solution quality, but VNS/LDS+CP is faster. For Graph11 and Graph13, VNS/LDS+CP is clearly the best one. This is certainly due to the fact that clusters are very large and strongly connected. Thus, the impact of our diversification mechanism is weaker since most clusters have few proper variables and tend to be quite similar. These results show the limits of MCS to reveal pertinents structures (i.e. weakly connected clusters of reasonable size).

Synthesis. These experiments clearly demonstrate the efficiency of DGVNS-2 compared with both VNS/LDS+CP and ID-Walk on structured problems like RLFAP, SPOT5 and tagSNP. The same holds for DGVNS-1, i.e., DGVNS-1 clearly outperforms both VNS/LDS+CP and ID-Walk on the same instances. For GRAPH instances, the results show the limits of the MCS decomposition method to provide pertinent neighborhood structures.

6 Conclusions

In this paper we have introduced three new strategies enabling better intensification and diversification in DGVNS. Experiments show that, achieving better compromise between these two components leads to very good results. Moreover, comparisons made with VNS/LDS+CP and ID-Walk show the relevance of our two schemes DGVNS-1 and DGVNS-2 to efficiently guide the exploration of the search space.

We are currently investigating two directions: parallelizing the exploration of clusters and using hyper-tree decomposition methods [10]. Contrary to tree decomposition, which consists in grouping the vertices in clusters (i.e. variables in subproblems), hyper-tree decomposition consists in grouping the constraints (or hyper-edges) in nodes of the hyper-tree.

Acknowledgements. This work is partly supported by the ANR (French Research National Agency) funded project FiCOLOFO ANR-10-BLA-0214.

References

1. http://www-sop.inria.fr/coprin/neveu/incop/presentation-incop.html
2. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM. J. on Algebraic and Discrete Methods 8, 277–284 (1987)
3. Bensana, E., Lemaître, M., Verfaillie, G.: Earth observation satellite management. Constraints 4(3), 293–299 (1999)
4. Cabon, B., de Givry, S., Lobjois, L., Schiex, T., Warners, J.P.: Radio link frequency assignment. Constraints 4(1), 79–89 (1999)

5. de Givry, S., Schiex, T., Verfaillie, G.: Exploiting tree decomposition and soft local consistency in weighted CSP. In: AAAI, pp. 22–27. AAAI Press (2006)
6. Dechter, R., Pearl, J.: Tree clustering for constraint networks. Artif. Intell. 38(3), 353–366 (1989)
7. Carlson, C.S., et al.: Selecting a maximally informative set of single-nucleotide polymorphisms for association analyses using linkage disequilibrium. Am. J. of Hum. Genetics 74(1), 106–120 (2004)
8. Fontaine, M., Loudni, S., Boizumault, P.: Guiding VNS with tree decomposition. In: ICTAI, pp. 505–512. IEEE (2011)
9. Gottlob, G., Lee, S.T., Valiant, G.: Size and treewidth bounds for conjunctive queries. In: PODS, pp. 45–54 (2009)
10. Gottlob, G., Miklós, Z., Schwentick, T.: Generalized hypertree decompositions: Np-hardness and tractable variants. J. ACM 56(6) (2009)
11. Hansen, P., Mladenovic, N., Perez-Brito, D.: Variable neighborhood decomposition search. Journal of Heuristics 7(4), 335–350 (2001)
12. Harvey, W.D., Ginsberg, M.L.: Limited discrepancy search. In: IJCAI (1), pp. 607–615. Morgan Kaufmann (1995)
13. Koster, A.M.C.A., Bodlaender, H.L., van Hoesel, S.P.M.: Treewidth: Computational experiments. ENDM 8, 54–57 (2001)
14. Larrosa, J., Schiex, T.: In the quest of the best form of local consistency for Weighted CSP. In: IJCAI, pp. 239–244 (2003)
15. Linhares, A., Yanasse, H.H.: Search intensity versus search diversity: a false trade off? Appl. Intell. 32(3), 279–291 (2010)
16. Loudni, S., Boizumault, P.: Combining VNS with constraint programming for solving anytime optimization problems. EJOR 191, 705–735 (2008)
17. Marinescu, R., Dechter, R.: AND/OR branch-and-bound search for combinatorial optimization in graphical models. Artif. Intell. 173(16-17), 1457–1491 (2009)
18. Mladenovic, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research 24, 1097–1100 (1997)
19. Neveu, B., Trombettoni, G., Glover, F.: `ID Walk`: A candidate list strategy with a simple diversification device. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 423–437. Springer, Heidelberg (2004)
20. Rish, I., Dechter, R.: Resolution versus search: Two strategies for SAT. J. Autom. Reasoning 24(1/2), 225–275 (2000)
21. Robertson, N., Seymour, P.D.: Graph minors. ii. Algorithmic aspects of tree-width. J. Algorithms 7(3), 309–322 (1986)
22. Sánchez, M., Allouche, D., de Givry, S., Schiex, T.: Russian doll search with tree decomposition. In: Boutilier, C. (ed.) IJCAI, pp. 603–608 (2009)
23. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: Maher, M.J., Puget, J.-F. (eds.) CP 1998. LNCS, vol. 1520, pp. 417–431. Springer, Heidelberg (1998)
24. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. 13(3), 566–579 (1984)
25. Terrioux, C., Jégou, P.: Hybrid backtracking bounded by tree-decomposition of constraint networks. Artificial Intelligence 146(1), 43–75 (2003)
26. van Benthem, H.: GRAPH: Generating radiolink frequency assignment problems heuristically (1995)

A Hybridized Particle Swarm Optimization with Expanding Neighborhood Topology for the Feature Selection Problem

Yannis Marinakis[1] and Magdalene Marinaki[2]

[1] Decision Support Systems Laboratory, Department of Production Engineering and Management, Technical University of Crete, Chania, Greece
marinakis@ergasya.tuc.gr
[2] Industrial Systems Control Laboratory, Department of Production Engineering and Management, Technical University of Crete, Chania, Greece
magda@dssl.tuc.gr

Abstract. This paper introduces a new algorithmic nature inspired approach that uses a hybridized Particle Swarm Optimization algorithm with a new neighborhood topology for successfully solving the Feature Selection Problem (FSP). The Feature Selection Problem is an interesting and important topic which is relevant for a variety of database applications. The proposed algorithm for the solution of the FSP, the Particle Swarm Optimization with Expanding Neighborhood Topology (PSOENT), combines a Particle Swarm Optimization (PSO) algorithm and the Variable Neighborhood Search (VNS) strategy. As, in general, the structure of the social network affects strongly a PSO algorithm, the proposed method by using an expanding neighborhood topology manages to increase the performance of the algorithm. As the algorithm starts from a small size neighborhood and by increasing (expanding) the size of the neighborhood, it ends to a neighborhood that includes all the swarm, it manages to take advantage of the exploration capabilities of a global neighborhood structure and of the exploitation abilities of a local neighborhood structure. In order to test the effectiveness and the efficiency of the proposed method we use data sets of different sizes and compare the proposed method with a number of other PSO algorithms and other algorithms from the literature.

Keywords: Feature Selection Problem, Particle Swarm Optimization, Expanding Neighborhood Topology, Variable Neighborhood Search.

1 Introduction

Particle Swarm Optimization (PSO) is a population-based swarm intelligence algorithm that was originally proposed by Kennedy and Eberhart [8]. PSO simulates the social behavior of social organisms by using the physical movements of the individuals in the swarm. Its mechanism enhances and adapts to the global and local exploration.

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 37–51, 2013.

There are two kinds of population topologies for the PSO algorithm: the global best population topology and the local best population topology [6]. In the global best PSO, the neighborhood for each particle is the entire swarm. In the local best PSO, each particle has a smaller neighborhood. Thus, as it is mentioned in [6], the performance of the PSO depends strongly on the structure of the social network. If a highly connected social network is adapted, then, it will achieve a faster convergence to a solution compared to a less connected network. However, the faster convergence may lead the algorithm to be stuck to a local optimum. On the other hand, if a low connected social network is used, then, the swarm converges slower which allows larger parts of the search space to be explored. To combine the advantages of the better exploration by neighborhood structures and the faster convergence of highly connected networks, Suganthan [20] combined the two approaches. The search is initialized with an *lbest* PSO with $n = 2$ (i.e. with the smallest neighborhoods). The neighborhood sizes are, then, increased as iterations increase until each neighborhood contains the entire swarm.

In [16], an improvement of the idea proposed in [20] and an application to a combinatorial optimization problem was presented. In this paper, a Particle Swarm Optimization with Expanding Neighborhood Topology (PSOENT) algorithm was presented and used for the solution of the Permutation Flowshop Scheduling Problem (PFSP). In this algorithm, the advantages of a local search neighborhood topology and of a global search neighborhood topology are used. The algorithm started with a local search neighborhood topology where the neighborhoods for each particle were equal to 2 and in each iteration the neighborhood was increased (expanded) until it became equal to the number of particles. Then, if the maximum number of iterations had not yet been reached, the neighborhood was initialized again and it followed the same procedure until the maximum number of iterations had been reached. Thus, there were no consecutive iterations in which the size of the neighborhood was the same. With this procedure each particle had more exploitation abilities by moving for a number of iterations in small swarms inside the whole swarm. Also, this procedure increased the exploration capabilities as for a number of iterations all the particles were moving as a single swarm. By using an expanding neighborhood topology, each particle participated in each iteration in more than one swarm and, thus, the interchange of the information between the swarms was becoming easier.

In this paper, we modify the neighborhood topology presented in [16]. In [16], the neighborhood was expanding with the number of iterations, while in the proposed algorithm the neighborhood is expanding based on the quality of the solutions. Each particle has its own neighborhood. Thus, while all particles begin with the same neighborhood topology when a solution of a particle is not improved for a consecutive number of iterations (it_{num}), then, only its neighborhood is expanding. With this strategy there is a possibility of having different neighborhood topologies for each particle which is a novelty of the proposed algorithm. This idea of changing the neighborhood topology of the swarm not with the number of iterations but when the swarm can not improve the global or

personal best solution is inspired by the basic idea of changing a neighborhood when the algorithm is trapped in a local optimum in the Variable Neighborhood Search (VNS) Algorithm [7]. Thus, as in VNS algorithm the neighborhood is expanding in order to find a better local optimum, in the proposed algorithm the search of a better direction that the particle will move in its neighborhood is expanding in order to search in a larger neighborhood when the particle is trapped in a local optimum. This incorporation of one of the basic characteristics of Variable Neighborhood Search algorithm in the Particle Swarm Optimization gave a more powerful version of PSO algorithm. Another characteristic of VNS algorithm that is incorporated in the PSO is the reinitialization of the search of the local best neighborhood. In the proposed algorithm, when the number of neighbors becomes equal to the number of particles, then, the search of the local best neighborhood is reinitialized from a very small neighborhood. In each iteration of the algorithm the search is realized from a different point (shaking procedure of the VNS) as the current position of each particle is different in each iteration.

The VNS algorithm is, also, applied in order to optimize the particles. The basic idea of the VNS algorithm is the successive search in a number of neighborhoods of a solution. It should be noted that in a Particle Swarm Optimization algorithm with the term "neighborhood" it is meant the topology of the swarm (the particles that affect another particle in the process of finding the optimum), while in a Variable Neighborhood Search algorithm with the term "neighborhood" it is meant different numbers of local search algorithms. In the rest of the paper when the term "neighborhood" is used together with the term "topology" we will refer to the Particle Swarm Optimization part of the algorithm and when it is used together with the term "structure" we will refer to the local search algorithms that are used inside the Variable Neighborhood Search algorithm. The proposed algorithm is used for the solution of the Feature Selection Problem (FSP). A number of data sets are used and the results are compared with other evolutionary and nature inspired algorithms. The proposed algorithm performs better than the other PSO-based techniques from the literature and better or equally good with other evolutionary algorithms from the literature. The rest of the paper is organized as follows: In the next section, a description of the FSP is presented. In the third section, the proposed algorithm, the Particle Swarm Optimization with Expanding Neighborhood Topology (PSOENT), is presented and analyzed in detail. Computational results are presented and analyzed in the fourth section while in the last section conclusions and future research are given.

2 Feature Selection Problem

Recently, there has been an increasing need for novel data-mining methodologies that can analyze and interpret large volumes of data. The proper selection of the right set of features for classification is one of the most important problems in designing a good classifier. The basic feature selection problem is an optimization problem with a performance measure for each subset of features to measure its ability to classify the samples.

A formulation of the problem is the following [1]:

- V is the original set of features with cardinality m.
- d represents the desired number of features in the selected subset, X, where $X \subseteq V$.
- $F(X)$ is the feature selection criterion function for the set X.

Let us consider a high value of F to indicate a better feature subset. Formally, the problem of feature selection is to find a subset $X \subseteq V$ such that $\mid X \mid = d$ and

$$F(X) = \max_{Z \subseteq V, |Z|=d} F(Z) \tag{1}$$

In the literature, many successful feature selection algorithms have been proposed. These algorithms can be classified into two categories. If feature selection depends on learning algorithm, the approach is referred to as a *wrapper model*. Otherwise, it is said to be a *filter model*. Filters, such as mutual information (MI), are based on the statistical tools. Wrappers assess subsets of features according to their usefulness to a given classifier [15,21]. Unfortunately, finding the optimum feature subset has been proved to be NP-hard. Many algorithms are, thus, proposed to find suboptimal solutions in comparably smaller amount of time.

3 Particle Swarm Optimization with Expanding Neighborhood Topology Algorithm

3.1 General Description

In this paper, an algorithm for the solution of the feature selection problem based on the Particle Swarm Optimization (PSO) algorithm is presented. This algorithm is combined with three nearest neighbor-based classifiers, the 1-Nearest Neighbor, the k-Nearest Neighbor and the Weighted k (wk)-Nearest Neighbor classifier. As it was mentioned earlier, there are two kinds of population topologies for the PSO algorithm: the global best (*gbest*) population topology and the local best (*lbest*) population topology [6]. In the *gbest* PSO, the neighborhood for each particle is the entire swarm. The social network employed by the *gbest* PSO reflects the star topology in which all particles are interconnected. Thus, the velocities of each particle are updated based on the information obtained from the best particle of the whole swarm. In the *lbest* PSO, each particle has a smaller neighborhood. In this case, the network topology corresponds to the ring topology where each particle communicates with only a limited number of other members of the swarm. The communication is usually achieved using the particles' indices. Thus, if the size of the neighborhood is equal to three, the selected neighbors for the particle i are the particles $i-1$ and $i+1$. Thus, the velocities of each particle are updated based on the information obtained from the best particle of the neighborhood. The use of particle's indices for the

creation of the neighborhood is preferred because it is very difficult and computationally expensive to calculate distances between all the particles to find the neighbors of each particle. Furthermore, if the indices are used, then, a particle may belong to more than one neighborhood having the possibility of spreading a good solution among different neighborhoods. Usually, the *gbest* PSO converges faster than the *lbest* PSO. On the other hand, the *lbest* PSO has larger diversity in its solutions and, thus, it is more difficult to be trapped in local minima [6].

Initially, we have to choose the population of the particles. Each particle is randomly placed in the d-dimensional space as a candidate solution (in the feature selection problem d corresponds to the number of activated features). One of the key issues in designing a successful algorithm for Feature Selection Problem is to find a suitable mapping between Feature Selection Problem solutions' and particles in Particle Swarm Optimization algorithm. Every candidate feature in PSO is mapped into a binary particle where the bit 1 denotes that the corresponding feature is selected and the bit 0 denotes that the feature is not selected. Afterwards, the fitness of each particle is calculated using the following equation:

$$OCA = \frac{\sum_{m=1}^{E} e_{mm}}{\sum_{m=1}^{E}\sum_{l=1}^{E} e_{ml}}\%. \tag{2}$$

The fitness function measures the quality of the members of the swarm. In this problem, the quality is measured with the overall classification accuracy. Thus, for each particle the classifiers (1-Nearest Neighbor, k-Nearest Neighbor or wk-Nearest Neighbor [5]) are called and the produced overall classification accuracy (OCA) gives the fitness function. In the fitness function we would like to maximize the OCA. The previously mentioned formula for OCA (Eq. 2) is defined taking into account that the accuracy of a E class problem can be described using a $E \times E$ confusion matrix. The element e_{ml} in row m and column l describes the number of samples of true class l classified as class m, i.e., all correctly classified samples are placed in the diagonal and the remaining misclassified cases in the upper and lower triangular parts.

In the Particle Swarm Optimization algorithm, the velocity of the i-th particle $v_i = (v_{i1}, v_{i2}, ..., v_{id})$ is defined as the change of its position. The flying direction of each particle is the dynamical interaction of individual and social flying experience. The algorithm completes the optimization through following the personal best solution of each particle and the global best value of the whole swarm. The position of each particle is represented by a n-dimensional vector in problem space $x_i = (x_{i1}, x_{i2}, ..., x_{in})$, $i = 1, 2, ..., N$ (N is the population size and n is the vector size), and its performance is evaluated on the predefined fitness function. The velocity v_{ij} represents the changes that will be made to move the particle from one position to another. Where the particle will move

depends on the dynamic interaction of its own experience and the experience of the whole swarm. Depending on the method used, there are three possible directions that a particle can follow: to follow its own path, to move towards the best position it had during the iterations ($pbest_{ij}$) or to move to the best particle's position ($gbest_j$ if a global neighborhood topology is used or $lbest_{ij}$ if a local neighborhood topology is used). The velocity equation in the constriction Particle Swarm Optimization algorithm is given by [4]:

$$v_{ij}(t+1) = \chi(v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) + c_2 rand_2(gbest_j - x_{ij}(t))) \quad (3)$$

where

$$\chi = \frac{2}{|2 - c - \sqrt{c^2 - 4c}|} \text{ and } c = c_1 + c_2, c > 4. \quad (4)$$

c_1 and c_2 are the acceleration coefficients, t is the number of the current iteration, $rand_1$ and $rand_2$ are two random variables in the interval [0, 1]. The acceleration coefficients c_1 and c_2 control how far a particle will move in a single iteration. Low values allow particles to roam far from target regions before being tugged back, while high values result in abrupt movement towards, or past, target regions [8]. A limiting factor (χ), called constriction factor, is used. The constriction factor is used in order to prevent explosion, to ensure convergence of the algorithm and to eliminate the factors that limit the velocities of the particles.

In the proposed algorithm the velocity equation is almost the same as the one used in constriction PSO. What changes is the term *gbest* which is replaced by the term *lbest*. The neighborhood may consist of three particles, five particles or more. The velocity equation is [6]:

$$v_{ij}(t+1) = \chi(v_{ij}(t) + c_1 rand_1(pbest_{ij} - x_{ij}(t)) + c_2 rand_2(lbest_{ij} - x_{ij}(t))) \quad (5)$$

where

$$lbest_{ij} \in \{N_i | f(lbest_{ij}) = \min\{f(x_{ij})\}, \forall x \in N_i\} \quad (6)$$

where f is the objective function value and the neighbor N_i is defined by [6]:

$$N_i = \{pbest_{i-n_{N_i}}(t), pbest_{i-n_{N_i}+1}(t), \cdots, pbest_{i-1}(t), pbest_i(t), pbest_{i+1}(t), \cdots, pbest_{i+n_{N_i}}(t)\}. \quad (7)$$

The basic PSO and its variants have successfully operated for continuous optimization functions. In order to extend the application to discrete space, Kennedy and Eberhart proposed a discrete binary version of PSO [9] where a particle moves in a state space restricted to zero and one on each dimension where each

v_{ij} represents the probability of bit x_{ij} taking the value 1. Thus, the particles' trajectories are defined as the changes in the probability and v_{ij} is a measure of individual's current probability of taking 1. If the velocity is higher it is more likely to choose 1, and lower values favor choosing 0. A sigmoid function is applied to transform the velocity from real number space to probability space:

$$sig(v_{ij}) = \frac{1}{1 + exp(-v_{ij})} \tag{8}$$

Thus, the position of a particle is updated using the following equation:

$$x_{ij}(t+1) = \begin{cases} 1, & \text{if } rand_3 < sig(v_{ij}) \\ 0, & \text{if } rand_3 \geq sig(v_{ij}) \end{cases} \tag{9}$$

where x_{ij} is the valued of the j-th dimension of particle x_i, and $x_{ij} \in \{0, 1\}$; $sig(v_{ij})$ is calculated according to Equation (8), $rand_3$ is a random number distributed in [0, 1]. As in basic PSO, a parameter V_{max} is incorporated to limit the v_{ij} so that $sig(v_{ij})$ does not approach too closely 0 or 1 [10]. Such implementation can ensure that the bit can transfer between 1 and 0 with a positive probability. In practice, V_{max} is often set at ± 4.

A Variable Neighborhood Search (VNS) [7] algorithm is applied in order to optimize the particles. The basic idea of the method is the successive search in a number of neighborhoods of a solution. With the term neighborhood it is meant different number of local search algorithms. The search is applied either with random or with a more systematic manner in order to escape the solution from a local optimum. This method takes advantage of the fact that different local search algorithms will lead to different local optimum. In this paper, the VNS algorithm is used in the following way. Initially, the number of local search algorithms is selected. The local search strategies for the Feature Selection Problem are the 2-opt, 1-0 insert, 2-0 insert, 1-1 interchange and 2-2 interchange neighborhoods. As we do not want to increase the complexity of the algorithm, it is decided to apply in each particle one local search combination of algorithms per iteration. For this reason, a VNS operator C_{VNS} is selected that controls which local search algorithm is applied. The C_{VNS} value is compared with the output of a random number generator, $rand_i(0, 1)$. If the random number is less or equal to the C_{VNS}, then, the first local search algorithm is used. Then, if the random number is less or equal to the $2 * C_{VNS}$, then, the second local search algorithm is used, and so on. As we would like to have not only simple local search algorithms but also their combinations we select ten local search algorithms, the five previously mentioned methods and five combinations (2-opt and 1-1 interchange, 2-opt and 1-0 insert, 1-0 insert and 2-2 interchange, 2-0 insert and 1-1 interchange and, finally, 2-opt, 1-1 interchange, 1-0 insert, 2-2 interchange and 2-0 insert). The C_{VNS} operator is set equal to 0.1. The algorithm stops when a maximum number of iterations has been reached.

4 Computational Results

The performance of the proposed methodology is tested on 8 data sets taken from the UCI Machine Learning Repository. The data sets were chosen to include a wide range of domains and their characteristics are given in Table 1. The data vary in terms of the number of observations from very small samples (Hepatitis with 80 observations) up to larger data sets (Spambase with 4601 observations). In two cases (Breast Cancer Wisconsin, Hepatitis) the data sets are appeared with different size of observations. This is performed because in these data sets there is a number of missing values. Two different ways were used to cope with the problem of missing values. In the first way where all the observations are used, we took the mean values of all the observations in the corresponding feature while in the second way where we have less values in the observations, we did not take into account the observations that had missing values. Some data sets involve only numerical features, and the remaining include both numerical and categorical features. For each data set, Table 1 reports the total number of features and the number of categorical features in parentheses. All the data sets involve 2-class problems and they are analyzed with 10-fold cross validation. The algorithm was implemented in Fortran 90 and was compiled using the Lahey f95.

As it has already been mentioned, three approaches that use different classifiers, the 1nn, the knn and the wknn, are used. In all algorithms the value of k is changed dynamically depending on the number of iterations. Each iteration uses different k. The reason why k does not have a constant value is that we would like to ensure the diversity of solutions in each iteration of the algorithm. The determination of k is done by using a random number generator with a uniform distribution $(0, 1)$ in each iteration. Then, the produced number is converted to an integer k (e.g., if the produced number is in the interval $0.2 - 0.3$, then $k = 3$).

Table 1. Data Sets Characteristics

Data Sets	Observations	Features
Australian Credit (AC)	690	14(8)
Breast Cancer Wisconsin 1 (BCW1)	699	9
Breast Cancer Wisconsin 2 (BCW2)	683	9
German Credit (GC)	1000	24 (13)
Heart Disease (HD)	270	13(7)
Hepatitis 1 (Hep1)	155	19 (13)
Hepatitis 2 (Hep2)	80	19 (13)
Ionosphere (Ion)	351	34
Spambase (spam)	4601	57
Pima Indian Diabetes (PID)	768	8

For comparison purposes, three other algorithms have been developed using the Particle Swarm Optimization algorithm with different neighborhood topologies or different velocities equation. Two of these algorithms, the Particle Swarm Optimization with Global Neighborhood Topology (PSOGNT) and the inertial Particle Swarm Optimization (iPSO), use a global neighborhood topology while the other, the Particle Swarm Optimization with Local Neighborhood Topology (PSOLNT), uses a local neighborhood topology. PSOGNT uses the constriction velocities equation while iPSO uses the inertia velocities equation. In the PSOLNT, a local search neighborhood topology is used with a constant neighborhood for all particles in all iterations. For the selection of the size of the neighborhood, a number of different experiments with different neighborhoods were conducted and the one that gave better results for the Feature Selection Problem was selected. The reason that the PSOGNT and the PSOLNT are used is that we would like to compare the algorithm with algorithms that have the same characteristics with the proposed algorithm but they do not use the main characteristic of the proposed algorithm, the Expanding Neighborhood Topology. Also, an Ant Colony Optimization algorithm and a Genetic Algorithm were developed. All these algorithms use the same local search strategy (VNS) with the one used in the proposed algorithm (PSOENT). For analytical description of the iPSO, the Ant Colony Optimization algorithm and the Genetic Algorithm used in this paper and how they are designed and applied for the solution of the Feature Selection Problem please see [17].

Table 2. Parameters for all algorithms

	PSOENT	PSOLNT	PSOGNT	iPSO	ACO	GA
particles/ants/individuals	200	200	200	200	200	200
iterations/generations	50	50	50	50	50	50
VNS iterations	20	20	20	20	20	20
c_1	2.05	2.05	2.05	2	-	-
c_2	2.05	2.05	2.05	2	-	-
w_{max}	-	-	-	0.9	-	-
w_{min}	-	-	-	0.01	-	-
Neighborhoods	2 to 50	5	50	50	-	-
Evaporation parameter q	-	-	-	-	0.5	-
Probability of crossover	-	-	-	-	-	0.8
Probability of mutation	-	-	-	-	-	0.2

The parameters of all algorithms were selected after thorough testing. A number of different alternative values were tested and the ones selected are those that gave the best computational results concerning both the quality of the solution and the computational time needed to achieve this solution. After the selection of the final parameters, 10 different runs with the selected parameters were performed for each data set. The results presented in the tables are the best results found for each data set. The selected parameters for all algorithms are presented

in Table 2. In this Table, in the places where there are not any values, these parameters are not used in the corresponding algorithm. In the first column the name of the parameter is given. In the cases where more than one names exist it means that in each algorithm the parameter is used with different name, for example, the members of the population in the PSO implementations are called particles, in the ACO implementations are called ants and in GA implementations are called individuals.

Table 3 shows the Overall Classification Accuracy (OCA) and the average selected features (SNF) for the proposed algorithm and for all algorithms used in the comparisons. As it can be seen, the proposed algorithm gives superior results compared to the results of the other five algorithms as the average overall classification accuracy is 92.48%, while the average overall classification accuracies for the other five algorithms are 91.25% for the PSOGNT algorithm, 91.07% for the PSOLNT algorithm, 90.78% for the inertia Particle Swarm Optimization, 90.33% for the Ant Colony Optimization and 88.58% for the Genetic Algorithm.

If we examine the performance of each classifier separately, the results of the proposed algorithm are always better as in the 1nn classifier the average overall classification accuracy of the proposed algorithm is 92.17%, while the average overall classification accuracies of the other five algorithms are 91.00% for the PSOGNT algorithm, 90.91% for the PSOLNT algorithm, 90.78% for the inertia Particle Swarm Optimization, 90.28% for the Ant Colony Optimization and 88.45% for the Genetic Algorithm. For the knn classifier the average overall classification accuracy of the proposed algorithm is 92.7%, while the average overall classification accuracies of the other five algorithms are 91.53% for the PSOGNT algorithm, 91.33% for the PSOLNT algorithm, 91.05% for the inertia Particle Swarm Optimization, 90.59% for the Ant Colony Optimization and 88.6% for the Genetic Algorithm. Finally, for the wknn classifier the average overall classification accuracy of the proposed algorithm is 92.59%, while the average overall classification accuracies of the other five algorithms are 91.21% for the PSOGNT algorithm, 90.99% for the PSOLNT algorithm, 90.49% for the inertia Particle Swarm Optimization, 90.18% for the Ant Colony Optimization and 88.70% for the Genetic Algorithm.

As it can be seen the ranking of the algorithms based on the average classification accuracies is the same in all cases (all classifiers). The most important remark taken from these results is that the results of PSOGNT and PSOLNT are equally efficient. However, the results of PSOENT are better compared to the results of these algorithms as the expanding neighborhood topology gives to the particle more exploration and exploitation capabilities and, thus, makes the algorithm more effective with better results in all cases. Another important conclusion is that all algorithms which use the constriction factor perform better than the one that uses the inertia factor.

The average selected features for all algorithms in all runs of the algorithms are presented in Table 3. In general, there are $2^{numberoffeatures} - 1$ possible feature combinations and, thus, in our cases the problem with the fewest number of feature combinations is the Pima Indian Diabetes (namely $2^8 - 1$), while the most

difficult problem is the Spambase where the number of feature combinations is $2^{57}-1$. The significance of the solution of the feature selection problem using the proposed method is demonstrated by the fact that with the PSOENT algorithm the best solutions were found by using less features than the other algorithms used in the comparisons. More precisely, in the most difficult instance, the Spambase instance, the PSOENT algorithm needed between 21.25 to 21.85 average number of features in order to find their best solutions, while the other three PSO inspired algorithms (PSOLNT, PSOGNT, iPSO) needed between 21.28 - 24.32 average number of features to find their best solutions and the two other evolutionary algorithms (ACO and GA) needed between 22.85 to 25.01 average number of features. For all data sets, the proposed algorithm needs 10.11 average number of features, the three PSO inspired algorithms need between 10.24 to 10.63 average number of features and the two other evolutionary algorithms need between 10.62 to 10.71 average number of features.

Table 3. Classification Results (OCA(%)) and average selected number of features (SNF) for all algorithms

Data Sets	Classifier	PSOENT		PSOLNT		PSOGNT		iPSO		ACO		GA	
		OCA	SNF	OCA	SNF	OCA	SNF	OCA	SNF	OCA	SNF	OCA	SNF
AC	1nn	88.25	7.48	86.98	7.75	87.15	7.78	86.95	8.17	87.88	8.21	86.42	8.12
	knn	93.18	7.52	91.15	7.82	92.05	7.85	90.01	7.98	89.37	8.05	87.12	8.11
	wknn	91.57	7.45	90.25	7.95	90.88	8.05	89.39	8.12	88.18	8.23	86.12	8.07
BCW1	1nn	99.2	4.55	98.37	4.57	98.15	4.95	99.15	4.98	98.65	5.01	98.01	4.57
	knn	99.25	4.85	99.05	5.15	99.21	5.07	98.85	4.95	98.41	5.07	97.37	5.12
	wknn	99.38	4.79	99.17	5.21	98.89	4.85	99.05	4.84	98.08	5.18	97.48	5.01
BCW2	1nn	99.42	5.15	99.15	5.18	99.28	5.21	99.23	5.15	99.17	5.21	98.55	5.34
	knn	99.17	5.08	99.21	5.16	99.15	5.48	99.09	5.43	98.79	5.73	97.88	5.66
	wknn	99.05	5.21	98.54	5.35	98.57	5.65	97.65	5.53	98.03	5.49	97.45	5.87
GC	1nn	78.52	13.88	76.35	14.28	77.31	15.58	76.23	15.28	75.37	15.52	70.96	15.31
	knn	80.21	14.25	77.28	14.31	78.17	15.42	78.01	15.31	77.57	15.01	76.01	14.87
	wknn	79.48	14.28	78.35	14.15	78.21	15.28	77.11	15.31	78.27	15.12	77.35	14.32
HD	1nn	93.21	6.89	92.12	7.05	91.15	6.95	90.49	6.88	90.38	6.92	88.45	6.91
	knn	92.57	6.95	90.18	6.85	92.14	6.51	91.24	6.91	91.78	6.98	87.45	6.87
	wknn	92.48	6.57	91.25	6.45	90.75	6.57	90.88	6.77	91.22	6.88	89.39	6.94
Hep1	1nn	98.65	11.35	97.35	11.21	98.24	10.58	98.15	11.17	97.08	11.02	94.37	10.78
	knn	97.68	11.22	96.88	10.88	96.75	10.61	96.92	10.79	96.23	10.88	94.32	11.12
	wknn	97.85	11.18	97.25	10.95	97.17	10.85	97.12	10.98	96.23	11.09	95.31	11.23
Hep2	1nn	100	9.85	100	10.25	100	10.08	100	9.88	100	9.85	98.85	10.88
	knn	98.63	10.15	98.14	10.17	98.34	10.25	98.23	10.95	97.15	10.85	95.23	10.87
	wknn	98.45	10.28	98.25	9.95	98.17	10.78	98.18	10.95	96.75	10.82	94.29	10.74
Ion	1nn	98.62	15.85	98.17	16.25	98.21	16.15	98.17	16.21	97.12	16.34	95.08	16.05
	knn	97.55	16.28	96.75	16.14	97.01	16.28	96.14	16.21	96.37	16.38	95.11	16.45
	wknn	97.52	15.57	95.49	15.28	96.85	15.98	97.01	16.02	96.44	15.85	94.21	15.91
Spam	1nn	86.57	21.85	84.32	22.31	85.24	22.17	84.28	22.98	83.01	22.85	82.23	23.12
	knn	87.21	21.35	85.21	21.28	84.18	22.24	83.79	24.32	83.44	24.65	81.32	22.76
	wknn	87.45	21.25	84.18	22.15	85.29	23.08	83.18	23.88	82.11	24.97	81.18	25.01
PID	1nn	79.24	4.18	76.31	4.41	75.28	4.35	75.18	4.28	74.11	4.37	71.57	4.21
	knn	81.55	4.08	79.41	4.35	78.34	4.41	78.21	4.37	76.37	4.26	74.19	4.23
	wknn	82.68	4.05	77.25	4.28	77.29	4.25	75.39	4.25	76.47	4.37	74.24	4.28
Average	Total	92.48	10.11	91.07	10.24	91.25	10.44	90.78	10.63	90.33	10.71	88.58	10.62
	1nn	92.17	10.10	90.91	10.33	91.00	10.38	90.78	10.49	90.28	10.53	88.45	10.53
	knn	92.7	10.17	91.33	10.21	91.53	10.41	91.05	10.72	90.59	10.78	88.6	10.61
	wknn	92.59	10.06	90.99	10.17	91.21	10.54	90.49	10.66	90.18	10.8	88.70	10.74

If we examine each classifier separately the proposed algorithm performs always better as in the 1nn classifier the proposed algorithm needs 10.10 average number of features, the three PSO inspired algorithms need between 10.33 to 10.49 average number of features and the two other evolutionary algorithms need 10.53 average number of features, in the knn classifier the proposed algorithm needs 10.17 average number of features, the three PSO inspired algorithms need between 10.21 to 10.72 average number of features and the two evolutionary algorithms need between 10.61 to 10.78 average number of features, and in the wknn classifier the proposed algorithm needs 10.06 average number of features, the three PSO inspired algorithms need between 10.17 to 10.66 average number of features and the two other evolutionary algorithms need between 10.74 to 10.8 average number of features.

A statistical analysis based on the Mann-Whitney U-test for all algorithms, in all runs, is presented in Table 4. In this Table, a value equal to 1 indicates a rejection of the null hypothesis at the 5% significance level, which means that the method is statistically significant different from the other methods. On the other hand, a value equal to 0 indicates a failure to reject the null hypothesis at the 5% significance level, meaning that no statistical significant difference exists between the two methods. As it can be seen from this Table, at the 5% significance level the results with the PSOENT are statistically significant different from the results with the PSOGNT, iPSO, ACO and GA and there is no statistical significant differences with the results of PSOLNT.

Table 4. Results of Mann - Whitney test for all algorithms

5% significance level						
	PSOENT	PSOLNT	PSOGNT	iPSO	ACO	GA
PSOENT	-	0	0	1	1	1
PSOLNT	0	-	0	1	1	1
PSOGNT	0	0	-	0	1	1
iPSO	1	1	0	-	1	1
ACO	1	1	1	1	-	1
GA	1	1	1	1	1	-

The results of the algorithm are, also, compared (Table 5) with the results of a number of metaheuristic approaches from the literature. In these implementations, the same data sets are used as the ones used in this paper and, thus, comparisons of the results can be performed. More precisely, in Table 5 the results of the proposed algorithm are compared with the results of the following algorithms:

1. The Parallel Scatter Search algorithm proposed by Garcia Lopez et al. [14]. In this paper, three different versions of Scatter Search are proposed, named Sequential Scatter Search Greedy Combination (SSSGC), Sequential Scatter Search Reduced Greedy Combination (SSSRGC) and Parallel Scatter Search (PSS).

2. The Particle Swarm Optimization Linear Discriminant Analysis (PSOLDA) proposed by Lin and Chen [11].
3. The Particle Swarm Optimization Support Vector Machines (PSOSVM1, PSOSVM2) proposed by Lin et al. [13].
4. The Simulated Annealing Support Vector Machines (SASVM1, SASVM2) proposed by Lin et al. [12].
5. A Particle Swarm Optimization algorithm with a Nearest Neighbour classifier (PSONN) proposed by Pedrycz et al. [18].
6. A Genetic Algorithm using an adjacency matrix-encoding, GWC operator, and fitness function based on the VC dimension of multiple ODTs combined with naive Bayes (GOV - genetic algorithm for ODTs using VC dimension upper bound) proposed by Rokach [19].
7. A Scatter Search (SS-ensemble) with different classifiers like Support Vector Machines (SVM), Decision Trees (DT) and Back Propagation Networks (BPN) proposed by Chen et al. [2].
8. Three different metaheuristics (GRASP, Tabu Search and a Memetic algorithm) proposed by Casado Yusta [1].

Table 5. Comparison of the proposed algorithm with other metaheuristic approaches from the literature (OCA (%))

Method	Data Set								Average
	AC	BCW	GC	HD	Hep	Ion	Spam	PID	
PSOENT-1nn	88.25	99.42	78.52	93.21	100	98.62	86.57	79.24	90.48
PSOENT-knn	93.18	99.25	80.21	92.57	98.63	97.55	87.21	81.55	91.27
PSOENT-wknn	91.57	99.38	79.48	92.48	98.45	97.52	87.45	82.68	91.13
SSSGC	-	95.22	-	74.99	-	87.75	-	67.92	81.47
SSSRGC	-	94.88	-	74.99	-	87.12	-	67.66	81.16
PSS	-	95.11	-	74.91	-	87.35	-	68.10	81.37
PSOLDA	84.5	96.5	75.6	84.7	-	92.2	-	76.7	85.03
PSOSVM1	91.03	99.18	81.62	92.83	-	99.01	-	82.68	91.06
PSOSVM2	88.09	97.95	79.00	88.17	-	97.50	-	80.19	88.48
SASVM1	92.19	99.38	-	93.33	-	99.07	-	82.22	93.24
SASVM2	88.34	97.95	-	87.97	-	97.50	-	80.19	90.39
PSONN	-	-	74.7	83.9	-	94.6	-	-	84.4
GOV	85.35	97.13	-	-	81.29	-	-	-	87.92
SS-ensemble	91.74	99.46	85.49	96.24	97.46	-	-	83.92	92.39
GRASP	-	93	92.7	-	-	90.4	84.6	-	90.18
Tabu Search	-	92.6	92.7	-	-	90.6	82.64	-	89.64
Memetic	-	91.8	92.6	-	-	89	79.76	-	88.29

More precisely, the proposed PSOENT algorithm gives better results in three data sets, the Australian Credit (AC), the spambase (Spam) and the Hepatitis. For the other five data sets, the algorithms that perform better are: for the Breast Cancer Wisconsin, the Pima Indian Diabetes and the Heart Disease, the Scatter Search - ensemble proposed by [2], for the German Credit, the GRASP and the

Tabu Search proposed by [1], and for the ionosphere, the Simulated Annealing Support Vector Machines (SASVM1) proposed by [12]. In the last column of Table 5 the average values of the results of all algorithms are presented. The proposed algorithm with the knn classifier is ranked third (91.27%) in twenty three algorithms in total after the SASVM1 algorithm (93.24%) and the SS-ensemble algorithm (92.39%). The other two versions of the algorithm the one with the 1nn classifier (90.48%) and the other with the wknn classifier (91.13%) are ranked in sixth and fourth places respectively. Another remark is that the proposed algorithm performs better than the four other algorithms that are based on Particle Swarm Optimization.

5 Conclusions

In this paper, a new algorithm based on the Particle Swarm Optimization for the solution of the Feature Selection Problem is presented. This algorithm is a hybridization of the Particle Swarm Optimization algorithm with the Variable Neighborhood Search algorithm. The algorithm uses a local neighborhood topology where the size of the neighborhood begins with a small size of the neighborhood and it is expanding as the number of iterations increases. As a number of different variants of the Particle Swarm Optimization algorithm has been published, mainly using a different equation for the calculation of the velocities, we used a constriction factor for the velocities equation. Another issue that we have to deal with was the fact that the PSO algorithm is suitable for continuous optimization problems. Thus, it was a challenge to find an effective transformation of the solutions of PSO in discrete values without loosing information from this procedure. The algorithm was tested in 8 data sets that are usually used in the literature and gave very good results. The main challenge was to give an algorithm that it could combine the advantages of the exploration capabilities of a global neighborhood structure with the exploitation abilities of a local neighborhood structure. This was achieved as the algorithm gave better results when it was compared with a global neighborhood version of PSO and a local neighborhood version of PSO (with constant size of the neighborhood). This fact demonstrates the efficiency of the algorithm when it is used for the solution of an NP-hard problem, like FSP. In the future, this algorithm will be used for the solution of other NP-hard combinatorial optimization problems.

References

1. Casado Yusta, S.: Different metaheuristic strategies to solve the feature selection problem. Pattern Recognition Letters 30, 525–534 (2009)
2. Chen, S.C., Lin, S.W., Chou, S.Y.: Enhancing the classification accuracy by scatter-search-based ensemble approach. Applied Soft Computing (2010), doi:10.1016/j.asoc.2010.01.024
3. Chen, Y., Miao, D., Wang, R.: A rough set approach to feature selection based on ant colony optimization. Pattern Recognition Letters 31, 226–233 (2010)

4. Clerc, M., Kennedy, J.: The particle swarm: Explosion, stability and convergence in a multi-dimensional complex space. IEEE Transactions on Evolutionary Computation 6, 58–73 (2002)
5. Duda, R.O., Hart, P.E., Stork, D.G.:Pattern Classification and Scene Analysis, 2nd edn. John Wiley and Sons, New York (2001)
6. Engelbrecht, A.P.: Computational Intelligence: An Introduction, 2nd edn. John Wiley and Sons, England (2007)
7. Hansen, P., Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130, 449–467 (2001)
8. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of 1995 IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
9. Kennedy, J., Eberhart, R.: A discrete binary version of the particle swarm algorithm. In: Proceedings of 1997 IEEE International Conference on Systems, Man, and Cybernetics, vol. 5, pp. 4104–4108 (1997)
10. Kennedy, J., Eberhart, R., Shi, Y.: Swarm Intelligence. Morgan Kaufmann Publisher, San Francisco (2001)
11. Lin, S.W., Chen, S.C.: PSOLDA: A Particle swarm optimization approach for enhancing classification accurate rate of linear discriminant analysis. Applied Soft Computing 9, 1008–1015 (2009)
12. Lin, S.W., Lee, Z.J., Chen, S.C., Tseng, T.Y.: Parameter determination of support vector machine and feature selection using simulated annealing approach. Applied Soft Computing 8, 1505–1512 (2008)
13. Lin, S.W., Ying, K.C., Chen, S.C., Lee, Z.J.: Particle swarm optimization for parameter determination and feature selection of support vector machines. Expert Systems with Applications 35, 1817–1824 (2008)
14. Garcia Lopez, F., Garcia Torres, M., Melian Batista, B., Moreno Perez, J.A., Moreno Vega, J.M.: Solving feature subset selection problem by a parallel scatter search. European Journal of Operational Research 169, 477–489 (2006)
15. Maldonado, S., Weber, R.: A wrapper method for feature selection using support vector machines. Information Sciences 179(13), 2208–2217 (2009)
16. Marinakis, Y., Marinaki, M.: Particle swarm optimization with expanding neighborhood topology for the permutation flowshop scheduling problem. Soft Computing (2013), doi:10.1007/s00500-013-0992-z
17. Marinakis, Y., Marinaki, M., Doumpos, M., Zopounidis, C.: Ant colony and particle swarm optimization for financial classification problems. Expert Systems with Applications 36(7), 10604–10611 (2009)
18. Pedrycz, W., Park, B.J., Pizzi, N.J.: Identifying core sets of discriminatory features using particle swarm optimization. Expert Systems with Applications 36, 4610–4616 (2009)
19. Rokach, L.: Genetic algorithm-based feature set partitioning for classification problems. Pattern Recognition Letters 41, 1676–1700 (2008)
20. Suganthan, P.N.: Particle swarm optimiser with neighborhood operator. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1958–1962 (1999)
21. Uncu, O., Turksen, I.B.: A novel feature selection approach: Combining feature wrappers and filters. Information Sciences 177(2), 449–466 (2007)

Interleaving Constraint Propagation: An Efficient Cooperative Search with Branch and Bound

Eric Monfroy[1], Broderick Crawford[2,3] and Ricardo Soto[2,4]

[1] LINA, Université de Nantes, France
FirstName.Name@univ-nantes.fr
[2] Pontificia Universidad Católica de Valparaíso, Chile
FirstName.Name@ucv.cl
[3] Universidad Finis Terrae, Chile
[4] Universidad Autónoma de Chile, Chile

Abstract. The main characteristic of any constraint solver is Constraint propagation. Then it is very important to be able to manage constraint propagation as efficiently as possible, we present a hybrid solver based on a Branch and Bound algorithm combined with constraint propagation to reduce the search space. Based on some observations of the solving process constraint propagation is triggered by some rules. The results show that constraint propagation is profitable, but also that it is too costly to be executed at each node of the search tree, we show that is possible to make reasonable use of constraint propagation.

1 Introduction

The correct setting and proper selection of the most appropriate algorithm for solving a given problem has been investigated many years ago [14]. As well the considerations of [14] are still valid and can be considered at least from two complementary points of view. These two points of view are: 1) selecting solving techniques or algorithms from a set of possible available techniques, and 2) tuning an algorithm with respect to a given instance of a problem.

Furthermore, to adapt them to the process during solving their settings should be changed. An Autonomous Search [10] system should provide the ability to advantageously modify its internal components when exposed to changing external forces and opportunities. It corresponds to a particular case of adaptive systems with the objective of improving its problem solving performance by adapting its search strategy to the problem at hand.

In [9], a general definition and a taxonomy of search processes with respect to their computation characteristics (solver that are able to modify or adjust their strategies and parameters either a priori or adaptively, self-adaptation or supervised adaptation, ...) is given. In [5], a framework for dynamic adaptation of enumeration strategies of a constraint propagation-based solver was proposed. In this paper, we use a subsequent instantiation of the framework [12][1].

[1] We use the term framework instantiation to refer to any code that uses the framework.

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 52–61, 2013.

This instantiation manages dynamically some strategies and components of a hybrid solver based on Branch and Bound (B&B) combined with Constraint Propagation in order to reduce the search space. To this end, the framework we propose observes the resolution process, analyses the observations, and makes some decision to possibly dynamically adapt strategies and trigger some functions of the solver. With respect to the classification of [9] we propose an autonomous and supervised adaptive solving framework.

According [4], our proposal is a kind of cooperative search technique too. *Cooperative search consists of a search performed by agents that exchange information about states, models, entire sub-problems, solutions or other search space characteristics.*

In [7], many classification schemes were proposed for hybrid and parallel metaheuristics. According the first taxonomy proposed for cooperative search algorithms (based on the types of algorithms being used in the cooperative system and the implementation) our proposal is in the category of serial heterogenous algorithms. This class involves having different algorithms running in a pipeline fashion. The output of each algorithm supplied as an input to the next algorithm. This class is identified in the hybrid taxonomy as a high-level relay technique.

With our cooperative search we don't focus on improving the resolution of a single problem, but we are interested in quickly finding solutions on average for a set of problems. The experimental results are more than promising.

This paper is organized as follows. We present our proposal in Section 2. Experimental results are discussed in Section 3 and we conclude in Section 4.

2 Cooperative Search Framework

The framework is based on 4 components exchanging information: the first component runs a solver or a solver cooperation/hybridization based on some tunable modules/functions and on some solving strategies; the second one observes resolution and takes snapshots, i.e., some kind of quantitative summaries of the observations; the third one analyses snapshots and draws some indicators (that may be qualitative) about strategy and function quality; and the fourth one makes decisions to update strategy priorities and trigger functions [11,12].

2.1 The SOLVE Component

It is a solver, or an hybrid solver for solving constraint problems. This (hybrid) solver has at disposal several possible strategies and several functions that can be changed or triggered using some rules of the UPDATE component. For the experimentations, we consider a branch and bound algorithm: while there remain subproblems, choose one (depth-first selection in our case) and treat it. Compared to an usual B&B, this algorithm is hybrid (Algorithm 1): each subproblem P may be reduced by a propagation phase (e.g., [1]) to reduce the search space by eliminating values of variables that cannot participate in a solution.

Algorithm 1. Sketch of the hybrid algorithm

```
1: procedure HYBRID_B&B
2:     Optimum ← −∞
3:     Nodes ← (P_0, −∞)
4:     while Nodes ≠ ∅ do
5:         P ← getSubproblem(Nodes)
6:         PropagationPhase(P) (conditional)
7:         RelaxationPhase(P)
8:         checkSolution(P)
9:         branching(P, (P_1, ..., P_k))
10:        for (1 ≤ i ≤ k) do
11:            Nodes ← Nodes ∪ (P_i, bound(P))
12:        end for
13:        OptimumSolution ← Solution
14:        OptimumValue ← Optimum
15:    end while
16: end procedure
```

This technique is rather common (e.g., [15]), thus, we just give an overview of the method. We consider two models of the problem to be solved. The first one, M_{CSP} is based on Constraint Satisfaction Problem (CSP) and uses finite domain variables (i.e., $X_{CSP} \in \{v_1, \ldots, v_n\}$), and the second one, M_{LPM}, is a Linear Programming Model (LPM) (i.e., based on continuous variables $v_1 < X_{LPM} < v_n$). The two models are equivalent: at each node of the search tree, the problem is modeled by both techniques. Information discovered in a model is communicated to the other one (e.g., if X_{CSP} is reduced to the domain $\{v_1, \ldots, v_n\}$, then the corresponding variable X_{LPM} can be constrained by $v_1 < X_{LPM} < v_n$, and so on for objective functions and in the opposite way).

The (conditional) propagation phase is applied to the M_{CSP} model to reduce the search space by removing values of variables that cannot satisfy some constraints. The obtained information (reduction of domains) is communicated to the M_{LPM} model; the relaxation phase is applied to the relaxation of the M_{LPM} model (i.e., M_{LPM} without integrity constraints). In case there is no propagation phase, the algorithm acts as a usual B&B: the relaxation is solved to determine the unsatisfiability of the problem or an integer solution.

Then, when checking solution, the algorithm evaluates if the solution is feasible. If the solution is an integer with a better value than the actual one, the optimum value and the optimal actual solution are updated. If the solution is integer, but not better, it is discarded. If it is not integer, the branching is done on non integer variables, generating the respective subproblems w.r.t. the enumeration strategy (in this case, after some preliminary tests, we fixed the lexicographic order to select the variable, and the largest value of the variable).

In the following, we use our framework to (de)activate the propagation phase: by just changing the updating rules we obtain different types of hybridization.

2.2 The OBSERVATION Component

It aims at observing and recording information of the current search tree, i.e., it spies the solving process of the SOLVE component. These observations (called **snapshots**) can be seen as an abstraction of the resolution state at a time t. Taking a snapshot consists in extracting (since search trees are too large) some information from a solving state.

In our experimentations, snapshots are taken at each node of the search tree and they contain the following data:

- Characteristic of the problem:
  - V_{total} : total number of variables
  - R_{eq} (R_{leq}, R_{geq} resp.): number of constraints of type $=$ ($\leq$, $\geq$ resp.)
- Measures of the search tree
  - N_{expl} : number of explored nodes
  - N_{failed} : failed nodes (not leading to a solution)
  - N_{sol} : nodes with solutions
  - $Backs$: number of backtrackings
  - N_{prop} : nodes in which the propagation phase happened
  - $Depth$: current depth of the tree search
- Measures of the solving process
  - T_{prop} : CPU time used in the propagation phase
  - T_{relax} : CPU time used in the relaxation phase
  - V_{fixen} : variables fixed by enumeration
  - V_{fprop} : variables fixed by propagation
  - VF : total of fixed variables (i.e., $V_{fixen} + V_{fprop}$)

2.3 The ANALYSE Component

It analyses the snapshots taken by the OBSERVATION: it evaluates the strategies and functions, and provides **indicators** to the UPDATE component. They can be extracted, computed, or deduced from one or several snapshots.

Numeric indicators (δn) are quantitative results computed from snapshots.: e.g., the depth of the search (δn_{depth}), the number of fixed variables (δn_{fix}), or fixed by enumeration (δn_{fixen}). Indicators can be more complex: e.g., the difference of depth between 2 snapshots to give information on the progress in the search tree, or the difference between the depth (δn_{depth}) of the search and the variables fixed by enumeration (δn_{fixen}) which gives the indicator (δn_{gap}) on how many unsuccessful enumerations were performed on the last variable.

Boolean indicators (δb) reflect properties. Simple ones can be related to problems (e.g., there is a univariate constraint or a hard variable was fixed). More complex properties can be related to a quantitative or qualitative analysis of the snapshots, such as thrashing in the case of a constraint propagation based solver [5,6].

Indicators that we used for our experimentations are described later on.

2.4 The UPDATE Component

It makes decisions using the indicators: it makes interpretations of the indicators, and then updates the strategies priorities and/or triggers some functions of the SOLVE component. The knowledge of the UPDATE component is contained in a set of rules. The head of such a rule is a conjunction of conditions on the indicators (disjunctions can be handled by several rules). There are two types of rules: for priority update ($\Rightarrow$rules) and for function call ($\rightarrow$rules).

For priority update rules ($\Rightarrow$rules), the body is a conjunction of updates of strategies priorities:

$$\bigwedge_{i=1}^{l}(\sum_{j\in J_i}\omega_j \times \delta n_j)\ op\ c_j \wedge \bigwedge_{i=1}^{k}\delta b_i \Rightarrow \bigwedge_{i=1}^{l} p_i = p_i + f_i(\delta n_1, ..., \delta n_l)$$

where:

- the ω_j are the weights of each numeric indicator δn_j in the condition, the c_j are constants, the J_i are subsets of all the indicators, and $op \in \{\leq, \geq, =\}$;
- the δb_i are some Boolean indicators;
- the f_i are functions over the indicators that returns real numbers to increase or decrease the priority p_i of the strategy i;
- and the $\bigwedge_{i=1}^{l}$ in the body of the rule is an abuse of language to mean that the l priorities can be updated.

For function rules ($\rightarrow$rules) the body requests the application of a function, possibly with some parameters:

$$\bigwedge_{i=1}^{l}(\sum_{j\in J_i}\omega_j \times \delta n_j)\ op\ c_j \quad \wedge \quad \bigwedge_{i=1}^{k}\delta b_i \quad \rightarrow \quad function(...))$$

When the head of a rule is fulfilled (i.e., conditions are verified), its body is executed: for $\Rightarrow$rules, the priorities of the strategies (e.g., enumeration strategies) are updated in the SOLVE component. Whereas for $\rightarrow$rules, functions are triggered in the SOLVE component.

3 B&B + Constraint Propagation Solver

We now experiment with a B&B + Constraint Propagation solver: snapshots, indicators, and rules enable us to simply change the strategies of hybridization. Here, we do not use priority of strategies (See [13] for other strategies and rules). We use rules of the second type to activate (*propagation*()) or deactivate (*nopropagation*()) propagation phases. Our goal is not to design the best solver, but 1) to show how simple it is to change hybridization strategies in our framework, 2) to observe the role of propagation on various problems, and 3) to give some hints on how to manage propagation.

The solving process was written in Gecode [16]; the propagation is achieved by the propagators of Gecode while the components of the B&B algorithm (e.g., relaxation) are achieved by *lp_solve* [3] is a linear (integer) programming solver based

Table 1. Problem instances

Problem	Variables	Constraints	Best known sol.	Time
mbacp1	572	473	1	0.81
mbacp2	714	656	0	8.44
mbacp3	856	346	-	3.05
mbacp12	717	570	-	-
mbacp13	859	578	-	-
mbacp23	859	650	-	-
mbacp123	862	882	-	-
scp65	1000	200	161	-
scpa1	3000	300	253	-
scpb1	3000	300	69	-
sppaa01	8904	823	56138	238.76
sppaa03	8627	825	49649	12.47
sppaa04	7195	426	26402	319.19
sppnw18	10757	124	340160	2.19
mknapcb4a	250	5	59312	-
mknapcb4b	100	10	23064	-
mknapcb4c	100	10	22801	-
mknapcb2-1	100	10	22131	-

on the revised simplex method and the B&B method for the integers; it can be used as a library. The tests were run on an Intel Xeon 1.6GHz computer, with 4GB of memory. Each run was stopped after a time out of 1200 seconds.

The measures to evaluate the performance of the various hybridizations are: **v**alue of the **f**irst-found integer solution and **C**PU time (in s.) to obtain it; the same measures for the **b**est-found integer solution; and **C**PU time (in s.) to prove optimality. The solvers and their hybridizations were tested with various instances of 4 types of problems: the Multiple Balanced Academic Curriculum Problem (mbacp), the set covering problem (scp), the set partitioning problem (spp), the multidimensional knapsack problem (mknap). The instances and model of MBACP can be found in [8], and of the other problems in [2]. The size of the instances, in terms of variables and constraints, together with the best known solution and the best known CPU time are given in Table 1.

As a reference, Table 2 shows the results obtained with *lp_solve* alone. The columns show the value of the first solution and the time required to compute it; the best solution; the time required for proving optimality ("-" when optimality is not proven before the 1200 s. of timeout). With the same model, Gecode is faster to find a first solution, but of worse quality; it thus must explore a larger search space to find the optimum which it never reached before the timeout.

3.1 Basic Hybridization

In basic hybridization we do not consider any updating rule: propagation is triggered at each node of the search tree and the snapshots and indicators are

Table 2. Results for lp_solve and the basic hybridization

	lp_solve					Basic hybridization				
Problem	First solution		Best solution		Opt.	First solution		Best solution		Opt.
	Value	Time	Value	Time	Time	Value	Time	Value	Time	Time
mbacp1	4	4.2	1	17.6	326.2	4	2.8	1	6.1	212.5
mbacp2	4	15.1	0	315.5	320.1	4	2.2	0	10.4	11.7
mbacp3	4	2.2	1	20.4	-	4	1.0	1	86.3	-
mbacp12	4	34.9	1	213.0	594.2	4	9.5	1	133.1	340.8
mbacp13	4	32.4	1	210.9	222.6	4	3.2	1	35.1	39.2
mbacp23	4	3.4	1	175.4	191.7	4	2.3	1	79.3	84.3
mbacp123	4	157.5	4	157.5	-	4	56.9	3	395.6	-
scp65	166	0.2	161	9.0	23.4	166	0.2	161	10.4	26.1
scpa1	271	0.8	253	107.0	167.5	271	0.9	253	118.6	184.7
scpb1	75	0.9	69	11.4	33.1	75	1.2	69	13.9	38.8
sppaa01	56363	25.1	56363	25.1	-	56363	33.4	56363	33.4	-
sppaa03	49734	11.8	49649	54.3	71.7	49734	17.1	49649	97.2	127.0
sppaa04	27802	10.6	27507	121.6	-	27802	16.8	27507	150.6	-
sppnw18	342950	1.1	340160	28.5	28.5	342950	1.7	340160	39.3	39.5
mknapcb4a	21277	0.1	23057	956.6	-	21277	0.0	23050	466.3	-
mknapcb4b	20189	0.1	22704	878.8	-	20189	0.0	22704	1182.8	-
mknapcb4c	20978	0.1	22131	858.6	1115.7	20978	0.0	22131	1124.3	-
mknapcb2-1	57318	0.2	59015	1140.7	-	57318	0.1	59013	1018.1	-

not used. The results are presented in Table 2 and following we give few comments since this basic hybridization should be seen as a reference for the next hybridizations that make use of our updating rules.

MBACP. Times for proving optimality are significantly better than with the individual solvers (up to 29 times quicker for *mbacp*2). The cost of propagation is rather small, between 2% to 4% (depending on the instance) of the total execution time. However, propagation is quite inefficient.

SCP. The basic hybridization worsen the CPU times to get both the first and best solution. The propagation cost is around 5%: however, the reason is that propagation is quite inefficient for these problems and thus stops quickly.

SPP. Propagation is here more efficient, but costs between 26% to 40% of the total time.

MKNAP. These are the most difficult problems for the solvers. It seems that numerous valid solutions must be evaluated and the search tree is much larger.

3.2 Hybridization Based on Propagation Rate

This idea is coming from previous experimental observations of the number of variables fixed by propagation w.r.t. the depth of the tree search. We perceived that for the MBACP instances, propagation fixes numerous variables in the first level of the search tree; deeper, propagation is not so effective, even not at all

after a while. For SPP, this behavior is even more obvious, especially in the first levels. The behavior is a bit different for the MKNAP instances: propagation is not effective in the first levels because the instances are less constrained, and thus offer numerous possible solutions. But after some steps and cutting the problem into subproblems, propagation is effective.

Thus, our idea is to achieve more propagation phases when we observe that propagation fixes variables, and less propagation phases otherwise. We define the threshold th as the minimum number of variables fixed by propagation. In the snapshots, we have the total number of variables fixed by propagation and the number of nodes we have explored. Hence, we can get an indicator to reflect the efficiency of propagation at a given depth of the search tree.

These indicators enabled us to implement this idea: F is the snapshot taken at the father node; $snapshots(h)$ represents the set of snapshots taken at depth h.

- $\delta n_1 = \#snapshots(\ Depth_F\ )$ represents the quantity of snapshots taken at depth $Depth_F$ of the father node.
- $\delta n_2 = \sum_{G \in snapshots(Depth_F)} (V_{fprop})_G$ is the total number of variables fixed by propagation. Since we know the number of snapshots and the total number of variables fixed by propagation in each of these observations, we can determine the effectiveness of propagation at the depth of the father node.

Rule r_1 triggers propagation as long as we do not have any observation that proves the inefficiency of propagation. Rule r_2 evaluates the efficiency of propagation in the observed nodes; if it is greater than th, then propagation is triggered. If no rule applies, then propagation is deactivated.

$$\begin{array}{lll} r_1: & \delta n_1 = 0 & \rightarrow \ Propagation() \\ r_2: & (\ \delta n_2/\delta n_1\ ) \geq th & \rightarrow \ Propagation() \end{array}$$

To test this hybridization (see Table 3), the parameter th will vary from 0 to 20: with $th = 0$, propagation is triggered at each node of the search tree; experimentally, we observed that if $th > 20$ then propagation is never triggered.

MBACP. For the larger instances (e.g., m12, m13 y m123) this hybridization improves the results w.r.t. the basic hybrid. With a low threshold, we could shorten CPU times from 30% to 40%. As observed initially in the basic hybrid, propagation is fast but rather inefficient for MBACP: thus, to trigger propagation, a low threshold is required.

SCP. There is no improvement for proving optimality: the timings are similar to the ones of the basic hybrid, and the "best" th produces the same results as *lp_solve* alone. This confirms the idea that propagation does not improve this type of problems and slows down the solving process.

SPP. The results are better than for the basic hybridization, but a little bit worse than *lp_solve* alone. Except for the instance sppnw18, for which this hybridization improves time for optimality w.r.t. to all the other solvers (hybrid

Table 3. Hybridization triggering propagation w.r.t. to propagation efficiency

Problem	Best th	First solution Value	First solution Time	Best solution Value	Best solution Time	Opt. Time
mbacp1	1	4	2.8	1	6.1	212.2
mbacp2	1	4	2.2	0	10.4	11.7
mbacp3	4	4	1.0	1	6.8	-
mbacp12	4	4	6.2	1	43.8	234.7
mbacp13	6	4	3.1	1	22.3	26.2
mbacp23	2	4	2.3	1	50.9	56.2
mbacp123	2	4	65.5	3	155.8	-
scp65	13	166	0.2	161	9.0	23.1
scpa1	12	271	0.9	253	107.5	167.9
scpb1	2	75	1.2	69	11.6	33.3
sppaa01	5	56363	33.3	56363	33.3	-
sppaa03	3	49734	17.6	49649	90.1	116.3
sppaa04	5	27802	17.0	27412	960.3	-
sppnw18	6	342950	1.7	340160	27.4	27.6
mknapcb4a	6	21277	0.0	23057	991.7	-
mknapcb4b	3	20189	0.0	22704	914.3	-
mknapcb4c	12	20978	0.0	22131	880.4	1145.7
mknapcb2-1	6	57318	0.1	59013	778.9	-

or not). For this type of problems, although propagation effectively reduces the search space and the search tree, the overhead is higher than the speed-up for the global solving process.

MKNAP. On average, this hybridization is a bit faster than the best tested solver (*lp_solve* for these instances). It also gives some good quality solutions. After a closer look at the mknapcb4b instance, we can tune the threshold in order to get better solutions than *lp_solve*; however, the cost is to perform propagation 40% of the total time.

4 Conclusion

In this paper, we presented how simple adaptive hybridization strategies can be designed by just changing some few rules. We experimented on a cooperative B&B + Constraint Propagation solver in which propagation can be triggered responding to some observations of the solving process. The results show that some phases of propagation are beneficial to the B&B algorithm, but also that propagation is too costly to be executed at each node. The hybridization strategies are thus crucial to tune when to perform or not propagation. Depending on the types of problems, we have also seen that different hybridizations lead to different improvements, i.e., none of the hybridization is always the best.

We thus plan to develop a hyperheuristic approach integrating a learning module that would enable to select the hybridization with respect to the problem to be solved.

References

1. Apt, K.R.: Principles of Constraint Programming. Cambridge Univ. Press (2003)
2. Beasley, J.E.: Or-library: distributing test problems by electronic mail. JORS 41(11), 1069–1072 (1990)
3. Berkelaar, M.: lpsolve—simplex-based code for linear and integer programming
4. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput. Surv. 35(3), 268–308 (2003)
5. Castro, C., Monfroy, E., Figueroa, C., Meneses, R.: An approach for dynamic split strategies in constraint solving. In: Gelbukh, A., de Albornoz, Á., Terashima-Marín, H. (eds.) MICAI 2005. LNCS (LNAI), vol. 3789, pp. 162–174. Springer, Heidelberg (2005)
6. Crawford, B., Soto, R., Monfroy, E., Palma, W., Castro, C., Paredes, F.: Parameter tuning of a choice-function based hyperheuristic using particle swarm optimization. Expert Syst. Appl. 40(5), 1690–1695 (2013)
7. El-Abd, M., Kamel, M.: A taxonomy of cooperative search algorithms. In: Blesa, M.J., Blum, C., Roli, A., Sampels, M. (eds.) HM 2005. LNCS, vol. 3636, pp. 32–41. Springer, Heidelberg (2005)
8. Gent, I., Walsh, T.: Csplib: a benchmark library for constraints. Technical report, APES-09-1999 (1999), `http://csplib.cs.strath.ac.uk/`
9. Hamadi, Y., Monfroy, E., Saubion, F.: What is autonomous search? In: The Ten years of CP-AI-OR. Springer (2010) (to appear)
10. Hamadi, Y., Monfroy, E., Saubion, F. (eds.): Autonomous Search. Springer (2012)
11. Monfroy, E., Castro, C., Crawford, B.: Adaptive enumeration strategies and metabacktracks for constraint solving. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 354–363. Springer, Heidelberg (2006)
12. Monfroy, E., Castro, C., Crawford, B., Figueroa, C.: Adaptive hybridization strategies. In: ACM Symposium on Applied Computing, pp. 922–923 (2011)
13. Monfroy, E., Castro, C., Crawford, B., Soto, R., Paredes, F., Figueroa, C.: A reactive and hybrid constraint solver. Journal of Experimental and Theoretical Artificial Intelligence 25(1), 1–22 (2013)
14. Rice, J.: The algorithm selection problem. Technical Report CSD-TR 152, Purdue Univ. (1975)
15. Rodosek, R., Wallace, M., Hajian, M.: A new approach to integrating mixed integer programming and clp. Baltzer Journal (1998)
16. Schulte, C., Tack, G., Lagerkvist, M.: Gecode: Generic constraint development environment. In: INFORMS Annual Meeting (2006)

Automatic Tuning of GRASP with Evolutionary Path-Relinking

L.F. Morán-Mirabal[1], J.L. González-Velarde[1], and M.G.C. Resende[2]

[1] Tecnológico de Monterrey, Monterrey, Mexico
{lmoran,gonzalez.velarde}@itesm.mx
[2] AT&T Labs Research, Florham Park, NJ 07932, USA
mgcr@research.att.com

Abstract. Heuristics for combinatorial optimization are often controlled by discrete and continuous parameters that define its behavior. The number of possible configurations of the heuristic can be large, resulting in a difficult analysis. Manual tuning can be time-consuming, and usually considers a very limited number of configurations. An alternative to manual tuning is automatic tuning. In this paper, we present a scheme for automatic tuning of GRASP with evolutionary path-relinking heuristics. The proposed scheme uses a biased random-key genetic algorithm (BRKGA) to determine good configurations. We illustrate the tuning procedure with experiments on three optimization problems: set covering, maximum cut, and node capacitated graph partitioning. For each problem we automatically tune a specific GRASP with evolutionary path-relinking heuristic to produce fast effective procedures.

Keywords: Randomized heuristics, GRASP, biased random-key genetic algorithm, automatic tuning.

1 Introduction

Combinatorial optimization problems can often be "hard" to solve optimally using exact methods. Heuristics have been proved to find optimal or good suboptimal solutions in less time than required by exact methods. An example of this kind of heuristic is GRASP [6,7], which iteratively builds feasible solutions, and improves them applying a local search procedure. GRASP can be further hybridized with other intensification procedures, such as path-relinking [11] and evolutionary path-relinking [10,23].

There are numerous ways to hybridize GRASP with path-relinking and/or evolutionary path-relinking, resulting in a heuristic that is controlled by parameters and configurations. Selecting these heuristic settings is usually done manually through extensive experimentation, which in turn is time-consuming and only considers a small number of combinations. An approach to address these difficulties is to use an automatic tuning procedure.

Automatic tuning procedures have been proposed in the literature, and have been shown to improve the performance of optimization algorithms when compared

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 62–77, 2013.

with variants using manually-tuned settings (see, e.g. [1] and [14]). This paper proposes an automatic tuning procedure for parameters in a GRASP with evolutionary path-relinking by using a biased random-key genetic algorithm [12].

A BRKGA is an evolutionary algorithm based on the random-key genetic algorithm of Bean [3]. It evolves a population of solutions encoded as vectors of *random keys* applying genetic operators such as crossover and mutation. As a result, the algorithm returns the *fittest* individual of an evolved population (i.e. a best-valued solution). To apply a BRKGA for automatic tuning, we assume each individual encodes a set of parameters of the algorithm being tuned, and its fitness is a measure of the algorithm's performance using the encoded settings.

2 GRASP with Evolutionary Path-Relinking

A *greedy randomized adaptive search procedure* (GRASP) [6,7] is a multi-start heuristic for combinatorial optimization. It applies local search to a series of solutions generated with a greedy randomized algorithm. As initially proposed, GRASP did not have any memory mechanism. Laguna and Martí [17] introduced a memory mechanism in GRASP, hybridizing it with path-relinking [11]. In the resulting heuristic, a pool of the best solutions found during the search is maintained by the algorithm. After each GRASP local minimum is produced, a solution is selected at random from the pool, and the solution space spanned by the two solutions is explored by path-relinking. Evolutionary path-relinking [2,10,23] uses the path-relinking operator in an attempt to improve the pool of elite solutions. Given a pool, evolutionary path-relinking applies path-relinking between pairs of pool solutions, updating the pool if better solutions are found.

The pseudo-code in Algorithm 1 illustrates a GRASP+evPR for a minimization problem. The algorithm begins in line 1 by initializing the incumbent solution value f^* to a large number while in line 2 the pool E_s of elite solutions is initialized empty. The variable `it2evPR`, which measures the number of iterations left until evolutionary path-relinking is called, is initialized in line 3. All GRASP+evPR iterations take place in lines 4 to 18 until some stopping criterion is met. In line 5, a randomized greedy solution x is constructed and local search is applied to it in line 6. The resulting local minimum is tested for inclusion in the elite pool E_s in line 7. If E_s is not yet full, then x is accepted if it differs from all solutions currently in E_s. Otherwise, if E_s is full, x is accepted if it is better than at least one solution in the pool. If x is better than all pool solutions, then replaces the worst pool solution. Otherwise, if it is better than at least one solution but not all, then it replaces the least different solution having worse cost. Path-relinking is not applied until the second GRASP iteration. From then on, a solution x_p is selected from E_s in line 9 and path-relinking is applied between x and x_p in line 10. The resulting solution x is tested for inclusion in the elite pool in line 11. Evolutionary path-relinking is invoked every i_e GRASP iterations. This condition is tested in line 13, and if triggered, the updated pool is returned in line 14. The counter `it2evPR` is then re-initialized in line 15. At the end of each iteration in line 17, this counter is reduced by one unit. As a result, an elite pool solution having minimum cost is returned by the procedure in line 19.

```
1  f* ← ∞;
2  E_s ← ∅;
3  it2evPR ← i_e;
4  while stopping criterion is not satisfied do
5      x ← GreedyRandomized();
6      x ← LocalSearch(x);
7      E_s ← UpdateElite(E_s, x);
8      if |E_s| ≥ 2 then
9          x_p ← SelectPoolSolution(E_s, x);
10         x ← PathRelinking(x, x_p);
11         E_s ← UpdateElite(E_s, x);
12     end
13     if it2evPR = 0 then
14         E_s ← evPathRelinking(E_s, x);
15         it2evPR ← i_e + 1;
16     end
17     it2evPR ← it2evPR − 1;
18 end
19 return argmin{f(x) | x ∈ E_s}
```

Algorithm 1. GRASP with evolutionary path-relinking

3 Automatic Tuning Using a BRKGA

Each GRASP+evPR component shown in Algorithm 1, may in fact represent different algorithms. Discrete and continuous parameters can be used to define which specific configuration of these components are used. Given that there may exist a large number of these parameters and that each can potentially take on many values, tuning the parameters manually may be time-consuming and hard to specify, making reproduction difficult. An alternative is automatic tuning of parameters, where an algorithm is used in the tuning process.

Adenso-Diaz and Laguna [1] and Hutter et al. [14] were among the first to consider automatic tuning procedures. Adenso-Diaz and Laguna proposed CALIBRA, a framework which combines two different Design of Experiments approaches along with a local search procedure to tune up to five parameters. Hutter et al. proposed PARAMILS, a tuning methodology that combines stochastic local search procedures and mechanisms that tackle properties found in algorithm configuration problems. Both CALIBRA and PARAMILS have been shown to improve academic solvers and heuristics.

Festa et al. [9] proposed an automatic tuning procedure using a biased random-key genetic algorithm (BRKGA). They propose the tuning procedure for an implementation of a GRASP with path-relinking heuristic for the generalized quadratic assignment problem (GQAP). They consider 30 parameters and show that their tuning improves algorithm performance with respect to manually tuned parameters on four out of five instances. Pedrola et al. [21] recently

applied the approach of Festa et al. [9] to automatically tune a GRASP heuristic for the multilayer IP/MPLS-over-Flexgrid optimization problem. They use five small traffic instances to tune a simple GRASP with three parameters.

BRKGAs evolve population of vectors of random keys (or individuals) applying Darwin's principle of survival of the fittest [12]. A BRKGA works with a fixed-size population $\mathcal{P}$ made up of $|\mathcal{P}|$ vectors of n randomly generated numbers in the real interval $(0, 1]$ (random keys). A *decoder* is a deterministic algorithm that takes as input a vector of random keys and outputs its fitness value.

At each generation of a BRKGA, the population is partitioned into a smaller set $\mathcal{P}_e$ of elite individuals and a larger set $\mathcal{P}_{\bar{e}}$ with the remaining individuals. The evolutionary dynamics of a BRKGA are as follows. First, all elite individuals are copied, without change, to the population of the next generation $\mathcal{P}^+$. Then, a set $\mathcal{P}_m$ of mutant individuals (i.e. newly generated vectors of random keys) is inserted into $\mathcal{P}^+$. The first two steps account for $|\mathcal{P}_e| + |\mathcal{P}_m|$ individuals and therefore $p_x = |\mathcal{P}| - |\mathcal{P}_e| - |\mathcal{P}_m|$ individuals are required for $\mathcal{P}^+$ to be complete. This is done through mating of p_x pairs of individuals from the current population, one from $\mathcal{P}_e$ and another from $\mathcal{P}_{\bar{e}}$. Individuals are selected for mating at random and with replacement.

Let a and b denote the elite and non-elite individuals to be mated, and let c denote the resulting offspring. Mating is done with parameterized uniform crossover [24], where a biased coin is tossed n times, to determine from which parent the offspring will inherit each key. The coin has probability $p_h > 0.5$ to result in heads. For $i = 1, \ldots, n$, the i-th component of c receives the i-th component of a if the coin toss results in heads or the i-th component of b otherwise. This way, c has a greater chance to inherit the keys of its elite parent. Also, since such parent is selected from the smaller set $\mathcal{P}_e$, it has a greater chance of mating than a non-elite parent. Bean [3] proposed a similar algorithm, except that parents are selected at random from the entire population and the coin flip does not necessarily favor the more fit parent.

A BRKGA is summarized in the pseudo-code of Algorithm 2. It takes as input the sizes of the population $\mathcal{P}$, the elite set $\mathcal{P}_e$, and mutant set $\mathcal{P}_m$ (such that $|\mathcal{P}_e| + |\mathcal{P}_m| \leq |\mathcal{P}|$ and $2 \times |\mathcal{P}_e| \leq |\mathcal{P}|$), the size of the random-key vector (n), and the coin-toss probability of heads ($p_h > 0.5$). In line 1 the initial population is generated. The algorithm runs through several generations, until a stopping criterion is met. The operations taken in each generation are expressed in lines 2 to 19. In line 3, the fitnesses of all new individuals in population $\mathcal{P}$ are evaluated. Population $\mathcal{P}$ is then partitioned in line 4 into a set $\mathcal{P}_e$ of elite individuals and a set $\mathcal{P}_{\bar{e}}$ with the remaining population. The population of the next generation $\mathcal{P}^+$ is initialized with the elite set of the current population in line 5. In lines 6 and 7 the mutant set $\mathcal{P}_m$ is generated and added to $\mathcal{P}^+$. The remainder of $\mathcal{P}^+$ is completed in lines 8 to 17. For each remaining individual, parents a and b are selected at random in lines 9 and 10 and mating is applied in lines 11 to 15 to produce offspring c, which is added to $\mathcal{P}^+$ in line 16. A generation is completed in line 18 by making the population of the current generation that of the next generation. Finally, the fittest individual of the final population is returned in line 20.

```
Data: |P|, |P_e|, |P_m|, n, p_h
Generate population P with individuals having n random-keys;
while stopping criterion is not satisfied do
    Evaluate fitness of each new individual in P;
    Partition P into sets P_e and P_ē;
    Initialize next population: P+ ← P_e;
    Generate mutants P_m each having n random-keys ∈ (0, 1];
    P+ ← P+ ∪ P_m;
    for i ← 1 to |P| − |P_e| − |P_m| do
        Select parent a at random from P_e;
        Select parent b at random from P_ē;
        for j ← 1 to n do
            Toss biased coin having probability p_h > 0.5 of heads;
            if Toss is heads then c[j] ← a[j];
            else c[j] ← b[j];
        end
        P+ ← P+ ∪ {c};
    end
    P ← P+;
end
return argmin{f(x) | x ∈ P}
```

Algorithm 2. Biased random-key genetic algorithm

3.1 Encoding and Decoding

Let n denote the number of algorithm configurations and parameters. These are encoded as a vector χ of n real-valued random keys, each in the range $(0, 1]$. Suppose an algorithm configuration consists of a finite set of components, each of which can take on a single state. For example, *path-relinking type* is a component which can take on a single state from the set { *forward*, *backward*, *back&forth*, *mixed* }. Each state from these finite sets correspond to a different interval in the range $(0, 1]$. A set with s states would associate state 1 with interval $(0, 1/s]$, state 2 with interval $(1/s, 2/s]$, and so on. To decide which state the i-th component takes, the decoder identifies which interval contains the random key $\chi[i]$ and assigns the state corresponding to that interval to the component. A real-value parameter in the range $(l, u]$ and encoded as $\chi[i]$ is decoded as $l + (u - l) \times \chi[i]$. For example the i-th parameter *path-relinking truncation* can take on any value in the real interval $(0.2, 0.7]$. Therefore, the random key $\chi[i] = 0.44$ is decoded as $0.2 + (0.7 - 0.2) \times 0.44 = 0.42$. Note that each decoding of a vector of random keys is independent of the other, and hence can be parallelized to speed up the automatic tuning procedure.

4 GRASP+evPR for Three Optimization Problems

In this section, we describe three GRASP+evPR heuristics for combinatorial optimization problems, which in turn will be tuned using BRKGA in Section 5.

4.1 Set Covering

Let $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$ be a set of n elements (i.e. the universe) and let $J = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_m\}$ be a collection of subsets of $\mathcal{U}$ with associated costs $c_1, c_2, \ldots, c_m$, respectively. The *set covering problem* (SCP) consists in finding a minimum cost collection of sets $\mathcal{S}$ from J, such that the union of the sets in $\mathcal{S}$ is $\mathcal{U}$. The cost of the cover is defined as $\sum_{\mathcal{J}_j \in \mathcal{S}} c_j$. Set covering is NP-hard [16].

Feo and Resende [6] introduced a GRASP for set covering. Their construction procedure is based on the greedy algorithm of Johnson [15]. This greedy algorithm starts with an empty cover $\mathcal{S} = \emptyset$ and among unselected subsets, selects a subset $\mathcal{J}_j^*$ that maximizes the ratio κ_j/c_j, where κ_j is the number of uncovered elements $e_i \in \mathcal{U}$ that become covered if $\mathcal{J}_j^*$ is added to the solution.

Instead of selecting an element that maximizes the ratio κ_j/c_j, our the construction phase creates a restricted candidate list (RCL) which consists of all unselected subsets $\mathcal{J}_j \in J \setminus \mathcal{S}$ such that $\kappa_j/c_j \geq g_{\max} - \alpha \times (g_{\max} - g_{\min})$, where $g_{\max} = \max\{\kappa_j/c_j \ : \ \mathcal{J}_j \in J \setminus \mathcal{S}\}$, $g_{\min} = \min\{\kappa_j/c_j \ : \ \mathcal{J}_j \in J \setminus \mathcal{S}\}$, and α is a real number such that $0 \leq \alpha \leq 1$. An element $\mathcal{J}_j^*$ is selected at random from the RCL and added to $\mathcal{S}$. This is repeated until $\mathcal{S}$ is a complete cover.

Three variants of GRASP construction are considered. The first uses a *fixed* value for α, while the second selects at *random* a value of α from the uniform interval $[\alpha_{\min}, \alpha_{\max}]$ each time construction is called. The third variant uses a value of α selected at random from a finite set of α values with probabilities favoring those values that produced better solutions in previous constructions. The number n_α of α values varies from 2 to 10 and the values go from 0 to 0.8. This approach, called *Reactive GRASP*, was proposed by Prais and Ribeiro [22].

There are three types of moves in our local search phase of GRASP: remove one superfluous cover element (*remove-1*); remove all superfluous cover elements in increasing order of cost c_j (*remove-all*); swap a cover element $\mathcal{J}_j^{\text{out}}$ with an unselected element $\mathcal{J}_j^{\text{in}}$ such that the cover is kept complete if the swap is made (*swap-2*). The swapped elements $\mathcal{J}_j^{\text{out}}$ and $\mathcal{J}_j^{\text{in}}$ are determined by scanning $\mathcal{S}$ in decreasing order of cost and $\mathcal{J} \setminus \mathcal{S}$ in increasing order of cost, respectively. Moves are done using one of two options: first improving and best improving.

Our path-relinking strategy defines the symmetric difference $\delta_{\mathcal{S},\mathcal{S}_t}$ between any two solutions $\mathcal{S}$ and $\mathcal{S}_t$, as the cover elements present in $\mathcal{S}$ but not in $\mathcal{S}_t$. At each path-relinking step, $\mathcal{S}$ can be in either one of two states: feasible or infeasible. In the feasible state, the greedy move selects from $\delta_{\mathcal{S},\mathcal{S}_t}$, the element whose inclusion or removal from $\mathcal{S}$ results in the largest cost reduction. On the other hand, in the infeasible state, the greedy move selects from $\delta_{\mathcal{S},\mathcal{S}_t}$, the element whose inclusion or removal results in the largest reduction in infeasibility. Moves do not necessarily have to be greedy. If path-relinking is greedy randomized, then a real parameter $\alpha_p \in [0, 1]$ determines the proportion of best elements in $\delta_{\mathcal{S},\mathcal{S}_t}$ that are placed in a RCL so that one can be selected at random.

Path-relinking comes in several flavors. In *forward* path-relinking, $\mathcal{S}_l$ is the local optimum found by local search and $\mathcal{S}_t$ is a solution selected at random from the elite pool. In *backward* path-relinking the roles of $\mathcal{S}_l$ and $\mathcal{S}_t$ are reversed. In *back&forth* path-relinking, *backward* is applied first and then *forward* is applied

next. In *mixed* path-relinking a path is started from $\mathcal{S}_l$ and another from $\mathcal{S}_t$, and both meet in the middle. Finally the entire path need not be explored and a truncated variant is possible where $\gamma = 20\%$ to 80% of the path is explored.

The evolutionary path-relinking phase is triggered every i_e GRASP iterations, where i_e is a tunable parameter. The algorithm produces a sequence of elite pools $E_s^0, E_s^1, \ldots$, starting from the current elite set, i.e. $E_s^0 = E_s$. At iteration k all elite solutions in pool E_s^k are copied to pool E_s^{k+1}. While there are pairs $\{x, y\}$ of solutions in E_s^k that have not been considered, a path-relinking operator is applied to the pair and the resulting solution z becomes a candidate to enter E_s^{k+1}. The procedure stops if the sorted solution values of E_s^k and E_s^{k+1} are identical, returning set E_s^{k+1} as a result. Otherwise, k is incremented by one unit, and another iteration takes place. Evolutionary path-relinking employs a greedy randomized move strategy with tunable RCL parameter $\alpha_q \in [0, 1]$.

4.2 Maximum Cut

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{1, \ldots, n\}$ is the set of vertices and $\mathcal{E}$ is the set of edges, and let $w_{i,j}$ be the weights associated with edges $(i, j) \in \mathcal{E}$. The *maximum cut problem* (max-cut) consists in finding a cut $\mathcal{C} \subseteq \mathcal{E}$ of maximum weight which partitions the vertices in $\mathcal{V}$ in two non-empty sets. The weight of the cut $\mathcal{C}$ is defined as $\sum_{(i,j)\in\mathcal{C}} w_{i,j}$. The decision version of the max-cut problem was proven to be NP-Complete by Karp [16].

Festa et al. [10] introduced a GRASP with path-relinking heuristic for the max-cut problem. The construction phase of their heuristic uses a greedy function that takes into account the contribution to the objective function achieved by assigning a particular vertex into one of the subsets that define the cut, i.e. $\mathcal{C}$ and $\bar{\mathcal{C}}$. The greedy function is related to the sum of the weights of its outgoing arcs. Their local search procedure works by starting from the first elements of $\mathcal{C}$ and $\bar{\mathcal{C}}$ and in turn checks whether moving the elements from one set to the other leads to an improvement of the objective function.

In the path-relinking phase of Festa et al. [10], a solution y is represented as an n-dimensional binary vector, such that $y_i = 1$ if vertex $v_i \in \mathcal{C}$, and $y_i = 0$ if vertex $v_i \in \bar{\mathcal{C}}$. The procedure starts by computing the symmetric difference between the initial solution x and the target solution t, defined to be $\Delta_{x,t} = \{i \mid x_i \neq t_i,\ i = 1, \ldots, n\}$. At each path-relinking step, an index from $\Delta_{y^k,t}$ is selected and used obtain the next solution in the path, i.e. y^{k+1}. This is done by evaluating, for each $i \in \Delta_{y^k,t}$, the cost change g_i resulting from flipping the value of y_i^k. The greedy move selects from $\Delta_{x,t}$, the index corresponding to the largest g_i value. Two path-relinking flavors, *forward* and *backward*, are considered. An evolutionary strategy is used as a post-processing phase.

Our construction phase is similar to the one of Festa et al., except that it starts by ranking edges in decreasing order of edge weights and creates an RCL, where an edge $(i, j) \in \mathcal{E}$ is part of the RCL if $w_{i,j} \leq w^* - \alpha \times (w^* - w_*)$ where $w^* = \max\{w_{i,j} : (i, j) \in \mathcal{E}\}$ and $w_* = \min\{w_{i,j} : (i, j) \in \mathcal{E}\}$. An edge $(i^*, j^*) \in$ RCL is selected at random, assigning endpoint i^* to $\mathcal{C}$ and endpoint j^* to $\bar{\mathcal{C}}$.

The remaining nodes are placed in the partitions as done in Festa et al. As with the SCP, three variants of construction are proposed: *fixed*, *random*, and *reactive*.

During local search, nodes are scanned in decreasing order of their degrees. We allow three types of moves: *move*-1, *move-x*, and *move-max*. In *move*-1, a single node is moved either from $\mathcal{C}$ to $\bar{\mathcal{C}}$ or vice-versa. In *move-x*, a tunable portion $x \in [1\%, 20\%]$ of the nodes are allowed to move. Finally, in *move-max*, there is no limit on the number of nodes that can change partition during the search. There are three options for scanning the nodes using the three moves described: *check-once*, *check-until*, and *variable*. In *check-once*, each node is scanned only once during an invocation of local search. In *check-until*, nodes are scanned until there is no further improving move available. In *variable*, a move and an option are selected at random each time local search is invoked.

Our path-relinking phase allows for greedy or greedy randomized moves. If greedy randomized, the tunable parameter α_p determines the size of the RCL. We allow for *forward*, *backward*, *back&forth*, and *mixed* configurations of path-relinking. As in the SCP, the path explored can be either complete or truncated according to a tunable parameter $\gamma \in [0.2, 0.8]$. Evolutionary path-relinking is triggered every i_e GRASP iterations and the path-relinking operator is greedy randomized with a tunable parameter $\alpha_q \in [0, 1]$.

4.3 Node Capacitated Graph Partitioning

Given a node- and edge-weighted directed graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of arcs. For each node $v \in \mathcal{V}$, let $p_v \in \mathbb{Z}^+$ denote a non-negative integer node weight and for each arc $(u, v) \in \mathcal{E}$, let $q_{u,v} \in \mathbb{Z}^+$ denote a non-negative integer arc weight. In the *node capacitated graph partitioning* problem (NCGPP) we wish to partition the set of nodes into n clusters $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ such that, for $i = 1, 2, \ldots, n$, the weight sum of the nodes assigned to cluster $\mathcal{C}_i$ is no greater than the capacity $c_i \in \mathbb{Z}^+$ of the cluster. Furthermore, we seek the assignment that minimizes the edge weight sum Q for all edges having endpoints assigned to different clusters. Mehrotra and Trick [19] and Ferreira et al. [8] were among the first to study this problem, proposing a branch and price algorithm and a branch and cut algorithm, respectively. Deng and Bard [5] proposed a GRASP with path-relinking heuristic for capacitated clustering, essentially the same problem.

The GRASP+evPR heuristic we consider for the NCGPP was proposed by Morán-Mirabal et al. [20]. The construction phase builds a solution one node to cluster assignment at a time. Assignments are made as long as the capacity of the cluster is not exceeded. Let $\bar{\mathcal{V}}$ be the set of all yet-unassigned nodes. When a cluster k being scanned is empty, a node $i \in \bar{\mathcal{V}}$ is assigned to k with a probability proportional to the sum of edge weights between i and all the nodes in $\bar{\mathcal{V}}$ (e.g. the greedy choice would select the node in $\bar{\mathcal{V}}$ with the maximum edge-weight sum). Once an assignment is made, the available capacity of the cluster c_k is updated.

When one node is assigned to k, the following assignments are selected using a greedy function $g(i)$ that considers the edge weight sum between the nodes

assigned to k and a node $i \in \bar{\mathcal{V}}$. An RCL is formed such that $g(i) \geq g^* - \alpha(g^* - g_*)\}$, where $g_* = \min\{g(i) \mid i \in \bar{\mathcal{V}}\}$, $g^* = \max\{g(i) \mid i \in \bar{\mathcal{V}}\}$, and the tunable parameter $\alpha \in \mathbb{R}$ is such that $0 \leq \alpha \leq 1$. A node in the RCL is selected uniformly at random and is assigned to cluster k. This implementation considers the same three variants of construction used in the SCP, i.e. *fixed*, *random*, and *reactive*.

Once the construction phase produces a solution, local search attempts to reduce the sum of edge weights Q by making changes in the assignment. Morán-Mirabal et al. [20] propose three local search move types that scan the nodes in increasing order of their total node weight: *move-1*, *move-max*, and *swap-2*. In *move-1*, the procedure is restarted at the first node in the permutation, whereas in *move-max* it proceeds to the next node in the permutation. In *swap-2*, pairs of node assignments are considered for swapping. The number of pairs considered for swapping is limited by the tunable parameter β, where $0.01 \leq \beta \leq 0.3$. Three local search options are considered: *check-once*, *check-until*, and *variable*. In *check-once*, each node is scanned only once. In *check-until*, nodes are scanned until there is no further improving moves. Finally, in *variable*, at each invocation of local search a move type and an option are chosen at random.

The path-relinking phase defines the symmetric difference $\Gamma(\Pi_s, \Pi_t)$ between solutions Π_s and Π_t as the set of nodes assigned to different clusters. At each path-relinking step a greedy function $h(i)$ that considers the ratio between the change in the sum of edge weights Q and the capacity utilization resulting from an assignment is used. The function penalizes moves leading to capacity deficits.

An RCL is defined such that $h(i) \geq h^* - \alpha_p(h^* - h_*)$, where $h_* = \min\{h(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, $h^* = \max\{h(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, and the tunable parameter $\alpha_p \in \mathbb{R}$ such that $0 \leq \alpha_p \leq 1$. A move is selected from the RCL uniformly at random. The path-relinking operator of Morán-Mirabal et al. [20] generates a sequence of neighboring solutions, each of which may be feasible or infeasible, and selects among the feasible solutions, a solution with the lowest value of Q. If all solutions explored are infeasible, then the path-relinking phase is deemed unsuccessful and GRASP+evPR continues to the next phase.

This implementation allows *forward*, *backward*, *back&forth*, and *mixed* path-relinking strategies, each of which explores a complete or truncated path. The amount of truncation is determined by the tunable parameter $\gamma \in [0.2, 0.7]$. Evolutionary path-relinking is triggered every i_e GRASP iterations, where i_e is a tunable parameter. Evolutionary path-relinking applies the greedy randomized path-relinking operator using a tunable parameter $\alpha_q \in [0, 1]$.

5 Experimental Results

In this section we define a set of benchmark instances and test the performance of the BRKGA tuning procedure for each of the three problems presented in Section 4.

5.1 Instances

For the SCP we consider 20 benchmark instances from Beasley [4]. These instances are from three different problem sets, ten from set 4, eight from set 5,

Table 1. Manual vs. tuned GRASP+evPR performance on set covering instances. Maximum time and average time-to-target (TTT) values are in seconds.

Instance			GRASP+evPR (manual)		GRASP+evPR (tuned)	
Name	Target	Max Time	Avg. Cost	Avg. TTT	Avg. Cost	Avg. TTT
scp41	429	3000	430.03	701.301	*430.00	38.518
scp42	512	5000	517.56	711.059	513.93	713.783
scp43	516	3000	518.78	633.946	516.36	299.205
scp44	494	400	495.13	114.815	*494.00	20.131
scp45	512	4000	514.70	200.115	512.98	706.002
scp46	560	3500	563.01	470.884	560.71	501.211
scp47	430	3500	430.96	686.634	430.35	591.294
scp48	492	4500	**492.99**	11.827	**492.98**	33.537
scp49	641	5000	648.24	1380.262	645.45	780.404
scp410	514	400	**514.28**	88.411	**514.00**	17.782
scp51	253	5500	255.00	1096.633	254.06	558.792
scp52	302	5500	307.83	1118.605	305.51	1777.903
scp53	226	4500	227.40	809.426	227.01	844.274
scp54	242	400	242.66	93.672	242.05	49.360
scp55	211	5000	211.98	70.063	211.71	659.301
scp56	213	250	**213.18**	69.990	**213.00**	19.600
scp58	288	4000	289.00	848.211	288.40	322.173
scp59	279	4000	279.61	578.748	279.46	662.917
scp61	138	5000	139.87	661.849	139.65	214.398
scp64	131	100	*131.00	8.337	*131.00	5.058

Boldface indicates there is no significant statistical difference when applying a Wilcoxon-Mann-Whitney test.
* indicates a Wilcoxon-Mann-Whitney test could not be applied since all runs returned the same cover cost.

and two from set 6. All instances have 200 rows, 1000 or 2000 columns, and a graph density of 2% or 6%.

For max-cut we consider 21 benchmark instances from Helmberg and Rendl [13]. These instances divide into seven subsets of three which share size and structure. The subsets range from 800 to 2000 nodes, 1600 to 19900 edges, and a graph density of 0.2% to 6.0%.

For the NCGPP we use a 20 of the synthetic instances from Morán-Mirabal et al. [20]. The instances divide into four subsets of five which share size and structure. Each subset is created combining number of nodes $(200, 400)$ and number of clusters $(15, 25)$ in the set.

5.2 The Experiments

For each problem, two types of experiments are done, automatic tuning experiments and comparison of algorithm performance using automatically-tuned and manually-tuned settings. All implementations of GRASP+evPR and BRKGA use an implementation of the Mersenne Twister algorithm [18] as their random-number generator. BRKGA was implemented using the API described in Toso and Resende [25]. All experiments were done using the Condor job control system [26] which submitted jobs to either a cluster running Intel Xeon X5650 processors at 2.67 GHz, or a cluster running Intel Xeon E5530 processors at 2.4 GHz.

A total of 16 tunable parameters are considered for each problem. They include the many options for construction, local search, path-relinking, and evolutionary path-relinking described in Section 4. To measure the fitness of individuals during BRKGA tuning, our decoder makes n_s independent GRASP+evPR runs of n_{it} iterations using the decoded parameters, and returns the average best solution found as its fitness. Therefore, the fittest individual of a BRKGA tuning procedure is the combination of GRASP+evPR parameters that return a best result for a given simulation. We take advantage of the decoding independence of individuals

and use t_{max} parallel threads in the tuning experiments. The actual number of threads used depends on the availability of the cluster to which a job is submitted.

Each GRASP+evPR heuristic was programmed from the ground up and during the implementation of each of its components, a manual tuning procedure was performed. Manual tuning considered a subset of the instances for each problem, therefore a single set of parameters and configurations were selected as a result. For ease of notation, from now on we refer to the manually-tuned parameters as *manual* and the automatically-tuned parameters as *tuned*. For each manual vs. tuned experiment, we use the Wilcoxon-Mann-Whitney test with confidence level 95% to assess the statistical significance of the results.

Set Covering. Each SCP instance was automatically tuned using a BRKGA with $|\mathcal{P}| = 100$, $|\mathcal{P}_e| = 20$, $|\mathcal{P}_m| = 15$, $p_h = 0.7$ and a stopping criterion of 20 generations. The fitness of each set of parameters and configurations in an individual were evaluated with $n_{it} = 200$, $n_s = 30$ and $|E_s| = 10$. Parameter t_{max} was set to 30 parallel decoding threads. On average, tuning of each instance took about 1900 minutes to complete.

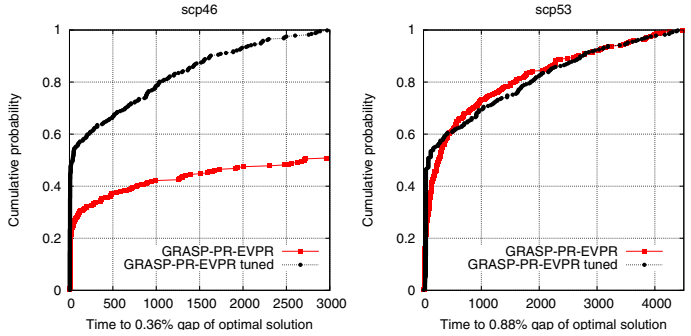


Fig. 1. Two examples of set covering GRASP+evPR performance (tuned vs. manual). Times are in seconds.

Most instances are automatically tuned to either *fixed* or *random* construction. Only two instances were tuned to *reactive*. All instances are tuned to *remove-all* as local search move and most of them are tuned to be *best improving*. All instances are tuned to use *back&forth* path-relinking and most of them are *greedy*. Also the majority are tuned to a truncated path with length $\leq 70\%$. Finally all instances are tuned to use evolutionary path-relinking with α_q values that are at most 0.57.

To measure the quality of the tuned parameters, a set of experiments was run for each instance and then compared against the results using a set of previously defined manual parameters. First a target with gap of 1% or less with respect to the optimal solution cost was selected and a maximum runtime to achieve

such target was set. Next, a total of 300 independent runs were made and the final solution costs and times taken were saved. Table 1 shows averages over the 300 runs. Note that in all but one instance the average solution cost is less with *tuned*. The average times vary from one instance to the other but in half of them a speed up is seen when using *tuned*. Also, whenever there is no significant statistical difference between both methods, *tuned* tends to be faster. Figure 1 shows examples of empirical runtime distributions for two of the instances considered. Each distribution shows how *tuned* GRASP+evPR either has a performance comparable to *manual* (`scp53`)or outperforms *manual* noticeably (`scp46`).

Maximum Cut. Each max-cut instance was automatically tuned using a BRKGA with $|\mathcal{P}| = 100$, $|\mathcal{P}_e| = 20$, $|\mathcal{P}_m| = 15$, $p_h = 0.7$ and 20 generations. The fitness of each individual was evaluated with $n_{it} = 200$, $n_s = 5$ and $|E_s| = 10$. Parameter t_{max} was set to 24 threads. The average time taken to tune each instance was about 960 minutes.

Most instances are tuned to either *fixed* or *random* construction. Only three instances are tuned to *reactive*. Almost half the instances are tuned to *variable* local search, and the rest are tuned to combinations of all types and options. Most are tuned to be either *back&forth* or *forward* path-relinking, and the majority are tuned to *greedy* and *complete* path-relinking. Evolutionary path-relinking is tuned to be used in all instances but it is mostly triggered every 100 iterations.

For the comparison experiments, a target cut weight was selected and a maximum runtime of 1000 seconds was set. Next, a total of 300 independent runs were made and the best solution value and time to target solution were saved. Table 2 shows the results. In all but four instances the average cut weight with *tuned* is greater or equal than that with *manual*. Note, however, that all four instances have an average cut weight found by *tuned* that is greater that the target and a difference of less than 1% from the average cut weight found by *manual*. Only two instances show no significant statistical difference when applying a Wilcoxon-Mann-Whitney text. In such two cases, either *tuned* or *manual* performed faster, hence no dominance is noticed. Figure 2 shows examples of empirical runtime distributions for two of the instances considered. Each distribution shows how tuned GRASP+evPR performs similarly to manual GRASP+evPR, but tends to find the target in shorter times.

Node Capacitated Graph Partitioning. Each NCGPP instance was automatically tuned using a BRKGA with $|\mathcal{P}| = 100$, $|\mathcal{P}_e| = 20$, $|\mathcal{P}_m| = 15$, $p_h = 0.7$ and 10 generations. The fitness of each individual was evaluated with $n_{it} = 200$, $n_s = 15$ and $|E_s| = 10$. Parameter t_{max} was set to 16 threads. The average tuning times for instances with 200 and 400 nodes were of 113 and 1162 minutes respectively.

The majority of instances are tuned to *fixed* construction, and only 5 to *random* construction. Most α values are tuned to less than 0.10, which correspond to less randomized constructions. Most instances were tuned to use *check-until* local search with a *move-max* option, except for two instances that were tuned

Table 2. Manual vs. tuned GRASP+evPR performance on max-cut instances. Maximum time and average time-to-target (TTT) values are in seconds.

Instance			GRASP+evPR (manual)		GRASP+evPR (tuned)	
Name	Target	Max Time	Avg. Weight	Avg. TTT	Avg. Weight	Avg. TTT
G1	11511	1000	11514.92	27.409	11535.12	36.629
G2	11498		11503.72	11.035	11507.74	7.724
G3	11515		11519.34	42.338	11529.15	22.635
G11	560	1000	561.56	16.539	560.86	14.343
G12	540		541.64	3.685	548.57	4.895
G13	566		566.41	3.558	567.98	4.063
G14	3030	1000	3030.98	43.272	3030.55	52.808
G15	2998		2999.59	6.628	3003.14	8.811
G16	3011		3012.3	16.155	3015.82	10.414
G22	13073	1000	13084.36	48.150	13092.49	84.789
G23	13124		13128.97	105.886	13152.98	101.529
G24	13138		13141.68	117.601	13147.13	124.904
G32	1368	1000	1371.10	45.979	1379.51	37.154
G33	1354		1358.24	131.353	1357.21	84.214
G34	1356		1360.61	123.018	1359.13	53.654
G35	7570	1000	7572.43	278.599	7575.37	212.086
G36	7564		**7566.49**	169.284	**7567.15**	137.809
G37	7549		7551.58	83.163	7558.36	99.105
G43	6568	1000	6571.14	26.686	6579.98	20.054
G44	6548		**6552.32**	12.877	**6552.69**	17.501
G45	6566		6568.71	29.672	6578.62	21.662

Boldface indicates there is no significant statistical difference when applying a Wilcoxon-Mann-Whitney test.

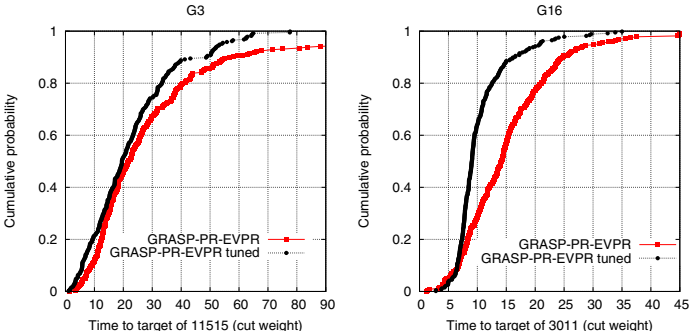


Fig. 2. Two examples of max-cut GRASP+evPR performance (tuned vs. manual). Times are in seconds.

to use *variable*, and three that were tuned to *move*-1. Most instances chose *mixed* and *forward* path-relinking. More than half of the instances are tuned to use *greedy randomized* path-relinking with a complete path. Almost half of the instances are tuned not to use evolutionary path-relinking.

For the comparison experiments, targets with gaps from 0.4% to 15.0% from the best known solution were selected and a maximum runtime was defined. Next, a total of 300 independent runs were made and the best solution edge weight and time to target solution were saved. Table 3 shows average results over the 300 runs. In all instances, the average solution edge weight is less with *tuned*. Moreover, we observe a speed up of up to a factor of 16 of the average time to target solution for *tuned* with respect to that of *manual*. Only one instance shows no significant statistical difference when applying a Wilcoxon-Mann-Whitney test, however for such instance, *tuned* proves to be much faster than *manual*. Figure 3 shows examples of empirical runtime distributions for

Table 3. Manual vs. tuned GRASP+evPR performance on node capacitated graph partitioning instances. Maximum time and average time-to-target (TTT) values are in seconds.

Instance			GRASP+evPR (manual)		GRASP+evPR (tuned)	
Name	Target	Max Time	Avg. Weight	Avg. TTT	Avg. Weight	Avg. TTT
200_15_1	86730	800	87554.05	274.329	85708.98	45.051
200_15_2	94972		97211.21	317.914	94425.63	41.467
200_15_3	79510		81850.21	346.083	79401.67	63.067
200_15_4	82560		84168.06	368.786	81895.81	24.324
200_15_5	104252		103575.66	215.103	103122.56	12.683
200_25_1	141726	800	142731.91	303.349	140939.68	21.826
200_25_2	144420		144539.23	275.136	143343.53	31.732
200_25_3	146894		146399.04	222.271	145855.27	24.237
200_25_4	138962		139291.66	268.845	138128.82	29.171
200_25_5	158726		160171.68	304.802	157995.25	34.455
400_15_1	426526	1200	422238.09	37.693	419604	13.403
400_15_2	394432		399262.75	488.385	391131.88	80.310
400_15_3	393648		391614.46	234.334	389258.68	16.603
400_15_4	369764		367769.97	373.395	365058.37	51.948
400_15_5	410252		406387.12	122.615	404703.72	18.911
400_25_1	594666	1200	**591782.73**	207.668	**591543.72**	14.993
400_25_2	596240		592543.43	49.910	588863.93	4.699
400_25_3	585676		582703.92	73.369	581049.76	8.255
400_25_4	531436		528504.74	114.473	527368.74	12.457
400_25_5	610986		608108.42	168.330	606347.47	24.644

Boldface indicates there is no significant statistical difference when applying a Wilcoxon-Mann-Whitney test.

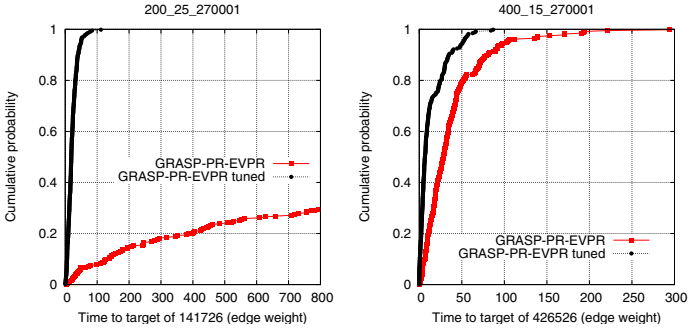


Fig. 3. Two examples of node capacitated graph partitioning GRASP+evPR performance (tuned vs. manual). Times are in seconds.

two of the instances considered. Each distribution shows how *tuned* outperforms *manual* with respect to both time and solution quality.

6 Concluding Remarks

Solving problems with heuristics involves the selection of parameters and configurations that alter the speed and solution quality of the algorithms used. Manual tuning can be tedious and time consuming without assuring that the tuned parameters can perform well on a different instance of the same problem.

This paper presents an automatic-tuning procedure of GRASP with evolutionary path-relinking heuristics by using a biased random-key genetic algorithm (BRKGA). The procedure evolves an initial pool of sets of parameters and configurations by making short runs and learning from the performance of each set in the pool.

The procedure is tested on benchmark instances of three optimization problems and results show that GRASP heuristics with automatically-tuned parameters tend to have better performance, both in terms of time to target solution and solution quality, than GRASP heuristics that use manually-tuned parameters.

Acknowledgment. This research was partially funded by Tecnológico de Monterrey Research Fund CAT128. It was done while the first author was a visiting scholar at AT&T Labs Research, in Florham Park, New Jersey.

References

1. Adenso-Diaz, B., Laguna, M.: Fine-tuning of algorithms using fractional experimental designs and local search. Operations Research 54, 99–114 (2006)
2. Aiex, R., Pardalos, P., Resende, M., Toraldo, G.: GRASP with path-relinking for three-index assignment. INFORMS J. on Computing 17, 224–247 (2005)
3. Bean, J.: Genetic algorithms and random keys for sequencing and optimization. ORSA Journal on Computing 2, 154–160 (1994)
4. Beasley, J.: An algorithm for set covering problem. European Journal of Operational Research 31, 85–93 (1987)
5. Deng, Y., Bard, J.: A reactive GRASP with path relinking for capacitated clustering. J. of Heuristics 17, 119–152 (2011)
6. Feo, T., Resende, M.: A probabilistic heuristic for a computationally difficult set covering problem. Operations Research Letters 8, 67–71 (1989)
7. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. J. of Global Optimization 6, 109–133 (1995)
8. Ferreira, C., Martin, A., de Souza, C., Weismantel, R., Wolsey, L.: The node capacitated graph partitioning problem: A computational study. Mathematical Programming 81, 229–256 (1998)
9. Festa, P., Gonçalves, J.F., Resende, M.G.C., Silva, R.M.A.: Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm. In: Festa, P. (ed.) SEA 2010. LNCS, vol. 6049, pp. 338–349. Springer, Heidelberg (2010)
10. Festa, P., Pardalos, P., Resende, M., Ribeiro, C.: Randomized heuristics for the MAX-CUT problem. Optimization Methods and Software 7, 1033–1058 (2002)
11. Glover, F.: Tabu search and adaptive memory programming – Advances, applications and challenges. In: Barr, R., Helgason, R., Kennington, J. (eds.) Interfaces in Computer Science and Operations Research, pp. 1–75. Kluwer Academic Publishers (1996)
12. Gonçalves, J., Resende, M.: Biased random-key genetic algorithms for combinatorial optimization. J. of Heuristics 17, 487–525 (2011)
13. Helmberg, C., Rendl, F.: A spectral bundle method for semidefinite programming. SIAM Journal on Optimization 10, 673–696 (1997)
14. Hutter, F., Hoos, H., Stützle, T.: Automatic algorithm configuration based on local search. In: Proceedings of the Twenty-second Conference on Artificial Intelligence (AAAI 2007), pp. 1152–1157 (2007)
15. Johnson, D.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9, 256–278 (1974)

16. Karp, R.: Reducibility among combinatorial problems. In: Miller, R., Thatcher, J. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, NY (1972)
17. Laguna, M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J. on Computing 11, 44–52 (1999)
18. Matsumoto, M., Nishimura, T.: Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Transactions on Modeling and Computer Simulation 8, 3–30 (1998)
19. Mehrotra, A., Trick, M.: Cliques and clustering: A combinatorial approach. Operations Research Letters 22, 1–12 (1997)
20. Morán-Mirabal, L., Gonzalez-Velarde, J., Resende, M.: Randomized heuristics for handover minimization in mobility networks. Technical report, AT&T Labs Research, Florham Park, New Jersey (August 2012)
21. Pedrola, O., Castro, A., Velasco, L., Ruiz, M., Fernández-Palacios, J.P., Careglio, D.: CAPEX study for a multilayer IP/MPLS-over-flexgrid optical network. J. of Optical Communications and Networking 4, 639–650 (2012)
22. Prais, M., Ribeiro, C.: Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. INFORMS J. on Computing 12, 164–176 (2000)
23. Resende, M., Werneck, R.: A hybrid heuristic for the p-median problem. J. of Heuristics 10, 59–88 (2004)
24. Spears, W., DeJong, K.: On the virtues of parameterized uniform crossover. In: Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 230–236 (1991)
25. Toso, R., Resende, M.: A C++ application programming interface for biased random-key genetic algorithms. Technical report, AT&T Labs Research, Florham Park, NJ (2012)
26. University of Wisconsin: Condor high throughput computing (2012), `http://research.cs.wisc.edu/condor` (last visited on June 25, 2012)

Combining Genetic Algorithm and Simulated Annealing Methods for Reconstructing HV-Convex Binary Matrices

Hadded Mohamed and Hasni Hamadi

National School of Computer Science, Manouba, Tunisia
CRISTAL Laboratory, RAMSIS pole, ENSI, Manouba, Tunisia
HaddedMohamed88@gmail.com, Hamadi.Hasni@ensi.rnu.tn

Abstract. In this paper, we consider the discret tomography problem (DTP), namely reconstruction convex binary matrices from their row and column sums respectively H and V, $RBM(H,V)$. This is reformulated as an integer programming problem. Since the problem is NP-complete, a new hybrid genetic algorithm with simulated annealing algorithm is proposed to find an approximate solution.

Keywords: Discret tomography, convex binary matrix, Integer programming, Genetic algorithm, simulated annealing, NP-complete.

1 Introduction

The goal of discrete tomography (DT) is to reconstruct discrete subset of $\mathbb{Z} * \mathbb{Z}$ or more generally of $\mathbb{Z}^n$ from a data set of $\mathbb{Z}$ or $\mathbb{Z}^{n-1}$, respectively. DT is applicable in many interesting sectors such as nondestructive testing, image processing, medical images, electron microscopy, data security, industrial tomography, material sciences, mathematical objects and schedules [3]. Among the problems of discrete tomography (DT) is the reconstruction of binary matrices from its horizontal and vertical projections which give respectively the number of ones in each row and in each column. This problem is polynomial and reconstruction algorithms in polynomial time exist[1]. However, possible reconstructions may exist[2]. To reduce the space of feasible solutions, geometric properties about the matrix to reconstruct should be considered such as convexity, periodicity and connectivity, etc. Here we will study the reconstruction of binary matrices problem under the constraints of horizontal projections H, vertical projections V and convexity. Since the problem is NP-complete [3], one way to solve it is to use approximation algorithms. Several solutions methods have been proposed to find an optimal or approximate solution for $RBM(H,V)$. Dahl and Flatberg [4] provide an approximate solution based on the Lagrangian decomposition. Jarray [5] provide an iterative approximation based on a longest path and a min-cost/max-flow model. Batenburg [6] has proposed a solution method based on the evolutionary algorithm and a local search method named HillClimbing to find hv-convex or nearly hv-convex matrices. Recently Jarray, Tlig [7] and

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 78–91, 2013.

Dakhli [8] have used simulated annealing and tabu search algorithm to reconstruct hv-convex matrices.

The remainder of this paper is organized as follows : In section 2, we introduce some definitions and notations. In section 3, we present the $RBM(H,V)$ problem and we reformulate it by using integer programming. In section 4 and 5, we present respectively simulated annealing and genetic algorithm approaches. In section 6, we present genetic algorithms based simulated annealing approach for reconstructing hv-convex binary matrices. In section 7, we present and discuss the numerical results. Conclusion and perspectives are reported in section 8.

2 Preliminaries

In this paper, we assume that all binary matrices studied are of size $m \times n$. We will study now the problem of reconstruction a binary matrix from its horizontal H and vertical V projections denoted by $RM(H,V)$.

Definition 2.1. Let $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$ be nonnegative integral vectors. We denote by $BM(H,V)$ the class of all binary matrices $A = [a_{i,j}]$ satisfying:

$$\sum_{j=1}^{n} a_{i,j} = h_i \quad ; \quad i = 1, \ldots, m$$
$$\sum_{i=1}^{m} a_{i,j} = v_j \quad ; \quad j = 1, \ldots, n$$

The vectors H and V are called the row and column projections of any matrix in $BM(H,V)$.

Problem 2.2 (decision prblem). Given $H = (h_1, \ldots, h_m)$ and $V = (v_1, \ldots, v_n)$ two nonnegative integer vectors.
Is there a $m \times n$ binary matrix respecting the horizontal projection H and the vertical projection V ?

Ryser [1] and Gale [9] derived necessary and sufficient conditions for the existence of a binary matrix from its orthogonal projections.

Definition 2.3. *A switching component* is a set of four cells (i,j), $(i,j+k)$, $(i+h,j)$ and $(i+h,j+k)$ such that the cells (i,j) and $(i+h,j+k)$ taking the value 1 and the cells $(i,j+k)$ and $(i+h,j)$ the value 0. The switching component is of the following form.

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} or \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The elementary switching operation is to interchange the 0's and 1's for these four cells. The orthogonal projections do not change after this operation.

Theorem 2.4. [1] Let A and B in $BM(H,V)$,$B \neq A$, Then A is transformable into B (or vice versa) by finite sequences of switching operations.

we will study now the problem of reconstruction a binary matrix from its orthogonal projections (H,V) and more similar to a model matrix M denoted by $RM(M,H,V)$ problem.

Problem 2.5 ($RM(M,H,V)$). Let M be a given binary matrix and H, V be given nonnegative integer vectors. $RM(M,H,V)$ is to find a binary matrix $A = [a_{i,j}]$ in $BM(H,V)$, such that the difference between A and M is minimal.

This problem consist to maximize the number of common 1's between A and M which is equivalent to solve the maximization problem $(max \sum_{i,j} a_{i,j} m_{i,j})$ under the orthogonal projections constraints: $(1) \sum_{j=1}^{n} a_{i,j} = h_i$, $1 \leq i \leq m$ and $(2) \sum_{i=1}^{m} a_{i,j} = v_j$, $1 \leq j \leq n$. Then $RM(M,H,V)$ may be reformulated as an integer linear program below.

$$(P) \begin{cases} max \sum_{i,j} a_{i,j} m_{i,j} \\ s.t \\ \sum_{j=1}^{n} a_{i,j} = h_i \quad \forall i \in \{1,m\} \\ \sum_{i=1}^{m} a_{i,j} = v_j \quad \forall j \in \{1,n\} \\ A \in \{0,1\}^{m \times n} \end{cases}$$

We use a max-flow/min-cost model in a complete bipartite graph to solve this problem, which was introduced by Batunburg [6]. Many standard algorithms are available for solving the max flow min cost problem in polynomial time as Busacker and Gowen.

3 HV-Convex Binary Matrix

3.1 Definitions

As the number of solutions in $RBM(H,V)$ may be high, it is important to study the reconstruction problem where we impose geometric constraints on binary matrices. A binary matrix is h-convex if the ones in each row are adjacent. Similarly a binary matrix is v-convex if the ones are adjacent in each column. A binary matrix is hv-convex if it is h-convex and v-convex (see Figure 1). Barcucci et al. [11] prove that the consistency problem for h-convex or v-convex is NP-complete. Woeginger [10] prove that the consistency problem is also NP-complete for hv-convex binary matrices. The reconstruction of hv-convex binary matrix problem is denoted by $RCBM(H,V)$.

3.2 Integer Programming Formulation

In this section, we will define the integer program for solving $RBM(H,V)$.

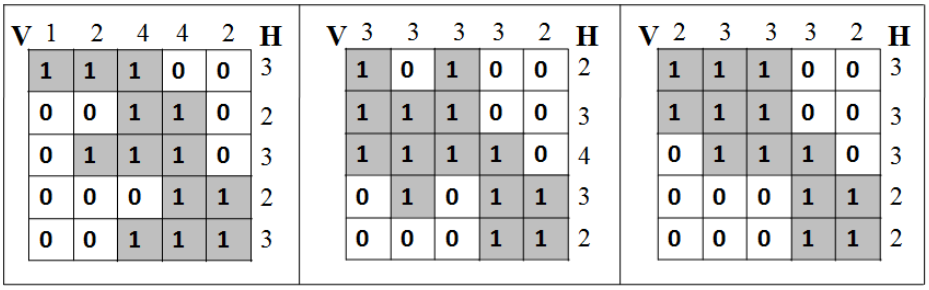


Fig. 1. {h, v and hv}-convex matrix

$RBM(H,V)$ is to find a binary matrix that respects the orthogonal projections H and V which maximizes the number of adjacent ones. Let X be a binary matrix and $x_{i,j}$ denote the entry of the matrix in position (i,j). The following function f counts the number of adjacent ones:

$$f(X) = \sum_{i=1}^{m} \sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} + \sum_{j=1}^{n} \sum_{i=1}^{m-1} x_{i,j}x_{i+1,j}$$

Thus $RBM(H,V)$ is equivalent to the integer program (P), which consist to maximize the following function f under constraints of orthogonal projections H and V.

$$(P)\begin{cases} max\ f(X) \\ s.t \\ \sum_{j=1}^{n} x_{i,j} = h_i \quad \forall i \in \{1,m\} \qquad (1) \\ \sum_{i=1}^{m} x_{i,j} = v_j \quad \forall j \in \{1,n\} \qquad (2) \\ a_{i,j} \in \{0,1\}^{m \times n} \end{cases}$$

Constraint (1) ensures that the sum of ones in row i is equal to the prescribed horizontal projection h_i. Constraint (2) ensures that the sum of ones in column j is equal to the prescribed vertical projection v_j. (1) and (2) ensure that $X \in BM(H,V)$.

3.3 Bounds

Let X be a binary matrix with orthogonal projections H and V.

I) $f(X) \leq 2\sum_{j=1}^{n} v_j - m - n$.
II) X is hv-convex if and only if $\alpha = f(X) = 2\sum_{j=1}^{n} v_j - m - n$.

Clearly, α is an upper bound for $f(X)$.

Proof. Let X be a binary matrix respecting the projections H and V.
I)

- The number of adjacent ones in the row i satisfies the following condition:

$\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} \leq \sum_{j=1}^{n} x_{i,j} - 1$

$\Leftrightarrow \quad \sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} \leq h_i - 1 \qquad (h_i = \sum_{j=1}^{n} x_{i,j})$

$\Leftrightarrow \quad \sum_{i=1}^{m}\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} \leq \sum_{i=1}^{m} (h_i - 1)$

$\Leftrightarrow \quad \sum_{i=1}^{m}\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} \leq \sum_{i=1}^{m} h_i - m \qquad \mathbf{(1)}$

- The number of adjacent ones in the column j satisfies the following condition:

$\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} \leq \sum_{i=1}^{m} x_{i,j} - 1$

$\Leftrightarrow \quad \sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} \leq v_j - 1 \qquad (v_j = \sum_{i=1}^{m} x_{i,j})$

$\Leftrightarrow \quad \sum_{j=1}^{n}\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} \leq \sum_{j=1}^{n} (v_j - 1)$

$\Leftrightarrow \quad \sum_{j=1}^{n}\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} \leq \sum_{j=1}^{n} v_j - n \qquad \mathbf{(2)}$

(1)+(2)

$\sum_{i=1}^{m}\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} + \sum_{j=1}^{n}\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} \leq \sum_{i=1}^{m} h_i + \sum_{j=1}^{n} v_j - m - n$

Or $X \in MB(H,V) \quad \Leftrightarrow \quad \sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j$ Thus

$n_x \leq 2\sum_{j=1}^{n} v_j - m - n$

II)

- The row i is h-convex if and only if the number of adjacent ones in row i is equal to $h_i - 1$:

$\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} = h_i - 1$

$\Leftrightarrow \quad \sum_{i=1}^{m}\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} = \sum_{i=1}^{m} h_i - m \qquad \mathbf{(1)}$

- The column j is v-convex if and only if the number of adjacent ones in column j is equal to $v_j - 1$:

$\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} = v_j - 1$

$\Leftrightarrow \quad \sum_{j=1}^{n}\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} = \sum_{j=1}^{n} v_j - n \qquad \mathbf{(2)}$

(1)+(2)

$\sum_{i=1}^{m}\sum_{j=1}^{n-1} x_{i,j}x_{i,j+1} + \sum_{j=1}^{n}\sum_{i=1}^{m-1} x_{i,j}x_{i+1,j} = \sum_{i=1}^{m} h_i + \sum_{j=1}^{n} v_j - m - n$

Or $X \in MB(H,V)$ Thus $\sum_{i=1}^{m} h_i = \sum_{j=1}^{n} v_j$ we deduce That

$f_1(X) = 2\sum_{j=1}^{n} v_j - m - n$

4 Simulated Annealing

4.1 Overview

Simulated annealing SA is an optimization technique based on the physical process of annealing. This method was proposed in 1982 by S. Kirkpatrick et al. from the Metropolis method (1953) that used to model the physical process of heating a material and then slowly lowering the temperature to decrease defects,

thus minimizing the system energy [13]. SA starts from any solution and a temperature T. In each step, SA generates a random neighboring solution to the current solution. If the neighboring solution is better, it becomes the current solution, otherwise, we calculate the increased cost and we accept it with a certain probability according to this increase.

4.2 Simulated Annealing for RBM(H,V)

Jarray and Tlig [7] have adopted SA algorithm for approximating hv-convex matrices. The neighborhood of a solution is defined as the matrix obtained by a single switching. The objective function is the functions $f(X)$ defined in section 3.2. The initial solution is generated by using the Rysers classical algorithm. SA algorithm for $RBM(H,V)$ can be described as in algorithm 1:

- $Temp$: Temperature.
- Tm : Minimum temperature.
- $Nmax$: Maximum number of iterations.
- β : Heat transfer coefficient.
- p : Random variable. $(0 < p < 1)$.

Algorithm 1. Simulated Annealing Algorithm

Input:
Matrix X_0, β, $Temp$, Tm and $Nmax$.
Ouput:
Matrix X that maximizes the objective function $f(X)$.
1. While $((Temp \geq Tm))$ **do**
1.1. $NbrIteration = 0$; ;
1.2. while $(NbrIteration < Nmax)$
1.2.1. Choose a switching component ;
1.2.2. Calculate the evaluation function F' of the neighboring solution ;
1.2.3. $\Delta F = F' - F$;
1.2.4. if $((\Delta F > 0)$ **or** $(p < \exp^{-(\Delta F/Temp)}))$ **then**
1.2.4.1. Accept this switching component ;
1.2.4.2. $F = F'$;
end if ;
1.2.5. $NbrIteration = NbrIteration + 1$;
end while ;
1.2. $Temp = \beta \times Temp$;
end while ;

Although SA is very simple in implementation, Batenburg found that simulated annealing, which only use local search operators, are not well suited for this task. The main reason for this is that the $RBM(H,V)$ problem usually has a great number of local optima and moving between different optima may require a large number of "uphill" steps [6].

5 Genetic Algorithm

5.1 Overview

Genetic algorithms GA or (Evolutionary Algorithm EA) are heuristics derived from biological evolution. This algorithm starts from a population of potential solutions randomly generated called individuals population. It uses an adaptive function to evaluate the quality of each individual, optimization operators "crossover and mutation" to explore the space of feasible solutions and selection process to direct the population to an optimal solution.

5.2 Genetic Algorithm for RBM(H,V)

In this section we describe the main steps of the genetic algorithm adapted for the reconstructing of hv-convex matrices problem.

- *Population elements coding*

Each individual is encoded as a binary matrix (0 and 1).

- *Generation of initial population*

The first step of GA is the genesis of the initial population formed by a set of binary matrices in $BM(H,V)$. The initial population generation procedure is to randomly read a binary matrix M and then apply the max flow min cost algorithm to obtain another matrix in $BM(H,V)$ and more similar to its original matrix M ($BM(M,H,V)$). These two steps are repeated T times where T is the size of the initial population.

- *Crossover operator*

Once the initial population is obtained, a crossover operator is applied to diversify the population in each generation and explore the search space. This operator randomly selected two parent individuals and generates two children by combining the genes of these parents. Among the most used types of crossover operator, uniform crossover, two point crossover. By several tests, we found that the two point crossover is very effective in terms of the obtained children quality and in terms of running time.
This operator we implemented may be explicited as follows. Let X and Y be two binary matrices parents of sizes $m \times n$ and a, b two randomly chosen integers in $[1, n]$ such that $a < b$. therefor, the two point crossover operator is to swap genes between a and b for both parents, we obtain two children matrices Z and T defined as follows.

$$Z = (z_{ij})_{1 \leq i \leq m} = \begin{cases} x_{i,j} \ , \ 1 \leq j \leq a \\ y_{i,j} \ , \ a < j \leq b \\ x_{i,j} \ , \ b < j \leq n \end{cases} \qquad T = (t_{ij})_{1 \leq i \leq m} = \begin{cases} y_{i,j} \ , \ 1 \leq j \leq a \\ x_{i,j} \ , \ a < j \leq b \\ y_{i,j} \ , \ b < j \leq n \end{cases}$$

Although the crossover operator generates two children matrices similar to their parents, these two children aren't in $BM(H,V)$ then it is necessary to use the formulation of a max flow/min cost model to correct the orthogonal projections

(see figure 3). This operator is applied with a probability indicated by P_c. Despite the fact that the combined action of crossover operator allows to diversify the population, it is possible that this action might remove the genes of an individual. For remedy for this problem, a mutation operator is applied because this operator is able to reintroduce these genes in the individual.

- *Mutation operator*

This operator can be viewed as a learning phase of the individual to its environment, The children are significantly improved before constitute the next generation. This operator exploits the adjaceny contraint of ones in each row and in each column, it reverses all the cells of value 1 which do not have any neighbor of value 1 and reverse all the cells of value 0 if the number of their neighbouring cells of value 1 higher to three. Note that the neighboring cells of cell (i, j) are the cells $(i, j-1)$, $(i, j+1)$, $(i-1, j)$ and $(i+1, j)$. This operator is applied with a probability indicated by P_m.

Algorithm 2. Mutation operator

Input:
Matrix: M.
Output:
Matrix: $M^{'}$.
1. for each cell m_{ij} **of** M **do**
1.1. if the sum of neighbouring ones to cell m_{ij} greater than 3 **then**
$m^{'}_{ij} = 1$;
else
if the sum of neighbouring ones to cell m_{ij} equal to 0 **then**
$m^{'}_{ij} = 0$;
else
$m^{'}_{ij} = m_{ij}$;
end if ;
end if ;
end for ;

- *Evaluation*

The evaluation of each individual is realized according to the objective function defined in Section 3.2 ($f(x)$). This step is performed at each iteration and according to this evaluation, individuals will be selected.

- *Selection*

The selection operator chooses a few individuals from both the old population and the new group of individuals according to their evaluations. Only individuals passing the selection test can access to the next generation and will participate in the production step while the others individuals are deleted.

There are several methods of selection.

- Roulette Wheel selection:
- Ranking selection:
- Selection tournament:
- Elitist selection:

The first three methods use random selection characters does not always guarantee that the best solution is retained, however the last method allows to select only the best individuals, then the new population is not diverse and we would arrive at a stagnation of the evolution.

To avoid these problems we have proposed another method which is to copy the 50% of the best individuals to ensure always that the best individuals are preserved and randomly copy the 50% other individuals to diversify the new population. Each individual is selected only once.

The GA algorithm for $RBM(H, V)$ is described in algorithm 3. By several experiments, We found this approach to be inadequate for $RBM(H, V)$. Indeed, the two crossover and mutation operators generally gives children of lower qualities relative to their parents(see figure 3). A solution To ensure that the child matrix has sufficient quality, we apply a local search operator called simulated annealing SA after the crossover or mutation operation.

6 Combining GA and SA Algorithm (GASA) for *RBM(H,V)*

The genetic algorithm GA and simulated annealing SA have some disadvantages that can be mitigated by hybridizing in a single architecture. GA is a global search algorithm based on the principle of diversification(research is remote from the neighbourhood). However, SA is a local search algorithm based on the principle of intensification(research is conducted in a neighborhood). Then the two methods are complementary because one allows to move quickly in the search space and thus diversify the research while the other explores intensively these areas of the search space. Their combination $GASA$ allows to quickly browse the interesting areas of the search space for explore it in detail. In each generation, two operators of crossover and mutation are applied with a probability, for each new configuration (s) thus generated, the simulated annealing $SA(s)$ is applied to improve the quality of s. Finally, a selection operator decides if the new individual should be introduced in the new population.

6.1 GASA Algorithm

The hybrid algorithm is proposed to find a binary matrix nearly hv-convex, this is equivalent to finding a binary matrix in $BM(H, V)$ that minimizes the objective function defined in section 3.2. This algorithm begins from an initial population randomly generated then each individual is improved by the simulated annealing. For each generation, new individuals (children) are produced by crossovor or by mutation but the latter are not in $BM(H, V)$ then it is necessary to correct their orthogonal projections by resolution of the $RBM(I, H, V)$ for each child

Algorithm 3. Genetic Algorithm GA

Inputs:
Size of problem: m, n.
Orthogonal projections: $H = (h_1, h_2, \ldots, h_m)$ and $V = (v_1, v_2, \ldots, v_n)$.
Size of the initial and new population: T, NT.
Maximum number of generations : $NbrGA$.
Number of new children per generation: γ.
Probability of crossover and mutation : P_c and P_m .

Output:
Binary matrix of size $m \times n$: M.

1. $Itr = 0$;
2. $P_I = \{\emptyset\}$;
3. Initialization the initial population P_{Itr} of size T consists of matrices in $BM(H, V)$;
4.while $(Itr < NbrGA)$ **do**
4.1.while $(t \leq \gamma)$ **do**
4.1.1. Produce a child matrix E by crossover operator according to the crossover probability;
4.1.2. Apply the mutation operator according to the mutation probability ;
4.1.3. Correction of orthogonal projections of the matrix E by the resolution of $RBM(E, H, V)$;
4.1.4. $P_{Itr} = P_{Itr} \cup \{E\}$;
end while ;
4.2. Calculate the function evalution for each individual in P_{Itr};
4.3. Evaluation of individuals;
4.4. Creation of the new population P_{Itr+1} containing the selected individuals from the population P_{Itr} ;
4.5. $Itr = Itr + 1$;
end while ;

I thus obtained. Then, the quality of each child is increased by using simulated operator. Finally, new population is evaluated according to the objective function and a selection process is applied to direct the entire population to an optimal solution. This principle is repeated until a solution for this problem is found or the number of iterations reaches the maximum number allowed. The complete algorithm of GASA is shown as flowchart in Fig. 2.

7 Computational Results

7.1 Choice of the Parameters

The choice of parameters helps to reduce the run time and to improve the quality of result. We did several experiments to find the best parameters for $GASA$. We found that our approach performs well when :

$$T = 40,\ NT = T,\ NbrGA = 100,\ P_c = 0,8,\ P_m = 0,05\ ,\gamma = 4,$$

$$Nmax = 100,\ Temp = m + n,\ Tm = 5 \text{ and } \beta = 0.9.$$

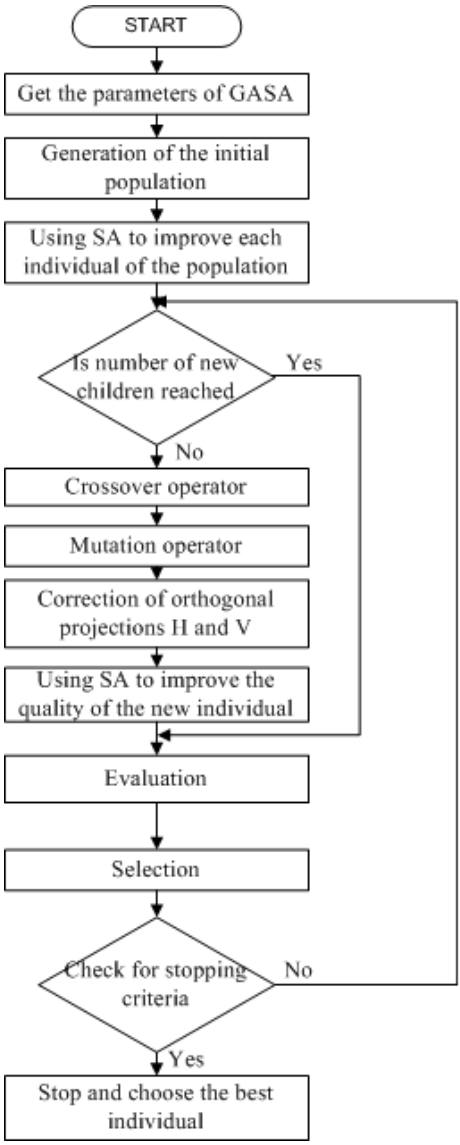


Fig. 2. Flowchart for GASA algorithm

7.2 Results

We have implemented our algorithms in Java language, using the jdk compiler. All our experiments were run on a Intel Centrino Duo 3.2GHz PC with 1Gb of memory.

We have tested $GASA$ approach by using a set of matrices of different sizes with a fixed number of hv-convex components described and generated in [12], [15] and [16]. A component is a maximal hv-convex connected set. The mim-cost max-flow models used by the algorithm are solved by the CS2 network flow library [14]. The table 1 shows the results with their run time. The first column gives the size of the problem and the number of hv-convex components. The following column α is the upper bound of the number of adjacent ones given in section 3.2 ($\alpha = 2\sum_{j=1}^{n} v_j - m - n$). The column Sol gives the objective function value for the perfect solution provided By GA, SA and $GASA$. Gap is the gap in(%) between the original matrix and the approximate matrix. The last column $T(s)$ indicates the running time.

For each problem size, $GASA$ algorithm converges to an hv-convex matrices ($Sol = 0$) or gives an approximate hv-convex matrix (Gap close to 0), we say that this approach is efficient. On the other hand, we observe that simulated annealing gives the best results in terms of quality and running time than $GASA$ and GA for matrices with a single hv-convex component for example (20×20, 1), but if

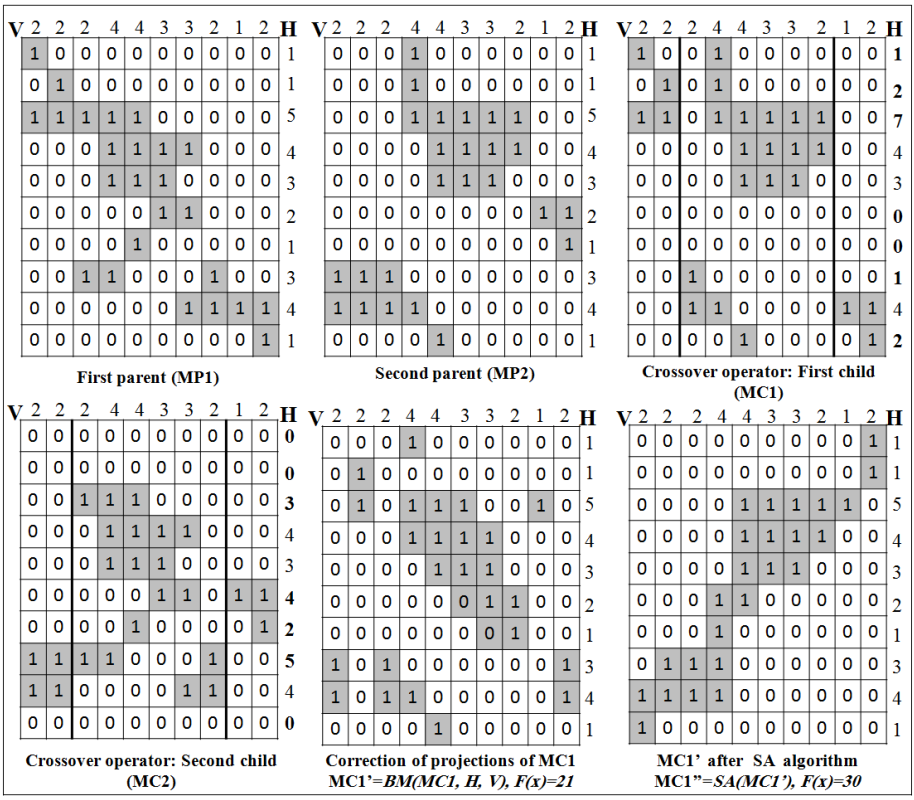


Fig. 3. An example of GASA approach

Table 1. *CPU running time and resultats in seconds for GA, SA and GASA*

Matrix	α	Genetic Algorithm			Simulated Annealing			GASA		
		Sol	GAP(%)	Time	Sol	GAP(%)	Time	Sol	GAP(%)	Time
10×10, 1	80	79	0.12	**6.79**	80	0	**0.34**	80	0	**0.43**
10×10, 2	36	34	5.55	**3.94**	36	0	**1**	36	0	**0.24**
10×10, 3	18	13	27.77	**2.67**	14	22.22	**3.24**	18	0	**0.23**
10×10, 4	66	66	0	**5.47**	45	31.88	**5**	66	0	**0.42**
20×20, 1	390	388	3.07	**186.4**	390	0	**8**	390	0	**10.2**
20×20, 2	156	132	15.38	**84.6**	150*	3.48	**10**	156	0	**55.4**
20×20, 3	90	72	20	**56.3**	60	33.33	**70**	90	0	**41.5**
20×20, 4	46	32	30.43	**37.8**	31	32.60	**25**	46	0	**24.2**
30×30, 1	954	937	1.77	**891.6**	954	0	**89**	954	0	**265.3**
30×30, 2	310	255	10.64	**352.1**	271	12.58	**101.2**	307	0.96	**140.1**
30×30, 3	256	216	15.62	**552.2**	217*	14.06	**93.99**	251	1.95	**120.7**
30×30, 4	164	141	14.02	**363.1**	120*	26.82	**115.1**	164	0	**111.2**
40×40, 1	408	341	16.42	**520.5**	408	0	**160.5**	408	0	**482.2**
40×40, 2	622	527	15.27	**606**	543*	12.7	**220.2**	609	2.09	**605.7**
40×40, 3	304	265	12.82	**670.2**	241*	20.72	**168.1**	290	4.61	**437**
40×40, 4	268	268	0	**254**	231*	9.05	**76**	268	0	**134**
50×50, 1	2578	2571	0.27	**840.5**	2578	0	**240**	2578	0	**840.1**
50×50, 2	480	410	14.58	**730.3**	421*	12.29	**156.1**	445	7.29	**730.6**
50×50, 3	720	641	11	**401**	633*	12.08	**202**	706	1.94	**400.4**
50×50, 4	580	502	13.44	**540.1**	489*	15.68	**322.3**	564	2.75	**522.2**

* indicates that the algorithm remains in a stagnation state.

the number of components increase the performance of SA decrease and $GASA$ becomes better than SA.

8 Conclusion

In this paper we have studied a variant of the NP-hard problem of reconstructing hv-convex binary matrices from horizontal and vertical projections. This problem is formulated as an integer problem. To solve it, we proposed a genetic algorithms based simulated annealing approach. For evaluation, We tested this approach on several matrices with different sizes and number of hv-convex components then we compared it with other existing methods GA and SA. Although we have used a few number of test matrices, the results seem to indicate that $GASA$ is more successful.

$GASA$ can be generalized to solve problems with additional constraints as periodicity and more than two directions of projections.

References

1. Ryser, H.J.: Combinatorial properties of matrices of zeros and ones. Canad. J. Math. 9, 371–377 (1957)
2. Wang, B., Zhang, F.: On the precise number of (0, 1)-matrices in a(r, s). Discrete Math. 187, 211–220 (1998)
3. Del Lungo, A., Nivat, M.: Reconstruction of connected sets from two projections. In: Herman, G.T., Kuba, A. (eds.) Discrete Tomography: Foundations, Algorithms and Applications, ch. 7, pp. 163–188. Birkhauser, Boston (1999)
4. Dahl, G., Fatberg, T.: Optimization and reconstruction of hv-convex (0,1)-matrices. Discrete Applied Mathematics 151, 93–105 (2005)
5. Jarray, F., Costa, M.-C., Picouleau, C.: Approximating hv-convex binary matrices and images from discrete projections. In: Coeurjolly, D., Sivignon, I., Tougne, L., Dupont, F. (eds.) DGCI 2008. LNCS, vol. 4992, pp. 413–422. Springer, Heidelberg (2008)
6. Batenburg, K.J.: An evolutionary algorithm for discrete tomography. Discrete Applied Mathematics 151, 36–54 (2005)
7. Jarray, F., Tlig, G.: A simulated annealing for reconstructing hv-convex binary matrices. Electronic Notes in Discrete Mathematics 36, 447–454 (2010)
8. Jarray, F., Tlig, G., Dakhli, A.: Reconstructing hv-convex images by tabu research approach. In: International Conference on Metaheuristics and Nature Inspired Computing, 3 p. (2010)
9. Gale, D.: A theorem on flows in networks. Discrete Mathematics 187, 1073–1082 (1957)
10. Woeginger, G.J.: The reconstruction of polyominoes from their orthogonal projections. Information Processing Letters 77, 225–229 (2001)
11. Barcucci, E., Del Lungo, A., Nivat, M., Pinzani, R.: The reconstruction of polyominoes from their orthogonal projections. Theoretical Computer Science 155, 321–347 (1996)
12. Hochstattler, W., Loebl, M., Moll, C.: Generating convex polyominoes at random. Discrete Mathematics 153, 165–176 (1996)
13. Aria, M.: An Integration of simulated annealing and genetic algorithm for travelling salesman problem. Journal Majalah Ilmiah Unikom 8 (May 2011)
14. Goldberg, A.V.: An efficient implementation of a scaling minimum-cost flow algorithm. Journal of Algorithms 22, 1–29 (1997)
15. Balázs, P.: A benchmark set for the reconstructin of hv-convex discrete sets. Discrete Appl. 157, 3447–3456 (2009)
16. Balázs, P.: Reconstruction of canonical *hv*-convex discrete sets from horizontal and vertical projections. In: Wiederhold, P., Barneva, R.P. (eds.) IWCIA 2009. LNCS, vol. 5852, pp. 280–288. Springer, Heidelberg (2009)

Experimental Analysis of Pheromone-Based Heuristic Column Generation Using irace

Florence Massen[1], Manuel López-Ibáñez[2], Thomas Stützle[2], and Yves Deville[1]

[1] ICTEAM, Université catholique de Louvain, Belgium
{Florence.Massen,Yves.Deville}@uclouvain.be
[2] IRIDIA, Université libre de Bruxelles, Belgium
{manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. Pheromone-based heuristic column generation (ACO-HCG) is a hybrid algorithm that combines ant colony optimization and a MIP solver to tackle vehicle routing problems (VRP) with black-box feasibility. Traditionally, the experimental analysis of such a complex algorithm has been carried out manually by trial and error. Moreover, a full-factorial statistical analysis is infeasible due to the large number of parameters and the time required for each algorithm run. In this paper, we first automatically configure the algorithm parameters by using an automatic algorithm configuration tool. Then, we perform a basic sensitivity analysis of the tuned configuration in order to understand the significance of each parameter setting. In this way, we avoid wasting effort analyzing parameter settings that do not lead to a high-performing algorithm. Finally, we show that the tuned parameter settings improve the performance of ACO-HCG on the multi-pile VRP and the three-dimensional loading capacitated VRP.

1 Introduction

An advantage of metaheuristics is that they can be adapted to different problem variants by adjusting their algorithmic components and parameter settings. The parameter configuration often has a crucial impact on the performance of an algorithm on a particular problem. However, complex algorithms typically have many parameters, resulting in a large number of possible configurations of the algorithm. Testing all possible configurations is typically intractable, particularly on problems where one run of the algorithm may take several hours. Due to this intractability, only a very limited subset of parameter configurations are tested when designing an algorithm, and such a subset is chosen by the designer based on her intuitions. This manual approach may easily miss the best-performing configurations for a metaheuristic.

Automatic algorithm configuration methods aim to identify high-performing configurations of an algorithm for a given problem [2,8]. Typically, the method is given a description of the parametric space of the algorithm (types and domains of the parameters), a set of training instances representative of the problem, and a computational budget (e.g., a maximum number of algorithm runs). The goal of the method is to find a high-performing parameter configuration for unseen instances of the same problem. When the goal is not simply to obtain the best performance, but also to understand

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 92–106, 2013.

the effect of parameters, the number of possible configurations that need to be evaluated and analyzed may be extremely large. This is the case even if most configurations are of little interest, since they produce poor results. In this paper, we propose to use automatic configuration as a first step before analyzing the effect of parameters.

We use this approach to improve the performance and analyze the parameters of pheromone-based heuristic column generation (ACO-HCG) for vehicle routing problems with black box feasibility (VRPBB) [11]. VRPBBs are an abstraction of rich vehicle routing problems. In the VRPBB, the problem structure remains unchanged with respect to a basic VRP, but a combinatorial side-problem needs to be solved to verify the feasibility of each route. Examples of such problems combine routing and loading [7] or routing and scheduling [12]. In the VRPBB, the combinatorial side-problem to be solved for every route is considered to be unknown, but a black-box function allowing to test the feasibility of a route is provided. An optimization procedure for the VRPBB should be independent of the particular side-problem. An example of such an optimization procedure is ACO-HCG, which combines ant colony optimization with an exact solver. At each iteration, ants generate a set of feasible routes probabilistically according to pheromone values. A solution is obtained by solving a relaxed set partitioning problem (RMP_{relax}) over the set of feasible routes. This solution is used to update the pheromone information, biasing the construction of new feasible routes by the ants in subsequent iterations. In addition, feasible routes are further improved using local search. This procedure continues until a time limit is reached, and the final solution is obtained by solving the integer set partitioning problem over the set of all feasible routes ever generated.

In this paper, we extend ACO-HCG with new parameters formalizing algorithmic design choices. Then, we use irace [10] to automatically find a high-performing configuration that improves the results of the original ACO-HCG. In particular, we improve previous results of ACO-HCG in two VRPBB problems, the multi-pile VRP (MPVRP) and the three-dimensional loading capacitated VRP (3L-CVRP). The new ACO-HCG also improves the best-known results from the literature on the MPVRP and the 3L-CVRP on several instances. Finally, we perform a basic sensitivity analysis of the parameters of ACO-HCG with the improved configuration generated by irace. This analysis shows that the new components play a significant role in the effectiveness of the algorithm.

2 Vehicle Routing Problems with Black Box Feasibility

In this section, we present the capacitated vehicle routing problem (CVRP), extend it to VRPs with black-box feasibility and present two example applications of the resulting VRPBB.

2.1 The Capacitated Vehicle Routing Problem and Black-Box Feasibility

The CVRP [17] is defined on a complete undirected and weighted graph $G = (V, E)$. In the set of vertexes $V = \{0, 1, \ldots, n\}$, 0 is the depot vertex and vertexes $1, \ldots, n$

correspond to the n customers that must be visited. The set of edges connecting every pair of vertexes in V is given in E. With each edge $(i, j) \in E$ s.t. $i, j \in V, i \neq j$ is associated a non-negative weight c_{ij} corresponding to the cost of traveling through edge (i, j), i.e., from vertex i to j. A homogeneous fleet of K vehicles is given for visiting the customers. All vehicles have a limited capacity Q. Each customer i $(i = 1, \ldots, n)$ has a given demand d_i. A solution to the CVRP corresponds to a set of routes. Each route is a sequence of vertexes, where the first and last vertex always correspond to the depot, while the remaining vertexes correspond to customers. No customer vertex may appear more than once in a route. Finally, the goal is to find a set of routes Sol such that (i) the number of routes in Sol does not exceed K; (ii) each customer $i \in V \setminus \{0\}$ is visited exactly once; (iii) the sum of the demands of the customers visited on a route does not exceed the vehicle capacity Q; (iv) the total traveling cost obtained by summing the weights of the used edges in minimized.

The VRPBB is based on the CVRP and a feasible solution to the VRPBB must also be feasible for the CVRP. However, in addition to the CVRP constraints, routes in the VRPBB must satisfy a fixed set of unknown constraints F. A route r is feasible with respect to F if and only if it satisfies all the constraints in F. We suppose that a deterministic black-box function is given to verify the feasibility of a route r w.r.t. F. The function is considered to be computationally expensive in comparison to common VRP-feasibility functions. In the following, we call a route black-box (BB)-feasible if it is feasible w.r.t. F, VRP-feasible if it respects the CVRP constraints, and feasible if it is VRP- and BB-feasible.

2.2 Applications of the VRPBB

In this paper, we consider two applications of the VRPBB, the MPVRP and the 3L-CVRP, which combine a basic VRP with two different types of loading constraints. We refer to [9] for an overview of such problems.

The Multi-Pile VRP. The MPVRP [4] is based on the CVRP, but there are no restrictions on the capacity nor on the number of vehicles ($Q = \infty$, $K = n$). However the loading space of the vehicles has a limited length L, width W and height H. Furthermore, the loading space is partitioned into p piles. Each customer i demands a set of m_i items I_i. Each item I_{ik} $(i = 1, \ldots, n;\ k = 1, \ldots, m_i)$ has a fixed width W, can take two possible lengths $l_{ik} \in \{L/p, L\}$ and has a height h_{ik}. A solution to the MPVRP must respect the CVRP constraints and a feasible loading for the items of the visited customers must exist. That is, no items must overlap (*non-overlapping*) and the items must fit in the loading space (*containment*). When a customer is visited, all its items must be directly accessible, i.e., its items are on top of the piles in the back of the truck (*sequential loading*). Existing works on the MPVRP use metaheuristic and exact approaches. Doerner et al. [4] propose heuristic approaches based on the fact that the loading problem to be solved is a generalization of the $P||C_{max}$ scheduling problem. Tricoire et al. [18] propose an exact approach to solve the loading problem, and a variable neighborhood search as well as a branch-and-cut method for the routing problem.

The Three-Dimensional Loading VRP. In the 3L-CVRP [7], vehicles in a homogeneous fleet have a limited capacity Q. Additionally, the loading space of the vehicles has a limited length L, a limited width W and a limited height H. With each customer i is associated a demand d_i and a set I_i of m_i items. An item I_{ik} $(i = 1, \ldots, n;\ k = 1, \ldots, m_i)$ corresponds to a three-dimensional box that is either fragile or non-fragile and has width w_{ik}, height h_{ik} and length l_{ik}. A solution to the 3L-CVRP is feasible if it respects the CVRP constraints and a feasible loading for the items of customers on a route exists. In order to be feasible, no items in a loading may overlap (*non-overlapping*) and all items need to fit into the volume of the vehicle (*containment*). Also, each box must be supported by a surface corresponding to at least 75% of the box's bottom surface (*support*). Moreover, only fragile items may be placed on top of fragile items (*fragility*). Finally, when visiting a customer, the unloading of the items of this customer must not be hindered by items belonging to customers yet to be visited (*LIFO policy*). Most of the approaches proposed for the 3L-CVRP are metaheuristics [7,16,6,3,22,15], except for an exact approach proposed recently [14]. In these approaches, the loading problem is typically solved using various packing heuristics.

3 Pheromone-Based Heuristic Column Generation for the VRPBB

The pheromone-based heuristic column generation algorithm (ACO-HCG) analyzed in this paper was originally proposed by Massen et al. [11]. It is based on the idea of reformulating the VRPBB as a set partitioning problem (SPP). In this section, we briefly explain this reformulation, summarize the ACO-HCG algorithm, and propose several extensions.

3.1 Reformulation as Set Partitioning Problem

The VRPBB may be reformulated as a SPP, where the goal is to choose from the set of all feasible routes ($\mathcal{R}$) a subset of at most K routes such that each customer appears exactly once and such that the total traveling cost is minimized. However, generating the set of all feasible routes $\mathcal{R}$ is, in general, intractable due to the exponential number of possible routes and the complexity of the black-box feasibility check. Therefore, a restricted version of the problem (*RMP*) is solved using a restricted set of feasible routes $\mathcal{R}^*$. This problem can be further relaxed, resulting in two variants: the linear relaxation of the *RMP* (called RMP_{relax}) or the integer *RMP* (called RMP_{int}). In both, customers may be visited more than once; thus, RMP_{int} is also a reduced form of the SPP to be solved. In ACO-HCG, collector ants generate new feasible routes that are iteratively added to $\mathcal{R}^*$.

3.2 Pheromone-Based Heuristic Column Generation (ACO-HCG)

The ACO-HCG algorithm iterates over three steps: (*i*) collector ants generate new feasible routes according to the pheromone information τ; (*ii*) the new routes are added to $\mathcal{R}^*$ and the corresponding RMP_{relax} is solved, obtaining a solution *Sol*; and (*iii*) *Sol* is used to update the pheromone information τ. After a time limit has been reached, the

SPP is solved over the set of feasible routes collected ($\mathcal{R}^*$) to produce the final solution to the VRPBB.

Collector Ants. Collector ants are based on the savings-based ants algorithm [13]. Each ant iteratively constructs a set of feasible routes, starting from an initial state where each customer i ($i = 1, \ldots, n$) is visited in a route of its own ($0-i-0$). At each step, the ant constructs a set Ω of potential route merges, selects a merge from Ω and executes the merge. A merge corresponds to the concatenation of two routes $0-i_1-i_2-\cdots-i_e-0$ and $0-j_1-j_2-\cdots-j_l-0$ producing route $0-i_1-i_2-\cdots-i_e-j_1-j_2-\cdots-j_l-0$. That is, edges $(i_e, 0)$ and $(0, j_1)$ are dropped and replaced by a new edge (i_e, j_1). The gain in cost resulting from this merge is computed as $\eta_{i_e j_1} = c_{i_e 0} + c_{0 j_1} - c_{i_e j_1}$.

The set Ω of merges considered by an ant is a subset of the set M of all merges resulting in VRP-feasible routes (VRP-feasible merges). The ant computes the attractiveness of each merge $h \in M$ as

$$\textit{attractiveness}(h) = \tau_{ij}^{\alpha} + \eta_{ij}^{\beta} \tag{1}$$

where (i, j) is the edge being introduced in merge h. The merges in M are then ordered by non-increasing attractiveness and considered one by one for inclusion in Ω. Since the BB-feasibility check is computationally expensive the feasibility information for every checked route is stored in a feasibility pool. If an ant encounters a merge resulting in a route that might help to reduce the total traveling cost (see the original paper [11] for more information) or it is unknown to the feasibility pool, then the ant checks the BB-feasibility (using either the feasibility pool or the black-box function), and the merge is added to Ω only if the route is BB-feasible. If the merge will not reduce the total traveling cost but the resulting route is known to be BB-infeasible, then the merge is nonetheless added to Ω, since this merge may allow discovering further feasible merges that would remain undiscovered otherwise. The construction of Ω stops once π feasible merges have been included, or all merges in M have been considered. Finally, a merge is selected from Ω and executed using roulette-wheel selection based on the attractiveness values. The ant stops its process once no further VRP-feasible merge is possible.

During the construction of Ω, the ants "collect" routes and add them to $\mathcal{R}^*$ only if they are BB-feasible. In addition, these routes are post-optimized before being inserted into $\mathcal{R}^*$ using a tabu search (TS) with infinite length tabu list using 2-opt and relocation moves.

Pheromone Update. After all ants finish collecting routes for $\mathcal{R}^*$, RMP_{relax} is solved, and the pheromones are updated using the resulting solution *Sol* for every edge $(i, j) \in E$ using the formula $\tau_{ij} = \rho\tau_{ij} + \sigma_{ij}\Delta\tau$. That is, for each edge $(i, j) \in E$, the current quantity of pheromones is evaporated (ρ is the trail persistence) and new pheromones are deposited. This quantity is relative to σ_{ij}, the number of times an edge (i, j) appears in *Sol* and a parameter $\Delta\tau$. Moreover, τ_{ij} is not allowed to drop below $\tau_{\min}$.

Extensions of ACO-HCG. We propose several extensions that were not considered in the original paper. First, the original algorithm solves RMP_{relax} at each iteration. Here, we consider solving RMP_{int} instead, in order to obtain an non-fractional solution

Table 1. Parameters considered for automatic configuration

Parameter	Domain	Description
π	$[10, 50] \in \mathbb{N}$	# (VRP and BB)-feasible merges in Ω
m	$[1, 10] \in \mathbb{N}$	# ants executed per iteration
α	$[0, 20] \in \mathbb{N}$	exp. factor for τ in merge attractiveness (Eq. 1)
β	$[0, 20] \in \mathbb{N}$	exp. factor for η in merge attractiveness (Eq. 1)
$\Delta\tau$	$[0.0, 1.0] \in \mathbb{R}$	pheromone update constant
ρ	$[0, 1] \in \mathbb{R}$	trail persistence
$\tau_{\min}$	$[0, 1] \in \mathbb{R}$	lower bound on pheromone level
useint	{never, always, ν}	solve RMP_{int} instead of RMP_{relax}
ν	$[2, 10] \in \mathbb{N}$	if *useint* $== \nu$, solve RMP_{int} instead of RMP_{relax} every ν iterations
strictness	{ strict, liberal }	strict / liberal ants
post-opt	{ ILS, TS }	ILS / TS for post-optimization
op	$\{+, \cdot\}$	use addition/multiplication operator in Eq. 1

at the expense of more computation time. This component is controlled by parameter $useint = \{never, always, \nu\}$. When $useint = \nu$, RMP_{int} is only used for some iterations, concretely, every ν iterations. Second, we implement iterated local search (ILS) using 2-opt, relocation and 4-opt double-bridge moves, as an alternative post-optimization method to the tabu search (TS) proposed in the original ACO-HCG. Third, we consider a stricter variant of the ants ("strict ants") that only include merges in Ω if they are both VRP- and BB-feasible. As explained above, the original ants, which we call "liberal", included BB-infeasible merges in Ω in some circumstances. Fourth, when computing the attractiveness (Eq. 1), the original algorithm sums the pheromone and heuristic values, which is fast to compute but if the ranges of the two values are very different then one will completely dominate the other. Here, we propose to use the product $attractiveness(h) = \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$, which is potentially slower, but it is more robust if τ^{α} and η^{β} have different ranges. These two alternatives are controlled by parameter *op*. A summary of all the algorithmic parameters and their domains is given in Table 1.

4 Experimental Setup

We analyze the components of ACO-HCG in a novel way. First, we find a high-performing parameter configuration by means of irace, an automatic algorithm configuration tool. Second, we examine the effect of each parameter starting from this high-performing configuration.

ACO-HCG was implemented in C++, compiled using gcc 4.4.6 and uses CPLEX 12.4 as the MILP solver. The black box for the MPVRP was provided by Tricoire et al. [18]. We use their exact approach with a time limit of 5 seconds. The black box used for the 3L-CVRP is a reimplementation of the loading approach proposed by Bortfeldt [3] using the same parameters. Experiments were run on a single core of an AMD

Opteron 6272 CPU (2.1 GHz, 16 MB L2/L3 cache size) running under Cluster Rocks Linux version 6/CentOS 6.3, 64bits.

As for the automatic algorithm configuration tool, we use irace [10], a publicly available implementation of Iterated F-Race [1]. Iterated F-Race starts by sampling a number of parameter configurations of a given algorithm uniformly at random. Then, at each iteration, it selects a set of elite configurations using a racing procedure and the non-parametric Friedman test. This racing procedure runs the algorithm configurations iteratively on a sequence of (training) problem instances, and discards configurations as soon as there is enough statistical evidence that they perform worse than the best one. After the race, the elite configurations are used to bias a local sampling model. The next iteration starts by sampling new configurations from this model, and racing is applied to these configurations together with the previous elite configurations. This procedure is repeated until a given budget of runs is exhausted. In this work, we tuned ACO-HCG using a budget of $5\,000$ runs.

Ideally, an automatic configuration method should produce an algorithm configuration that performs well on unseen instances of the same problem. In other words, the method should generalize over the given set of instances, and not overtune the algorithm to those specific instances. To prevent such overtuning, we use *training* instances for the tuning process that are different from the *test* instances used for comparison and in the analysis of parameters. The *test* instances are benchmark instances from the literature and available online: see [20] for MPVRP, and [19] for 3L-CVRP.

The *training* instances were generated by perturbing the benchmark instances from the literature. Only slight perturbations of some customer properties were allowed in order to not destroy the underlying problem structure. Each customer property was perturbed with a probability of 95%. A perturbation replaces the value of the property by *current value* $+ r \cdot \max_{\text{prop}}$, where r is a number selected uniformly at random in the interval $[-\delta, \delta]$ and $\max_{\text{prop}}$ is the maximal value for the considered property in the original instance. Different values for δ were considered ($\delta \in \{0.05, 0.1, 0.15\}$ for MPVRP, and $\delta \in \{0.1, 0.15\}$ for 3L-CVRP).

For the MPVRP instances only predefined types of items are available [4], and, hence, the demand of a customer corresponds to the number of items demanded per type. The following customer properties were considered for perturbation: x-coordinate and demand of one randomly selected type of item. For the 3L-CVRP, the following customer properties were considered: x-coordinate, demand, randomly selected dimension of randomly selected item, fragility of randomly selected item. Five different combinations of these properties were considered.

With each generated instance is associated a time limit (on the route generation phase). This time limit corresponds to the limit associated with the original instance in [18] for the MPVRP (1800 seconds) and [7] (1800, 3600 and 7200 seconds based on the instance size) for the 3L-CVRP.

5 Experimental Results

In this section, we first compare the parameter configuration obtained automatically using irace with the manual configuration of ACO-HCG. Then, we analyze the parameters of ACO-HCG one by one, starting from the automatically obtained configuration.

As mentioned above, the automatic configuration uses a set of training instances, generated by us, and the comparison and analysis uses a different set of test instances from the literature. The performance of the algorithm is measured by computing for each instance the relative percentage deviation (%-deviation) with respect to the best-known solution from the literature. We only consider best-known solutions obtained with the same loading algorithms that we use here as black-box functions ([18,11] for the MPVRP and [3,11] for the 3L-CVRP). We call these solutions *best-bb* in the remainder of this paper. The %-deviation is computed as $100 \cdot \frac{z - z_{\text{best}}}{z_{\text{best}}}$, where z is the solution cost obtained by a run of the algorithm and z_{best} is the best-bb solution cost for the same instance. We use the Wilcoxon signed-rank test with confidence level 95% to assess the statistical significance of the results.

5.1 Manual vs. Automatic Parameter Configurations

We carry out the automatic configuration of ACO-HCG using irace, the parameter domains given in Table 1, and the set of training instances. A run of irace stops after $5\,000$ runs of ACO-HCG. We run irace two times, once for the MPVRP training instances and another time for the 3L-CVRP training instances. Thus, we obtain two "automatic" configurations of ACO-HCG. Table 2 shows these automatic configurations and the two "manual" configurations that were reported in the original paper [11]. However, two parameters of the original algorithm are different in this manual configuration. First, in the original paper a sum ($op = \{+\}$) was used for computing attractiveness. This has been changed to a multiplication ($op = \{\cdot\}$) in order to be coherent with the standard attractiveness formulation used in the ACO literature [5]. Second, in the original algorithm the parameter π was implicitly defined in terms of two other parameters and the instance size, while here π is a single parameter, which simplifies the analysis. The setting of π in the manual configuration is a close approximation to the value that would be obtained given the default values of the two parameters replaced and typical instance sizes. The manual configurations were based on preliminary experiments and standard ACO parameters, and they were found to be competitive with existing approaches.

Some notable differences between the settings of the automatic configurations and the manual ones are that the former have larger value of β, a larger number of ants (m), a lower pheromone persistence (ρ), they solve RMP_{int} in some iterations (ν), and the ants are *strict* instead of *liberal*. The last two parameter settings are actually extensions of the ACO-HCG that we propose in this paper, and selected by irace on its own. In fact, we also propose ILS as an alternative post-optimization method (*post-opt*), but irace did not select it. The parametric analysis in the next section indeed indicates that ILS does not bring any improvement over TS in any of the two benchmark problems.

The automatic and manual configurations of ACO-HCG are compared in Fig. 1 in terms of %-deviation from the best-bb cost on the test instances. Each point in the plot shows the mean %-deviation over 20 independent runs (with different random seeds) on the same test instance. The two configurations perform equally on the same instance if the point is on the diagonal, the automatic configuration performs better if the point is under the diagonal, and the manual configuration performs better if the point is above the diagonal. Moreover, the symbols denote whether the differences observed are statistically significant.

Table 2. Parameter configurations of ACO-HCG

Problem	**Config.**	π	m	α	β	$\Delta\tau$	ρ	$\tau_{\min}$	useint (ν)	*strict.*	*post-opt*	*op*
MPVRP	Manual	13	1	5	5	0.15	0.95	0.20	never	liberal	TS	mult
	Automatic	10	9	1	10	0.69	0.34	0.79	$\nu = 7$	strict	TS	mult
3L-CVRP	Manual	13	5	5	5	0.15	0.95	0.20	never	liberal	TS	mult
	Automatic	41	10	3	9	0.66	0.45	0.29	$\nu = 6$	strict	TS	mult

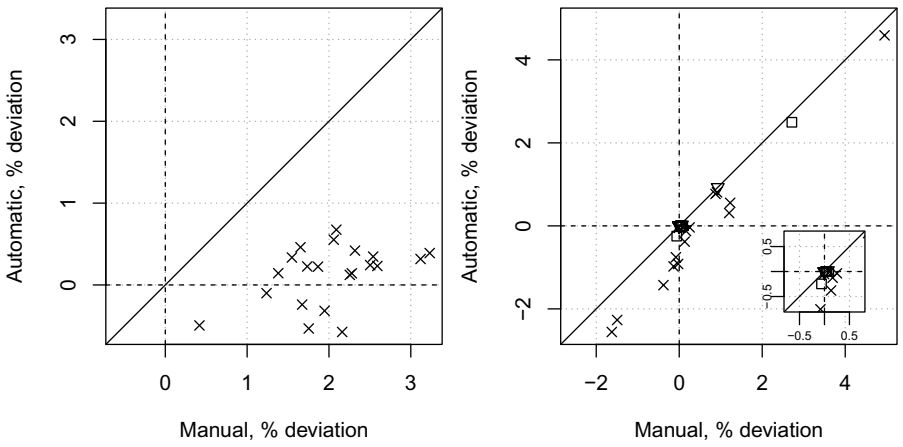
For the MPVRP (Fig. 1(a)), the improvement of the automatic configuration over the manual one is considerable. In particular, all the differences are statistically significant. Moreover, for many instances, the automatic configuration obtains an average result that is better than the best-bb solution. For the 3L-CVRP (Fig. 1(b)), the improvement is smaller. Nonetheless, the automatic configuration is never worse than the manual configuration and it is significantly better on a few instances. In Table 3 the manual and automatic configuration are compared to the best known solutions in literature using all kinds of loading algorithms (and thus possibly obtained using loading algorithms different from the ones used as black box functions in this work). The automatic configuration is able to find new best solutions for 16 out of the 21 MPVRP instances and 3 out of the 27 3L-CVRP instances. Complete tables with worst values and standard deviation are provided as supplementary material [21]

5.2 Experimental Analysis of the ACO-HCG Parameters

In this section, we systematically examine several algorithm parameters. In contrast to how such parametric analyses are carried out in the literature, we adopt a different approach that exploits the benefits of automatic configuration tools. In particular, we do not consider a fully-factorial experimental design, since the number of parameters and the computation time required by each run make such an approach intractable. Instead, we start from the high-performing parameter configuration automatically obtained in the previous section, and examine parameter settings that disable or replace one algorithmic component at a time.

Pheromone Information (α). By setting $\alpha = 0$, we disable the influence of the pheromone information. The result is a noticeable deterioration in quality in most MP-VRP instances (Fig 2(a)) and some 3L-CVRP instances (Fig 2(e)). This suggests that the pheromone information plays a positive role in the performance of the algorithm, probably helping to diversify the routes produced by the ants.

Savings Heuristic (β). By setting $\beta = 0$, we disable the use of the savings heuristic η in the attractiveness equation. The savings heuristic guides the ants to build cost-efficient routes, and, hence, disabling it leads to a substantial quality deterioration in both problems (MPVRP, Fig. 2(b), and 3L-CVRP, Fig. 2(f)). For small instances of the 3L-CVRP, however, the differences are typically minor. This is due to the high value for parameter π. In fact, for small instances with only few customers, setting π, that is, the number of feasible merges in Ω, to a high value, results in most possible merges



(a) MPVRP (b) 3L-CVRP

Fig. 1. Comparison between manual and automatic configuration. Each point gives the mean %-deviation from the best-bb solution over 20 runs with different random seed on the same test instance. The symbols denote whether there is a statistically significant difference (✕) or not (□), or all the runs obtained the same cost (▽).

being included in Ω. These are then checked for feasibility and added to $\mathcal{R}^*$. However, for large instances, the setting of parameter π excludes many interesting merges that would have a high heuristic value if $\beta \neq 0$. In summary, the savings heuristic remains essential for the generation of high-quality routes.

Learning Mechanism (ρ). By setting $\rho = 0$, the pheromones are reset at every iteration, and only the amount deposited in the current iteration has an effect. Hence, this setting disables the learning mechanism of ACO and forces the ants to focus on the solution found in the current iteration. Given that the results do not show a clear effect of the learning mechanism (Fig. 2(d) and 2(h)), but that completely disabling the pheromone information ($\alpha = 0$, as discussed above) does deteriorate quality, we conclude that the pheromone information provides a diversification mechanism rather than learning the best edges over time.

Strict vs. Liberal Ants. Whereas *strict* ants may only execute merges resulting in VRP- and BB-feasible routes, *liberal* ants may also execute merges resulting in only VRP-feasible routes. The rationale of *liberal* ants is that BB-infeasible routes might be merged in order to produce BB-feasible routes. This, of course, depends on the black box at hand. In the case of the MPVRP, such a situation cannot arise and, hence, *strict* ants produce much better results (Fig. 2(c)). While for the 3L-CVRP two infeasible routes can theoretically be concatenated to produce a feasible route, such a situation does not seem to occur frequently in the benchmark instances available (Fig. 2(g)). Thus, the choice of *strict* ants rather than *liberal* ants in the automatic configuration seems to be justified.

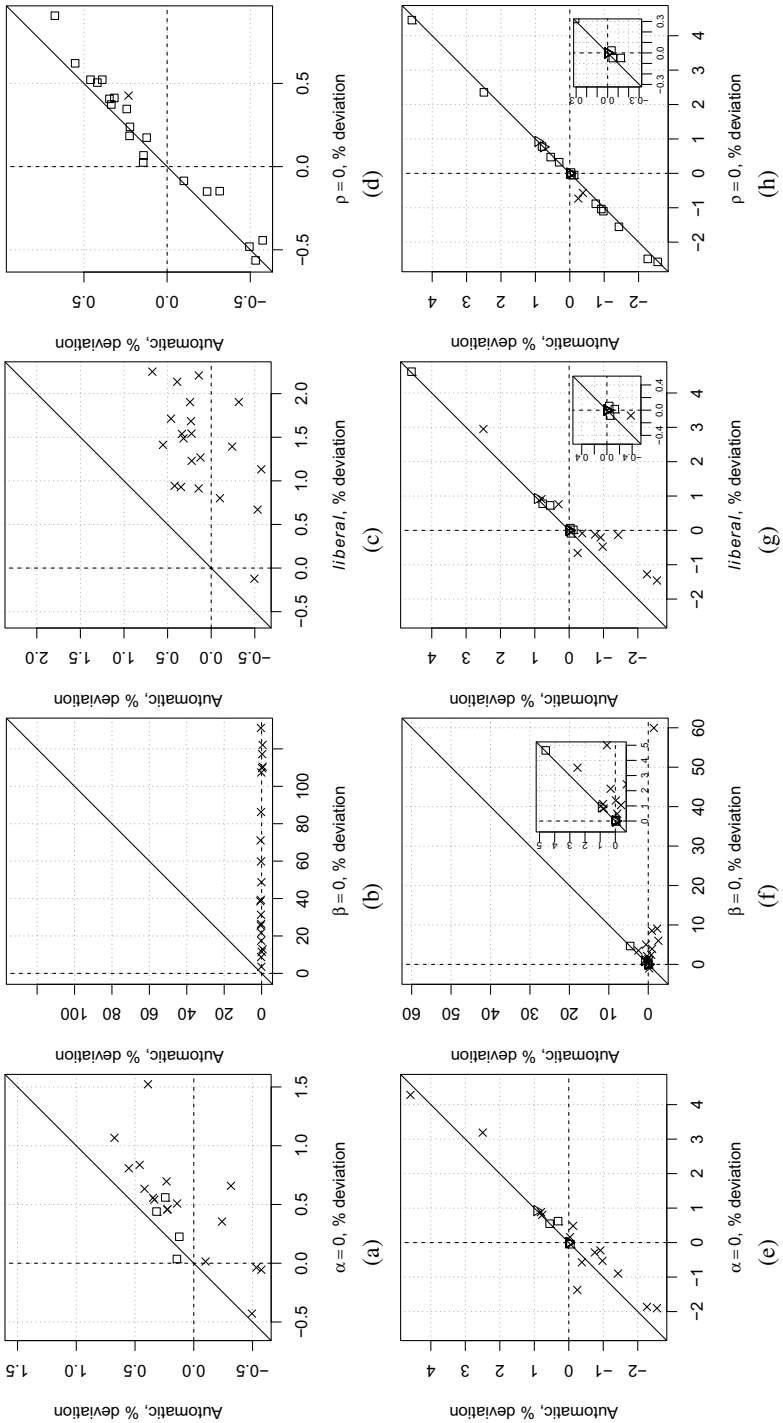


Fig. 2. Each plot shows the effect of changing one parameter of the "automatic" configuration. Results for the MP-VRP are in the upper row; results for 3L-CVRP in the lower row. Each point gives the mean %-deviation from the best-bb solution over 20 runs with different random seed on the same test instance. The symbols denote whether there is statistically significant difference (×) or not (□), or all the runs obtained the same cost (▽).

Table 3. Comparison of "manual" and "automatic" configuration with best-known results. $z_{min/avg}$=best and average solution value, sec_{tt}=total execution time in seconds, gap_{avg}=average relative percentage deviation w.r.t. best published solution. All results over 20 independent runs. The relative percentage deviation for one run is computed as $100 \cdot \frac{z - z_{\text{best}}}{z_{\text{best}}}$ where z is the solution value for the given run and z_{best} the best published solution value. For the automatic configuration results in **bold** indicate a tie or improvement over the best published solution value, results in *italic* indicate a tie or improvement over the average solution value obtained using the manual configuration.

(a) Comparison with best-known solutions for the MPVRP

	Best	Manual Conf.				Automatic Conf.			
	z_{best}	z_{min}	z_{avg}	sec_{tt}	g_{avg}	z_{min}	z_{avg}	sec_{tt}	g_{avg}
CMT01-1	587.29	590.45	599.38	1820	2.06	**587.29**	*590.54*	1833	0.55
CMT01-2	615.12	617.82	628.99	1807	2.25	**615.11**	*615.88*	1811	0.12
CMT01-3	623.45	623.91	632.06	1807	1.38	**623.44**	*624.35*	1806	0.14
CMT02-1	978.66	976.70	984.47	1811	0.59	**974.47**	*975.53*	1848	-0.32
CMT02-2	897.62	901.91	911.51	1826	1.55	**897.51**	*900.63*	1825	0.34
CMT02-3	888.38	895.49	903.79	1813	1.73	889.26	*890.39*	1814	0.23
CMT03-1	1188.18	1198.09	1218.03	1833	2.51	**1180.21**	*1184.93*	1819	-0.27
CMT03-2	1218.96	1229.23	1241.71	1833	1.87	1219.13	*1221.68*	1926	0.22
CMT03-3	1156.84	1170.63	1186.82	1836	2.59	**1154.11**	*1159.52*	1814	0.23
CMT04-1	1624.98	1631.73	1660.10	1881	2.16	**1607.63**	*1615.64*	1900	-0.57
CMT04-2	1552.27	1564.80	1578.23	1864	1.67	**1543.37**	*1548.54*	1940	-0.24
CMT04-3	1541.81	1563.87	1578.30	1862	2.37	**1541.35**	*1545.32*	1899	0.23
CMT05-1	2035.77	2052.01	2074.49	1950	1.90	**2019.80**	*2027.90*	2029	-0.39
CMT05-2	1833.41	1866.50	1892.65	1909	3.23	**1832.18**	*1840.55*	2022	0.39
CMT05-3	1948.84	1974.25	1996.54	1946	2.45	**1946.30**	*1952.23*	1907	0.17
CMT06-1	2240.57	2242.65	2292.50	2072	2.32	**2238.56**	*2249.97*	2111	0.42
CMT06-2	2070.04	2107.45	2142.76	2021	3.51	2089.17	*2096.93*	2000	1.30
CMT06-3	2154.19	2169.29	2189.78	1870	1.65	**2153.45**	*2164.11*	1847	0.46
CMT07-1	1136.55	1151.85	1160.34	1855	2.09	1140.14	*1144.25*	1835	0.68
CMT07-2	1217.45	1226.17	1232.51	1855	1.24	**1214.58**	*1216.23*	1854	-0.10
CMT07-3	1157.67	1171.37	1188.00	1894	2.62	**1152.16**	*1161.79*	1847	0.36
AVG					2.08				0.19

(b) Comparison with best-known solutions for the 3L-CVRP

	Best	Manual Conf.				Automatic Conf.			
	z_{best}	z_{min}	z_{avg}	sec_{tt}	g_{avg}	z_{min}	z_{avg}	sec_{tt}	g_{avg}
3l-cvrp01	291.00	302.02	302.13	1800	3.82	302.02	*302.02*	1800	3.79
3l-cvrp02	334.96	334.96	334.96	1800	0.00	**334.96**	*334.96*	1800	0.00
3l-cvrp03	392.46	385.53	392.23	1800	-0.06	**385.53**	*391.49*	1800	-0.25
3l-cvrp04	437.19	437.19	437.19	1800	0.00	**437.19**	*437.19*	1800	0.00
3l-cvrp05	443.61	447.73	447.73	1800	0.93	447.73	*447.73*	1800	0.93
3l-cvrp06	498.16	498.16	498.27	1800	0.02	**498.16**	*498.16*	1800	0.00
3l-cvrp07	768.85	769.68	769.68	1800	0.11	769.68	*769.68*	1800	0.11
3l-cvrp08	805.35	845.50	851.01	1800	5.67	845.50	*848.12*	1800	5.31
3l-cvrp09	630.13	630.13	630.67	1800	0.09	**630.13**	*630.13*	1800	0.00
3l-cvrp10	817.38	826.66	827.52	3601	1.24	826.66	*826.66*	3603	1.14
3l-cvrp11	778.10	776.19	778.03	3600	-0.01	**776.19**	*777.72*	3600	-0.05
3l-cvrp12	612.25	612.25	612.88	3600	0.10	**612.25**	*612.25*	3600	0.00
3l-cvrp13	2645.95	2661.62	2670.06	3600	0.91	2661.62	*2667.32*	3601	0.81
3l-cvrp14	1368.42	1392.06	1405.56	3602	2.71	1385.00	*1402.58*	3604	2.50
3l-cvrp15	1341.14	1336.21	1343.34	3611	0.16	**1336.21**	*1339.46*	3618	-0.13
3l-cvrp16	698.61	698.61	698.61	3600	0.00	**698.61**	*698.61*	3600	0.00
3l-cvrp17	866.40	866.40	866.84	3600	0.05	**866.40**	*866.40*	3600	0.00
3l-cvrp18	1207.72	1205.11	1222.25	3612	1.20	**1205.11**	*1211.46*	3614	0.31
3l-cvrp19	741.74	741.31	743.59	7203	0.25	**741.31**	*741.47*	7203	-0.04
3l-cvrp20	586.92	581.13	586.12	7236	-0.14	**577.39**	*581.19*	7215	-0.98
3l-cvrp21	1042.72	1080.24	1089.93	7272	4.53	1075.16	*1080.22*	7217	3.60
3l-cvrp22	1147.80	1154.12	1161.89	7220	1.23	1148.82	*1154.18*	7211	0.56
3l-cvrp23	1119.05	1104.06	1118.14	7229	-0.08	**1101.47**	*1110.60*	7213	-0.76
3l-cvrp24	1096.88	1112.49	1117.60	7225	1.89	1107.15	*1111.89*	7225	1.37
3l-cvrp25	1407.36	1389.35	1402.02	7346	-0.38	**1373.24**	*1387.27*	7240	-1.43
3l-cvrp26	1430.15	1553.04	1568.21	7291	9.65	1544.40	*1555.92*	7216	8.79
3l-cvrp27	1455.27	1493.33	1503.80	7316	3.33	1483.59	*1489.56*	7235	2.36
AVG					1.38				1.03

Sum vs. Multiplication in Attractiveness Equation. The results clearly worsen when using a sum ($op = +$) in the attractiveness equation. Since the values of the savings heuristic are much larger than the pheromone values, summing both neglects the effect of the pheromones. In fact, the plots (available as supplementary material [21]) are almost identical to those where the pheromone information is disabled ($\alpha = 0$, Fig 2(a) and 2(e)). Therefore, the use of multiplication is recommended.

We also analyzed other parameters of ACO-HCG, however, for the sake of conciseness, we only briefly summarize our findings here, and provide the full plots as supplementary material [21]. In particular, we analyzed the parameter *useint*, which controls whether the solution used to update the pheromone information is obtained by solving RMP_{int} or RMP_{relax}. For the MPVRP, while never using the integer solution does not have a significant effect on most instances, always using it slightly deteriorates the quality in some instances. For the 3L-CVRP, it does not matter whether we use RMP_{relax} or RMP_{int} to update the pheromones. When comparing ILS vs. TS as the post-optimization method, we observe that, in the MPVRP, solution quality does deteriorate in some instances when using ILS instead of TS, and in the 3L-CVRP, a slight deterioration can be observed on a couple of instances. Finally, if the number of ants is set to one ($m = 1$), results improve slightly on a few instances and get slightly worse in others. In the case of the 3L-CVRP, setting $m = 1$ slightly deteriorates quality in a majority of the instances.

6 Conclusion

In this paper, we carried out a parametric analysis of ACO-HCG for the VRPBB in a novel way. As a first step, we obtained a high-performing configuration of ACO-HCG by means of automatic configuration for two variants of the VRPBB, namely, MPVRP and 3L-CVRP. This automatic configuration significantly improves the results obtained by the default configuration of ACO-HCG. The default configuration was developed based on intuition and a few preliminary experiments, and it was found to be competitive with the state of the art. However, due to the long computation times required and the large number of parameters, further improving the parameter configuration by traditional methods was deemed intractable. Automatic configuration tools allowed us to overcome this difficulty and test new algorithmic components. In fact, the new configuration is able to improve the best-known solutions on many instances.

As a second step, we systematically analyzed the parameters of ACO-HCG, starting from the automatically-found configuration, and disabling or replacing one component at a time to observe its effect on quality. In this way, we identified four components that have a large effect on the results, namely, the pheromone information, the savings heuristic, the strictness of the ants when considering feasible routes, and the equation used for computing the attractiveness. Moreover, although the use of pheromone information is necessary to complement the savings heuristic, we also determined that the learning mechanism of ACO does not have a strong effect. These insights should lead to future improvements in the algorithm.

The analysis methodology used here can easily be applied to other algorithms (using other automatic configuration tools besides irace). We expect that the larger improvements and best insights will be obtained when analyzing hybrid algorithms with many

parameters that require long runs. In that case, the default configuration is probably far from optimal and previous studies may have missed insights that are only relevant for high-performing parameter configurations.

Future work should consider new algorithmic components that may improve ACO-HCG, other black-box functions found in the literature, and the effect of the parameters on the computation time required by the algorithm. We will also perform a more elaborate sensitivity analysis of the parameters, evaluating the influence of variations in their values.

Acknowledgments. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Manuel López-Ibáñez and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a postdoctoral researcher and a research associate, respectively. The authors also acknowledge support from the FRFC project "*Méthodes de recherche hybrides pour la résolution de problèmes complexes*". This research and its results have also received funding from the COMEX project within the Interuniversity Attraction Poles Programme of the Belgian Science Policy Office. Florence Massen is supported by the National Research Fund, Luxembourg.

References

1. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)
2. Birattari, M.: Tuning Metaheuristics: A Machine Learning Perspective. SCI, vol. 197. Springer, Heidelberg (2009)
3. Bortfeldt, A.: A hybrid algorithm for the capacitated vehicle routing problem with three-dimensional loading constraints. Comp. & Op. Res. 39(9), 2248–2257 (2012)
4. Doerner, K.F., Fuellerer, G., Hartl, R.F., Gronalt, M., Iori, M.: Metaheuristics for the vehicle routing problem with loading constraints. Networks 49(4), 294–307 (2007)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press (2004)
6. Fuellerer, G., Doerner, K.F., Hartl, R.F., Iori, M.: Metaheuristics for vehicle routing problems with three-dimensional loading constraints. EJOR 201(3), 751–759 (2010)
7. Gendreau, M., Iori, M., Laporte, G., Martello, S.: A tabu search algorithm for a routing and container loading problem. Trans. Sci. 40(3), 342–350 (2006)
8. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
9. Iori, M., Martello, S.: Routing problems with loading constraints. TOP 18, 4–27 (2010)
10. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
11. Massen, F., Deville, Y., Van Hentenryck, P.: Pheromone-based heuristic column generation for vehicle routing problems with black box feasibility. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) CPAIOR 2012. LNCS, vol. 7298, pp. 260–274. Springer, Heidelberg (2012)
12. Prescott-Gagnon, E., Desaulniers, G., Drexl, M., Rousseau, L.M.: European driver rules in vehicle routing with time windows. Trans. Sci. 44(4), 455–473 (2010)

13. Reimann, M., Doerner, K., Hartl, R.F.: D-ants: Savings based ants divide and conquer the vehicle routing problem. Comp. & Op. Res. 31(4), 563–591 (2004)
14. Ren, J., Tian, Y., Sawaragi, T.: A relaxation method for the three-dimensional loading capacitated vehicle routing problem. In: 2011 IEEE/SICE International Symposium on System Integration (SII), pp. 750–755. IEEE (2011)
15. Ruan, Q., Zhang, Z., Miao, L., Shen, H.: A hybrid approach for the vehicle routing problem with three-dimensional loading constraints. Comp. & Op. Res. (2011)
16. Tarantilis, C., Zachariadis, E., Kiranoudis, C.: A hybrid metaheuristic algorithm for the integrated vehicle routing and three-dimensional container-loading problem. IEEE Transactions on Intelligent Transportation Systems 10(2), 255–271 (2009)
17. Toth, P., Vigo, D. (eds.): The Vehicle Routing Problem. SIAM Monographs on Discrete Mathematics and Applications (2002)
18. Tricoire, F., Doerner, K.F., Hartl, R.F., Iori, M.: Heuristic and exact algorithms for the multi-pile vehicle routing problem. OR Spectrum 33(4), 931–959 (2011)
19. Benchmark instances for the 3L-CVRP, `http://www.or.deis.unibo.it/research.html`
20. Benchmark instances for the MPVRP, `http://prolog.univie.ac.at/research/VRPandBPP/`
21. Supp. material, `http://becool.info.ucl.ac.be/resources/ACO-HCG-IRACE`
22. Zhu, W., Qin, H., Lim, A., Wang, L.: A two-stage tabu search algorithm with enhanced packing heuristics for the 3L-CVRP and M3L-CVRP. Comp. & Op. Res. 39(9), 2178–2195 (2012)

A New Hybrid Metaheuristic – Combining Stochastic Tunneling and Energy Landscape Paving

Kay Hamacher

Dept. of Computer Science, Dept. of Physics & Dept. of Biology,
Technical University Darmstadt, Schnittspahnstr. 10, 64287 Darmstadt, Germany

Abstract. (Hybrid) metaheuristics such as simulated annealing, genetic algorithms, or extremal optimization play a most prominent role in global optimization. The performance of these algorithms and their respective sampling behavior during the search process are themselves interesting problems. Here, we show that a combination of two approaches – namely Energy Landscape Paving (ELP) and Stochastic Tunneling (STUN) – can overcome known problems of other Metropolis-sampling-based procedures. We show on grounds of non-equilibrium statistical mechanics and empirical evidence on the synergistic advantages of this combined approach and discuss simulations for a complex optimization problem.

1 Introduction

Global Optimization (GO) is an active area of research – both from a conceptual point of view and under experimental analysis. Applications range from the protein structure prediction problem (see e.g. [19,28] and references therein), over machine learning [29], vehicle routing [8], assembly line assignments [6], to quantitative finance [20].

A GO procedure constitutes a search process in configurational and "function value" space. The search dynamics in these spaces has become an interesting area of experimental research in itself [21,9,27,14,23].

In particular, with the event of simulated annealing [17] Monte-Carlo/ Metropolis [22] based metaheuristics became more and more popular due their astonishing success. The tuning of internal parameters such as the cooling schedule is the pitfall for unexperienced users, although a better understanding was achieved over decades of application, see e.g. [24,3]. In the realm of simulated annealing research it has become custom to investigate *minimization* as the optimization goal. We will stick to this notion and regard *optimization* as a *minimization* task[1].

1.1 Previous Work

Previous Work – Combining Metaheuristics. The general idea to combine successful approaches – either two metaheuristics or a metaheuristic with an

[1] *Maximization* is a trivial transformation to render it a *minimization* problem.

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 107–117, 2013.

analytical method – was developed quite some time ago. E.g., incorporating tabu search into other approaches was proposed [1,10], multi-level procedures were suggested [32], and Metropolis sampling on an effective function, which is locally optimized via, e.g., conjugate gradient methods [26], was successfully applied [25,31].

Two metaheuristics which we will use are described in the following section.

Previous Work – Stochastic Tunneling & Energy Landscape Paving. Energy landscape paving (ELP) was suggested to overcome the major problem of simulated annealing approaches: freezing of the search dynamics in suboptimal local minima [15]. To this end, the original objective function $f(\boldsymbol{x})$ is replaced by the following altered version

$$f_{\mathrm{ELP}}(\boldsymbol{x}) := f(\boldsymbol{x}) + \frac{\kappa}{\int H(q')dq'} \cdot H(q) \tag{1}$$

where $H(q)$ is a histogram of a quantity q that characterizes potential solutions. $H(q)$ "counts" the number of times solutions $\boldsymbol{x}$ are visited that are characterized by the same value of q. We assume that this quantity can be computed by a function $q(\boldsymbol{t})$ for all $\boldsymbol{t}$ of the domain of definition of f. To normalize the impact of the histogram, we divide by the sum of entries of H.

The parameter κ is a scaling parameter and quantifies the importance of the adaptation procedure via $H(q)$. The function $f_{\mathrm{ELP}}(\boldsymbol{x})$ is then sampled with a Metropolis-criterion based metaheuristic.

In typical applications, q is often chosen to be (a) easy to compute and (b) a scalar. In particular, property (b) is important to avoid the "curse of dimensionality" [7] often encountered in applied computing. Typical examples in regard to property (a) are, e.g., the number of river crossings in traveling salesperson problems or the number of secondary structure motifs in protein structures.

However, $q := f$ is a valid choice, too. This particular q is easily implemented, and is not domain-specific. Due to these reasons, we have used the objective function value itself as a characteristic q in our new methodology (see Sec. 2).

In Algorithm 1 we show in pseudo-code the ELP metaheuristic.

The rationale of ELP is as follows: whenever a "region" characterized by the same value of q was visited frequently during the run, the effective function f_{ELP} will show larger values. This effectively makes it easier for the Metropolis process to leave this region. Thus, a "trapping" can not occur, as – after sufficient number of re-visits – the function f_{ELP} will become larger and larger until the search process is driven from this region to another one.

Another approach to deal with trapping phenomena in search dynamics was developed via a non-linear transformation of the objective function: here, the objective function is mapped to an effective one that "damps" those function values which have already been visited in the past. The rationale here is, that

Algorithm 1. Energy landscape paving (ELP) which runs for G iterations. Here, `rand()` is a random number $\mathcal{U}(0,1)$ and $\mathcal{N}(\boldsymbol{x}_{g-1})$ is a vector from the neighborhood of $\boldsymbol{x}_{g-1}$. The histogram $H(q)$ keeps track of visited solutions that are characterized by a quantity q.

Require: randomly chosen start solution $\boldsymbol{x}_0$, an empty histogram $H(q)$
  $f_0 := f(\boldsymbol{x}_0)$
  $f_{\text{best}} := f_0$
  $q_0 := q(\boldsymbol{x}_0)$
  update $H(q_0) \leftarrow H(q_0) + 1$
  for $1 \leq g \leq G$ **do**
    $\boldsymbol{t} :=$ draw from $\mathcal{N}(\boldsymbol{x}_{g-1})$
    if $\exp\left(-\beta \cdot (f_{\text{ELP}}(\boldsymbol{t}) - f_{\text{ELP},g-1})\right) <$ `rand()` **then**
      ▷ Metropolis criterion for f_{ELP} of Eq. 1
      $\boldsymbol{x}_g := \boldsymbol{t}$
      $f_g = f(\boldsymbol{t})$
      $q_g := q(\boldsymbol{t})$
      if $f_g < f_{\text{best}}$ **then**
        $f_{\text{best}} := f_g$ and $\boldsymbol{x}_{\text{best}} := \boldsymbol{t}$
    else
      $\boldsymbol{x}_g := \boldsymbol{x}_{g-1}$ and $f_g = f_{g-1}$ and $q_g := q_{g-1}$
    update $H(q_g) \leftarrow H(q_g) + 1$

function values larger than the current best guess f_{best} of the (global) minimum are of no interest.

Therefore, the search process during Metropolis sampling can traverse regions with larger function values than f_{best} faster to avoid trapping in unimportant – potentially not seen – minima.

To this end, the objective function $f(\boldsymbol{x})$ is transformed to

$$f_{\text{STUN}}(\boldsymbol{x}) := 1 - e^{-\gamma \cdot (f(\boldsymbol{x}) - f_{\text{best}})} \tag{2}$$

and sampled via a Metropolis process again. The resulting algorithm is shown in Algorithm 2. Obviously, there is a subtle connection to the well-know tabu search approach [11,12].

Note, that we do *not* need to evaluate f on its whole domain of definition. Rather, STUN just requires to evaluate an exponential of a specific funtion value. Therefore, no curse of dimensionality occurs in the computation of f_{STUN}.

1.2 Our Contribution: Synergistically Combining STUN and ELP

Here, we will propose a different combination of metaheuristics: we will combine two approaches to avoid "trapping" phenomena which were identified to be of devastating importance for the suboptimal performance of simulated annealing in real-world applications [16].

These two approaches – Stochastic Tunneling (STUN) and Energy Landscape Paving (ELP) – are described above. In Sec. 2 we will motivate our newly developed procedure and discuss its potential merits. Sec. 4 shows empirical results

Algorithm 2. Stochastic Tunneling (STUN) which runs for G iterations. Here, `rand()` is a random number $\mathcal{U}(0,1)$ and $\mathcal{N}(\boldsymbol{x}_{g-1})$ is a vector from the neighborhood of $\boldsymbol{x}_{g-1}$.

Require: randomly chosen start solution $\boldsymbol{x}_0$
  $f_0 := f(\boldsymbol{x}_0)$
  $f_{\text{best}} := f_0$
  $\tilde{f}_0 := 1 - \exp\left(-\gamma \cdot (f(\boldsymbol{x}_0) - f_{\text{best}})\right)$
  for $1 \leq g \leq G$ **do**
    $\boldsymbol{t} :=$ draw from $\mathcal{N}(\boldsymbol{x}_{g-1})$
    $\tilde{f}(\boldsymbol{t}) := 1 - \exp\left(-\gamma \cdot (f(\boldsymbol{t}) - f_{\text{best}})\right)$
    if $\exp\left(-\beta \cdot \left(\tilde{f}(\boldsymbol{t}) - \tilde{f}_{g-1}\right)\right) <$ `rand()` **then** ▷ Metropolis criterion for $\tilde{f}$
      $\boldsymbol{x}_g := \boldsymbol{t}$
      $f_g = f(\boldsymbol{t})$
      $\tilde{f}_g = \tilde{f}(\boldsymbol{t})$
      if $f_g < f_{\text{best}}$ **then**
        $f_{\text{best}} := f_g$ and $\boldsymbol{x}_{\text{best}} := \boldsymbol{t}$
    else
      $\boldsymbol{x}_g := \boldsymbol{x}_{g-1}$ and $f_g = f_{g-1}$ and $\tilde{f}_g = \tilde{f}_{g-1}$

on the average performance for a test problem introduced in Sec. 3. The paper ends with a discussion and some conclusions.

2 A Combined Algorithm

Trapping of stochastic search procedures severely hinders any of these randomized algorithms to progress towards better solutions [16]. We propose to combine the ideas of ELP and STUN to avoid trapping to allow synergistic behavior of these approaches. At the same time, the ELP approach puts a penalty – via $H(q)$ in Eq. 1 – on revisiting regions previously frequently looked into. This might help to cope with a problem of STUN: its progress will saturate after the non-linear transformation of Eq. 2 has deleted too much structure of the search space by "flattening" the effective objective function f_{STUN} too much [13].

Therefore, we will work on the following effective objective function and sample it with Metropolis-like dynamics:

$$f_{\text{TAAO}}(\boldsymbol{x}) := 1 - e^{-\gamma \cdot (f(\boldsymbol{x}) - f_{\text{best}})} \; + \; \kappa \cdot H(q) \tag{3}$$

That is, we transform with increasingly better f_{best} the local minima and saddle-points away, while pushing the search process from too frequently visited regions away (via $H(q)$).

In the following, we will refer to this procedure as Tunneling-And-Avoidance-Optimization (TAAO). We lay out the approach in Algorithm 3.

Algorithm 3. Tunneling-And-Avoidance-Optimization (TAAO) which runs for G iterations. Again, `rand()` is a random number $\mathcal{U}(0,1)$ and $\mathcal{N}(\boldsymbol{x}_{g-1})$ is – again – a vector from the neighborhood of $\boldsymbol{x}_{g-1}$.

Require: randomly chosen start solution $\boldsymbol{x}_0$
  $f_0 := f(\boldsymbol{x}_0)$
  $f_{\text{best}} := f_0$
  $q_0 := q(\boldsymbol{x}_0)$
  $\tilde{f}_0 := 1 - \exp\left(-\gamma \cdot (f(\boldsymbol{x}_g) - f_{\text{best}})\right)$
  update $H(q_0) \leftarrow H(q_0) + 1$
  for $1 \leq g \leq G$ **do**
    $\boldsymbol{t} :=$ draw from $\mathcal{N}(\boldsymbol{x}_{g-1})$
    $\tilde{f}(\boldsymbol{t}) := 1 - \exp\left(-\gamma \cdot (f(\boldsymbol{t}) - f_{\text{best}})\right) + \frac{\kappa}{\int H(q')dq'} \cdot H(q(\boldsymbol{t}))$
    if $\exp\left(-\beta \cdot \left(\tilde{f}(t) - \tilde{f}_{g-1}\right)\right) <$ `rand()` **then** ▷ Metropolis criterion for $\tilde{f}$
      $\boldsymbol{x}_g := t$
      $f_g = f(t)$
      $\tilde{f}_g = \tilde{f}(t)$
      $q_g := q(\boldsymbol{t})$
      if $f_g < f_{\text{best}}$ **then**
        $f_{\text{best}} := f_g$ and $\boldsymbol{x}_{\text{best}} := \boldsymbol{t}$
    else
      $\boldsymbol{x}_g := \boldsymbol{x}_{g-1}$ and $f_g = f_{g-1}$ and $\tilde{f}_g = \tilde{f}_{g-1}$ and $q_g := q_{g-1}$
    update $H(q_g) \leftarrow H(q_g) + 1$

3 A Test Instance

In this study, we employ a combinatorial optimization problem to investigate experimentally the performance of the proposed hybrid algorithm of Sec. 2.

We want to determine the lowest energy of Ising spin-glasses [4] with Gaussian distributed interaction couplings. This problem is well suited as efficient benchmarks and known solutions are available [30]. The optimization problem reads

$$\begin{aligned} \min E(\boldsymbol{s}) &= \sum_{<i,j>} J_{ij} s_i s_j \\ \bigvee_{i \in \{1 \ldots N\}} s_i &\in \left[-\frac{1}{2}; \frac{1}{2}\right]. \end{aligned} \tag{4}$$

The summation $< i, j >$ in Eq. 4 is restricted to nearest neighbors. Here, we will restrict ourselves to N spins on a 2D regular lattice of side length $\sqrt{N}$. All interaction parameters J_{ij} are drawn from a normal distribution. The Ising ground state problem of Eq. 4 can considered a binary non-linear programming problem. The optimal solution is contained in the vector $\boldsymbol{s}^* := (s_1^*, s_2^*, \ldots, s_N^*)$.

In the subsequent parts of this study we will always report values obtained for an ensemble of 50 independently created Ising spin systems; that is, we have 50 times drawn N values J_{ij}, optimized each systems 10 times independently,

and taken the algebraic mean over these $50 \cdot 10 = 500$ runs as an indicator for the *average* performance.

4 Results

Here, we present empirical results on (1) the average performance in terms of relative error, (2) the search dynamics itself, and (3) a first sensitivity analysis towards a modified mechanism in TAAO. For the latter, we will use a variant of Algorithm 3: we reset the histogram $H(q)$ whenever we encounter a new best optimum $\boldsymbol{x}_{\text{best}}$; thus we use the ELP-behavior only between successive improvements of our best guess of $f_{\text{best}} := \boldsymbol{x}_{\text{best}}$ – this variant will be called "TAAO & Reset" in the following.

4.1 Performance Benchmarking – Quality of Solutions

As laid out above, we performed 500 independent runs on a sufficiently large combinatorial optimization problem. To demonstrate the superior performance we show in Fig. 1 the relative error in the obtained best function values[2].

Clearly, we see superior performance for the obtained minima in the case of STUN and TAAO. Both are superior than ELP alone. STUN shows the previously criticized saturation phenomena [13] from $2 \cdot 10^7$ iterations on. If we look closer into the performance of both STUN and TAAO in Fig. 1b, we observe two effects: (a) TAAO is obtaining better minima (some factor of two in relative error) and (b) the saturation sets in at higher iteration numbers (some $8 \cdot 10^7$ iterations).

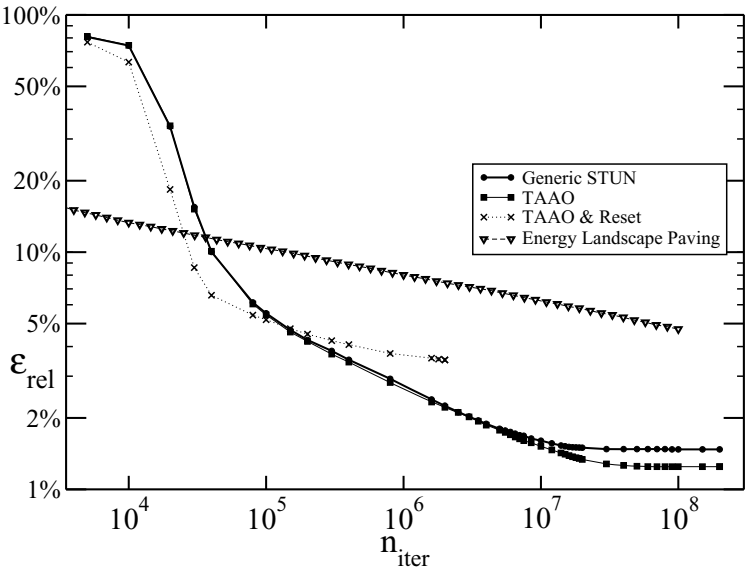
With these results we have demonstrated that the approach of Algorithm 3 prevents the search dynamics to saturate in terms of relative error. But the results are event more promising beyond this point: it is known [18] that the number of states in Ising spin classes is for small energies an exponential in the energy gap, that is in the value $E - E^*$ of Eq. 4 where E^* is the (known) absolute minimal energy of a particular incarnation of a Ising spin system.

Therefore, the number of states at $\epsilon := E - E^*$ is close to $\exp(a \cdot \epsilon)$. From the results of Fig. 1b we can conclude that the number of configurations that the TAAO-result is away from the true ground state is *exponentially smaller* than the number of states STUN would have to traverse additionally to reach TAAO's performance.

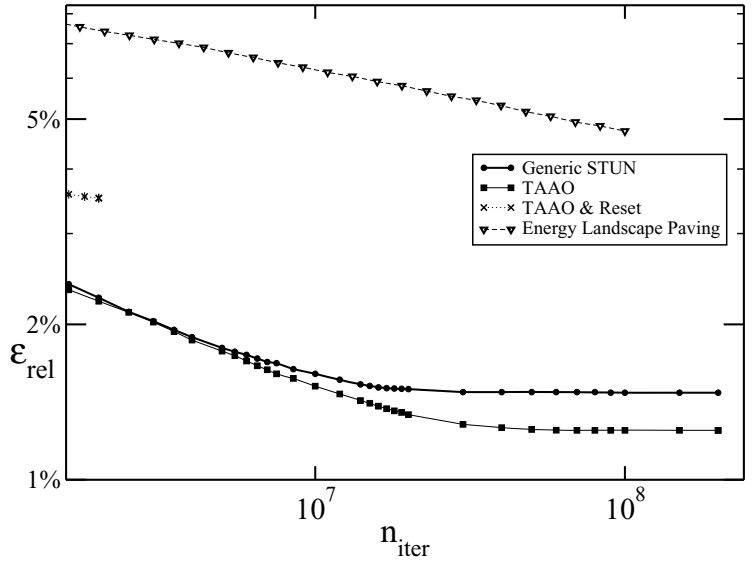
4.2 Statistical Properties of Search Dynamics

Previously, the dynamics of randomized algorithms for optimization were already subject to analysis [21,9,27,13]. In Fig. 2 we compare the auto-correlation functions of function values visited by all optimization schemes, where the auto-correlation function reads

[2] Exact solutions were obtained from [30].



(a) The averaged relative error ϵ_{rel} for the methods under investigation. Error bars were smaller than symbol sizes and therefore omitted. The results for Energy Landscape Paving were taken from [14] and the data for the Generic STUN algorithm from [13].



(b) Zoomed version of Fig. 1a

Fig. 1

$$\sigma^2 := \langle (f - \langle f \rangle)^2 \rangle_T \text{ with} \tag{5}$$

$$\langle f(f) \rangle_T := \frac{1}{T} \sum_{t=1}^{T} f(f_t) \text{ and thus} \tag{6}$$

$$\sigma^2 = \langle f^2 \rangle_T - \langle f \rangle_T^2. \tag{7}$$

This quantity measures the correlations of function values as a time-series: each search procedure creates such a series during its progress and the more correlated the values are, the more a search procedure exploits structure in search space. If $\sigma \approx 0$, then there exists hardly any correlation in the function values and the metaheuristic would randomly choose different function values, thus effectively perform blind guessing – hardly a promising approach. Now, Fig. 2 shows that TAAO shows *always* higher σ, thus exploits correlations better than other approaches.

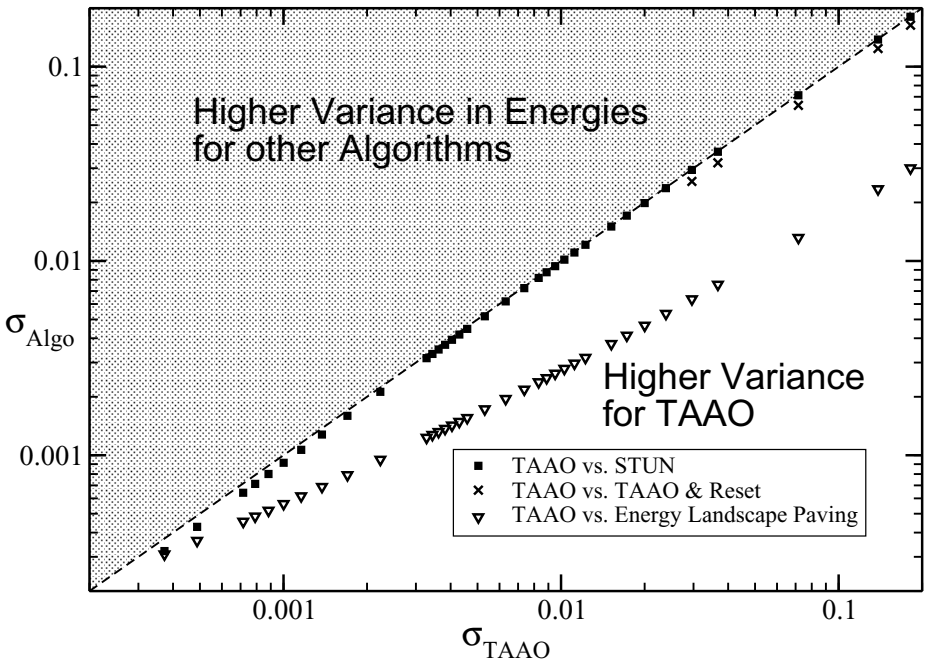


Fig. 2. Comparison of the fluctuations during the runs between the TAAO methodology as laid out in the text and the various other procedures investigated. Note that no point lies in the gray-shaded, upper triangle – therefore we find empirically $\sigma_{\text{TAAO}} > \sigma_a$ for all $a \in$ [STUN; TAAO&Reset; Adaptation; ELP].

These results suggest that the subtle interplay of STUN transformation and region avoidance via $H(q)$ of ELP provides for a better exploitation of any structure in the search problem.

4.3 Sensitivity Analysis to $H(q)$ Choices

As mentioned above, we also wanted to partially elucidate the interplay of STUN transformation and $H(q)$. The effect of using ELP-like behavior between successive improvements in f_{best} only and reseting the histogram upon each improvement (and thus a new effective objective function) is also shown in Figs. 1a and 1b with the label "TAAO & Reset".

Clearly, this variant is significantly worse performing than STUN or TAAO in its original formulation. From this differential diagnosis we can conclude that it is *not* the ELP mechanism on a singular transformed function that improves the performance. Rather, the ELP-like histogram supports only better performance when it acts constantly and globally during the whole run. This is only compatible to the insight that $H(q)$ improves the overall performance if applied to the whole chain of transformations of successive best values f_{best}. Therefore, $H(q)$ helps to avoid the previously criticized behavior of STUN to transform "too much" and leading to a golf-course-like landscape. Rather, $H(q)$ in the original TAAO builds regions where no golf player is allowed anymore in the future.

5 Conclusions

In the spirit of previous work on hybrid metaheuristics [1,10,32], we have combined two distinct randomized, Monte-Carlo inspired optimization schemes to synergistically improve the search dynamics. We call this new scheme Tunneling-And-Avoidance-Optimization (TAAO).

Based on empirical evidence from a very hard optimization problem – the ground state of Ising spin glasses – we have shown that TAAO performs better than its isolated ingredients (STUN and ELP) which are related to tabu search. We were also able to show that previously proposed mechanism to quantify the search dynamics, namely the auto-correlation function of visited function values, are applicable and also show the superior performance of TAAO. In our application, the exponentially distributed local minima values lead to the conclusion that TAAO has an exponential better performance than STUN alone.

Upon further investigation, we were able to show that the region-tabu property of the ELP mechanism alleviates STUN's saturation phenomena after long runs.

Our test instance for a combinatorial optimization problem — the Ising spin glass of Eq. 4 — shows exponential slowing down for any randomized search dynamics and for Metropolis-based optimization procedure in particular [4]. Furthermore, Ising spin glass optimizations were shown to belong to the class of NP hard problems [2]. These properties make it good candidate to assess the performance of a newly proposed metaheuristics.

Note, however, that an optimization procedure is *never* applicable to *all* optimization problems: the strictly proven "no-free-lunch" theorem by Wolpert and Macready [33] states that all optimization procedures will — averaged over all optimization problems — show the exact same performance. Therefore, an improved performance of *any* metaheuristics can only be shown for a single optimization problem. In this sense, the superior performance of TAAO in the Ising

spin glass instance implies inferior performance for other problems — as is the case for all optimization schemes.

Acknowledgments. The author gratefully acknowledge the financial support from the Deutsche Forschungsgemeinschaft (DFG) Graduiertenkolleg 1657 (Projekt 3C) at the Technische Universität Darmstadt.

References

1. Arito, F., Leguizamón, G.: Incorporating tabu search principles into aco algorithms. In: Blesa, et al. (eds.) [5], pp. 130–140
2. Barahona, F.: On the computational complexity of ising spin glass models. Journal of Physics A: Mathematical and General 15(10), 3241 (1982), http://stacks.iop.org/0305-4470/15/i=10/a=028
3. Bentner, J., Bauer, G., Obermair, G.M., Morgenstern, I., Schneider, J.: Optimization of the time-dependent traveling salesman problem with monte carlo methods. Phys. Rev. E 64, 036701 (2001)
4. Binder, K., Young, A.: Spin glasses: Experimental facts, theoretical concepts, and open questions. Rev. Mod. Phys. 58(4), 801–976 (1986)
5. Blesa, M.J., Blum, C., Di Gaspero, L., Roli, A., Sampels, M., Schaerf, A. (eds.): HM 2009. LNCS, vol. 5818. Springer, Heidelberg (2009)
6. Chaves, A.A., Lorena, L.A.N., Miralles, C.: Hybrid metaheuristic for the assembly line worker assignment and balancing problem. In: Blesa, et al. (eds.) [5], pp. 1–14
7. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Searching in metric spaces. ACM Comput. Surv. 33(3), 273–321 (2001)
8. Doerner, K.F., Schmid, V.: Survey: Matheuristics for rich vehicle routing problems. In: Blesa, M.J., Blum, C., Raidl, G., Roli, A., Sampels, M. (eds.) HM 2010. LNCS, vol. 6373, pp. 206–221. Springer, Heidelberg (2010)
9. Doye, J.P.K., Wales, D.J.: Thermodynamics of global optimization. Phys. Rev. Lett. 80(7), 1357–1360 (1998)
10. Fernandes, S., Lourenço, H.R.: Optimised search heuristic combining valid inequalities and tabu search. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) HM 2008. LNCS, vol. 5296, pp. 87–101. Springer, Heidelberg (2008)
11. Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. 13, 533–549 (1986)
12. Glover, F., Laguna, M.: Tabu Search. Kluwer, Dordrecht (1997)
13. Hamacher, K.: Adaptation in stochastic tunneling global optimization of complex potential energy landscapes. Europhys. Lett. 74(6), 944–950 (2006)
14. Hamacher, K.: Energy landscape paving as a perfect optimization approach under detrended fluctuation analysis. Physica A 378(2), 307–314 (2007)
15. Hansmann, U., Wille, L.T.: Global Optimization by Energy Landscape Paving. Phys. Rev. Lett. 88(23), 068105 (2002)
16. Ingber, L.: Simulated annealing: Practice versus theory. Mathematical and Computer Modelling 18(11), 29–57 (1993), http://www.sciencedirect.com/science/article/pii/089571779390204C
17. Kirkpatrick, S., Gelatt, C., Vecchi, M.: Optimization by simulated annealing. Science 220, 671–680 (1983)

18. Klotz, T., Schubert, S., Hoffmann, K.: The state space of short-range Ising spin glasses: the density of states. The European Physical Journal B-Condensed Matter and Complex Systems 2(3), 313–317 (1998)
19. Liwo, A., Lee, J., Ripoll, D.R., Pillardy, J., Scheraga, H.A.: Protein structure prediction by global optimization of a potential energy function. PNAS 96(10), 5482–5485 (1999)
20. Maringer, D., Parpas, P.: Global optimization of higher order moments in portfolio selection. J. Glob. Opt. 43, 219–230 (2009)
21. Mertens, S.: Random Costs in Combinatorial Optimization. Phys. Rev. Lett. 84(6), 1347–1350 (2000)
22. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. Chem. Phys. 21(6), 1087–1092 (1953)
23. Middleton, A.A.: Improved extremal optimization for the ising spin glass. Physical Review E (Statistical, Nonlinear, and Soft Matter Physics) 69(5), 055701 (2004), http://link.aps.org/abstract/PRE/v69/e055701
24. Munakata, T., Nakamura, Y.: Temperature control for simulated annealing. Phys. Rev. E 64(4), 046127 (2001)
25. Nayeem, A., Vila, J., Scheraga, H.A.: A comparative study of the simulated-annealing and monte carlo-with- minimization approaches to the minimum-energy structures of polypeptides: [met]-enkephalin. J. Comp. Chem. 12(5), 594–605 (1991)
26. Notay, Y.: Flexible conjugate gradients. SIAM Journal on Scientific Computing 22(4), 1444–1460 (2000)
27. Prügel-Bennett, A., Shapiro, J.L.: Analysis of genetic algorithms using statistical mechanics. Phys. Rev. Lett. 72(9), 1305–1309 (1994)
28. Schug, A., Wenzel, W., Hansmann, U.: Energy landscape paving simulations of the trp-cage protein. J. Chem. Phys. 122, 194711 (2005)
29. Sexton, R.S., Dorsey, R.E., Johnson, J.D.: Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation. Decision Support Systems 22(2), 171–185 (1998)
30. Simone, C., Diehl, M., Jünger, M., Mutzel, P., Reinelt, G.: Exact ground states of ising spin glasses: New experimental results with a branch-and-cut algorithm. J. Stat. Phys. 80, 487 (1995)
31. Wales, D.J., Scheraga, H.A.: Global Optimization of Clusters, Crystals, and Biomolecules. Science 285(5432), 1368–1372 (1999)
32. Walshaw, C.: Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In: Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M. (eds.) Hybrid Metaheuristics. SCI, vol. 114, pp. 261–289. Springer, Heidelberg (2008)
33. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997), citeseer.ist.psu.edu/wolpert96no.html

Workgroups Diversity Maximization: A Metaheuristic Approach

Marco Caserta[1] and Stefan Voß[2]

[1] IE Business School, Maria de Molina 12, 28006, Madrid, Spain
mcaserta@faculty.ie.edu
[2] University of Hamburg, Von-Melle-Park 5, 20146, Hamburg, Germany
stefan.voss@uni-hamburg.de

Abstract. Workgroup assignment problems commonly appear in various settings including international business schools. Especially if diverse people, like students, need to be divided into workgroups one may seek environments where diversity is fostered by generating heterogeneous workgroups. We study a problem of workgroups diversity maximization, i.e., the problem of building workgroups with the goal of maximizing intra-group diversity, while minimizing inter-group heterogeneity. For solving this problem with different objectives we propose a hybrid metaheuristic approach which combines local search techniques with a population based metaheuristic, including the cross entropy method as well as path relinking as ingredients. Numerical results are presented on some real-world instances.

1 Introduction

In this paper, we study the problem of creating workgroups with the aim of maximizing intra-group diversity, while minimizing inter-group heterogeneity. Let us consider an organization whose workforce is composed of a large body of workers with a diverse set of skills. Each worker is classified along a set of dimensions, *e.g.*, age, gender, native tongue, educational background, past experience, etc. The goal of the problem is to create teams maximizing diversity within each group, while making the set of teams as homogeneous as possible. In this context, the word "diversity" refers to differences in a range of qualities and characteristics among individuals. On the other hand, a set of groups is homogeneous if the overall set of characteristics of the members of each workgroup is similar, thus distributing people with similar characteristics as much as possible over the workgroups.

This study is motivated by the workgroup assignment problem commonly addressed in international business schools. Typically, business schools are attended by an extremely diverse body of students. Such students are divided in class sections and, within each section, in workgroups. To expose students to a richer experience, business schools attempt to create an environment in which diversity is fostered by generating sections and workgroups with heterogeneity in mind.

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 118–129, 2013.

Although the main application of the problem presented in this paper is the workgroup assignment problem faced by business schools, a number of authors have pointed out that similar problems are encountered in other realms of application, *e.g.*, when assigning employees to project teams, work packets (tours, routes, etc.) to workers, when scheduling final exams at universities, or in the VLSI design.

The workgroup assignment problem has been object of study for over two decades. Previous works on this problem in more or less different settings can be found, e.g., in [2,11,7,8,12,1,5,6]. To clarify those different settings, let us consider two among those papers. First, the seminal paper [11] which presents a decision support system designed to address the students assignment problem. The authors thoroughly examined the problem in the format arising at one of the major European business schools and they proposed a constructive heuristic approach for the assignment of students to class sections and, within each section, to workgroups. The guiding criterion was a measure of similarity, *i.e.*, they iteratively assigned students in such a way that similar students were placed in different sections and workgroups. Secondly, [12] provided a comparison of five different heuristic rules for the creation of maximally diverse groups. The authors compared and contrasted one constructive method arising from the students workgroup assignment problem and four switching methods drawn from the final exam scheduling problem. All methods were driven by the same objective function, *i.e.*, a measure of overall diversity of the resulting partitioning. In order to have a measure of the goodness of the final results, they also proposed an integer bound.

In this paper, we present a hybrid metaheuristic approach for the workgroup diversity problem that combines local search techniques with a population based metaheuristic. The major contributions of this paper are:

- We introduce a set of "hard" constraints that limit the way in which teams are created by preventing individuals with certain attributes to belong to the same team. In the context of the student assignment problem, it could be the case that students that were together in the same teams in previous semesters are not allowed to be assigned to the same team in subsequent semesters. Such limitations actually change the problem itself, since it is no longer possible to ensure that instances of such problem have at least a feasible solution.
- We consider the "general" assignment problem, in which class sections do not need to be of the same size due to, *e.g.*, limitations in rooms availability. In line with that, we develop a set of fitness functions that take into account the relative size of each group.

2 A Mathematical Model for the Workgroup Diversity Problem

The problem studied in this paper can be seen as belonging to the class of assignment problems, in which a (larger) set of entities, *e.g.*, students, is assigned

to a (smaller) set of tasks, *e.g.*, class sections. The objective function of the assignment problem is defined in such a way that maximal diversity is achieved. Such objective function can be expressed in a number of ways, measuring, *e.g.*, the distance among students, the variance of the assignment, etc.

Due to the original motivation behind this work, in the following, we will use the terms students to refer to the entities to be assigned, and, interchangeably, teams or sections to identify the groups students are assigned to. Let us assume we are given a pool of n students, each described by m attributes. Such students must be assigned to a set of K teams. The basic information about each student $i = 1, \ldots, n$ is collected via a matrix $A = \{a_{ij}\}$, where $a_{ij} = 1$ indicates that student i has characteristic, or attribute, j, while $a_{ij} = 0$ indicates that student i does not have attribute j. Without loss of generality, we assume that each attribute is of binary nature, given that, whenever an ordinal attribute is given, this can always be transformed in a set of mutually exclusive binary attributes. For example, let us assume that we use $a'_{iw} \in \{1, \ldots, 10\}$ to indicate the value of a given nominal attribute w, where the attribute can only take values in the set $\{1, \ldots, 10\}$. It is always possible to express such nominal attribute as a collection of ten binary attributes $a_{ij} \in \{0, 1\}$, with $j = 1, \ldots, 10$, with the additional constraints that $\sum_{j=1}^{10} a_{ij} = 1$ for every student. Thus, in the following, we assume that all nominal attributes have been transformed into a set of corresponding binary attributes.

Given the set of attributes $\mathcal{A} = \{1, \ldots, m\}$, we partition such set into $\mathcal{A} = \mathcal{A}_d \cup \mathcal{A}_f$, where A_d is the set of desirable attributes, *i.e.*, students with these attributes should be dispersed over different teams or sections as much as possible, while A_f is the set of forbidden attributes, *i.e.*, students with these attributes cannot be assigned to the same team or section. In a similar fashion, the attributes' matrix A can also be partitioned into two submatrices A_d and A_f, in such a way that A_d is a matrix of size $n \times |\mathcal{A}_d|$ that contains all the values of the desirable attributes, while A_f is a matrix of size $n \times |\mathcal{A}_f|$ containing all the values of the forbidden attributes.

The following set of constraints can be used to model the workgroup assignment problem:

$$\sum_{k=1}^{K} x_{ik} = 1, \quad i = 1, \ldots, n \tag{1}$$

$$\sum_{i=1}^{n} x_{ik} \geq L_k, \quad k = 1, \ldots, K \tag{2}$$

$$\sum_{i=1}^{n} x_{ik} \leq U_k, \quad k = 1, \ldots, K \tag{3}$$

$$\sum_{i=1}^{n} a_{ij} x_{ik} \leq 1, \quad k = 1, \ldots, K, \quad j \in \mathcal{A}_f \tag{4}$$

$$x_{ik} \in \{0, 1\}, \quad i = 1, \ldots, n, \quad k = 1, \ldots, K \tag{5}$$

where x_{ik} is a binary decision variables, whose value equal to 1 indicates that student i is assigned to team k. Constraints (1) account for the fact that each student must be assigned to exactly one team; constraints (2)–(3) define the minimum (L_k) and maximum (U_k) number of students assigned to each team, while constraints (4) ensure that no two students with the same "forbidden" attribute can be assigned to the same team. It is worth noting that, due to constraints (4), we cannot even ensure that the problem has a feasible solution.

With respect to the objective function, we follow an approach that resembles that of [1], in which a number of alternative objective functions were proposed and tested. The peculiarity of our study is that, contrary to what is assumed by previous works, we do not assume that the team sizes must be the same. Therefore, we adjust the metric used to define the different objective functions accordingly.

Z_1: Proportional Entropy. This objective function is derived from information theory concepts and resembles the one introduced in [7]. Let us define

$$p_{kj} = \frac{\sum_i a_{ij} x_{ik}}{\sum_i x_{ik}}$$

the proportion, *i.e.*, percentage, of students assigned to team k that enjoy attribute j, for each team and attribute. The objective function is, thus:

$$\max Z_1 = \sum_{k=1}^{K} \sum_{j=1}^{m} -p_{kj} \ln p_{kj} \tag{6}$$

In Equation (6), we assume that the product $p \ln p$ is set to zero whenever the corresponding p is equal to zero. The difference between the proposed Z_1 measure and the one presented, *e.g.*, in [7], is that the proposed measure takes into account the relative size of the team, with respect to the overall population. In other words, to maximize function Z_1, students will be distributed over teams taking into account the size of each team.

Z_2: Total Proportional Deviation. This measure is a variation of the Total Absolute Deviation measure presented in [7]. Let us first define

$$\overline{p}_j = \frac{1}{K} \sum_{k=1}^{K} \frac{\sum_i a_{ij} x_{ik}}{\sum_i x_{ik}}$$

the average percentage of students with attribute j in each group. We thus define the following objective function:

$$\min Z_2 = \sum_{k=1}^{K} \sum_{j=1}^{m} \left| \frac{\sum_i a_{ij} x_{ik}}{\sum_i x_{ik}} - \overline{p}_j \right| \tag{7}$$

Equation (7) computes the total deviation in percentages with respect to each team and attribute. Again, the main advantage of this measure is that it takes

into account the size of each team and, therefore, provides a measure of the percentage of students within each group enjoying a certain attribute. The goal here is to minimize the deviation of each group percentage from the average percentage, in such a way that each group has approximately the same percentage of students with a given attribute.

Z_3: Total Absolute Deviation. This measure is the Total Absolute Deviation measure presented in [7], except for the fact that we take into account the relative size of each workgroup within the total population. We thus define the following objective function:

$$\min Z_3 = \sum_{k=1}^{K} \sum_{j=1}^{m} \left| \sum_i a_{ij} x_{ik} - \sum_i a_{ij} \frac{\sum_l x_{lk}}{n} \right| \tag{8}$$

Equation (8) provides a measure of deviation between the real (first term in Equation (8)) and theoretically optimal (second term in Equation (8)) number of students in a group with a given attribute. The theoretically optimal number of students with a given attribute in a team is computed relative to the size of the group itself. Thus, the larger the group, the larger the number of students with an attribute in that team.

The major difference between Equation (7) and Equation (8) lies in the unit measure: While Equation (8) is expressed in number of people, Equation (7) is a relative measure and is expressed as percentage. The inherent advantage of Equation (7) is that, if we divided Z_2 by $m \times K$, we would get a standardized value, *i.e.*, Z_2 would take values between 0 and 1.

Z_4: Pairwise Distance. This measure is a modified version of the Z_5 presented in [7]. Here again we take into account the number of students in each team. The following objective function is thus defined:

$$\min Z_4 = \sum_{j=1}^{m} \sum_{k=1}^{K-1} \sum_{l=k+1}^{K} \left| \frac{\sum_i a_{ij} x_{ik}}{\sum_i x_{ik}} - \frac{\sum_i a_{ij} x_{il}}{\sum_i x_{il}} \right| \tag{9}$$

Equation (9) accounts for the pairwise difference in the number of students with a given attribute between any two groups. In the equation, the first term accounts for the number of students with attribute j in team k (relative to the size of that group), while the second term computes the number of students with that same attribute j within group l (again weighted with respect to the size of that group). By minimizing the total sum of pairwise differences, we are aiming at minimizing the inter-groups difference.

3 A Pool-Based Metaheuristic Algorithm

In this section, we present the relevant features of the proposed algorithm. The algorithm is composed of four different steps, presented in Figure 1.

In the initialization phase, we define the pool size $|\Omega|$ and the insertion criterion `ic`. With respect to the proposed approach, we define $|\Omega| = 10$ while the insertion criterion `ic` is related to the fitness value, *i.e.*, a solution is inserted into the pool if its fitness value is better than the fitness value of the worst solution currently in Ω.

The second step of the algorithm defines a population-based metaheuristic with the aim of generating a variety of solutions. The best solutions found during this phase are inserted into Ω. We use a cross entropy scheme to pupulate Ω; details are provided below.

Once the pool Ω has been populated, we make an attempt to improve the quality of the solutions in the pool by means of a simple nested neighborhood search. We iteratively select a solution from the pool and we perform all the 2-opt exchanges, using the steepest ascent method. In other words, given the current solution, we try out all the feasible 2-opt exchanges and we select the one that generates the maximum improvement in the fitness value. The 2-opt scheme stops when no further improvement can be obtained with an exchange.

When the 2-opt scheme reaches a local optimum, we perform a 3-opt exchange using the steepest ascent method. Once again, the 3-opt scheme stops when a local optimum is reached.

Finally, the last step of the algorithm implements a path relinking approach, in which a trajectory leading from an incumbent solution $\mathbf{x}^l$ to a target solution $\mathbf{x}^t$ is defined. At each step, the *distance* between incumbent and target solutions is reduced by executing a 2-opt swap over $\mathbf{x}^l$. Given the two solutions $\mathbf{x}^l = \left\{x^l_{ik}\right\}$ and $\mathbf{x}^t = \{x^t_{ik}\}$, we define the distance between the two as a hamming distance:

$$H\left(\mathbf{x}^l, \mathbf{x}^t\right) = \sum_{i=1}^{n} \sum_{k=1}^{K} |x^l_{ik} - x^t_{ik}| \tag{10}$$

At each iteration of the path relinking, we select the 2-opt swap that, while reducing the hamming distance (10) by at least one, maximizes the improvement in the fitness value of the newly obtained solution.

Let us now present the details of the Cross Entropy (CE) scheme used in Step 2. (See [9] and [4] for a tutorial and a comprehensive overview of the CE. See also [3] for an application and fine-tuning technique for CE.) The assignment of a student to a team can be seen as a stochastic process governed by a set of probabilities. Let us suppose we are given an $n \times K$ probability matrix $P = \{p_{ik}\}$, where each term $p_{ik} \in [0, 1]$ represents the probability of assigning student i to team k. Matrix P is a stochastic matrix, since the sum of the probabilities per row is equal to one, *i.e.*, $\sum_k p_{ik} = 1$. Given such probability matrix P, one could generate an assignment of students to teams.

In order to ensure the feasibility of the generated solution, the following rules should be taken into account:

(R_1) Each student i should be assigned to a single team, as imposed by Constraints (1). Therefore, whenever a student i is assigned to a team k, we set $p_{ik} = 1$ and $p_{iw} = 0$, for all $w \neq k$.

S1 : Initialize Pool.
- Setup data structure to collect solutions into solution pool Ω.
- Define pool size $|\Omega|$ and insertion criterion `ic`.

S2 : Populate Pool.
- Apply the Cross Entropy scheme to populate the pool Ω.
- Define CE population size N, quantile ratio ρ, and smoothing factor α using Response Surface Methodology.
- Run the CE algorithm while stopping criteria `sc` are not verified and add solution to Ω if `ic` is satisfied.

S3 : Local Search.
- Apply a 2-opt and 3-opt schemes to improve the quality of the solutions in Ω.
- For each solution in Ω, apply a 2-opt mechanism using steepest-ascent, *i.e.*, as long as the current solution can be improved with a 2-opt exchange.
- Once the current solution can no longer be improved with a 2-opt exchange, apply a 3-opt exchange using steepest-ascent.

S4 : Path Relinking.
- Apply a Path Relinking scheme using all the solutions from the pool Ω.
- Set as *target solution* the current best solution found so far, *i.e.*, $\mathbf{x}^t$.
- For each solution $\mathbf{x}^l \in \Omega$, transform x^i into $\mathbf{x}^t$ via 2-opt swaps.
- If a new best solution is visited, save the new best solution.

Fig. 1. A Pool-based Metaheuristic Algorithm

(R_2) Each team k should have a minimum of L_k and a maximum of U_k students, as indicated by Constraints (2) and (3). Therefore, as long as the minimum number of students per team is not reached, we need to ensure that there still exists a positive probability of assigning students to that team, *i.e.*, $\sum_i p_{ik} > 0$. Conversely, once the maximum number of students per team has been reached, we need to set the probability of assigning further students to that team to zero, *i.e.*, $\sum_i p_{ik} = 0$.

(R_3) As imposed by Constraints (4), students with the same attribute value for attributes in the set of forbidden attributes $\mathcal{A}_f$ should not be assigned to

the same team. We decided to treat this set of hard constraints as "soft" constraints, by penalizing, in the fitness functions, assignments for which Constraints (4) were not satisfied. In other words, Constraints (4) were relaxed in a Lagrangean fashion and appropriate values for the multipliers were determined, to penalize violations of any of the Constraints (4).

To exploit the stochastic nature of the proposed approach, we generate a population of N assignments, *e.g.*, $\mathbf{x}^1, \ldots, \mathbf{x}^w, \ldots, \mathbf{x}^N$, drawn under the probability matrix P. Next, using the basic idea of the CE, we use the "Maximum Likelihood Estimator" method to revise the probability matrix and to generate a new matrix P^1 that better reflects the best individuals within the current population. Thus, we adjust the current probability values p_{ik} to reflect how likely it is that student i is assigned to team k in a high-quality solution. Once we obtain the new probability matrix P^1, we draw a new population of size N. Hopefully, such matrix better describes high quality solutions obtained in the previous generation and, therefore, the chance of obtaining high quality permutations based upon the new matrix is higher.

This process of "probability matrix update" and "population generation" can be iterated until a stopping criterion `sc` is reached, *i.e.*, either the P matrix converges to a binary matrix (therefore, the process converged to a unique solution in the solution space) or a pre-specified maximum number of iterations, say 30 (see Section 4), has been reached.

The "Maximum Likelihood Estimator" method is used to modify the probabilities p_{ik} in such a way that the new stochastic matrix better reflects the chance of obtaining high quality solutions. Let us assume that, based upon the current stochastic matrix P^t, we have generated a population of size N, *i.e.*, assignments $\mathbf{x}^1, \ldots, \mathbf{x}^N$. Let us now find, within the current population, the objective function value of the $(1-\rho)\%$ quantile, *i.e.*, the value γ for which $\rho\%$ of the population has a better objective function value and $(1-\rho)\%$ has a worse objective function value.

We modify the transition probability matrix using the following updating rule:

$$\hat{p}_{ik} = \frac{\sum_{w=1}^{N} I_{\{\mathbf{x}^w : x_{ik}^w = 1\}} \times I_{\{f(\mathbf{x}^w) \geq \gamma\}}}{\rho N} \tag{11}$$

where $f(\mathbf{x}^w)$ is the fitness value of assignment $\mathbf{x}^w$, and $I_{\{\bullet\}}$ is the indicator function, whose value is 1 if condition $\bullet$ is true, and 0 otherwise. Therefore, we use the following two indicator functions:

$$I_{\{\mathbf{x}^w : x_{ik}^w = 1\}} = \begin{cases} 1 & \text{if student } i \text{ is assigned to team } k \text{ in assignment } \mathbf{x}^w, \\ 0 & \text{otherwise;} \end{cases}$$

$$I_{\{f(\mathbf{x}^w) \geq \gamma\}} = \begin{cases} 1 & \text{if } f(\mathbf{x}^w) \geq \gamma, \\ 0 & \text{otherwise.} \end{cases}$$

Remark. As pointed out by [4], in order to prevent the CE from converging too fast to a suboptimal solution, a *smoothing factor* α (typically $0.7 \leq \alpha \leq 0.9$) could be used in the updating rule. Therefore, to foster a more thorough exploration of the solution space, at each iteration t we use the following updating rule:

$$p_{ik}^{t+1} = \alpha \hat{p}_{ik} + (1 - \alpha) p_{ik}^{t}. \tag{12}$$

4 Computational Results and Statistical Analysis

In this section, we summarize the results obtained by the proposed algorithm on real-world instances obtained by IE Business School. We will present how the algorithm performs when used on six large instances derived from the MBA program at IE Business School, Madrid, Spain. Those instances are taken from different semesters, and belong to the international MBA program as well as the Spanish MBA program. The fact that instances belong to different programs allow to test how the algorithm performs when dealing with students with different profiles. The size of each instance is characterized by two values: The number of students n, and the number of teams to be formed m. The testbed is composed of instances whose size spans from $n = 316$ and $m = 10$ (the largest instance) to $n = 57$ and $m = 9$ (the smallest one).

The algorithm proposed in this paper was coded in C++ and compiled using the GNU g++ 4.5.2 compiler on a dual core Pentium 1.8GHz Linux workstation with 4Gb of RAM. Throughout the computational experiment we kept the pool size $|\Omega|$ constant to ten, while the values of the CE parameters, *i.e.*, N, ρ, and α were determined using the Response Surface Methodology, as illustrated in [3]. The maximum number of iterations of the CE was kept constant to 30 throughout the computational experiment phase.

Since each metric is expressed using a different unit measure, a comparison of fitness values among the functions is not really meaningful. Therefore, we decided to test the behaviour of each function with respect to the others. In other words, we wanted to know to which extent the solution found using a given fitness function was able to produce good values for the other fitness functions. If, for example, fitness function Z_1 produces solutions that are of good quality not only when evaluated with respect to Z_1 but also when evaluated using $Z_2, \ldots, Z_4$, we can claim that the fitness function Z_1 is *robust*.

Therefore, to estimate the robustness of a fitness function, we solved each instance of the problem with each function. Next, for each obtained solution, we computed the fitness value using all the functions and we determined the ranking of these solutions with respect to the same function. One might argue about the use of the term robustness in our context as there are various other options to define robustness (see, *e.g.*, [10]). In different words we could also investigate possible correlations between different functions.

As an example, let us consider the case of instance `A1.3-2012`. The table below summarizes the results. In Table 1, each row corresponds to an execution of the

algorithm using the corresponding Z_i function. For example, when Z_1 was used as objective function of problem (1)–(5), the algorithm found a solution, *i.e.*, $\mathbf{x}^1$, whose objective function value, computed using Z_1, was $Z_1(\mathbf{x}^1) = 181.321$. However, when the same solution was evaluated using fitness function Z_2, we obtained $Z_2(\mathbf{x}^1) = 89.5733$. Similarly, we got $Z_3(\mathbf{x}^1) = 537.44$, and $Z_4(\mathbf{x}^1) = 452.867$.

We next ran the algorithm using fitness function Z_2. Let us indicate the best solution obtained by the algorithm with $\mathbf{x}^2$. The second row of the table provides the values of $Z_i(\mathbf{x}^2)$. The same approach was repeated using Z_3 and Z_4 as objective functions, as presented in Table 1.

Table 1. Objective function values of the different fitness functions on instance `A1.3-2012`. Arrows indicate whether a function is to be maximized (Z_1) or minimized (Z_2, Z_3, Z_4).

Fitness Function Used	Functions Evaluation			
	$\uparrow Z_1$	$\downarrow Z_2$	$\downarrow Z_3$	$\downarrow Z_4$
Z_1	181.321	89.5733	537.44	452.867
Z_2	180.137	89.6233	537.74	453.867
Z_3	179.794	89.5467	537.28	451.833
Z_4	180.289	89.4300	636.58	452.233

Using Table 1, we can provide some information about the robustness (as well as possible correlations) of a given fitness function. We now want to rank, columnwise, solutions $\mathbf{x}^1, \ldots, \mathbf{x}^4$, assigning a score of 1 to the best solution and 4 to the worst one. For example, in column Z_1, we observe that the best solution is $\mathbf{x}^1$, followed by $\mathbf{x}^4$, $\mathbf{x}^2$, and $\mathbf{x}^3$. A similar process for each column of Table 1 leads to the creating of the ranking, as presented in Table 2.

Table 2. Ranking of the different fitness functions on instance `A1.3-2012`. Arrows indicate whether a function is to be maximized (Z_1) or minimized (Z_2, Z_3, Z_4).

Fitness Function Used	Functions Ranking				Avg
	$\uparrow Z_1$	$\downarrow Z_2$	$\downarrow Z_3$	$\downarrow Z_4$	
Z_1	1	3	2	3	2.5
Z_2	3	4	3	4	3.75
Z_3	4	2	1	1	2.25
Z_4	2	1	4	2	1.5

Let us now present the computational results obtained over six real-world instances. Table 3 presents the results, in terms of ranking, over these instances. In the table, column one provides the name of the instance, while columns two and three specify the number of students and the number of teams. Columns

four to seven provide the average rank value of each fitness function. The rank value is computed as presented in Tables 1 and 2. Values presented in the table are averaged over 10 runs per instance and fitness function. Therefore, a total of $6 \times 10 \times 4 = 240$ runs and $240 \times 4 = 960$ function evaluations have been carried out to fill out Table 3.

Table 3. Computational results on real-world instances. Ranking is computed cross-evaluating each solution using all the fitness functions. Each instance is solved ten times using the same fitness function. The table contains a total of 240 runs and 960 evaluations.

Name	n	m	Z_1	Z_2	Z_3	Z_4
A1.1-2012	60	9	1.97	2.92	2.31	2.79
F2012a	140	3	2.67	2.77	3.24	2.89
F2012b	80	2	2.75	2.77	3.24	2.95
sA4-2012	57	9	2.23	2.16	2.35	3.25
A1.3-2012	60	9	2.14	2.76	2.37	2.71
F2011	316	10	2.92	3.1	3.15	3.12

From Table 3 we evince that fitness function Z_1 is the most robust, since it produces higher ranking, *i.e.*, higher quality solutions. Interestingly, the user of the algorithm, when called to select different solutions obtained using different fitness functions, expressed a clear preference for the solutions generated by fitness function Z_1. Thus, the empirical evidence obtained using the ranking table and the preference of the user seem to be aligned.

As a final remark, it is worth noting that the best solution found by the algorithm was found either in step 3 (73% of the time) or in step 4 (21% of the time). The remaining 7% of the time, the cross entropy scheme, *i.e.*, step 2 of the algorithm, produced a solution that was not improved by steps 3 and 4.

5 Conclusions and Future Work

In this paper, we presented a model and an hybrid algorithm for the workgroup diversity maximization problem. This problem aims at finding an assignment of workers to groups that minimizes the inter-group heterogeneity while maximizing the intra-group diversity. The problem finds application in different realms, spanning from the business schools assignment problem to the VLSI design.

The solution approach proposed in the paper is hybrid in nature, where local search techniques are intertwined with a population-based method. The algorithm has been tested on six real-world instances provided by a business school. The size of the instances varies, along with the number of teams to be created. The testbed has been used to determine the robustness of different fitness functions in terms of solution quality (which also allows to investigate possible correlations between the functions). The computational results collected from

the testbed are in line with the preferences expressed by the user. Both are in accordance in identifying one fitness function as superior compared with the others.

Future studies should be focused on a few interesting extensions: (i) generating a valid bound to objectively determine the quality of a solution; and (ii) using statistical analysis to rigorously assert whether the proposed fitness functions present statistically significant differences (and, therefore, to create a robust ranking of such criteria).

As our problem definition incorporates different ideas compared to how this type of problem is modeled or operationalized regarding possible objectives and constraints, it is difficult to directly compare the various problems and related solution methods. In future research we would also be interested to compare the proposed concepts once transfered between problem settings. Moreover, a valid extension would be to combine the first fitness function with one of the other functions using a bi-level programming approach.

References

1. Baker, K.R., Powell, S.G.: Methods for Assigning Students to Groups: A Study of Alternative Objective Functions. Journal of the Operational Research Society 53(4), 397–404 (2002)
2. Beheshtian-Ardekani, M., Mahmood, M.A.: Development and Validation of a Tool for Assigning Students to Groups for Class Projects. Decision Sciences 17(1), 92–113 (1986)
3. Caserta, M., Quiñonez, E.: A Cross Entropy-Lagrangean Hybrid Algorithm for the Multi-item Capacitated Lot-sizing Problem with Setup Times. Computers & Operations Research 36(2), 530–548 (2009)
4. De Boer, P., Kroese, D.P., Mannor, S., Rubinstein, R.Y.: A Tutorial on the Cross-Entropy Method. Annals of Operations Research 134, 19–67 (2005)
5. Desrosiers, J., Mladenovic, N., Villeneuve, D.: Design of Balanced MBA Student Teams. Journal of the Operational Research Society 56(1), 60–66 (2005)
6. Fan, Z.P., Chen, Y., Zeng, S.: A Hybrid Genetic Algorthmic Approach to the Maximally Diverse Grouping Problem. Journal of the Operational Research Society 62(7), 1423–1430 (2011)
7. Mingers, J., O'Brien, F.A.: Creating Students Groups with Similar Characteristics: A Heuristic Approach. Omega 23(3), 313–321 (1995)
8. O'Brien, F.A., Mingers, J.: A Heuristic Algorithm for the Equitable Partitioning Problem. Omega 25(2), 215–223 (1997)
9. Rubinstein, R.Y., Kroese, D.P.: The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte Carlo Simulation, and Machine Learning. Springer, Berlin (2004)
10. Scholl, A.: Robuste Planung und Optimierung: Grundlagen - Konzepte und Methoden - Experimentelle Untersuchungen. Physica, Heidelberg (2004)
11. Weitz, R.R., Jelassi, M.T.: Assigning Students to Groups: A Multi-Criteria Decision Support System Approach. Decision Sciences 23(3), 746–757 (1992)
12. Weitz, R.R., Lakshminarayanan, S.: An Empirical Comparison of Heuristic Methods for Creating Maximally Diverse Groups. Journal of the Operational Research Society 49(6), 635–646 (1998)

Balancing Bicycle Sharing Systems: Improving a VNS by Efficiently Determining Optimal Loading Operations

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

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

Abstract. Public bike sharing systems are important alternatives to motorized individual traffic and are gaining popularity in larger cities worldwide. In order to maintain user satisfaction, operators need to actively rebalance the systems so that there are enough bikes available for rental as well as sufficient free slots for returning them at each station. This is done by a vehicle fleet that moves bikes among the stations. In a previous work we presented a variable neighborhood search metaheuristic for finding effective vehicle routes and three different auxiliary procedures to calculate loading operations for each candidate solution. For the most flexible auxiliary procedure based on LP, the current work provides a new, practically more efficient method for calculating proven optimal loading operations based on two maximum flow computations. The different strategies for determining loading operations are further applied in combination controlled by an additional neighborhood structure. Experimental results indicate that this combined approach yields significantly better results than the original variable neighborhood search.

1 Introduction

Public bicycle sharing systems are booming worldwide in many major cities as they augment public transport very well [1,2]. Modern systems have automated rental stations where users can easily rent bikes and return them elsewhere. In order to achieve a high degree of acceptance, operators need to actively rebalance the system in order to ensure that there are enough bikes as well as parking slots for returning them at any station at almost all times. This balancing is typically done by a vehicle fleet with trailers. So far, drivers mostly follow their experience and intuition when planning the transportation routes. It was not until recently that researchers have started to consider this transportation planning problem from an optimization point of view.

The *Balancing Bicycle Sharing System* (BBSS) problem is related to the well-studied vehicle routing problem (VRP). However, there are significant differences such as allowing multiple visits at stations and that an arbitrary number of bikes may be loaded or unloaded at each visit. We consider the static variant of the

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 130–143, 2013.

BBSS problem in which user activities during the rebalancing process are neglected. It can be regarded as a capacitated single commodity split pickup and delivery VRP. So far, only few algorithms have been published for the BBSS problem, and due to specific application characteristics, they address significantly different variants.

Chemla et al. [2] consider the problem with only one vehicle and achieving perfect balance as hard constraint. They describe a branch-and-cut algorithm utilizing a relaxed mixed integer linear programming (MIP) model and a tabu search for locally improving incumbent solutions. Benchimol et al. [3] also assume balancing as hard constraint and focus on approximation algorithms for selected special situations. Raviv et al. [4] propose four MIP models for different problem variants and compare their assets and drawbacks on instances with up to 60 stations. Their objective function minimizes user dissatisfaction and ignores tour lengths as well as the number of loading operations.

Contardo et al. [5] consider different MIP models for the dynamic scenario where demands need to be satisfied over time. They propose a hybrid approach using column generation and Benders decomposition that is able to handle instances with up to 100 stations.

Schuijbroek et al. [6] describe the decomposition of the problem into separate single-vehicle routing problems by solving a polynomial-size clustering problem. They apply a clustered MIP heuristic in two versions, with and without additional cuts. In addition, they present a constraint programming model that represents the problem as a scheduling problem. Results on instances of up to 135 stations and five vehicles show that the approaches outperform a MIP model operating on the full unclustered problem.

In [7], we propose a variable neighborhood search (VNS) metaheuristic for finding effective vehicle routes that employs an auxiliary algorithm for calculating meaningful loading operations for each considered candidate set of routes. Three alternatives have been studied for this auxiliary algorithm, with the most precise but also slowest one being based on linear programming (LP). While the first two methods are restricted to the so-called monotonic case, where stations may not be used as temporary buffers for the redistribution of bikes, the LP-approach is more flexible.

The current work improves upon these methods by introducing a practically significantly more efficient method for determining proven optimal loading operations for the general case based on two maximum flow computations. We also investigate a unified approach where multiple strategies for determining loading instructions are applied in combination. This is achieved by an additional neighborhood structure that determines the best suited strategy. Computational tests are performed on instances derived from real-world scenarios, indicating that the unified approach performs significantly better. However, it is also shown that in most cases the quality-loss of restricting the algorithm to monotonicity, i.e., not allowing buffering, is only small.

2 Problem Definition

Formally the BBSS problem is defined on a complete directed graph $G_0 = (V_0, A_0)$, where node set $V_0 = V \cup \{0\}$ consists of nodes for the rental stations V plus the vehicles' depot 0. Each arc $(u, v) \in A_0$ has associated a travel time $t_{u,v} > 0$ that includes a surcharge for parking and loading/unloading bikes. By $G = (V, A),\ A \subset A_0$ we denote the subgraph induced by the stations V only.

Each station $v \in V$ has associated a capacity of bikes $C_v \geq 0$, i.e., the number of available parking positions, the number of bikes it initially contains $p_v \geq 0$, and a target number of bikes it should contain after rebalancing $q_v \geq 0$. A fleet of vehicles $L = \{1, \ldots, |L|\}$ is available for transporting bikes. Each vehicle $l \in L$ has a capacity of bikes $Z_l > 0$ and starts and ends its route at the depot 0.

When serving a station, we must not only consider the traveling time given by the corresponding arc, but also the duration needed for parking the vehicle at the station and for performing loading operations. We can simply add the parking time and the time needed for an average number of loading operations to the traveling time, and therefore do not need to handle them separately anymore. In the following sections it is assumed that the traveling times have already been preprocessed in this sense.

A solution consists of two parts. The first part is the route for each vehicle $l \in L$ specified by an ordered sequence of visited stations $r_l = (r_l^1, \ldots, r_l^{\rho_l})$ with $r_l^i \in V$, $i = 1, \ldots, \rho_l$ and ρ_l representing the number of stops. Stations may be visited multiple times by the same or different vehicles. The total time t_l of a route may never exceed a given time limit $\hat{t}$. As the start and end point of each tour is the depot 0, it is not explicitly stored but assumed to be prepended and appended, respectively. The second part are the loading instructions $y_{l,v}^i, \in \{-Z_l, \ldots, Z_l\}$ with $l \in L$, $v \in V$, and $i = 1, \ldots, \rho_l$, specifying how many bikes are picked up if $y_{l,v}^i > 0$ or delivered if $y_{l,v}^i < 0$, respectively, at vehicle l's i-th stop at station v.

The following conditions must hold: The number of bikes available at each station $v \in V$ never exceeds C_v, for any vehicle $l \in L$ its capacity Z_l may never be exceeded, and the total time t_l of a tour

$$t_l = t_{0,r_l^1} + \sum_{i=2}^{\rho_l} t_{r_l^{i-1}, r_l^i} + t_{r_l^{\rho_l}, 0} \tag{1}$$

may not exceed the time limit $\hat{t}$.

Let a_v be the final number of bikes at each station $v \in V$ after rebalancing.

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

The primary objective is to minimize the deviation from the target number of bikes $\delta_v = |a_v - q_v|$ at each station $v \in V$, i.e., its *disbalance*, and secondarily we aim at minimizing the number of loading/unloading operations as well as the overall time required for all routes, i.e.,

$$\min \quad \alpha^{\text{bal}} \sum_{v \in V} \delta_v + \alpha^{\text{load}} \sum_{l \in L} \sum_{i=1}^{\rho_l} |y^i_{l,r^i_l}| + \alpha^{\text{work}} \sum_{l \in L} t_l, \tag{3}$$

where $\alpha^{\text{bal}} \gg \alpha^{\text{load}}, \alpha^{\text{work}} \geq 0$ are scaling factors controlling the relative importance of the respective terms. Note that an improvement in the balance is always considered as better than any improvements in the secondary objectives; throughout this paper we use $\alpha^{\text{bal}} = 1,\ \alpha^{\text{load}} = \alpha^{\text{work}} = 1/100\,000$. Despite this small weight, in case of equal balance the secondary objectives become important criteria to distinguish solutions and therefore must not be neglected. Otherwise, e.g. obviously unnecessary stops or loading and unloading actions may occur.

A natural simplification that may be exploited is to consider *monotonicity* regarding the fill levels of stations. Let $V_{\text{pic}} = \{v \in V \mid p_v \geq q_v\}$ denote pickup stations and $V_{\text{del}} = \{v \in V \mid p_v < q_v\}$ denote delivery stations. A vehicle is only allowed to load bikes at pickup stations and unload them at delivery stations. Depending on the excess or shortage of bikes at a station, vehicles are only allowed to load or unload bikes at it, respectively. In this way the number of bikes decreases or increases monotonically, and consequently the order in which different vehicles visit a station does not matter. Monotonicity simplifies the task of finding optimal loading instructions for a given set of routes considerably [7]. On the downside, enforcing monotonicity may exclude better solutions that e.g. use stations as buffers to temporarily store bikes or to transfer bikes between vehicles, see Fig. 1.

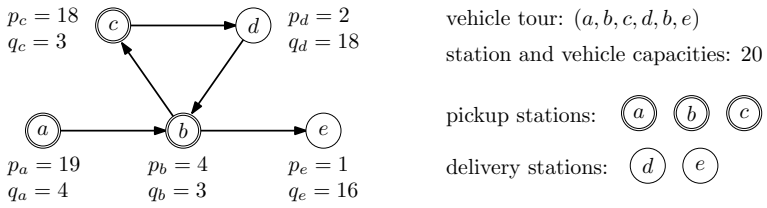


Fig. 1. Example where the restriction to monotonicity yields a worse solution. With monotonicity, the best possible loading instructions are $y_1 = (+15, +1, +4, -16, 0, -4)$ resulting in a total disbalance of 22. In the general case, node b can be used as buffer and loading instructions $y_1 = (+15, -14, +15, -16, +15, -15)$ yield perfect balance.

3 Variable Neighborhood Search for BBSS

Our metaheuristic for BBSS follows the general variable neighborhood search (VNS) principle [8] and is described in detail in [7]. An embedded Variable Neighborhood Descent (VND) is used for deterministic local improvement as intensification, while the outer VNS relies on stochastic shaking in larger neighborhoods for diversification.

An initial solution is derived by a greedy heuristic that iteratively constructs vehicle routes by always appending a feasible station for which the possible gain

in balance divided by the additional travel time is maximal. As vehicles have to terminate their routes empty, special attention is paid when considering pickup-stations: It is estimated how many bikes can still be delivered after the potential visit of a pickup-station and the possible balance gain is adjusted accordingly, i.e., the number of bikes that may be picked up is restricted correspondingly. Thus, loading instructions are set in this construction heuristic in a purely greedy way, i.e., by always picking up or delivering as many bikes such that the balance gain is locally maximized at each station visit.

In contrast, the VNS with its embedded VND searches the space of vehicle routes exploiting eleven different neighborhood structures only, and corresponding loading instructions are always derived for each considered set of routes by an auxiliary algorithm. Three alternatives have been investigated in [7]: a greedy heuristic (GH), a maximum flow approach for the monotonic case (MF-MC), and a linear programming approach for the general case (LP).

GH considers the stations in the order as they are visited in each tour and tries to bring each station as far as possible towards balance. Due to its greedy nature, the derived loading instructions do not necessarily provide the best possible overall balance and/or minimal number of loading/unloading activities. MF-MC sets up a flow network according to the given routes. By computing a maximum flow on this network, it is possible to obtain optimal loading instructions yielding the lowest achievable imbalance and a minimal number of loading operations under the assumption of monotonicity. While GH is fastest, MF-MC still is computationally very efficient, taking only about 1.8 times longer on average in our experimental evaluation. In order to overcome the monotonicity restriction, LP solves a minimum cost flow problem on a more sophisticated network via linear programming. On average our implementation of LP with CPLEX 12.4 needs about 90 times longer than MF-MC. This huge disadvantage, unfortunately, implies that the VNS can only perform substantially less iterations, and this aspect cannot be compensated by the higher quality of the loading instructions. In Section 4 we will present a new, computationally significantly more efficient approach to obtain optimal loading instructions for the general case.

The following subsections summarize the VND/VNS neighborhood structures. We employ several classical neighborhood structures that were already successfully applied in various VRPs together with new structures exploiting specifics of BBSS. Concerning the classical neighborhood structures, we primarily based our design on the experience from [9].

3.1 VND Neighborhood Structures

The following neighborhoods are applied in the given order and searched in a best improvement fashion. Preliminary experiments with a dynamic reordering strategy did not yield any significant advantage. All considered candidate tours are checked for feasibility, infeasible solutions are discarded. For each feasible solution one of the above mentioned methods for deriving loading instructions is applied. Obsolete station visits where no loading actions performed are removed from the tours.

Remove station: Considers all single station removals to avoid unnecessary visits.

Insert unbalanced station: Considers the insertion of any yet unbalanced station at any possible position.

Intra-route 2-opt: The classical 2-opt neighborhood of the traveling salesman problem applied individually to each route.

Replace station: Considers the replacement of each single station by another yet unbalanced station.

Intra or-opt: Considers all solutions in which sequences of one, two, or three consecutive stations are moved to a different place within the same route.

2-opt* inter-route exchange: Considers all feasible exchanges of arbitrarily long end segments of two routes.

Intra-route 3-opt: A restricted form of the well-known 3-opt neighborhood, individually applied to each route: For any partitioning of a route into three nonempty subsequences r_l =(a,b,c), the routes (b,a,c) and (a,c,b) are considered.

3.2 VNS Neighborhoods Structures

Shaking selects solutions randomly from the following neighborhood, which are all parameterized by δ, yielding a total of 24 specific neighborhoods. In contrast to the VND, created routes that violate the time budget are repaired by removing stations from the end. The neighborhoods are again applied in the given order, and each derived candidate solution is locally improved by the VND before deciding upon its acceptance.

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

Exchange Sequence: Exchange two randomly selected segments of length one to $\min(\delta, \rho_l)$ between two randomly chosen routes. $\delta \in \{1, \ldots, 5, \rho_l\}$.

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

Destroy and Recreate (D&R): Select a random position in a randomly chosen route, remove all nodes from this position up to the end, and recreate a new end segment by applying a randomized version of the greedy construction heuristic. The randomization is done in the typical GRASP-like way [10] with the threshold parameter set to $\delta \in \{0\%, 4\%, 8\%, 12\%, 16\%, 20\%\}$.

4 Maximum Flow Based Method for the General Case

Similarly as in [2,7], we set up a flow network, which is illustrated in Fig. 2. By $t(r_l^i)$ we denote the time when vehicle l makes its i-th stop at station r_l^i. Let

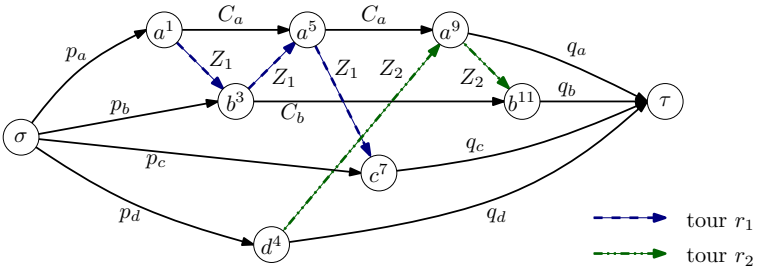


Fig. 2. Exemplary flow network for vehicle routes $r_1 = (a, b, a, c)$ and $r_2 = (d, a, b)$

$G_{\mathrm{f}} = (V_{\mathrm{f}}, A_{\mathrm{f}})$ be a directed multi-graph with node set $V_{\mathrm{f}} = \{\sigma, \tau\} \cup V_t$, where σ and τ are source and target nodes, respectively, and $V_t = \{v^j \mid v = r_l^i,\ j = t(r_l^i),\ i = 1, \ldots, \rho_l,\ l \in L\}$; i.e., we have a node v^j for each station v and time j when some vehicle makes a stop at v. Let $V^{\mathrm{first}} = \{v^{j_{\min}} \in V_t \mid j_{\min} = \min\{j \mid v^j \in V_t\}\}$, i.e., the nodes representing the first visits of all stations among all routes, and $V^{\mathrm{last}} = \{v^{j_{\max}} \in V_t \mid j_{\max} = \max\{j \mid v^j \in V_t\}\}$, i.e., the nodes representing the last visits of all stations. Arc set $A_{\mathrm{f}} = A_\sigma \cup A_\tau \cup A_R \cup A_V$ consists of:

- $A_\sigma = \{(\sigma, v^j) \mid v^j \in V^{\mathrm{first}}\}$ with capacities p_v.
- $A_\tau = \{(v^j, \tau) \mid v^j \in V^{\mathrm{last}}\}$ with capacities q_v.
- $A_R = \bigcup_{l \in L} A_{R,l}$ with $A_{R,l} = \{(u^j, v^k) \mid u = r_l^i,\ v = r_l^{i+1},\ j = t(r_l^i),\ k = t(r_l^{i+1}),\ i = 1, \ldots, \rho_l - 1\},\ \forall l \in L$, i.e., arcs representing the flow induced by the vehicles. Capacities are Z_l. Note that multiple arcs exist between two nodes if two (or more) vehicles leave and arrive at the same stations exactly at the same time.
- $A_V = \bigcup_{v \in V} A_v$, $A_v = \{(v^{j_1}, v^{j_2}), \ldots, (v^{j_{\max}-1}, v^{j_{\max}})\}$ with $(v^{j_1}, \ldots, v^{j_{\max}})$ being the sequence of nodes representing visits of station v sorted according to time. These arcs model the bikes staying at a station, capacities are C_v.

Step 1 – minimizing disbalance: In the first step, we calculate the maximum (σ, τ)-flow on this network. As argued in [2,7] the value of this maximum flow corresponds to the maximum achievable reduction of disbalance. In the ideal case all arcs $A_\sigma \cup A_\tau$ are fully saturated in the solution, indicating that the target values q_v can be achieved at all visited stations v. Loading instructions $y_{l,v}^i$ are obtained by taking the flow differences among successive arcs $(u^j, v^k) \in A_R$ for each vehicle and each stop.

However, these loading instructions may be infeasible if an arc $(\sigma, v^j) \in A_\sigma$ is not saturated. In this case, there are actually more bikes at station v than assumed in the flow network, and delivering bikes to this station may exceed the station's capacity.

Step 2 – saturating arcs A_σ: To repair the above situation, we perform a second stage maximum flow computation, modifying the loading instructions to

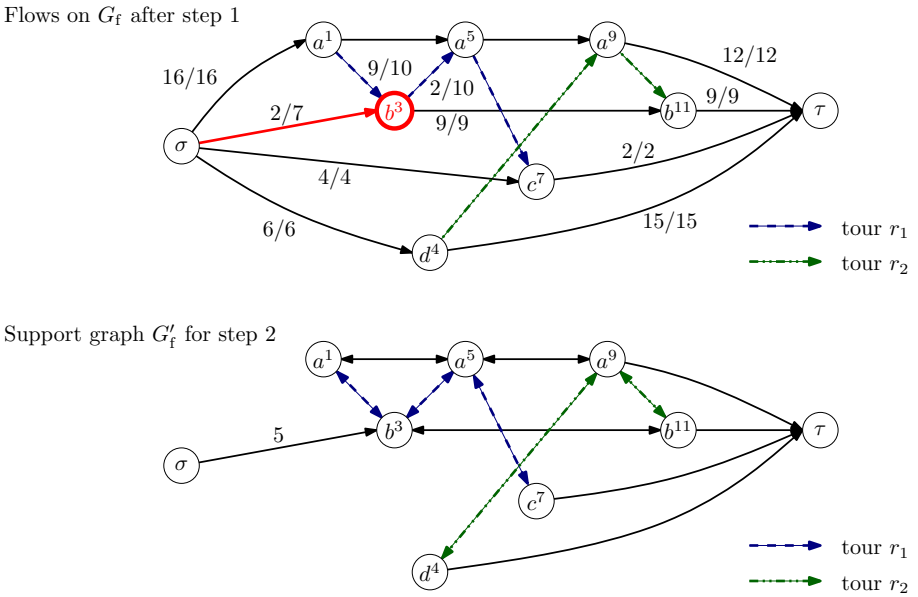


Fig. 3. Example where the arc (σ, b^3) is not saturated after the first maximum flow computation and therefore the solution is infeasible. In step 2 the remaining commodities of (σ, b^3) must be rooted through G'_f. (Flow and capacity values on several arcs are omitted for better readability).

become feasible while the total imbalance remains unchanged. The basic idea is to increase the capacities of the A_τ arcs and push additional flow through the network in order to saturate all arcs A_σ. Let $f(u^j, v^k)$ denote the flow on an arc (u^j, v^k). We derive from $G_f = (V_f, A_f)$ a support graph $G'_f = (V_f, A'_f)$ with the same node set but a modified arc set $A'_f = A'_\sigma \cup A'_\tau \cup A'_R \cup A'_V$ with:

- $A'_\sigma = \{(\sigma, v^j) \in A_\sigma \mid f(\sigma, v^j) < p_v\}$ with capacities $C_v - f(\sigma, v^j)$.
- $A'_\tau = \{(v^j, \tau) \in A_\tau\}$ with capacities $C_v - f(v^j, \tau)$.
- A'_R contains arcs $(u^j, v^k) \in A_R$ with residual capacities $Z_l - f(u^j, v^k)$ and corresponding reverse arcs (v^k, u^j) with capacities $f(u^j, v^k)$.
- A'_V contains arcs $(v^j, v^k) \in A_V$ with residual capacities $C_v - f(v^j, v^k)$ and corresponding reverse arcs (v^k, v^j) with capacities $f(v^j, v^k)$.

Subsequently, we perform a second maximum flow computation on G'_f, which always saturates every arc in A'_σ since $p_v \leq C_v,\ \forall v \in V$ holds. By modifying the original flows on A_R with the new flows $f'(u^j, v^k)$ on $(u^j, v^k) \in A'_R$, we obtain corrected flows $f_{\text{corr}}(u^j, v^k) = f(u^j, v^k) + f'(u^j, v^k) - f'(v^k, u^j),\ \forall (u^j, v^k) \in A_R$. Loading instructions derived from these flows are feasible and optimal with respect to the achievable balance. Figure 3 shows an example of G'_f.

Step 3 – minimizing the number of loading operations: While we cannot do better with respect to balance, we so far neglected the second term of the objective

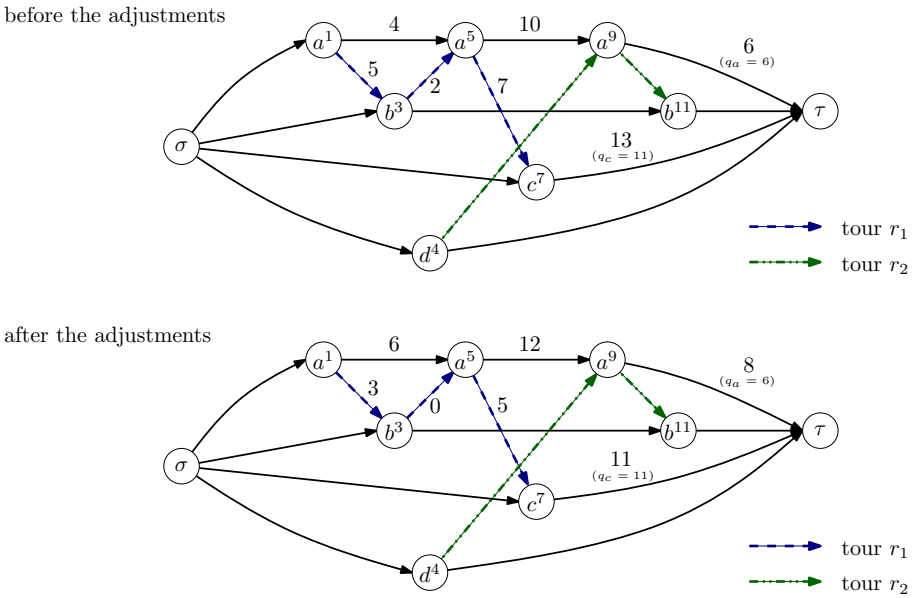


Fig. 4. Example of how the flow values can be adjusted in order to decrease the number of loading operations by 8

function (3) which aims at avoiding unnecessary loading operations. In order to further optimize this aspect, we adjust the corrected flow values f_{corr} on G_{f} in a way that we do not change the overall flow, but the load is shifted from transportation arcs A_R (which influence the loading instructions) to arcs A_V and A_τ. We aim at reaching the final number of bikes at each station as early as possible so that the number of loading operations is minimized. The algorithm works as follows:

(1) Consider all nodes in V_t according to the visiting times:
(2) The next node $v^j \in V_t$ either has an outgoing arc $a = (v^j, \tau) \in A_\tau$ or $a = (v^j, v^k) \in A_V,\ j < k$.
(3) Find a path P in G_{f} from v^j that does not contain arc a and either ends in a node $v^k,\ j < k$ or in node τ. If no such path exists, continue at (2).
(4) Let P' be the alternate path from v^j to the end node of P starting with a and otherwise including only arcs from $A_v \cup A_\tau$. The goal is to adjust the flows f_{corr} on all arcs in P and P' by a common value Δ where
   – flow values on arcs of P are decreased by Δ and must not become negative or less than q_v, and
   – flow values on arcs of P' are increased by Δ and may not exceed their capacity limits.

   If P and P' do not end at τ, we use Δ directly as the adjustment value. If the paths end at τ, the situation becomes more complicated since adjustments modify the balance of some stations. Let (u^i, τ) be the last arc

in P and (v^j, τ) the last arc in P'. Let $bal(P) = f_{\text{corr}}(u^i, \tau) - q_u$ and $bal(P') = f_{\text{corr}}(v^j, \tau) - q_v$. We have to consider the following four cases.
(a) If $bal(P) \geq 0$ and $bal(P') \geq 0$, adjust by $\min(\Delta, bal(P))$.
(b) If $bal(P) \geq 0$ and $bal(P') < 0$, adjust by $\min(\Delta, \max(bal(P), bal(P')))$.
(c) If $bal(P) < 0$ and $bal(P') \geq 0$, do not adjust.
(d) If $bal(P) < 0$ and $bal(P') < 0$, adjust by $\min(\Delta, bal(P'))$.

(5) Repeat (3) and (4) until no further adjustments are possible. If any adjustments were found, restart the algorithm from (1), else continue at (2).

An example of this procedure is given in Fig. 4. First the arc (a^1, a^5) is considered and we can find improvements on paths $P = \langle a^1, b^3, a^5 \rangle$ and $P' = \langle a^1, a^5 \rangle$. Increasing the flow on arcs of P' and decreasing the flow on arcs of P by 2 does not change the final balance of any stations, but reduces the number of loading operations by 4. The second improvement can be found on arc (a^5, a^9) and paths $P = \langle a^5, c^7, \tau \rangle$ and $P' = \langle a^5, a^9, \tau \rangle$. Increasing the flow on arcs of P' and decreasing the flow on arcs of P by 2 now changes the balance at stations a and c. However, shifting the two excessive bikes from c to a is cost-neutral in terms of final balance, but reduces the number of loading operations by 4.

Note that the restart at step (5) is necessary since adjustments on paths from a station v with later visiting times may have an impact on paths from an earlier station u by enabling adjustments that were not possible when u was considered. It can be shown that loading instructions derived from the resulting flows after this procedure are optimal with respect to balance and the number of loading operations as long as the route is local optimal with respect to the VND neighborhoods, i.e., it does not contain any unnecessary stops that can be removed without increasing the disbalance.

5 Combined Approach

Experiments with the new maximum flow based method for the general case indicated a substantial speedup in comparison to the LP-based method. Nevertheless, also this strategy is still significantly slower than the simple greedy method. Experiments shown in the next section indicate that it remains questionable whether the advantage of potentially better solutions outweighs the disadvantage of higher running times. We therefore also investigated algorithms, where the individual strategies for determining loading instructions (except the LP-based method which is now clearly dominated) are applied in combination.

The following approach turned out to work particularly well: We start with the fast greedy method as default strategy and introduce an additional VND neighborhood structure, inserted at the fourth position: All available strategies for determining loading instructions are applied and their results are compared. The best strategy, which is the one yielding the best solution or in case of ties the fastest one, is remembered and from now on also applied as default strategy for all successively created candidate solutions until the solution for which this strategy was found best is discarded in a VNS iteration due to a better non-descending solution. In this latter case, the default strategy is reset to the fast greedy method.

6 Computational Results

We tested our approach on benchmark instances[1] from [7]. These instances are based on real-world data provided by Citybike Wien[2] running a bike-sharing system with currently 92 stations. The instances are characterized by the number of stations $|V| \in \{10, 20, 30, 60, 90\}$, the number of vehicles $|L| \in \{1, 2, 3, 5\}$, and the shift length $\hat{t} \in \{120, 240, 480\}$. 30 instances are considered for each combination of a subset of practically meaningful configurations. The algorithms have been implemented using GCC 4.6 and each test run was performed on a single core of an Intel Xeon E5540 machine with 2.53 GHz. Each run was terminated when no improvement could be achieved within the last 5 000 VNS iterations or after one hour of CPU time. In the first case we consider the heuristic search as converged, major further improvements would be highly unlikely. For all maximum flow calculations we use the push-relabel method by Cherkassky and Goldberg [11], while CPLEX 12.4 is used in the LP approach.

Table 1 shows average results of our VNS with the five methods for deriving loading instructions: GH, MF-MC, and LP from [7] and the two new methods denoted by MF-GC (for "maximum flow based, general case") and COMB (for "combined approach"). For each algorithm variant and each instance class we list the number of instances (runs) for which the algorithm variant yielded the best results (#best), the mean objective value of the finally best solutions ($\overline{\text{obj}}$), the corresponding standard deviation (sd), and the median run time until the best solutions have been found ($\overline{\text{time}}$). Note that objective values must be compared with care due to the small scaling factor for the secondary objectives ($\alpha = 1/100\,000$). In case of two solutions achieving the same balance, objective value differences may be very small, but they might nevertheless indicate practically important differences in the lengths of routes or the number of loading instructions. Thus, we consider #best to be a better indicator for analyzing performance differences than objective value differences. Maximum #best-values are printed bold for each instance class. Note that in comparison to the results published in [7], all former methods exhibit slightly better average objective values due to some small but significant improvements in the implementation.

As one might expect, we observe that in general GH is clearly the fastest variant while LP is slowest. On average over the larger instances for which the runs were terminated by the time limit of one hour, GH could perform 110 times more iterations than LP. MF-MC increased the running time over GH per iteration on average by about 120% and MF-GC by about 290%. They are thus substantially faster than LP but still considerably slower than GH. In contrast, the combined strategy increases the average runtime only moderately by about 20%. Thus, the number of iterations COMB could perform for the larger instances where it has been terminated by the run time limit was not dramatically lower than the iteration number of GH.

[1] Available at `https://www.ads.tuwien.ac.at/w/Research/Problem_Instances`
[2] `http://www.citybikewien.at/`

Table 1. Results of the VNS with different variants for deriving loading instructions

Inst. set			VNS/GH				VNS/MF-MC				VNS/LP				VNS/MF-GC				VNS/COMB			
$\|V\|$	$\|L\|$	$\hat{t}$	#best	obj	sd	time	#best	obj	sd	time	#best	obj	sd	time	#best	obj	sd	time	#best	obj	sd	time
10	1	120	28	28.334	9.911	0.0	29	28.334	9.911	0.0	29	28.334	9.911	0.1	**30**	28.334	9.911	0.0	29	28.334	9.911	0.0
10	1	240	28	4.269	3.551	0.0	29	4.269	3.551	0.0	29	4.269	3.551	1.2	**30**	4.202	3.575	0.0	28	4.269	3.551	0.0
10	1	480	26	0.003	0.000	0.0	**29**	0.003	0.000	0.0	**29**	0.003	0.000	1.2	**29**	0.003	0.000	0.0	27	0.003	0.000	0.0
10	2	120	27	10.002	6.302	0.0	27	10.002	6.302	0.0	**30**	9.936	6.247	0.4	29	9.936	6.247	0.0	28	10.069	6.356	0.0
10	2	240	**30**	0.003	0.000	0.0	**30**	0.003	0.000	0.0	29	0.003	0.000	4.9	**30**	0.003	0.000	0.1	28	0.003	0.000	0.0
10	2	480	26	0.003	0.000	0.0	**29**	0.003	0.000	0.0	28	0.003	0.000	1.2	28	0.003	0.000	0.0	27	0.003	0.000	0.0
20	1	120	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.1	**30**	85.868	14.761	0.0	**30**	85.868	14.761	0.0
20	1	240	28	43.003	11.335	0.0	28	43.003	11.383	0.0	**29**	42.936	11.298	6.5	**29**	43.003	11.335	0.2	**29**	43.003	11.335	0.0
20	1	480	**29**	1.672	2.171	0.7	27	1.672	2.171	1.7	26	1.805	2.187	155.3	27	1.672	2.171	12.7	25	1.672	2.170	0.6
20	2	120	28	55.336	13.373	0.0	28	55.336	13.373	0.0	29	55.336	13.373	2.8	**30**	55.336	13.373	0.1	**30**	55.336	13.373	0.0
20	2	240	20	4.139	3.748	3.2	**22**	4.205	3.726	3.3	16	4.205	3.726	237.3	**22**	4.272	3.704	17.4	18	4.205	3.800	1.3
20	2	480	22	0.006	0.000	2.1	25	0.006	0.000	5.2	20	0.006	0.000	240.9	24	0.006	0.000	7.9	**26**	0.006	0.000	2.0
30	2	120	24	104.802	17.877	0.0	26	104.802	17.877	0.0	**27**	104.736	17.768	7.9	26	104.802	17.792	0.2	26	104.603	17.625	0.0
30	2	240	15	34.472	10.682	4.8	10	34.672	10.781	10.4	13	35.006	10.735	1155.6	**18**	34.739	10.808	58.8	15	34.539	10.966	3.7
30	2	480	16	0.009	0.000	26.1	17	0.009	0.000	36.6	5	0.009	0.000	1490.6	16	0.009	0.000	240.6	**19**	0.009	0.000	31.6
30	3	120	21	78.204	17.343	0.1	27	77.737	17.065	0.5	**23**	78.204	17.438	53.4	**23**	78.004	17.356	1.7	**23**	77.937	17.280	0.3
30	3	240	**11**	7.208	4.475	28.1	7	7.141	4.539	38.4	1	7.675	4.551	1808.4	**11**	6.875	4.384	323.2	7	7.008	4.259	17.9
30	3	480	13	0.009	0.000	32.9	13	0.009	0.000	42.5	4	0.009	0.000	1479.0	**16**	0.009	0.000	392.3	**16**	0.009	0.000	46.3
60	1	120	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.7	**30**	325.601	26.357	0.0	**30**	325.601	26.357	0.0
60	1	240	26	267.336	23.811	0.5	24	267.336	23.794	0.6	26	267.336	24.276	77.6	**28**	267.136	23.842	1.8	26	267.203	23.766	0.4
60	2	480	4	60.412	13.629	309.1	7	60.212	13.889	1006.1	0	63.745	13.803	2969.3	5	61.412	14.202	2094.8	**14**	59.879	13.845	474.1
60	3	240	8	126.742	20.003	68.2	**11**	126.942	20.475	129.8	2	129.209	20.828	2946.9	8	126.876	19.928	593.5	8	127.342	20.153	65.5
60	3	480	8	6.284	3.921	906.2	7	6.350	4.071	2508.3	0	10.617	5.096	2856.3	2	6.884	3.471	2648.4	**15**	5.950	3.805	990.6
60	5	120	8	196.940	28.665	6.6	13	196.474	29.203	12.9	6	197.074	29.005	865.1	13	196.474	29.147	97.2	**18**	196.207	29.390	9.2
60	5	240	7	41.814	13.568	408.2	7	41.548	13.304	720.8	0	46.814	13.293	3085.9	4	42.348	12.251	2342.3	**14**	41.148	12.710	427.2
90	1	120	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.4	**30**	519.801	23.266	0.0	**30**	519.801	23.266	0.0
90	1	240	**29**	456.936	21.125	0.8	28	456.936	21.281	2.8	**29**	456.870	21.209	110.2	**29**	456.936	21.125	6.7	**29**	456.936	21.125	0.9
90	1	480	15	352.540	17.702	17.3	14	352.673	18.280	102.4	7	353.740	18.365	815.3	**16**	352.673	18.317	328.7	**16**	352.673	18.021	35.8
90	2	120	25	478.736	21.987	0.3	23	478.936	21.818	0.6	25	478.869	21.900	70.2	**27**	478.803	21.881	2.0	**27**	478.803	21.975	0.3
90	2	240	13	368.140	18.479	32.0	**14**	368.206	18.105	83.7	7	369.406	18.187	1708.2	13	368.273	18.343	209.6	13	368.540	18.292	37.0
90	2	480	**11**	205.279	13.250	493.1	6	205.013	13.500	1733.8	0	210.879	14.070	2987.4	4	206.679	13.102	2663.3	9	205.146	12.819	875.2
90	3	120	23	441.938	20.946	1.2	21	441.871	20.724	3.3	23	441.804	21.362	221.2	**25**	441.604	20.647	11.4	23	441.738	20.882	2.9
90	3	240	9	294.809	16.320	99.6	9	293.609	16.020	304.4	0	297.609	15.361	2598.0	**12**	294.276	16.109	1133.2	6	294.476	15.971	198.0
90	3	480	11	100.352	9.571	2007.1	6	101.218	10.300	2526.2	0	110.485	9.919	2364.8	0	104.418	9.970	2759.2	**13**	99.952	9.119	2320.3
90	5	120	5	376.141	20.109	13.3	9	376.140	19.874	23.2	10	376.141	19.333	1631.5	**12**	375.807	20.405	161.7	10	375.741	20.299	19.2
90	5	240	**10**	174.282	13.541	594.7	**10**	174.415	12.653	1451.5	0	185.482	13.108	3040.4	3	177.348	13.007	2479.2	9	174.149	13.472	769.2
90	5	480	**15**	0.961	1.638	2888.7	2	1.361	1.688	3018.2	0	8.494	3.223	2657.8	0	3.028	2.149	3091.2	14	1.028	1.364	3139.7
TOTAL			709	5252.403		7944.9	704	5251.737		13767.2	592	5308.335		38654.0	709	5263.486		21679.4	**755**	**5249.203**		9469.2

Concerning the total number of instances on which an algorithm performed best (sum of all #best) COMB is the clear winner with 755 instances, followed by GH and MF-GC with the same value of 709 instances, and closely behind MF-MC with 704 instances. Final objective values give a similar picture although observed differences are generally very small. Wilcoxon signed-rank tests indicate the statistical significance of the superiority of COMB over all other approaches and that LP is dominated by all other approaches with error probabilities of less than 3%.

It is relatively hard to derive more detailed conclusions of the performances w.r.t. the parameters $|V|$, $|L|$, and $\hat{t}$, but we can observe the general tendency that LP can only compete on small instances, while GH, MF-MC, MF-GC, and COMB perform comparably well over the whole range of benchmark instances.

7 Conclusions and Future Work

We presented a new method for calculating proven optimal loading instructions for given routes to be used within a heuristic VNS framework for obtaining approximate solutions for the static problem of balancing a bicycle sharing system. This method, called MF-GC, is not restricted by monotonicity and is based on two sequential maximum-flow calculations and a final phase for minimizing the number of loading operations. The experiments performed have shown that this new method is in practice substantially faster than LP. The VNS with MF-GC is therefore able to perform much more iterations than with LP in reasonable time, and consequently significantly better final solutions are obtained. Nevertheless, the greedy heuristic calculation of loading instructions is still a very strong competitor, since it is even faster and yields results that are typically very close to optimal, despite the fact that it produces only monotonic solutions. These results also show, that only infrequently significant advantages can be gained by exploiting non-monotonicity, i.e., by using stations as buffers or transferring bikes from one vehicle to another.

Finally, we applied the strategies GH, MF-MC, and MF-GC in combination, controlled by an additionally introduced VND-neighborhood structure. This approach turned out to exploit the benefits of the individual strategies in a beneficial way – it applies the more expensive MF-MC and MF-GC only on a more regular basis when an advantage could already be gained in a predecessor solution. This approach performed best with high statistical significance, although objective value differences still remain small.

In future work we want to investigate further large neighborhood structures that might be based on ejection chains or mixed integer programming. Furthermore the underlying problem definition needs to be extended to cover the dynamic case, in which bikes rented and brought back during the rebalancing process are also considered. As these demands are not exactly known in advance, this extended problem variant becomes a stochastic online problem.

Acknowledgements. This work is supported by the Austrian Research Promotion Agency (FFG) under contract 831740. We thank Matthias Prandtstetter, Andrea Rendl, and Markus Straub from the Austrian Institute of Technology (AIT) and City Bike Wien for the collaboration in this project, constructive comments and for providing the data used in our test instances.

References

1. DeMaio, P.: Bike-sharing: History, impacts, models of provision, and future. Journal of Public Transportation 12(4), 41–56 (2009)
2. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization 10(2), 120–146 (2013)
3. Benchimol, M., Benchimol, P., Chappert, B., De la Taille, A., Laroche, F., Meunier, F., Robinet, L.: Balancing the stations of a self service bike hire system. RAIRO – Operations Research 45(1), 37–61 (2011)
4. Raviv, T., Tzur, M., Forma, I.A.: Static repositioning in a bike-sharing system: models and solution approaches. EURO Journal on Transportation and Logistics, 1–43 (2013)
5. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Technical Report CIRRELT-2012-09, CIRRELT, Montreal, Canada, submitted to Transportation Science (2012)
6. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Technical Report 2013-E1, Tepper School of Business, Carnegie Mellon University (2013)
7. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In: Middendorf, M., Blum, C. (eds.) EvoCOP 2013. LNCS, vol. 7832, pp. 121–132. Springer, Heidelberg (2013)
8. Mladenović, N., Hansen, P.: Variable neighborhood search. Computers and Operations Research 24(11), 1097–1100 (1997)
9. Pirkwieser, S., Raidl, G.R.: A variable neighborhood search for the periodic vehicle routing problem with time windows. In: Prodhon, C., Wolfler-Calvo, R., Labadi, N., Prins, C. (eds.) Proceedings of the 9th EU/MEeting on Metaheuristics for Logistics and Vehicle Routing, Troyes, France (2008)
10. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics, pp. 219–249. Kluwer Academic Publishers (2003)
11. Cherkassky, B.V., Goldberg, A.V.: On implementing the push-relabel method for the maximum flow problem. Algorithmica 19(4), 390–410 (1997)

Automatic Design of Hybrid Stochastic Local Search Algorithms

Marie-Eléonore Marmion, Franco Mascia,
Manuel López-Ibáñez, and Thomas Stützle

IRIDIA, CoDE, Université Libre de Bruxelles (ULB), Brussels, Belgium
{mmarmion,fmascia,manuel.lopez-ibanez,stuetzle}@ulb.ac.be

Abstract. Many stochastic local search (SLS) methods rely on the manipulation of single solutions at each of the search steps. Examples are iterative improvement, iterated local search, simulated annealing, variable neighborhood search, and iterated greedy. These SLS methods are the basis of many state-of-the-art algorithms for hard combinatorial optimization problems. Often, several of these SLS methods are combined with each other to improve performance. We propose here a practical, unified structure that encompasses several such SLS methods. The proposed structure is *unified* because it integrates these metaheuristics into a single structure from which we can not only instantiate each of them, but we also can generate complex combinations and variants. Moreover, the structure is *practical* since we propose a method to instantiate actual algorithms for practical problems in a semi-automatic fashion. The method presented in this work implements a general local search structure as a grammar; an instantiation of such a grammar is a program that can be compiled into executable form. We propose to find the appropriate grammar instantiation for a particular problem by means of automatic configuration. The result is a semi-automatic system that, with little human effort, is able to generate powerful hybrid SLS algorithms.

Keywords: Stochastic local search, generalized local search structure, grammar, automatic algorithm design.

1 Introduction

Many stochastic local search (SLS) methods manipulate a single solution at each of the search steps [11]. Examples of such SLS methods (also called metaheuristics) include classical iterative best- and first-improvement algorithms [20], iterated local search (ILS) [15], simulated annealing (SA) [13], variable neighborhood search (VNS) [10], random iterative improvement (RII) [20,11], probabilistic iterative improvement (PII) [11], and iterated greedy (IG) [21], among others. Successful algorithms for hard combinatorial problems are often the result of an effective engineering of such SLS methods or of an appropriate combination of ideas from various of these methods. However, despite the plethora of possibilities, algorithm designers rarely consider but a few methods when tackling

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 144–158, 2013.

a new problem. We believe that this is due to two main reasons. First, the hybridization of such techniques is not a trivial task in terms of designing how the different parts should interact and implementing all possible interactions. Second, the effort required to design and analyze experiments that evaluate the different components and parameters of a hybrid algorithm is significant, thus considering a large number of hybrid algorithms may seem prohibitive at first.

In this paper, we propose a semi-automatic system that, with little human effort, is able to generate powerful hybrid SLS algorithms. We achieve this by combining two different proposals. First, we propose a unified structure that encompasses many of the above mentioned SLS methods proposed in the literature. We describe this unified structure as a grammar, from which we may instantiate not only SLS algorithms following one specific SLS methods, but also SLS algorithms that are composed of algorithmic components that are taken from various of these individual methods. Hence, our proposed grammar defines a very large space of possible hybrid SLS algorithms.

Our second proposal is to find the best instantiation of the grammar for a given problem by means of automatic configuration tools. Automatic configuration tools are typically used for tuning the parameters of optimization algorithms, given a set of training instances and a description of the parameter space [12,14]. However, automatic configuration tools are also effective at instantiating heuristics from grammars [16].

In our approach, most human effort is devoted to implement and improve the problem-specific components (neighborhood moves, perturbations, heuristics), which are often the key to the success of an algorithm in a specific problem. Given these components and a set of training instances representative of the problem, the system takes care of generating a large number of hybrid SLS algorithms, and selects the best-performing on the training instances. This generation and selection process is mainly limited by the computation power available.

There are other two key characteristics of our proposal. First, the unified structure embodied by the grammar allows reusing the same few problem-specific components to generate a large number of different algorithms. The implementation of reusable components is based on ParadisEO [3], a framework that allows algorithm designers to reuse basic components to build their own algorithms. The system we propose in this work goes a step beyond and builds the algorithms automatically. The second key characteristic is that the algorithms instantiated from the grammar are stand-alone programs that are compiled to executable code. Therefore, the overhead introduced by the flexibility of the system is minimized, and the resulting automatically-crafted SLS algorithms are competitive with hand-crafted algorithms.

The paper is structured as follows. Section 2 introduces our unified structure for SLS algorithms and Section 3 how this structure can be implemented through a grammar and how SLS algorithms are configured. Section 4 describes our benchmark problem; experimental results are then given in Section 5 before we conclude and outline directions for extending our proposals in Section 6.

ILS Algorithm
1: $s_0 :=$ Initialization()
2: $s^* :=$ ILS(s_0)
3: **return** s^*

Function ILS(s_0)
Require: perturbation, ls, acceptanceCriterion, stop
1: $s^* :=$ ls(s_0)
2: **repeat**
3: $s' :=$ perturbation(s^*)
4: $s'' :=$ ls(s')
5: $s^* :=$ acceptanceCriterion(s'', s^*)
6: **until** termination criterion (stop) is satisfied
7: **return** s^*

Fig. 1. The iterated local search (ILS) algorithm

2 Generalized Local Search Structure

In this paper, we focus on SLS algorithms that work on a single solution at a time. The algorithm may internally keep a memory of multiple solutions, such as the best solution found so far, but there is the concept of the *current solution*, whose neighborhood is being explored.

We propose a generalized local search (GLS) structure modeled after iterated local search (ILS) [15]. ILS, as shown in Fig. 1, starts from an initial solution s_0, applies an improvement method (usually referred as local search, ls), and then three steps are repeated until the termination criterion is met: the current solution is perturbed to generate a new one, local search is applied to the new solution, and the acceptance criterion (in the simplest case of acceptance criteria) accepts the new solution or stays with the current one. ILS contains the most important elements of any hybrid LS algorithm, which are a *perturbation* operator, a subsidiary *local search*, and an *acceptance criterion*.

The *perturbation* is a transformation of the input solution. In ILS, this is typically a small random transformation of the solution but it may also be a random re-initialization. A perturbation may be one simple move in a neighborhood space, but it may also be composed of k applications of a simple move, and k may be even vary during the search either based on feedback of the search process or according to a pre-defined schedule. The *local search* can range from a simple iterative improvement over short runs of an SA algorithm to a full-fledged ILS. It could also be that no local search algorithm is used at all.

The *acceptance criterion* determines which solution will replace the current solution. The most basic acceptance criterion (*improveAccept*) accepts only solutions that are better (strictly or not) than the best solution found so far. Other acceptance criteria allow worse solutions to be accepted in order to increase the exploration of the search space. For instance, the *probAccept* criterion accepts a worsening solution with a probability $p \in [0, 1]$. For example, if $p = 1$, every new solution is accepted (*alwaysAccept*). A *thresholdAccept* criterion accepts a worsening solution if the relative deviation between the best and the current solution is below a threshold. In simulated annealing, a worse solution is accepted according to the Metropolis criterion (*metropolisAccept*). This criterion uses a cooling schedule that starts from an initial temperature, the temperature

Table 1. Classical SLS algorithms modeled after the ILS scheme

Name	Perturbation	Local Search	Acceptance Criterion
SA [13]	one move	∅	Metropolis
PII [11]	one move	∅	prob.
RII [20,11]	one move	∅	probRandom
VNS [10]	variable move	first-improv. descent	improvingStrictly
IG [21]	deconstruction-construction	"any"	"any"

is decreased according to the cooling schedule until the algorithm stops after reaching a final temperature. Often, the temperature is decreased periodically after a number of iterations (*span*).

By considering different alternatives for each of these components, we can replicate many of the SLS methods proposed in the literature. For instance, simulated annealing (SA) can be replicated by defining the perturbation operator as a simple move operator in a neighborhood, using no subsidiary local search, and using the Metropolis acceptance criterion. With these components, the scheme given for ILS above will actually replicate a classical SA algorithm.

Another example is variable neighborhood search (VNS) [10]. VNS executes an iterative improvement method at each iteration, and varies the strength of the perturbation depending on whether the resulting solution improves the best so far. This is equivalent to an ILS with a specialized *variable move* operator, iterative improvement as local search, and an *improveAccept* acceptance criterion. Other classical SLS algorithms can be modeled after ILS in a similar manner, as shown in Table 1.

In addition to replicating these classical SLS algorithms, the GLS structure proposed here can also reproduce more complex combinations of SLS algorithms. For example, an ILS algorithm can use a different ILS algorithm (with different perturbation and/or acceptance criterion) as a subsidiary local search, which, in turn may use SA as its own subsidiary local search. We call *recursion* the possibility of an ILS to embed another ILS. The level of recursion is the number of embedded ILSs. This ability of combining simple components to generate hybrid local searches allows designing powerful algorithms. However, it raises the question of how to find high-performing algorithms for a particular problem, among all the possible combinations. The next section deals with this question.

3 Implementation

3.1 A Practical Implementation of the GLS Structure

In this section, we describe how to implement the GLS structure proposed above in order to generate practical algorithms for a given problem. Our method consists of three parts. First, we use a generative grammar to describe the design space defined by the GLS structure. Second, we use a re-usable framework of source code components as the underlying implementation of the grammar. This implementation includes both problem-independent code, which can be re-used

in any problem, and problem-specific components, which must be developed for each problem. Finally, we use automatic algorithm configuration tools to search the design space and generate high-performing instantiations of the grammar, given a set of training instances representative of a problem.

3.2 A Grammar Description of the GLS Structure

A practical implementation of the GLS structure will contain many components that interact. Implementing such a GLS structure as a unique monolithic algorithm is a complex task. Moreover, the fact that a local search can be embedded within another in arbitrary ways complicates such implementation. The alternative that we propose here is to implement only the individual components, with clearly defined interfaces, and directly generate specific algorithms by combining these components. This is a typical problem in genetic programming, where grammars are often used to represent the design space of an algorithm [17].

A grammar is a set of derivation rules that describes how the symbols in a language can be combined to produce valid sentences. In our case, the valid sentences are local search algorithms encoded in `C++`, but for clarity we will describe the algorithms in pseudo-code. Fig. 2 shows the grammar that describes the GLS structure proposed in the previous section. Each line is a production rule of the form `<non-terminal> ::=` *expression* that describes how the non-terminal on the left-hand side can be replaced by the expression on the right-hand side. Expressions may contain terminal and/or non-terminals. Alternative expressions are separated with the symbol "|". The non-terminal symbol `<algorithm>` defines the starting point for instantiating an algorithm from the grammar.

The first three rules in the grammar describe the main structure of the GLS structure proposed earlier (see Fig. 1). The next three rules describe the basic components of our GLS structure, that is, the perturbation operator (`<perturb>`), local search (`<ls>`), and acceptance criterion (`<accept>`). Since the rule `<ls>` can expand to `<ils>` which contains again `<ls>`, a local search can be embedded within another local search (recursion). The other rules describe the alternatives available for the various components. Our grammar explicitly contains classical local search algorithms, but defined in terms of ILS, as detailed in Table 1. Moreover, the grammar also allows problem-specific components (`<pbs_...>`), which can be implemented for each problem tackled.

The possibility of adding problem-specific components is an advantage of our proposed method. Such components are critical for the success of SLS algorithms. For example, in this way problem specific construction and destruction mechanisms can be incorporated and be used in the destruction/construction phase (`<deconst-construct_perturb>`) of an IG algorithm. Hence, our grammar must account for such components. A practical implementation of our method also requires to define other problem-specific components in order to describe the representation of the problem, neighborhood operators, the objective function and how to read an instance of the problem. For simplicity, we do not include these in our exposition, but they are implemented in a similar fashion.

```
     <algorithm> ::= <initialization> <ils>
<initialization> ::= random | <pbs_initialization>
           <ils> ::= ILS(<perturb>, <ls>, <accept>, <stop>)

<perturb> ::= none | <initialization> | <pbs_perturb>
     <ls> ::= <ils> | <descent> | <sa> | <rii> | <pii> | <vns> | <ig> | <pbs_ls>
 <accept> ::= alwaysAccept | improvingAccept  <comparator>
            | prob(<value_prob_accept>) | probRandom | <metropolis>
            | threshold(<value_threshold_accept>) | <pbs_accept>

<descent> ::= bestDescent(<comparator>, <stop>)
            | firstImprDescent(<comparator>, <stop>)
    <sa>  ::= ILS(<pbs_move>, no_ls,  <metropolis>, <stop>)
    <rii> ::= ILS(<pbs_move>, no_ls, probRandom, <stop>)
    <pii> ::= ILS(<pbs_move>, no_ls, prob(<value_prob_accept>), <stop>)
    <vns> ::= ILS(<pbs_variable_move>, firstImprDescent(improvingStrictly),
                  improvingAccept(improvingStrictly), <stop>)
     <ig> ::= ILS(<deconst-construct_perturb>, <ls>, <accept>, <stop>)

<comparator> ::= improvingStrictly | improving
<value_prob_accept> ::= [0, 1]
<value_threshold_accept> ::= [0, 1]
<metropolis> ::= metropolisAccept(<init_temperature>, <final_temperature>,
                                  <decreasing_temperature_ratio>, <span>)
 <init_temperature> ::= {1, 2,..., 10000}
<final_temperature> ::= {1, 2,..., 100}
<decreasing_temperature_ratio> ::= [0, 1]
<span> ::= {1, 2,..., 10000}
```

Fig. 2. A simplified view of the grammar for the GLS structure

Finally, each ILS in the proposed grammar has its own termination criterion (`<stop>`), which is typically a maximum computation time. If there is more than one level of ILS algorithms, the total computation time must be divided among them, such that the inner level does not consume all available time. We adopt here a simple scheme. The top-level ILS stops once the total time is consumed. Each subsequent level stops after consuming a ratio of the time allocated to its parent ILS. This ratio is controlled by a parameter $t_{\text{ls}} \in \{0.1, 0.2, \ldots, 1\}$ for each level of ILS. These ratios have to be tuned in order to generate an efficient hybrid SLS algorithm.

In practice, an instantiation of the grammar produces an algorithm that is mapped to source code implementing the individual components. In our case, the implementation of the components is done using Paradiseo [3], an open-source `C++` framework whose purpose is to facilitate the design of metaheuristics by providing a library of reusable components. The idea is that an algorithm designer can re-use the available algorithm components or implement her own components, and freely combine these components to design new algorithms. Our proposal goes a step beyond this idea, since in our proposed method the algorithm designer can focus on implementing problem-specific components, while the grammar takes care of describing the possible algorithm designs given the available components. The next section describes how to automatically find a high-performing SLS algorithm for a given problem, among all the possible algorithm designs that can be generated from the grammar.

3.3 Automatic Generation of Hybrid LS Metaheuristics

Given a particular problem, our goal is to find the highest-performing instantiation of the grammar given above. As mentioned above, techniques such as genetic programming [17] and grammatical evolution [2] are often used for this task. Recently, we have shown how to instantiate IG algorithms from a grammar by means of a parametric representation [16]. The use of a parametric representation has certain advantages and enables the use of state-of-the-art automatic configuration tools for offline parameter tuning. In that work, we show that a parametric representation produced better IG algorithms than the representation used by grammatical evolution. Here we explore the much larger space of SLS algorithms defined by the proposed GLS structure.

We follow the method described in our previous work [16] to generate a parametric description of the grammar. This requires defining a maximum limit to the number of ILS levels in the final algorithm, that is, a maximum number of applications of the rule `<ls>` in the grammar. This limit has an influence on the number of parameters required to describe the grammar. In the next section, we explore the effect of this limit on the results.

From the parameter description and given a set of training instances representative of the problem, we apply an automatic configuration tool to search the space of possible algorithm designs. Here, we use irace [14], a publicly available implementation of Iterated F-Race [1]. Nonetheless, any automatic configuration tool that handles large numbers of categorical, numerical and conditional parameters with complex constraints would be appropriate.

Each parameter configuration tested by irace is an instantiation of the grammar, which is mapped to C++ code and compiled into an executable. This executable is then run on various training instances by irace in order to determine its performance. A complete description of the irace procedure is beyond the scope of the paper. It suffices to say that the irace procedure stops after exhausting a given budget of algorithm runs, and that it returns the SLS algorithm configuration that it identified as the best performing one during the tuning.

4 Experimental Setup

We test our proposed method on the permutation flowshop scheduling problem with weighted tardiness (*PFSP-WT*). In contrast to our previous work [16], our aim here is to automatically generate a hybrid SLS algorithm that matches or outperforms the current state-of-the-art algorithm for the *PFSP-WT*. First, we briefly describe the *PFSP-WT*. Then, we summarize the state-of-the-art algorithm and the problem-specific components added to our grammar. Finally, we describe the experimental setup.

4.1 The *PFSP-WT*

The permutation flowshop scheduling problem (PFSP) encompasses a variety of problems that are typical of industrial production environments. The common

goal of various PFSPs is to schedule n jobs on m machines with the condition that all jobs must be processed in the same order and jobs are not allowed to pass each other. Each job i requires, on each machine j, a fixed, non-negative processing time p_{ij}.

In the *PFSP-WT*, we are asked to determine a schedule that minimizes the total weighted tardiness. Each job i has a due date d_i, which denotes the desired completion time of the job on the last machine, and a priority weight w_i, which denotes its importance. The *tardiness* of a job i is defined as $T_i = \max\{C_i - d_i, 0\}$, where C_i is the completion time of job i on the last machine, and the *total weighted tardiness* is given by $\sum_{i=1}^{n} w_i \cdot T_i$. This problem is *NP*-hard even for a single machine [5].

4.2 Local Search Components for the *PFSP-WT*

To the best of our knowledge, a state-of-the-art algorithm for the *PFSP-WT* was proposed by Dubois-Lacoste et al. [7]. This algorithm, henceforth called soa-IG, is an iterated greedy algorithm that works as follows. An initial solution is constructed using a modified version of the well-known NEH algorithm [19] called NEH-WSLACK. In NEH-WSLACK [6], the WSLACK heuristic provides the initial order for the NEH algorithm, and the jobs are inserted in the solution in the order that minimizes the partial objective function, i.e., computed using the jobs present in the partially constructed solution. The local search in soa-IG is a first-improvement descent using a swap neighborhood and with a maximum number of swaps, fixed to $2 \cdot (n-1)$ swaps, where n is the number of jobs. The perturbation operator consists in removing d jobs randomly from the solution. These jobs are re-inserted one by one to minimize the partial objective function. Finally, in the acceptance criterion, a new solution that is worse than the current one is accepted with a probability given by $\exp(100 \cdot (f(\pi) - f(\pi'))/(f(\pi) \cdot T_c))$, where T_c is a user-defined parameter, $f(\pi)$ is the objective value of the current solution and $f(\pi')$ is the objective value of the new one. Dubois-Lacoste et al. [7] suggest the settings $d = 5$ and $T_c = 1.2$.

We add the aforementioned components to the grammar of our GLS structure as additional problem-specific components (Fig. 3). In particular, we add two initialization methods, NEH with and without the WSLACK heuristic (`NEH` and `NEH-WSLACK`). In addition to the random destruction-construction perturbation used by soa-IG, we add further problem-specific perturbations based on classical neighborhood move operators (insert, exchange and swap) and a strength parameter k that controls the number of random moves applied per perturbation. The value of k may be fixed or vary during the run (`var_`) as in VNS. The problem-specific local search used by SOA is added to the grammar (`soa_ls`). Moreover, the `pbs(_variable)_move` used in the grammar (see Fig 2) are set to the insert move. Note that, the descents also use the insert move to define the neighborhood. Finally, we add the acceptance criterion of soa-IG as an additional acceptance criterion.

```
<pbs_initialization> ::= NEH | NEH-WSLACK
<pbs_perturb> ::= <deconst-construct_perturb>
                | <perturb_move>(<k>)
                | var_<perturb_move>(<k>)
<perturb_move> ::= insert | swap | exchange
<k> ::=  {1,2,...,10}
<deconst-construct_perturb> ::= soa_ig_perturb(<d>)
<d> ::= {1,2,...,10}
<pbs_ls> ::= soa_ig_ls

<pbs_variable_move> ::= var_<pbs_move>(<k>)
<pbs_move> ::= insert
<pbs_accept> ::= soa_ig_accept(<Tc>)
<Tc> ::= [0,1]
```

Fig. 3. The extended grammar for the *PFSP-WT*

4.3 Experimental Protocol

We assess the potential of the proposed method by generating three hybrid SLS algorithms for the *PFSP-WT*, and comparing them with soa-IG. In particular, we generate three algorithms (ALS1, ALS2, and ALS3) for tackling the *PFSP-WT* from our GLS structure, by allowing different levels of recursion.

The procedure for generating these three algorithms is as follows. We consider the grammar presented in Fig. 2 and the *PFSP-WT*-specific extensions discussed above (Fig. 3). For each level of recursion, we automatically generate a parameter description. Indeed, the recursion leads to an increasing number of parameters. With one level of recursion, i.e., a single ILS, the grammar is represented by 80 parameters. Of these 80 parameters, 27 are categorical and represent possible algorithmic choices, 25 are integer-valued, and 28 are real-valued. With two or three levels of recursion, the number of parameters increases to 127 and 174, respectively. Any combination of the values that can be assumed by these parameters defines a different hybrid SLS algorithm implemented in `C++` and compiled with GCC 4.7.2 with options "`-Ofast -flto -march=native`". All experiments are carried out on a single core of AMD Opteron 6272 processors (2.1GHz) running CentOS 6.2 Linux.

The parameter description is given to irace together with a number of training instances. As training instances, we generated 10 random *PFSP-WT* instances for each number of jobs in $\{50, 60, 70, 80, 90, 100\}$ and with 20 machines, following the procedure described by Minella et al. [18]. Within irace, a specific algorithm, i.e., a specific parameter configuration, is evaluated by running it on a training instance with a time limit of 30 CPU-seconds. A single run of irace stops after exhausting a given budget of evaluations. Since the number of parameters is different according to the level of recursion, we used different budgets for the different runs of irace; concretely, 30 000 evaluations for generating ALS1, 40 000 for generating ALS2, and 50 000 for generating ALS3.

The three algorithms ALS1, ALS2 and ALS3 generated by irace are then run on a set of test instances of size `50x20` and `100x20`, different from the set of training instances. Also soa-IG is run on the same instances. To avoid differences due to implementation details, we have instantiated soa-IG as one

```
ALS1 Algorithm: ILS(IG)
  s_0 := NEH-WSLACK()
  s* := ILS(perturb_move_insert(k = 6),
            ILS(soa_ig_perturb(d = 9),
                firstImprDescent(strict,
                                 t_ls = 0.5),
                soa_ig_accept(T_c = 0.8956),
                t_ls = 0.8)
            improvingAccept,
            t_ls = maxTime)
  return s*
```

```
ALS2 Algorithm: ILS(ILS(VNS)) :
  s_0 := NEH()
  s* := ILS(perturb_none,
            ILS(perturb_none,
                ILS(variable_move_insert(k = 1),
                    firstImprDescent(strict,
                                     t_ls = 0.4),
                    improvingStrictlyAccept,
                    t_ls = 0.4),
                metropolisAccept(1548, 56, 0.7447, 7401),
                t_ls = 0.8),
            improvingAccept,
            t_ls = maxTime)
  return s*
```

```
ALS3 Algorithm: ILS(ILS(VNS)) :
  s_0 := NEH-WSLACK()
  s* := ILS(perturb_move_exchange(k = 7)),
            ILS(soa_ig_perturb(d = 5),
                ILS(variable_move_insert(k = 3),
                    firstImprDescent(strict, t_ls = 0.3),
                    improvingStrictlyAccept, t_ls = 0.4),
                metropolisAccept(4969, 48, 0.8356, 8954), t_ls = 0.8),
            alwaysAccept, t_ls = maxTime)
  return s*
```

Fig. 4. Hybrid LS algorithms automatically generated for *PFSP-WT*

specific SLS algorithm through our grammar, taking care that the algorithm is implemented correctly in this way. These test instances were generated by Minella et al. [18] from well-known PFSP instances [22]. Each run is repeated 30 times with different random seeds.

5 Experimental Results

The three algorithm (ALS1, ALS2 and ALS3) generated by irace are shown in Fig. 4. The first one (ALS1) is an IG algorithm within a classical ILS. It uses the NEH-WSLACK initialization, then executes a classical ILS with a k-`insert` move as perturbation, IG as the subsidiary local search, and an `improving` acceptance criterion. The IG has a time limit of $0.8 \cdot maxTime$, and it is represented by an ILS with the construction/deconstruction operator of soa-IG as the perturbation, a first-improvement descent as the subsidiary local search, and the IG acceptance criterion. The first-improvement descent has a time limit of $0.5 \cdot 0.8 \cdot maxTime$. (Note that the first-improvement descent will actually terminate much before its maximum time limit upon finding a local optimum; in fact, the time limits mentioned here and in the following do actually not restrict the computation times of iterative improvement algorithms.)

ALS2 is a VNS algorithm included in an ILS that is itself included in an ILS. ALS2 uses the NEH initialization, then executes a classical ILS without perturbation, an ILS as the subsidiary local search, and an `improving` acceptance criterion. The subsidiary ILS has a time limit of $0.8 \cdot maxTime$, again no perturbation, a VNS as the subsidiary local search, and a `Metropolis` acceptance criterion. The VNS has a time limit of $0.4 \cdot 0.8 \cdot maxTime$, and it is represented as

an ILS with a *variable* `insert` move perturbation, a first-improvement descent as the subsidiary local search, and the `improvingStrictly` acceptance criterion. The first-improvement descent has a time limit of $0.4 \cdot 0.4 \cdot 0.8 \cdot maxTime$.

ALS3 is also a VNS algorithm included in an ILS that is itself included in an ILS. Although three levels of recursion were allowed when generating ALS3, this algorithm only has two levels as ALS2. ALS3 uses the NEH-WSLACK initialization, then executes a classical ILS with a k-`exchange` move as perturbation, an ILS as the subsidiary local search, and an acceptance criterion that always accepts a new solution. The subsidiary ILS has a time limit of $0.8 \cdot maxTime$, and uses the construction/deconstruction operator of soa-IG as the perturbation, a VNS as the subsidiary local search, and a `Metropolis` acceptance criterion. The VNS has a time limit of $0.4 \cdot 0.8 \cdot maxTime$ and it is represented as an ILS with a *variable* `insert` move perturbation, a first-improvement descent as the subsidiary local search, and the `improvingStrictly` acceptance criterion. The first-improvement descent has a time limit of $0.3 \cdot 0.4 \cdot 0.8 \cdot maxTime$ seconds.

Comparison with the State-of-the-Art Algorithm. To assess the performance of the three automatically generated algorithms, we run them 30 times on the test instances and compare them with soa-IG. Fig. 5 and 6 show the solution cost reached by each algorithm on each instance. Table 2 gives the best and mean solution. The behavior of the algorithms is slightly different depending

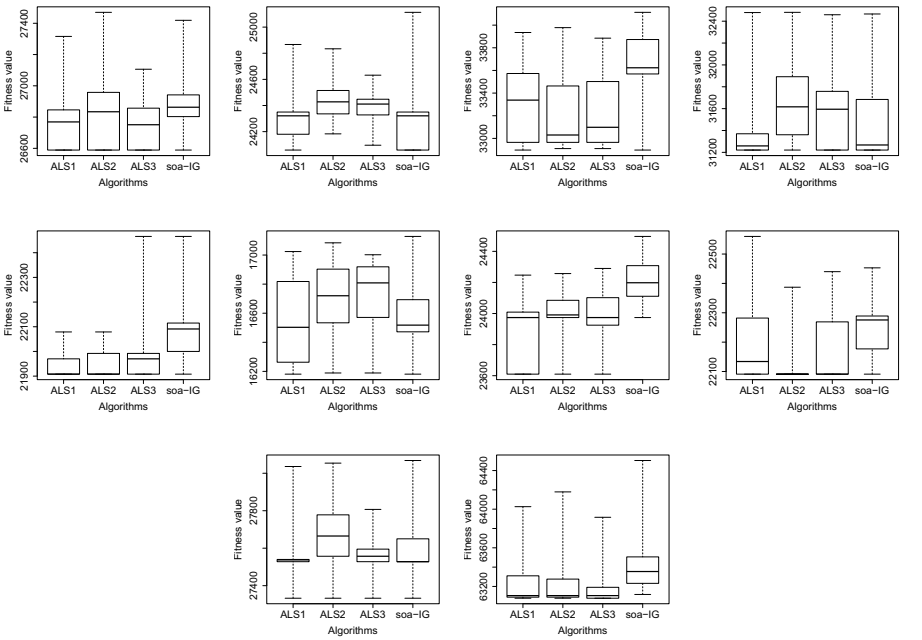


Fig. 5. Solution costs obtained by the three automatic SLS algorithms (ALS1, ALS2 and ALS3) and soa-IG on the `50x20` instances

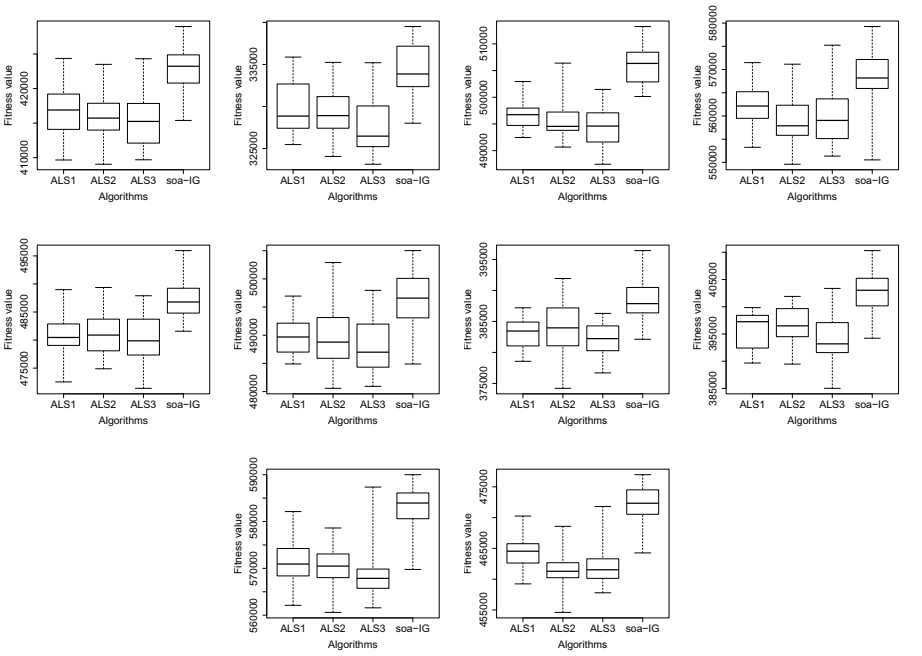


Fig. 6. Solution costs obtained by the three automatic SLS algorithms (ALS1, ALS2 and ALS3) and soa-IG on the `100x20` instances.

on the instance size. The performance of the automatically generated SLS algorithms on the `50x20` instances matches the quality obtained by soa-IG in most instances, and they are noticeably better on a few. On the `100x20` instances, the automatic SLS algorithms clearly outperform soa-IG.

In order to assess the performance over each set of instances, we perform a statistical analysis based on the Friedman test for analyzing non-parametric unreplicated complete block designs, and its associated post-hoc test for multiple comparisons [4]. First, we pair the runs performed on the same instance using the same random seed. This is the blocking factor, and the different algorithms are the treatment factor. Algorithms are ranked within each block, lower solution cost corresponds to lower rank. If the Friedman test rejects the hypothesis that the different algorithms obtain the same mean rank, then we calculate the difference (ΔR) between the sum of ranks of each algorithm and the best ranked one (with the lowest sum of ranks). We also calculate the minimum difference between the sum of ranks of two algorithms that is statistically significant (ΔR_α), given a significance level of $\alpha = 0.05$. Table 3 gives the results of this analysis, applied separately to the two sets of instances of size `50x20` and `100x20`. We indicate in bold face the best strategy (the one having the lowest sum of ranks) and those that are not significantly different from the best one. In both cases, the best ranked algorithm is significantly better than the rest. However, the best

Table 2. Solution costs obtained by the three automatic SLS algorithms and soa-IG on the test instances

Instances		ALS1		ALS2		ALS3		soa-IG	
		Best	Avg	Best	Avg	Best	Avg	Best	Avg
50x20	ta051	26589	26806.2	26589	26824.9	26589	**26756.1**	26589	26899.6
	ta052	24059	**24273.2**	24183	24443.2	24096	24390.8	24059	24333.5
	ta053	32897	33307.8	32910	**33183.7**	32910	33206.8	32897	33634.6
	ta054	31221	**31470.2**	31221	31663.3	31221	31572.7	31221	31488.9
	ta055	21908	**21936.2**	21908	21948.3	21908	21975.9	21908	22094.7
	ta056	16181	**16516.4**	16189	16711.6	16189	16740.7	16181	16556.7
	ta057	23610	**23869**	23610	23990.4	23610	23953.9	23974	24211.2
	ta058	22091	22207.7	22091	**22131.9**	22091	22166.8	22091	22262.1
	ta059	27333	**27521.3**	27333	27685.1	27333	27573.1	27333	27577.9
	ta060	63078	63286.3	63078	63235.9	63078	**63179.9**	63117	63456
100x20	ta081	409667	416932.6	409052	415941.5	409697	415306.3	415388	422625
	ta082	325472	329803.8	324060	329161.3	323133	**327466.6**	328014	334437.1
	ta083	492455	496922.6	490669	495669.7	487450	**494569.8**	500142	505772
	ta084	553249	562380.8	549600	**558824.5**	551359	559419.9	550536	568600.5
	ta085	472546	480861	474883	481147.7	471402	**479941.3**	481576	487291.6
	ta086	484905	490357.9	480575	489379	480926	**488144.7**	484892	496511.1
	ta087	378567	382931.6	374208	384024.5	376694	**382277.9**	382122	388511.8
	ta088	389673	395809.4	389475	396729.7	385029	**394056.2**	394226	402836.5
	ta089	562109	571495.2	560593	570489.5	561570	**568465.7**	569769	582829.9
	ta090	459232	464206.4	454597	**461262.9**	457784	462177.3	464264	471961.3

Table 3. Statistical analysis based on the Friedman-test. The second column gives the minimum difference in the sum of ranks that is statistically significant (ΔR_α), given a significance level of $\alpha = 0.05$. For each instance set, algorithms are ordered according to the rank obtained. The numbers in parenthesis are the difference of ranks relative to the best algorithm. The algorithm that is significantly better than the other ones is indicated in bold face.

Instances ΔR_α	**Algorithms** (ΔR)
50x20 57.92	**ALS1**, ALS3 (75), ALS2 (115.5), soa-IG (221.5)
100x20 47.04	**ALS3**, ALS2 (100), ALS1 (143), soa-IG (573)

algorithm is different in each case. Notably, soa-IG is consistently ranked as the worst by a large margin, especially on the 100x20 instances. These results are consistent with the observations above. Therefore, our conclusion is that the current state of the art can be matched and outperformed by the automatically generated algorithms.

6 Conclusion

Hybridizations of stochastic local search (SLS) methods that manipulate a single solution at each step of the search are among the most effective non exact algorithms for tackling hard combinatorial optimization problems [11]. Nonetheless, designing such hybrid SLS algorithms is an arduous task that requires a significant amount of effort in implementation, experimental setup and analysis. In practice, algorithm designers only consider a few ad-hoc combinations of SLS

algorithms. In this paper, we have shown that the process of designing such algorithms can become mostly automatic. In particular, we have proposed a unified and practical generalized local search (GLS) structure.

We have shown that the GLS structure *unifies* the formulation of various simple SLS methods and their possible combinations (hybridizations) into a single structure. In fact, the best algorithms generated when applied to the *PFSP-WT* are complex hybrids that combine ILS with IG, VNS and even a different ILS. Our proposal is also *practical*, in the sense that it generates algorithms that are as efficient as if they were hand-crafted by a competent programmer. Two properties of our proposal are key for obtaining such efficiency. First, instead of a complex algorithmic framework with many parameters, our system generates specific algorithms from a grammar description of the GLS structure. These specific algorithms, which contain only a small fraction of all the algorithmic components available in the grammar, are generated directly as `C++` code and compiled. Second, our grammar description allows algorithm designers to include problem-specific components, which are often crucial for obtaining high-performing SLS algorithms. The system takes care of combining, testing and selecting (or discarding) these problem-specific components among all the available algorithmic components.

We have evaluated our proposal by applying our method to the *PFSP-WT* comparing it the a state-of-the-art IG algorithm. Our experimental results showed that the three automatically generated SLS algorithms are able to outperform it on well-known *PFSP-WT* instances from the literature.

Despite this initial success, there is considerably room for improvement. First, we used a moderate amount of computing effort for the automatic generation of algorithms. Therefore, the real potential of the proposed GLS structure may not have been exhausted. Second, the definition of the GLS structure is a work in progress. Future work will extend the GLS structure presented here, and its implementation, to include additional SLS algorithms, for example, greedy randomized adaptive search procedure (GRASP) [8] and Tabu Search [9]. Additionally, possible re-designs of specific aspects of the GLS structure and its implementation may be considered once more computational results are gathered. Third, additional techniques may prove to be useful for avoiding too complex SLS algorithm designs. Finally, we will apply the proposed method to other hard combinatorial problems, with the aim of improving the state of the art.

Acknowledgments. This work was carried out during the tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement no 246016.

References

1. Balaprakash, P., Birattari, M., Stützle, T.: Improvement strategies for the F-race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.) HM 2007. LNCS, vol. 4771, pp. 108–122. Springer, Heidelberg (2007)

2. Burke, E.K., Hyde, M.R., Kendall, G.: Grammatical evolution of local search heuristics. IEEE Transactions on Evolutionary Computation 16(7), 406–417 (2012)
3. Cahon, S., Melab, N., Talbi, E.G.: ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10(3), 357–380 (2004)
4. Conover, W.J.: Practical Nonparametric Statistics, 3rd edn. John Wiley & Sons, New York (1999)
5. Du, J., Leung, J.Y.T.: Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research 15(3), 483–495 (1990)
6. Dubois-Lacoste, J.: A study of Pareto and Two-Phase Local Search Algorithms for Biobjective Permutation Flowshop Scheduling. Master's thesis, IRIDIA, Université Libre de Bruxelles, Belgium (2009)
7. Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: A hybrid TP+PLS algorithm for bi-objective flow-shop scheduling problems. Computers & Operations Research 38(8), 1219–1236 (2011)
8. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–113 (1995)
9. Glover, F.: Tabu search – Part I. INFORMS Journal on Computing 1(3), 190–206 (1989)
10. Hansen, P., Mladenovic, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)
11. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco (2005)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
13. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science 220, 671–680 (1983)
14. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium (2011)
15. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search: Framework and applications. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, ch. 9, 2nd edn., pp. 363–397. Springer (2010)
16. Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T.: From grammars to parameters: Automatic iterated greedy design for the permutation flow-shop problem with weighted tardiness. In: 7th International Conference on Learning and Intelligent Optimization, LION 7. LNCS. Springer (to appear, 2013)
17. Mckay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: A survey. Genetic Programming and Evolvable Machines 11(3-4), 365–396 (2010)
18. Minella, G., Ruiz, R., Ciavotta, M.: A review and evaluation of multiobjective algorithms for the flowshop scheduling problem. INFORMS Journal on Computing 20(3), 451–471 (2008)
19. Nawaz, M., Enscore Jr., E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA 11(1), 91–95 (1983)
20. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization – Algorithms and Complexity. Prentice Hall, Englewood Cliffs (1982)
21. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 177(3), 2033–2049 (2007)
22. Taillard, É.D.: Benchmarks for basic scheduling problems. European Journal of Operational Research 64(2), 278–285 (1993)

GRASP and Variable Neighborhood Search for the Virtual Network Mapping Problem*

Johannes Inführ and Günther R. Raidl

Vienna University of Technology
Favoritenstraße 9–11/1861, 1040 Vienna, Austria
{infuehr,raidl}@ads.tuwien.ac.at

Abstract. Virtual network mapping considers the problem of fitting multiple virtual networks into one physical network in a cost-optimal way. This problem arises in Future Internet research. One of the core ideas is to utilize different virtual networks to cater to different application classes, each with customized protocols that deliver the required Quality-of-Service. In this work we introduce a Greedy Randomized Adaptive Search Procedure (GRASP) and Variable Neighborhood Search (VNS) algorithm for solving the Virtual Network Mapping Problem. Both algorithms make use of a Variable Neighborhood Descent with ruin-and-recreate neighborhoods. We show that the VNS approach significantly outperforms the previously best known algorithms for this problem.

Keywords: Virtual Network Mapping, Variable Neighborhood Search, GRASP.

1 Introduction

The Internet as it exists today suffers from ossification [20]. It is hard or even impossible to introduce new technologies, even though they would bring large improvements in Quality-of-Service. Examples for such technologies include Explicit Congestion Notification [21] or Differentiated Services (a Quality-of-Service framework) [5]. The most prominent example is probably IPV6 [9], which was first specified in 1998 and is still not implemented completely, despite the obvious demand. The main reason why upgrades are so problematic is that changes to the underlying technology, such as employed protocols, would be very disruptive for the users who depend on the Internet working exactly as it does now.

Network virtualization has been identified as a central technology for alleviating the ossification of the Internet in the Future Internet research community [3,4]. It is already being successfully employed in scientific network testbeds such as GENI [1], G-Lab [25] or PlanetLab [8]. In this context, network virtualization is used to share large scale research networks among different research groups. Each group uses its own virtual network to perform experiments, without fear of interference by other groups even though they are using the same

* This work has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT10-027.

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 159–173, 2013.

underlying physical network. With network virtualization, changes to the Internet technology can be employed in an incremental and non-disruptive manner. Old and new technologies can coexist in different virtual networks. However, virtualization does not have to be just a device to gradually move from one technology to the next. Having multiple virtual networks in place could be the preferred state, because it allows specialization of the virtual networks to better cater to the requirements of different application classes. For a survey on network virtualization, its application and available technologies, see [6].

The Virtual Network Mapping Problem (VNMP) arises in this context. The multitude of virtual networks (VNs), each with different characteristics and protocols, still has to be realized by utilizing the available physical network infrastructure (the substrate) and the available resources. Additionally, VNs have to be realized in such a way that they fulfill the required specification with respect to Quality-of-Service parameters such as available communication bandwidth and delay.

In this work, we introduce a Greedy Randomized Adaptive Search Procedure (GRASP) and a Variable Neighborhood Search (VNS) algorithm for solving the VNMP. Instead of simple Local Search, both algorithms make use of a Variable Neighborhood Descent with ruin-and-recreate neighborhoods [24]. We will show that the VNS approach significantly outperforms the previously best known algorithms from the literature.

The rest of this work is structured as follows: Section 2 defines the VNMP formally, followed by a discussion of the relevant background in Section 3. The GRASP and VNS approaches are presented in Sections 4 and 5. Section 6 contains the results of the experimental evaluation of our proposed algorithms and their comparison to other algorithms presented in the literature. We conclude in Section 7.

2 The Virtual Network Mapping Problem

Three types of information are required to fully specify a VNMP: The substrate network (i.e. the physical network) with its available resources, the virtual networks (VNs) that need to be realized with their resource requirements and the location constraints between the nodes of the VNs and the substrate nodes.

A directed graph $G = (V, A)$ with node set V and arc set A models the substrate network. Each substrate node $i \in V$ has a CPU power of $c_i \in \mathbb{N}^+$. This CPU power is used by the VN nodes mapped to i, but also by all implementations of VN arcs traversing it. We assume that routing one unit of bandwidth (BW) requires one unit of CPU power. It is inconsequential whether this BW is simply relayed or has originated from a virtual node mapped to the substrate node. Even if both, the sending and receiving virtual node are mapped to the same substrate node, CPU capacity is required to route traffic from one virtual node to the other. Substrate arcs $e \in A$ have a BW capacity $b_e \in \mathbb{N}^+$ and a delay $d_e \in \mathbb{N}^+$ that is incurred when data is sent across e.

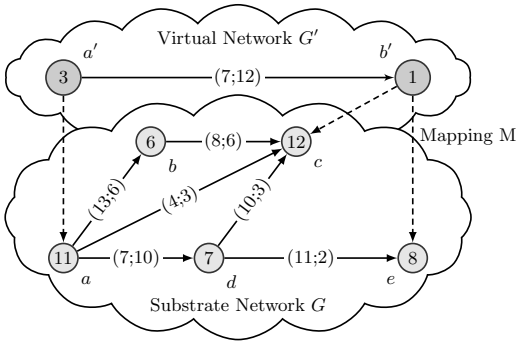


Fig. 1. An illustrative VNMP instance

The disconnected components of another directed graph $G' = (V', A')$ model the virtual networks. Each node $k \in V'$ requires a CPU power $c_k \in \mathbb{N}^+$. Each arc $f \in A'$ has a bandwidth requirement $b_f \in \mathbb{N}^+$ and a maximum allowed delay $d_f \in \mathbb{N}^+$.

The substrate nodes that a virtual node k is allowed to use are defined by the set $M \subseteq V' \times V$. By $s(a)$ and $t(a)$, $\forall a \in A \cup A'$, we denote arc a's source and target nodes, respectively.

Two components are required to specify a valid VNMP solution: A mapping $m : V' \to V$ such that $(k, m(k)) \in M,\ \forall k \in V'$ and a substrate path $P_f \subseteq A$ from $m(s(f))$ to $m(t(f))$ for every $f \in A'$ that does not exceed d_f. The total CPU load on each $i \in V$ (caused by virtual nodes hosted on i and traversing BW) is not allowed to exceed c_i and the BW capacities b_e have to be respected, too.

The objective of the VNMP is to minimize the total substrate usage cost. A price of $p_i^V \in \mathbb{N}^+$ has to be paid for every $i \in V$ that hosts at least one virtual node. Using a substrate arc $e \in A$ costs $p_e^A \in \mathbb{N}^+$. The sum of incurred node and arc usage costs is the total substrate usage cost C_{u}.

Already finding a valid solution to the VNMP is NP-hard [2]. Therefore we cannot expect an optimization approach to always be able to find valid solutions (which may not even exist) within practical time. To get around this problem, we allow the possibility of adding CPU power to the substrate, each additional unit costing C^{CPU}, and increasing the available BW on substrate arcs, costing C^{BW} per additional unit. The sum of the costs for additional resources is the additional resource cost C_{a}. For valid solutions to the VNMP, $C_{\mathrm{a}} = 0$. We call a VNMP instance solved if a valid solution could be obtained. We set $C^{\mathrm{CPU}} = 1$ and $C^{\mathrm{BW}} = 5$ in this work, which are values we also used in [19], to reflect the fact that it is cheaper to add additional CPU capacity to a router than to increase the BW of a network link. We use C_{a} as primary objective that has to be minimized. Only if two solutions have the same C_{a} the one with lower C_{u} is preferred. This has the advantage of guiding solutions towards validity during search.

Figure 1 shows a simple VNMP instance. The virtual network G' contains two virtual nodes, showing their CPU requirement, and a virtual arc connecting them, labeled with its BW requirement and allowed delay. The substrate network G contains the physical network nodes showing their CPU resources and the available links between the nodes, labeled by their BW capacity and the delay that is incurred when data is transmitted across them. The dashed lines show M, i.e. the allowed locations of the virtual nodes. Usage costs have been omitted for clarity. This example has only one valid solution, as b' cannot actually be mapped to c, even though c has enough resources available. The path implementing the virtual connection cannot use b, because it does not have enough resources to route the required BW. The direct connection from a to c lacks the required BW capacity, and the path (a, d, c) incurs too much delay. So the only valid solution is to map a' to a, b' to e and use the path (a, d, e) to implement the virtual arc between a' and b'.

3 Background and Related Work

The Greedy Randomized Adaptive Search Procedure (GRASP) [10] is a metaheuristic for combinatorial optimization problems. It works by continually repeating two steps. The first step is the randomized greedy construction of a solution to the problem to be solved. A second step is applying a local improvement technique to the constructed solution. These two steps are repeated until a termination criterion (like runtime or number of iterations) is reached. The best found solution is the final result of GRASP. How the randomized greedy solution construction works is a central aspect of GRASP. It iteratively builds a solution by adding components that seem good (but not necessarily the best) according to a greedy criterion. All possible components are collected in a candidate list (CL). The restricted candidate list (RCL) is created from the CL, usually by selecting all components from CL that are good enough (only a limited deviation from the perceived best alternative) or by selecting the best ones until the RCL has a specified length. The actual component that is added to the solution is selected uniformly at random from the RCL. This procedure usually leads to promising and at the same time diversified solutions for local optimization. A comprehensive overview of GRASP can be found in [11,22]. For hybridization techniques see [12].

The General Variable Neighborhood Search (VNS) [13] algorithm is built around Variable Neighborhood Descent. In Variable Neighborhood Descent, a local search is performed systematically switching between a series of neighborhood structures until a solution is reached that is local optimal w.r.t. all neighborhood structures. VNS adds diversification by applying random moves, called shaking, in successively larger neighborhood structures to escape the basins of attraction of local optima. VNS is a very successful metaheuristic for combinatorial optimization problems, for more details and a survey of applications see [14].

The VNMP appears in the literature as Network Testbed Mapping [23], Virtual Network Embedding [7], Virtual Network Assignment [29] and Virtual

Network Resource Allocation [26]. Embedding virtual networks into a shared substrate is always the central problem. Differences arise with the considered resources. For example, the authors of [29] do not consider any resources explicitly, [23,26] use bandwidth and [7,27] add CPU power. We extend on the latter by also considering the interaction between routing and hosting virtual networks and supporting delay constraints for the virtual connections. There are different methods for constraining the allowed mapping locations of virtual nodes present in the literature. The nodes of a VN might be required to be located at different substrate nodes [29], the mapping might be predetermined [26] or a a distance limit between a virtual node and the substrate node that hosts it might be in effect [7]. The VNMP model we utilize in this work can represent all of these variants and is thus most flexible. As for the paths used to implement a virtual connection, there are two approaches: using a single path or using multiple paths. Using multiple paths [7,28]) has the advantage that the problem of finding the implementations for the virtual connections becomes polynomially solvable when bandwidth may be arbitrarily split. However, using multiple paths to implement a virtual connection makes its observed behavior much more erratic, as it depends on multiple physical links. Therefore, we utilize here only a single path to implement virtual connections.

4 GRASP

A key component for a well working GRASP approach is the randomized greedy heuristic. We use the best identified construction heuristic configuration from our previous work in [19] as basis for randomization. In a nutshell, as long as virtual arcs are implementable (its source and target node have been mapped), the virtual arc f with the smallest fraction of d_f to shortest possible delay between $m(s(f))$ to $m(t(f))$ is implemented by the path with the least increase in C_{u} without increasing C_{a}. If no such virtual arc exists, the unmapped node with the highest total CPU requirement (CPU requirement of the virtual node and BW of connected virtual arcs) is selected from the VN that has the highest sum of total CPU requirements. It is mapped to the substrate node with the highest amount of free CPU capacity, ties are broken by using the amount of free incoming and outgoing bandwidth. Based on our previous work, we know that the substrate node selection strategy is the most influential for the overall performance of the construction heuristic. Therefore we concentrate on randomizing this strategy and keep all other parts of the randomized construction heuristic deterministic. We introduce a parameter $\alpha \in [0, 1]$ that controls the level of randomization. When selecting a suitable substrate node for a virtual node, we collect a list of possible targets sorted by the available CPU and BW, the candidate list. Let $f_{\mathrm{Best}}^{\mathrm{CPU}}$ denote the free CPU capacity and $f_{\mathrm{Best}}^{\mathrm{BW}}$ the free BW capacity of the node that would have been selected by the deterministic strategy. We build the restricted candidate list by selecting all nodes i with $f_i^{\mathrm{CPU}} \geq \alpha f_{\mathrm{Best}}^{\mathrm{CPU}} \wedge f_i^{\mathrm{BW}} \geq \alpha f_{\mathrm{Best}}^{\mathrm{BW}}$. If $f_{\mathrm{Best}}^{\mathrm{CPU}}$ or $f_{\mathrm{Best}}^{\mathrm{BW}}$ is negative (i.e. more resources are used than actually are available), α is replaced by $2-\alpha$ in the relevant acceptance criterion. The actual mapping target is chosen uniformly at random from the restricted candidate list.

After the randomized greedy solution is generated, a local improvement strategy is applied. For comparison purposes, we choose the same method the Genetic Algorithm (GA) presented in [18] uses. It is a Variable Neighborhood Descent approach based on three ruin-and-recreate [24] neighborhoods, which are searched in a first-improvement fashion. They remove a part of a solution and reconstruct it using a construction heuristic designed for this rebuilding task (CH3 from [19]). The following short description of the used neighborhoods skips this rebuilding step. The first neighborhood is the set of all solutions reachable by removing the mapping of a single virtual node. The second neighborhood is the set of all solutions reachable by clearing a substrate arc, which means that all virtual arc implementations using this arc are removed. The third neighborhood is the set of all solutions reachable by clearing a substrate node. This means that all virtual nodes mapped to the substrate node, and all virtual arc implementations using this node, are removed from the solution. This neighborhood configuration was selected because it offers a good balance between required runtime and solution quality. Also, preliminary experiments showed that using simple Local Search is not competitive. In this work, we will call this configuration VND. We use VND without timelimit to improve solutions generated by the randomized greedy heuristic. If the found solution is better than the best solution found so far, we keep it. Then we repeat the randomized construction and improvement steps until the timelimit is reached, the best found solution is the result of GRASP.

5 VNS

Our proposed VNS algorithm uses a single type of shaking neighborhood in multiple configurations. Let this neighborhood be called $N^s(v)$, with $v \in [0, 1]$ as parameter controlling the shaking vigor. N^s is based on the idea of clearing substrate nodes. When $N^s(v)$ is applied to a VNMP solution, N^s randomly selects $\lceil v \cdot |V| \rceil$ substrate nodes. All virtual arc implementations that traverse the selected nodes are removed from the solution. All virtual nodes mapped to the selected substrate nodes are mapped to a substrate node that is allowed by M but not selected. If no such node exists, the mapping remains unchanged. The resulting solution is completed and improved by VND to create the final solution of one VNS iteration. During the execution of VNS we apply N^s with different values for v. The used values are determined by two parameters, the base neighborhood size v_b and the count of iterations that have not resulted in an improvement of the best found solution n_ni. At the beginning of a new iteration, $N^s(v_\mathrm{b} n_\mathrm{ni})$ is applied to the currently best found solution and the result is improved by VND. If the solution created in this manner is better than the currently best known solution, n_ni is reset to one, otherwise n_ni is increased by one. The upper limit for n_ni is n_max. If this value is exceeded, n_ni is reset to one. The largest shaking neighborhood applied during VNS is $N^s(v_\mathrm{b} n_\mathrm{max})$. Values for v_b and n_max have to be chosen such that $v_\mathrm{b} n_\mathrm{max} \leq 1$. The shaking and improvement steps are applied until the timelimit is reached. The initial solution

```
VNMPSolution best=initialize();
n_ni=1;
while(!terminate()){
      VNMPSolution candidate=shake(N^s(v_b n_ni),best);
      applyVND(candidate);

      if(candidate<best){ //New best solution found
            best=candidate;
            n_ni=1;
      }else{
            ++n_ni;
            if(n_ni>n_max)n_ni=1;
      }
}
return best;
```

Listing 1.1. VNS for the VNMP

for VNS is built by the same method as for GRASP, but without randomization. Listing 1.1 shows the general outline of the proposed VNS.

6 Results

The proposed GRASP and VNS algorithms have been tested on the instances available from [16]. This instance set contains VNMP instances from 20 to 1000 substrate nodes, with 30 instances of each size. Every instance includes 40 VNs that have to be implemented. The VNs have different properties to cover different use-cases, like high BW requirements for P2P applications or low delays for VoIP applications. Table 1 shows the main properties of the used instances, for more information see [19]. To analyze the behaviour of the proposed algorithms in different load cases, we also tested with instances from the instance set that had some of their VNs removed. A load of 0.5 means that only 50% of the available VNs were used. We considered load levels of 0.1, 0.5, 0.8 and 1. This results in

Table 1. Properties of the used VNMP instances: average number of substrate nodes (V) and arcs (A), virtual nodes (V') and arcs (A') and the average number of allowed map targets for each virtual node ($M_{V'}$)

Size	$\|V\|$	$\|A\|$	$\|V'\|$	$\|A'\|$	$\|M_{V'}\|$
20	20	40.8	220.7	431.5	3.8
30	30	65.8	276.9	629.0	4.9
50	50	116.4	398.9	946.9	6.8
100	100	233.4	704.6	1753.1	11.1
200	200	490.2	691.5	1694.7	17.3
500	500	1247.3	707.7	1732.5	30.2
1000	1000	2528.6	700.2	1722.8	47.2

a total of 840 VNMP instances, 120 for every size and 210 for every load level, so the results presented later in this section are averages of 120 or 210 runs respectively. All algorithms compared in this section have been run on one core of an Intel Xeon E5540 multi-core system with 2.53 GHz and 3 GB RAM per core. A CPU-time limit of 200 seconds was applied for sizes up to 100 nodes, 500 seconds for larger instances. All reported results of statistical tests are based on a paired Wilcoxon signed rank test with a 5% level of significance.

Our design goal for the proposed algorithms was finding good solutions to the VNMP with respect to the objective function. This is significantly different from finding just valid solutions. The following results will show cases where algorithms find better results on average while solving fewer instances. To recognize the algorithms that create good solutions, we cannot look for low substrate usage costs C_{u}, because higher values might be better if the additional resource costs C_{a} are lower. Therefore we use the following ranking procedure as introduced in [19]. The achieved results of each algorithm under comparison for a specific instance are sorted in ascending order. The algorithm that achieved the best results gets rank 0, the second best rank 1 and so on. Algorithms with the same result get the same rank. To have a value that is comparable across different instances, the rank is divided by the maximum rank to create the relative rank R_{rel}. If all results are the same (i.e. the highest rank is zero), R_{rel} is zero as well for all algorithms. A R_{rel} of 0.1 means that the results of the algorithm in question are within the top 10% of compared algorithms.

Section 6.1 compares the performance of the GRASP approach for different values of α, Section 6.2 analyzes the performance of the VNS approach for different shaking neighborhood configurations and Section 6.3 shows a comparison of the best GRASP and VNS approach with other approaches from the literature.

6.1 GRASP

To evaluate the influence of α on the GRASP approach, we tested values for α from 0 (completely random initial solution) to 0.9 in 0.1 increments and 0.99 (very similar initial solutions). The average performance depending on the instance size can be seen in Table 2. The symbol next to the relative rank shows the relation to the best R_{rel} based on a statistical test, $>$ means that the reported R_{rel} is significantly larger than the best, $=$ means that no significant difference could be observed. Immediately visible is the tendency of the best α value to rise with the instance size. For size 20, $\alpha \in [0, 0.4]$ yields the best results w.r.t. R_{rel}, while for size 1000 $\alpha \in [0.5, 0.8]$.

The reason for this behaviour is that for small instances, the randomized construction heuristic does not have to make as many random choices as for the larger instance sizes. Therefore, to get the same search space coverage w.r.t initial solutions, α has to be small for small instances. The results for the larger instances show that if α is too small, then the performance degrades, because the initial solution is far too random. Another contributing factor is that VND takes longer to optimize a very random initial solution, as can be seen by the iteration counts. Therefore, fewer iterations can be performed in the same amount of time.

Table 2. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different values of α per instance size

	Size	GR-0.00	GR-0.10	GR-0.20	GR-0.30	GR-0.40	GR-0.50	GR-0.60	GR-0.70	GR-0.80	GR-0.90	GR-0.99
R_{rel}	20	0.278 =	0.216 =	0.219 =	0.235 =	**0.215** =	0.331 >	0.384 >	0.445 >	0.516 >	0.618 >	0.842 >
	30	0.431 >	0.337 >	0.318 =	0.288 =	**0.266** =	0.341 >	0.337 >	0.461 >	0.551 >	0.626 >	0.826 >
	50	0.549 >	0.512 >	0.463 >	0.362 >	**0.311** =	0.329 =	0.402 >	0.484 >	0.536 >	0.640 >	0.868 >
	100	0.870 >	0.665 >	0.468 >	0.362 =	**0.319** =	0.359 =	0.410 >	0.384 >	0.460 >	0.554 >	0.692 >
	200	0.889 >	0.737 >	0.488 >	0.361 >	**0.301** =	0.301 =	0.313 =	0.339 =	0.436 >	0.531 >	0.740 >
	500	0.856 >	0.718 >	0.511 >	0.449 >	0.381 >	**0.306** =	0.325 =	0.358 >	0.390 >	0.488 >	0.617 >
	1000	0.902 >	0.665 >	0.623 >	0.529 >	0.425 >	0.354 =	**0.338** =	0.341 =	0.375 =	0.426 >	0.470 >
Its.	20	1998	2323	2641	2916	3142	3291	3453	3677	3751	3836	3897
	30	780	896	1034	1153	1248	1399	1534	1588	1676	1667	1664
	50	282	329	390	442	481	497	531	553	565	567	566
	100	39	47	56	62	66	71	77	82	83	87	87
	200	45	54	66	78	90	98	103	112	116	127	129
	500	15	18	22	26	30	33	36	39	41	41	42
	1000	4	5	7	7	9	10	11	12	12	12	12
Solv.	20	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	99.2
[%]	30	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	100	100.0	100.0	100.0	99.2	98.3	97.5	97.5	99.2	99.2	97.5	95.8
	200	95.8	99.2	99.2	99.2	98.3	97.5	97.5	96.7	95.8	95.0	91.7
	500	71.7	80.8	81.7	81.7	76.7	80.8	78.3	75.8	77.5	75.0	70.0
	1000	34.2	58.3	57.5	55.8	60.0	55.0	54.2	55.8	55.0	55.0	55.8

Note that for finding valid solutions, low α values seem to be beneficial, even for large instances.

Table 3 shows the influence of α for different load cases. Again we can observe that higher values of α allow for more iterations, but they do not lead to improved performance for high load. Instead, a value for $\alpha \in [0.4, 0.5]$ seems to be best suited when performance at a specific load level across different sizes is most important. Low α values are again beneficial for finding valid solutions.

Based on these results, we select the GRASP approach with $\alpha = 0.4$ for further comparisons.

6.2 VNS

To analyze the influence of different shaking neighborhood configurations, we tested $n_{\max} \in 2, 5, 10$ and $v_{\text{b}} \in 0.01, 0.05, 0.1$ to cover the range from very small changes with few shaking neighborhoods (i.e. few different configurations for N^s) to large changes with a lot of neighborhoods. Table 4 shows the performance of different neighborhood configurations based on instance size. The different configurations are labeled as "VNS-$n_{\max}$.v_{b}", e.g. VNS-2.05 uses $n_{\max} = 2$ and $v_{\text{b}} = 0.05$. We can see a similar behaviour to GRASP. For smaller sizes, large shaking neighborhoods are beneficial, while large instance sizes require small neighborhoods for the best levels of performance. Smaller shaking neighborhoods lead to an increased number of iterations in the same amount of time. Also note the similarity in number of iterations between VNS-5.05 and VNS-2.10, caused by the very similar maximum shaking neighborhood sizes. Indeed, between sizes

Table 3. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different values of α per load

	Load	GR-0.00	GR-0.10	GR-0.20	GR-0.30	GR-0.40	GR-0.50	GR-0.60	GR-0.70	GR-0.80	GR-0.90	GR-0.99
R_{rel}	0.10	0.520 >	0.413 >	0.330 >	**0.288** =	0.299 =	0.335 =	0.419 >	0.495 >	0.573 >	0.683 >	0.766 >
	0.50	0.744 >	0.586 >	0.445 >	0.384 >	**0.307** =	0.313 =	0.324 =	0.354 =	0.473 >	0.574 >	0.773 >
	0.80	0.782 >	0.634 >	0.521 >	0.430 >	**0.305** =	0.318 =	0.312 =	0.350 >	0.397 >	0.482 >	0.695 >
	1.00	0.682 >	0.568 >	0.470 >	0.377 =	**0.356** =	0.360 =	0.380 =	0.407 >	0.423 >	0.482 >	0.654 >
Its.	0.10	1529	1737	1990	2193	2348	2490	2638	2781	2871	2917	2947
	0.50	108	142	173	208	239	267	298	322	330	333	336
	0.80	80	105	120	137	155	169	180	190	192	195	201
	1.00	90	114	126	139	151	160	167	173	175	176	173
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	0.50	90.0	97.1	97.6	97.1	97.1	96.7	96.7	96.7	96.2	95.7	95.7
	0.80	81.0	90.5	89.0	89.5	90.0	87.6	87.6	89.0	88.6	86.7	85.2
	1.00	72.9	77.1	78.1	76.7	74.8	76.2	74.3	72.9	73.8	73.3	69.0

Table 4. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different shaking neighborhood configurations per instance size

	Size	VNS-2.01	VNS-5.01	VNS-10.01	VNS-2.05	VNS-5.05	VNS-10.05	VNS-2.10	VNS-5.10	VNS-10.10
R_{rel}	20	0.389 >	0.436 >	0.396 >	0.416 >	0.240 >	0.244 >	0.340 >	0.198 =	**0.163** =
	30	0.402 >	0.404 >	0.394 >	0.344 >	0.305 >	0.293 >	**0.229** =	0.304 >	0.283 =
	50	0.457 >	0.390 =	0.356 =	0.396 =	**0.333** =	0.368 =	0.338 =	0.390 =	0.472 >
	100	0.432 >	0.371 =	0.372 =	**0.349** =	0.486 >	0.506 >	0.425 >	0.578 >	0.630 >
	200	0.460 >	0.360 =	**0.316** =	0.376 >	0.490 >	0.554 >	0.479 >	0.591 >	0.685 >
	500	0.467 >	0.420 >	**0.344** =	0.395 =	0.500 >	0.520 >	0.547 >	0.658 >	0.624 >
	1000	0.468 >	**0.339** =	0.366 =	0.462 >	0.459 >	0.518 >	0.579 >	0.596 >	0.665 >
Its.	20	7327	7327	7345	7325	6796	5812	6822	5701	4742
	30	3721	3699	3670	3590	3132	2577	3156	2511	2010
	50	1766	1758	1664	1583	1321	1042	1327	1001	786
	100	415	389	346	311	237	181	233	169	127
	200	504	450	399	349	270	208	260	192	147
	500	157	143	124	110	85	67	83	62	48
	1000	51	44	39	33	25	20	25	18	14
Solv.	20	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	30	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
	100	98.3	99.2	100.0	100.0	99.2	100.0	100.0	100.0	99.2
	200	93.3	95.8	97.5	95.8	95.0	99.2	99.2	98.3	100.0
	500	74.2	74.2	76.7	79.2	76.7	77.5	80.0	76.7	76.7
	1000	55.8	55.0	59.2	53.3	55.0	54.2	55.0	55.0	53.3

50 and 500, there is no significant difference between the two configurations. Larger shaking neighborhoods seem to increase the chance of finding valid solutions. The results indicate that increasing the shaking neighborhood size in multiple small steps works better than few large steps. This can be seen with configurations that have the same maximum shaking neighborhood size. VNS-10.01 and VNS-2.05 show no significant difference in R_{rel}, except for sizes 200 and 1000 where using smaller steps is significantly better. The difference is more pronounced for VNS-10.05 and VNS-5.10. Until size 50 there is no difference in performance, for larger instances using smaller steps is significantly better.

Table 5. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) and fraction of solved instances (Solv.) in percent for different shaking neighborhood configurations per load

	Load	VNS-2.01	VNS-5.01	VNS-10.01	VNS-2.05	VNS-5.05	VNS-10.05	VNS-2.10	VNS-5.10	VNS-10.10
R_{rel}	0.10	0.393 >	0.289 =	0.253 =	0.242 =	**0.239** =	0.250 =	0.250 =	0.304 >	0.390 >
	0.50	0.464 >	0.408 =	**0.360** =	0.407 >	0.415 >	0.458 >	0.436 >	0.486 >	0.516 >
	0.80	0.446 =	0.451 =	**0.408** =	0.467 >	0.471 >	0.479 >	0.486 >	0.559 >	0.535 >
	1.00	0.454 =	**0.407** =	0.432 =	0.449 =	0.482 >	0.528 >	0.506 >	0.546 >	0.572 >
Its.	0.10	6220	6176	6100	5992	5451	4640	5472	4542	3744
	0.50	924	905	877	853	712	550	708	526	410
	0.80	483	477	458	444	360	274	367	262	204
	1.00	339	335	329	312	257	196	256	186	141
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
[%]	0.50	95.2	96.7	98.1	96.7	97.1	97.1	97.6	97.1	96.2
	0.80	87.1	87.1	88.6	85.7	86.7	87.1	88.6	87.1	88.6
	1.00	72.9	72.9	75.2	76.7	73.8	76.2	76.2	75.7	74.8

The influence of the shaking neighborhood configuration across different load cases can be seen in Table 5. Small shaking neighborhoods lead to the best performance. Load 0.10 is an exception, as larger shaking neighborhoods achieve the best results. As for the configurations with the same maximum shaking neighborhood size, smaller steps are an significant advantage for half of the load cases.

Based on these results, we chose VNS-10.01 for further comparison.

6.3 Comparison

In this section, we compare our proposed algorithms GR-0.4 and VNS-10.01 with approaches from the literature. These are GA-D-VND, the Genetic Algorithm for the VNMP introduced in [18], B-VND, the Variable Neighborhood Descent algorithm with the best performance with respect to R_{rel} from [19] and FLOW, a multicommodity-flow based integer linear programming formulation presented in [17] with small modifications to match the VNMP model used in this work. FLOW was solved by CPLEX 12.4 [15]. We also compare to VND, the Variable Neighborhood Descent algorithm used within GR-0.4, VNS-10.01 and GA-D-VND. The timelimits used in [18] were the same as the ones used in this work. The results of VND and B-VND had a timelimit of 1000 seconds and FLOW had 10000 seconds. Note that we do not directly compare FLOW to the other algorithms, because it can fail to generate any solution to a VNMP instance due to runtime or memory limits. This is true starting with instances of size 50, and for size 1000 FLOW only generates a solution for 30 out of 120 instances. However, for instances with a result generated by FLOW, this result was used for the calculation of R_{rel}. For comparison purposes, missing results were treated as $R_{\text{rel}} = 1$ and as instance that could not be solved, so these values can be directly compared with the other reported results. The reported values for the average runtime and C_{a} are only based on instances where FLOW generated a solution and are therefore not directly comparable to the other results.

Table 6. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) or runtime, fraction of solved instances (Solv.) in percent and average C_{a} for different solution methods per instance size

	Size	GR-0.4	VNS-10.01	GA-D-VND	VND	B-VND	FLOW
R_{rel}	20	0.476 >	0.222 =	**0.192** =	0.912 >	0.753 >	0.000
	30	0.518 >	**0.222** =	0.241 =	0.920 >	0.705 >	0.000
	50	0.598 >	**0.214** =	0.275 >	0.930 >	0.696 >	0.029
	100	0.606 >	**0.187** =	0.368 >	0.916 >	0.564 >	0.362
	200	0.577 >	**0.197** =	0.413 >	0.859 >	0.372 >	0.655
	500	0.628 >	0.489 >	0.538 >	0.846 >	**0.171** =	0.750
	1000	0.623 >	0.592 >	0.589 >	0.569 >	**0.228** =	0.817
Its. / t[s]	20	3142	7345	8185	0.2	0.4	131.2
	30	1248	3670	3899	0.7	1.3	1338.8
	50	481	1664	1663	2.1	4.2	2832.1
	100	66	346	314	16.0	29.7	6117.2
	200	90	399	333	40.2	119.7	7140.3
	500	30	124	94	126.6	605.1	3211.1
	1000	9	39	23	397.1	828.1	9114.5
Solv. [%]	20	100.0	100.0	100.0	96.7	97.5	100.0
	30	100.0	100.0	100.0	100.0	100.0	100.0
	50	100.0	100.0	100.0	99.2	98.3	97.5
	100	98.3	100.0	100.0	95.0	97.5	64.1
	200	98.3	97.5	95.8	90.0	98.3	35.0
	500	76.7	76.7	76.7	73.3	90.8	25.0
	1000	60.0	59.2	58.3	57.5	61.7	18.3
C_{a}	20	0.0	0.0	0.0	13.1	4.5	0.0
	30	0.0	0.0	0.0	0.0	0.0	0.0
	50	0.0	0.0	0.0	4.9	2.1	0.0
	100	2.5	0.0	0.0	6.3	3.3	19142.5
	200	0.4	6.1	3.0	19.0	1.0	71648.7
	500	47.1	68.5	77.3	97.6	13.9	3413.8
	1000	245.5	311.2	215.9	184.1	198.9	3952.2

Note that we show the average runtime only for VND, B-VND and FLOW, since the other algorithms were run until the timelimit was reached, so we show the performed iterations for them instead. For reference, the average runtime when considering different load cases is 328.5 seconds.

Table 6 shows the performance of the compared algorithms in relation to each other. It can be seen that the results achieved by the GR-0.4 are disappointing. The GRASP approach is significantly outperformed by the VNS and GA algorithms. However, using GRASP around VND is significantly better than using VND alone, except for size 1000, where both perform equally well. B-VND can only be beaten or matched up to size 100, then B-VND achieves significantly better results. The VNS approach works far better, achieving the best solutions

Table 7. Average relative rank R_{rel} and its relation to the best result, average number of iterations (Its.) or runtime, fraction of solved instances (Solv.) in percent and average C_{a} for different solution methods load

	Load	GR-0.4	VNS-10.01	GA-D-VND	VND	B-VND	FLOW
R_{rel}	0.10	0.497 >	**0.215** =	0.315 >	0.876 >	0.528 >	0.045
	0.50	0.616 >	**0.294** =	0.384 >	0.913 >	0.450 >	0.393
	0.80	0.602 >	**0.333** =	0.397 >	0.841 >	0.484 >	0.517
	1.00	0.586 >	**0.371** =	0.400 =	0.771 >	0.532 >	0.538
Its. /	0.10	2348	6100	6354	5.6	41.7	1946.6
t[s]	0.50	239	877	1016	50.2	218.3	3216.1
	0.80	155	458	537	111.2	316.2	4441.3
	1.00	151	329	385	166.0	331.6	5668.0
Solv.	0.10	100.0	100.0	100.0	100.0	100.0	95.7
[%]	0.50	97.1	98.1	97.1	95.7	99.0	61.0
	0.80	90.0	88.6	88.6	85.7	91.9	48.6
	1.00	74.8	75.2	74.8	68.1	77.1	46.2
C_{a}	0.10	0.0	0.0	0.0	0.0	0.0	567.3
	0.50	0.5	6.9	0.6	0.7	0.1	4306.8
	0.80	18.3	29.6	21.4	30.0	11.4	20967.5
	1.00	150.0	183.9	147.3	155.0	116.3	43651.8

for sizes 30 to 200. For size 20, the GA approach works marginally better. Keep in mind however, that we selected a shaking configuration for the VNS that was significantly worse for the smallest instance sizes than the alternatives, so it should be possible to at least match the GA with a different configuration. For the two largest sizes, VNS is beaten by B-VND, partly because the B-VND had more runtime available (and also made use of it) as evidenced by the average runtimes. Also, it is not a coincidence that there is no significant difference between the GRASP, VNS, GA and VND approaches for size 1000. They all use VND as local improvement strategy, and as can be seen by the iteration count, not enough iterations could be performed to reap the benefits of the more involved heuristics within the available runtime. FLOW generates the best results for sizes 20 to 50, but based on the runtimes it is only competitive for size 20. Also note the quick degradation of the number of solved instances and the average C_{a} compared to the heuristic approaches.

For solving instances at a specific load level, Table 7 shows that the VNS approach is the best choice across all load levels, achieving significantly better results than all of the other compared algorithms. There is no reason to use GR-0.4, it is matched or outmatched by B-VND within the same or lower runtime. For the least challenging problem class (load of 0.1), FLOW again achieves better results than the heuristics. Note however that it does not even find valid solutions for all instances in this class while requiring a lot more runtime.

7 Conclusions

In this work, we have presented a GRASP and VNS algorithm for solving the Virtual Network Mapping Problem. We have shown that the VNS algorithm produces significantly better results than the GA and VND approaches previously introduced. It also compares favourably against an integer linear programming approach. Based on the presented results, we can conclude that the main idea of VNS (successively larger random moves away from local optima) works better than learning from a set of good solutions (GA) or improving good random solutions (GRASP) for the Virtual Network Mapping Problem. The comparison is fair since the same improvement strategy (VND) was used, the parameters of all algorithms have been optimized and the same timelimits were employed.

The main direction for future work will be testing the presented algorithms in an online setting that allows for the arrival and departure of virtual networks. In particular, the fact that GA produces a set of good solutions instead of a single one might prove useful.

References

1. GENI.net Global Environment for Network Innovations, `http://www.geni.net`
2. Andersen, D.: Theoretical Approaches to Node Assignment. Unpublished Manuscript (December 2002), `http://www.cs.cmu.edu/~dga/papers/andersen-assign.ps`
3. Anderson, T., Peterson, L., Shenker, S., Turner, J.: Overcoming the Internet impasse through virtualization. Computer 38(4), 34–41 (2005)
4. Berl, A., Fischer, A., de Meer, H.: Virtualisierung im Future Internet. Informatik-Spektrum 33, 186–194 (2010)
5. Carlson, M., Weiss, W., Blake, S., Wang, Z., Black, D., Davies, E.: An architecture for differentiated services. IETF, RFC 2475 (1998)
6. Chowdhury, N., Boutaba, R.: A survey of network virtualization. Computer Networks 54(5), 862–876 (2010)
7. Chowdhury, N., Rahman, M., Boutaba, R.: Virtual network embedding with coordinated node and link mapping. In: INFOCOM 2009, pp. 783–791. IEEE (2009)
8. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. 33, 3–12 (2003)
9. Deering, S., Hinden, R.: Internet protocol, version 6 (ipv6) specification (December 1998), `http://tools.ietf.org/html/rfc2460`
10. Feo, T., Resende, M.: Greedy randomized adaptive search procedures. Journal of Global Optimization 6, 109–133 (1995)
11. Festa, P., Resende, M.: An annotated bibliography of grasp–part i: Algorithms. International Transactions in Operational Research 16(1), 1–24 (2009)
12. Festa, P., Resende, M.G.C.: Hybrid GRASP heuristics. In: Abraham, A., Hassanien, A.-E., Siarry, P., Engelbrecht, A. (eds.) Foundations of Computational Intelligence Volume 3. SCI, vol. 203, pp. 75–100. Springer, Heidelberg (2009)
13. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. European Journal of Operational Research 130(3), 449–467 (2001)

14. Hansen, P., Mladenović, N., Moreno Pérez, J.: Variable neighbourhood search: methods and applications. 4OR 6, 319–360 (2008)
15. IBM ILOG: CPLEX 12.4, http://www-01.ibm.com/software/integration/optimization/cplex-optimizer
16. Inführ, J., Raidl, G.R.: The Virtual Network Mapping Problem benchmark set, https://www.ads.tuwien.ac.at/projects/optFI/
17. Inführ, J., Raidl, G.R.: Introducing the virtual network mapping problem with delay, routing and location constraints. In: Pahl, J., Reiners, T., Voß, S. (eds.) INOC 2011. LNCS, vol. 6701, pp. 105–117. Springer, Heidelberg (2011)
18. Inführ, J., Raidl, G.R.: A memetic algorithm for the virtual network mapping problem. In: Lau, H., Van Hentenryck, P., Raidl, G. (eds.) The 10th Metaheuristics International Conference, MIC13, Singapore (2013), submitted for review
19. Inführ, J., Raidl, G.R.: Solving the Virtual Network Mapping Problem with Construction Heuristics, Local Search and Variable Neighborhood Descent. In: Middendorf, M., Blum, C. (eds.) EvoCOP 2013. LNCS, vol. 7832, pp. 250–261. Springer, Heidelberg (2013)
20. National Research Council: Looking Over the Fence at Networks. National Academy Press (2001)
21. Ramakrishnan, K.K., Floyd, S., Black, D.: The addition of explicit congestion notification (ECN) to IP. IETF, RFC 3168 (2001)
22. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: Handbook of Metaheuristics, pp. 219–249 (2003)
23. Ricci, R., Alfeld, C., Lepreau, J.: A solver for the network testbed mapping problem. SIGCOMM Comput. Commun. Rev. 33(2), 65–81 (2003)
24. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. Journal of Computational Physics 159(2), 139–171 (2000)
25. Schwerdel, D., Günther, D., Henjes, R., Reuther, B., Müller, P.: German-lab experimental facility. In: Berre, A.J., Gómez-Pérez, A., Tutschku, K., Fensel, D. (eds.) FIS 2010. LNCS, vol. 6369, pp. 1–10. Springer, Heidelberg (2010)
26. Szeto, W., Iraqi, Y., Boutaba, R.: A multi-commodity flow based approach to virtual network resource allocation. In: Global Telecommunications Conference, GLOBECOM 2003, vol. 6, pp. 3004–3008. IEEE (2003)
27. Yeow, W.L., Westphal, C., Kozat, U.: Designing and embedding reliable virtual infrastructures. In: Proceedings of the Second ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures, VISA 2010, pp. 33–40. ACM, New York (2010)
28. Yu, M., Yi, Y., Rexford, J., Chiang, M.: Rethinking virtual network embedding: substrate support for path splitting and migration. ACM SIGCOMM Computer Communication Review 38(2), 17–29 (2008)
29. Zhu, Y., Ammar, M.: Algorithms for assigning substrate network resources to virtual network components. In: Proceedings of the 25th IEEE International Conference on Computer Communications, pp. 1–12 (2006)

Hybrid Metaheuristics for the Far From Most String Problem

Daniele Ferone[1], Paola Festa[1], and Mauricio G.C. Resende[2]

[1] Department of Mathematics and Applications "R. Caccioppoli"
University of Napoli FEDERICO II, Italy
paola.festa@unina.it, danieleferone@gmail.com
[2] AT&T Labs Research, Florham Park, NJ, USA
mgcr@research.att.com

Abstract. Among the sequence selection and comparison problems, the *Far From Most String Problem* (FFMSP) is one of the computationally hardest with applications in several fields, including molecular biology where one is interested in creating diagnostic probes for bacterial infections or in discovering potential drug targets.

In this article, several hybrid metaheuristics are described and tested. Extensive comparative experiments on a large set of randomly generated test instances indicate that these randomized hybrid techniques are both effective and efficient.

Keywords: Computational biology, Molecular structure prediction, Protein and sequences alignment, Combinatorial optimization, Hybrid metaheuristics.

1 The Far From Most String Problem (FFMSP)

The FFMSP is one of the so called *string selection and comparison problems*, that belong to the more general class of problems known as *sequences consensus*, where a finite set of sequences is given and one is interested in finding their *consensus*, i.e. a new sequence that agrees as much as possible with all the given sequences. In other words, the objective is to determine a sequence called consensus, because it represents in some way all the given sequences. For the FFMSP, the objective is to find a sequence that is far from as many as possible sequences of a given set of sequences having all the same length.

To formally state the problem, the following notation is needed:

- an *alphabet* $\Sigma = \{c_1, c_2, \ldots, c_k\}$ is a finite set of elements, called *characters*;
- $s^i = (s^i_1, s^i_2, \ldots, s^i_m)$ is a sequence of length m ($|s^i| = m$) on Σ ($s^i_j \in \Sigma,\ j = 1, 2, \ldots, m$);
- given two sequences s^i and s^l on Σ such that $|s^i| = |s^l|$, $d_H(s^i, s^l)$ denotes their Hamming distance and is given by

$$d_H(s^i, s^l) = \sum_{j=1}^{|s^i|} \Phi(s^i_j, s^l_j), \tag{1}$$

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 174–188, 2013.

where s^i_j and s^l_j are the characters in position j in s^i and s^l, respectively, and $\Phi : \Sigma \times \Sigma \to \{0, 1\}$ is the predicate function such that

$$\Phi(a, b) = \begin{cases} 0, \text{ if } a = b; \\ 1, \text{ otherwise.} \end{cases}$$

– given a set of sequences $\Omega = \{s^1, s^2, \ldots, s^n\}$ on Σ ($s^i \in \Sigma^m$, $i = 1, 2, \ldots, n$) d^{Ω}_H denotes the Hamming distance among the sequences in Ω and it is given by

$$0 \leq d^{\Omega}_H = \min_{i,l=1,\ldots,n \mid i<l} d_H(s^i, s^l) \leq m. \tag{2}$$

The FFMSP consists in determining a sequence far from as many as possible sequences in the input set Ω. This can be formalized by saying that given a threshold t, a string s must be found maximizing the variable x such that

$$d_H(s, s^i) \geq t, \text{ for } s^i \in P \subseteq \Omega \text{ and } |P| = x, \tag{3}$$

or, equivalently

$$d^{P \cup \{s\}}_H \geq t, \text{ for } P \subseteq \Omega \text{ and } |P| = x. \tag{4}$$

Computational intractability of the general sequences consensus problem was first proved in 1997 by Frances and Litman [11] and in 1999 by Sim and Park [22]. Among sequences consensus problems, the FFMSP is one of the hardest from a computational point of view, as proved in 2003 by Lanctot et al. [18], who demonstrated that for sequences over an alphabet Σ with $|\Sigma| \geq 3$, approximating the FFMSP within a polynomial factor is NP-hard.

Given theoretical computational hardness results, polynomial time algorithms for the FFMSP can yield only solutions with no constant guarantee of approximation. In such cases, to find good quality solutions in reasonable running times and to overcome the inner intractability of the problem from a computational point of view, heuristic methods must be devised. The first attempt in this direction was done in 2005 by Meneses et al. [19], who proposed a heuristic algorithm consisting of a simple greedy construction followed by an iterative improvement phase. Later, in 2007 Festa [5] designed a GRASP and very recently in 2012, Mousavi et al. [20] devised a new function to be used in alternative to the objective function when evaluating neighbor solutions during the local search phase.

In this paper, we designed, implemented, and tested several pure and hybrid metaheuristics. The scope of the hybrid metaheuristics designing has been to combine the main characteristics of the pure metaheuristics themselves in the attempt to take advantage of their best properties in terms of computation time and solution quality.

The remainder of this article is organized as follows. In Section 2, we propose various randomized heuristics for finding approximate solutions of the FFMSP, based on the instantiation of several metaheuristics and their hybrids. Computational results are reported in Section 3. Concluding remarks and insights about further improvements of the proposed techniques are given in the last section.

2 Hybrid Metaheuristics

In the last decades, a considerable amount of scientific papers has empirically shown that suitable combinations of concepts and characteristics from different metaheuristics can lead to the design of hybrid robust techniques that produce higher solution quality than the individual metaheuristics themselves, especially when solving difficult real-world combinatorial optimization problems.

Following this trend, to find good quality solutions to the FFMSP, we have considered several types of hybridizations. In particular, we have designed, implemented, and tested the following pure and hybrid multistart iterative heuristics:

- ◇ a pure GRASP, inspired by [5];
- ◇ a GRASP that uses Path-relinking for intensification;
- ◇ a pure VNS;
- ◇ a VNS that uses Path-relinking for intensification;
- ◇ a GRASP that uses VNS to implement the local search phase; and
- ◇ a GRASP that uses VNS to implement the local search phase and Path-relinking for intensification.

As any multistart iterative heuristic, stopping criteria in all the above listed techniques could be maximum number of iterations, maximum number of iterations without improvement of the incumbent solution, maximum running time, or solution quality at least as good as a given target value.

algorithm GRASP(t, m, Σ, $f_t(\cdot)$, $\{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma}$, Seed)
1 s_{best}:=$\emptyset$; $f_t(s_{best})$:=$-\infty$;
2 **for** $j = 1$ **to** $m\rightarrow$
3 $V_j^{\min}$:=$\min_{c\in\Sigma} V_j(c)$; $V_j^{\max}$:=$\max_{c\in\Sigma} V_j(c)$;
4 **endfor**
5 **while** stopping criterion not satisfied$\rightarrow$
6 $[s, \{\mathrm{RCL}_j\}_{j=1}^m]$:=GrRand($m$, Σ, $\{V_j(c)\}_{j\in\{1,\dots,m\}}^{c\in\Sigma}$, $V_j^{\min}$, $V_j^{\max}$, Seed);
7 s:=LocalSearch(t, m, s, $f_t(\cdot)$, $\{\mathrm{RCL}_j\}_{j=1}^m$);
8 **if** $(f_t(s) > f_t(s_{best}))$ **then**
9 s_{best}:=s;
10 **endif**
11 **endwhile**
12 **return**(s_{best});
end GRASP

Fig. 1. Pseudo-code of a GRASP for the FFMSP

2.1 A Pure GRASP

Each GRASP iteration consists of a construction phase [3, 4], where a solution is built in a *greedy*, *randomized*, and *adaptive* manner, and a local search phase

which starts at the constructed solution and applies iterative improvement until a locally optimal solution is found. Repeated applications of the construction procedure yields diverse starting solutions for the local search and the best overall local optimal solution is returned as the result. The reader can refer to [8–10] for a study of a generic GRASP metaheuristic framework and its applications.

```
function GrRand(m, Σ, {V_j(c)}_{j∈{1,...,m}}^{c∈Σ}, V_j^min, V_j^max, Seed)
1  for j = 1 to m→
2    RCL_j:=∅; α:=Random([0,1], Seed);
3    μ:=V_j^min + α · (V_j^max − V_j^min);
4    for all c ∈ Σ→
5      if (V_j(c) ≤ μ)) then
6        RCL_j:=RCL_j∪{c};
7      endif
8    endfor
9    s_j:=Random(RCL_j, Seed);
10 endfor
11 return(s, {RCL_j}_{j=1}^m);
end GrRand
```

Fig. 2. Pseudo-code of the GRASP construction for the FFMSP

A complete solution is iteratively constructed in the construction phase, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a greedy function that measures the (myopic) benefit of selecting each element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL). For the FFMSP, in 2007 [5] a GRASP has been proposed to find suboptimal solutions for the FFMSP and Figure 1 depicts its pseudo-code, where $f_t : \Sigma^m \mapsto \mathbb{N}$ denotes the objective function to be maximized according to (3) and (4).

Figure 2 reports the pseudo-code of the construction procedure that iteratively builds a sequence $s = (s_1, \ldots, s_m) \in \Sigma^m$, selecting one character at time. The greedy function is related to the occurrence of each character in a given position. In fact, as in [5], for each position $j \in \{1, \ldots, m\}$ and for each character $c \in \Sigma$, we compute $V_j(c)$ as the number of times c appears in position j in any of the strings in Ω. The pure greedy choice would consist in selecting the character c with the lowest greedy function value $V_j(c)$. To define the construction mechanism for the RCL, let

$$V_j^{\min} = \min_{c \in \Sigma} V_j(c), \qquad V_j^{\max} = \max_{c \in \Sigma} V_j(c).$$

Denoting by $\mu = V_j^{\min} + \alpha \cdot (V_j^{\max} - V_j^{\min})$ the cut-off value (line 3), where α is a parameter such that $0 \leq \alpha \leq 1$ (line 2), the RCL is made up by all characters whose value of the greedy function is less than or equal to μ (line 6). A character is then randomly selected from the RCL (line 9).

```
function LocalSearch(t, m, s, f_t(·), {RCL_j}_{j=1}^m)
1  max:=f_t(s); change:=.TRUE.;
2  while (change)→
3    change:=.FALSE.;
4    for j = 1 to m→
5      temp:=s_j;
6      for all c ∈RCL_j→
7        s_j:=c;
8        if (f_t(s) > max) then
9          max:=f_t(s); temp:=c; change:=.TRUE.; break;
10       endif
11     endfor
12     s_j:=temp;
13   endfor
14 endwhile
15 return(s);
end LocalSearch
```

Fig. 3. Pseudo-code of the GRASP local search for the FFMSP

The basic step of the local search described in Figure 3 is slightly different from the one implemented in [5]. In our GRASP, it consists in investigating all positions $j \in \{1, \ldots, m\}$ (loop in lines 4–14) and changing the character in position j in the sequence s to another character in RCL_j. Instead, in [5] the position j and the new character in position j are selected at random. Moreover, the random selection of the new character in position j involves the set of all characters occurring in that position in all the given sequences in Ω.

The current solution is replaced by the first improving neighbor (lines 8–11). The search stops after all possible moves have been evaluated and no improving neighbor was found, returning a local optimal solution (line 16).

2.2 A Pure VNS

Contrary to other metaheuristics based on local search methods, VNS [16] is based on the exploration of a dynamic neighborhood model. It explores increasingly distant neighborhoods of the current best found solution.

Let N_k, $k = 1, \ldots, k_{max}$, be a set of pre-defined neighborhood structures and let $N_k(s)$ be the set of solutions in the kth-order neighborhood of a solution s. In the first phase, a neighbor $s' \in N_k(s)$ of the current solution is applied. Next, a

solution s'' is obtained by applying local search to s'. Finally, the current solution jumps from s to s'' in case the latter improved the former. Otherwise, the order of the neighborhood is increased by one and the above steps are repeated until some stopping condition is satisfied.

```
algorithm VNS(t, m, Σ, f_t(·), k_max, Seed)
1  s_best:=∅; f_t(s_best):=−∞;
2  while stopping criterion not satisfied→
3    k:=1; s:=BuildRand(m, Σ, Seed); /* pure randomly */
4    while (k ≤ k_max)→
5      s':=Random(N_k(s), Seed);
6      s'':=LocalSearch(t, m, s', f_t(·), {RCL_j}_{j=1}^m);
7      if (f_t(s'') > f_t(s)) then
8        s:=s''; k:=1;
9        if (f_t(s'') > f_t(s_best)) then s_best:=s'';
10       endif
11     else k:=k + 1;
12     endif
13   endwhile
14 endwhile
15 return(s_best);
end VNS
```

Fig. 4. Pseudo-code of a VNS for the FFMSP

In the case of the FFMSP, the kth-order neighborhood is defined by all sequences that can be derived from the current sequence s by selecting k positions $j_1, \ldots, j_k$ and changing $s_{j_1}, \ldots, s_{j_k}$ with a character in $\mathrm{RCL}_{j_1}, \ldots, \mathrm{RCL}_{j_k}$, respectively. The same local search strategy used within the pure GRASP algorithm described in Section 2.1 is used in the VNS heuristic, whose pseudo-code is reported in Figure 4.

2.3 Path-Relinking

Path-relinking is a heuristic proposed in 1996 by Glover [12] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [13–15].

Starting from one or more elite solutions, paths in the solution space leading towards other guiding elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. At each iteration, all moves that incorporate attributes of the guiding solution are analyzed and the move that best improves (or least deteriorates) the initial solution is chosen.

Figure 5 illustrates the pseudo-code of the Path-relinking for the FFMSP. It is applied to a pair of sequences $(s, \hat{s})$, where s is a given input solution and $\hat{s}$

is a solution (*sufficiently different* from s – see Section 3) selected at random (line 1) from an elite set $\mathcal{E}$ of solutions that has a fixed size that does not exceed `MaxElite`. Their common elements are kept constant, and the space of solutions spanned by these elements is searched with the objective of finding a better solution. This search is done by exploring a path in the solution space linking the worst solution s' between s and $\hat{s}$ to the best one (line 3). s' is called the *initial solution* and $\hat{s}$ the *guiding solution*.

```
algorithm Path-relinking(t, m, f_t(·), s, E, Seed)
1   ŝ:=Random(E, Seed);
2   f* := max{f_t(s), f_t(ŝ)}; s* := arg max{f_t(s), f_t(ŝ)};
3   s' := arg min{f_t(s), f_t(ŝ)}; ŝ:=s*;
4   Δ(s', ŝ):={i = 1, ..., m | s'_i ≠ ŝ_i};
5   while (Δ(s', ŝ) ≠ ∅) →
6         i* := arg max{f_t(s' ⊕ i) | i ∈ Δ(s', ŝ)};
7         Δ(s' ⊕ i*, ŝ) := Δ(s', ŝ) \ {i*};
8         s' := s' ⊕ i*;
9         if (f_t(s') > f*) then
10              f* := f_t(s'); s* := s';
11        endif;
12  endwhile;
13  return (s*);
end Path-relinking
```

Fig. 5. Pseudo-code of a Path-relinking for the FFMSP

The procedure then computes (line 4) the symmetric difference $\Delta(s', \hat{s})$ between the two solutions as the set of components for which the two solutions differ:

$$\Delta(s', \hat{s}) := \{i = 1, \ldots, m \mid s'_i \neq \hat{s}_i\}.$$

Note that, $|\Delta(s', \hat{s})| = d_H(s', \hat{s})$ and $\Delta(s', \hat{s})$ represents the set of moves needed to reach $\hat{s}$ from s', where a move applied to the initial solution s' consists in selecting a position $i \in \Delta(s', \hat{s})$ and replacing s'_i with $\hat{s}_i$.

Path-relinking generates a path of solutions $s'_1, s'_2, \ldots, s'_{|\Delta(s', \hat{s})|}$ linking s' and $\hat{s}$. The best solution s^* in this path is returned by the algorithm (line 13).

The path of solutions is computed in the loop in lines 5 through 12. This is achieved by advancing one solution at a time in a greedy manner. At each iteration, the procedure examines all moves $i \in \Delta(s', \hat{s})$ from the current solution s' and selects the one which results in the highest cost solution (line 6), i.e. the one which maximizes $f_t(s' \oplus i)$, where $s' \oplus i$ is the solution resulting from applying move i to solution s'. The best move i^* is made, producing solution $s' \oplus i^*$ (line 8). The set of available moves is updated (line 7). If necessary, the best solution s^* is updated (lines 9–11). Clearly, the algorithm stops as soon as $\Delta(s', \hat{s}) = \emptyset$.

2.4 Hybrid GRASP with Path-Relinking

Since GRASP iterations are independent of one another, it does not make use of solutions produced throughout the search. One way to add memory to GRASP is its hybridization with Path-relinking. In 1999 the first proposal of such a hybrid method was published by Laguna and Martí [17]. It was followed by several extensions, improvements, and successful applications [2, 6, 7].

Into the pure GRASP algorithm described in Section 2.1 we have integrated Path-relinking applied at each GRASP iteration to pairs $(s, \hat{s})$ of solutions, where s is the locally optimal solution obtained by GRASP local search and $\hat{s}$ is randomly chosen from a pool with a limited number `MaxElite` of high quality solutions found along the search. The pseudo-code for the proposed GRASP with Path-relinking hybrid algorithm is shown in Figure 6.

algorithm `GRASP+PR`(t, m, Σ, $f_t(\cdot)$, $\{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma}$, `Seed`, `MaxElite`)
1 s_{best}:=$\emptyset$; $f_t(s_{best})$:=$-\infty$; $\mathcal{E}$:= $\emptyset$; $iter$:=0;
2 **for** $j = 1$ **to** $m\rightarrow$
3 $V_j^{\min}$:=$\min_{c\in\Sigma} V_j(c)$; $V_j^{\max}$:=$\max_{c\in\Sigma} V_j(c)$;
4 **endfor**
5 **while** stopping criterion not satisfied$\rightarrow$
6 $iter$:=$iter + 1$;
7 $[s, \{\mathrm{RCL}_j\}_{j=1}^m]$:=`GrRand`($m$, Σ, $\{V_j(c)\}_{j\in\{1,\ldots,m\}}^{c\in\Sigma}$, $V_j^{\min}$, $V_j^{\max}$, `Seed`);
8 s:=`LocalSearch`(t, m, s, $f_t(\cdot)$, $\{\mathrm{RCL}_j\}_{j=1}^m$);
9 **if** ($iter \leq$ `MaxElite`) **then**
10 $\mathcal{E} := \mathcal{E} \cup \{s\}$;
11 **if** ($f_t(s) > f_t(s_{best})$) **then** s_{best}:=s;
13 **endif**
14 **else**
10 $\overline{s}$:=`Path-relinking`(t, m, $f_t(\cdot)$, s, $\mathcal{E}$, `Seed`);
15 `AddToElite`($\mathcal{E}$, $\overline{s}$);
11 **if** ($f_t(\overline{s}) > f_t(s_{best})$) **then** s_{best}:=$\overline{s}$;
13 **endif**
13 **endif**
11 **endwhile**
12 **return**(s_{best});
end `GRASP+PR`

Fig. 6. Pseudo-code of a hybrid GRASP with Path-relinking for the FFMSP

The pool of elite solutions is originally empty (line 1). The best solution $\overline{s}$ found along the relinking trajectory is considered as a candidate to be inserted into this pool. If the pool is not full ($|\mathcal{E}| \leq$ `MaxElite`), the candidate is simply inserted. Otherwise, if the pool is full, the procedure `AddToElite` evaluates its insertion into $\mathcal{E}$. In more detail, if $\overline{s}$ is better than the best elite solution, then $\overline{s}$

replaces the worst elite solution. If the candidate is better than the worst elite solution, but not better than the best, it replaces the worst if it is *sufficiently different* (see Section 3) from all elite solutions.

2.5 Hybrid GRASP with VNS

As underlined in Subsection 2.2, until a stopping criterion is met, at each iteration VNS chooses a neighbor sequence s from the neighborhood of the current solution at random. In our hybrid GRASP with VNS, VNS is applied as local search and its starting solution is the sequence s output of the GRASP construction procedure.

2.6 Hybrid VNS with Path-Relinking

As is the case for GRASP, VNS described in Section 2.2 also can be hybridized with Path-relinking, as intensification phase. At each VNS iteration Path-relinking is applied to pairs $(s, \hat{s})$ of solutions, where s is the locally optimal solution obtained by VNS and $\hat{s}$ is randomly chosen from the `MaxElite` high quality solutions found along the search.

2.7 Hybrid GRASP with VNS and Path-Relinking

This hybrid procedure is simply obtained by replacing the local search phase of the GRASP procedure with VNS and applying at the end of each major iteration Path-relinking as intensification procedure.

3 Experimental Results

In this section, we present numerical results on computational experiments with the heuristics proposed in this article. We describe first the computer environment and the problem instances. Then, we describe implementation details and the used combination of values for the parameters of the heuristics. Finally, we report on the experimental evaluation of the different algorithms.

Our codes have been written in the *C language*, compilated with "*cc (GCC) 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)*", and ran on an "*Intel Core i7 Quad core @ 2.67 GHz RAM 6GB*" with *Linux* (*Ubuntu 11.10*) operating system.

Problem instances were generated at random. In the set of test instances, the sequence length m ranges from 300 to 800, the number n of sequences in Ω ranges from 100 to 200, and threshold t varies from $75\%m$ to $85\%m$. For each problem size, the algorithms were run for 100 random instances and average solution value was computed. The results obtained are summarized in Table 1, where for each problem type, in the first column the instance size (m, n, and t) is reported. The remaining columns report the average objective function values (z) obtained by each algorithm and the corresponding average running times (in seconds). We make the following observations:

Table 1. Average objective function values obtained by each algorithm and the corresponding average running times (in seconds)

	GRASP		GRASP+PR		GRASP+VNS+PR		VNS		VNS+PR	
n, m, t	z	Time	z	Time	z	Time	z	Time	z	Time
100, 300, 0.75	**100**	1.37	**100**	1.41	**100**	1.71	94.47	72.45	**100**	7.44
100, 300, 0.80	67.86	1.67	76.17	3.21	**76.68**	33.28	19.98	71.02	48.58	77.43
100, 300, 0.85	3.56	1.72	10.17	4.17	**12.23**	31.36	1.12	37.65	3.53	44.19
100, 600, 0.75	**100**	1.56	**100**	1.89	**100**	1.11	91.78	278.94	**100**	31.91
100, 600, 0.80	65.35	2.31	78.83	11.77	**80.47**	122.65	8.51	264.66	20.72	295.49
100, 600, 0.85	1.21	1.28	3.58	11.07	**4.36**	91.87	0.04	152.64	0.91	186.71
100, 800, 0.75	100	1.84	100	1.34	100	2.48	87.36	549.60	100	63.00
100, 800, 0.80	67.76	1.42	80.37	21.10	**82.30**	251.26	4.41	450.63	10.94	527.76
100, 800, 0.85	0.30	2.98	0.97	31.45	**2.51**	148.98	0.66	273.98	0.63	329.48
200, 300, 0.75	197.78	1.22	**200**	1.85	**200**	3.70	180.08	135.61	**200**	47.91
200, 300, 0.80	76.50	1.39	93.70	6.52	**94.45**	65.44	36.71	150.51	66.81	160.37
200, 300, 0.85	2.83	1.59	9.26	8.59	**11.12**	68.20	2.16	86.60	4.62	104.13
200, 600, 0.75	**200**	1.94	**200**	1.61	**200**	3.39	178.11	545.50	**200**	75.43
200, 600, 0.80	62.80	1.63	83.91	24.91	**86.17**	274.93	11.93	588.35	33.41	625.66
200, 600, 0.85	0.98	1.79	1.51	31.22	**2.38**	174.73	0.71	305.29	0.96	369.29
200, 800, 0.75	**200**	1.04	**200**	1.33	**200**	1.61	175.06	947.80	**200**	193.30
200, 800, 0.80	44.66	1.75	71.28	43.63	**72.44**	519.59	6.37	987.35	17.12	1102.47
200, 800, 0.85	0.86	1.55	1.93	59.20	**3.71**	311.81	0.15	544.21	0.49	659.02

- the stopping criterion for all algorithms was `MaxIterations` = 500 or the obtainment of an incumbent solution with objective function value $z = n$ (i.e., an optimal solution);
- the maximum order k_{max} in the dynamic neighborhood model used in VNS and in the hybrid VNS with Path-relinking and the hybrid GRASP with VNS and Path-relinking was set to 30;
- the maximum number `MaxElite` of elite solutions in the hybrid heuristics invoking Path-relinking as intensification procedure was set to 10;
- in Path-relinking, for inclusion of the candidate solution $\overline{s}$ in the elite set $\mathcal{E}$ when $\overline{s}$ is better than the worst elite set solution but not better than the best, $\overline{s}$ is inserted (replacing the worst solution) if it *sufficiently different* from all elite solutions, i.e. if $|\Delta(\overline{s}, \epsilon)| \geq \frac{m}{2}$, for all $\epsilon \in \mathcal{E}$;
- on all instances, the hybrid GRASP with VNS and Path-relinking found a better quality solution as compared to the competitor heuristics;
- the hybrid GRASP with Path-relinking found best results for 7 out of the 18 instances; hybrid VNS with Path-relinking found best results for 6 instances, while the pure GRASP and the pure VNS found the best solution for only 5 and 0 instances, respectively;
- at the expense of increased running times, the integration of Path-relinking in the pure metaheuristics was beneficial in terms of solution quality;
- looking at the objective function values achieved by GRASP with Path-relinking and GRASP with VNS and Path-relinking, at the expense of

increased running times, the use of VNS in the local search phase of GRASP was beneficial.

Given the random component of each proposed algorithm and since their running times per iteration vary substantially, we have performed two further experiments.

Table 2. Average objective function values obtained by each algorithm after 90 seconds of computation

n, m, t	GRASP	GRASP+PR	GRASP+VNS+PR	VNS	VNS+PR
100, 300, 0.75	**100**	**100**	**100**	94	**100**
100, 300, 0.8	71.07	**79.61**	78.12	23.43	49.54
100, 300, 0.85	6.41	**13.18**	11.86	1.02	3.27
100, 600, 0.75	**100**	**100**	**100**	91.79	**100**
100, 600, 0.8	70.24	**80.13**	78.05	6.38	11.29
100, 600, 0.85	2.73	**4.98**	4.48	0.03	0.12
100, 800, 0.75	**100**	**100**	**100**	85.18	**100**
100, 800, 0.8	70.07	**82.64**	79.45	3.71	8.42
100, 800, 0.85	1.17	**1.84**	1.65	0	0.03
200, 300, 0.75	199.81	**200**	**200**	179.34	**200**
200, 300, 0.8	81.75	**100**	95.11	34.71	61.67
200, 300, 0.85	4.82	**11.90**	11.03	2.32	3.70
200, 600, 0.75	**200**	**200**	**200**	172.41	**200**
200, 600, 0.8	66.23	**88.49**	80.31	10.25	19.10
200, 600, 0.85	1.03	**2.42**	1.73	0.09	0.97
200, 800, 0.75	**200**	**200**	**200**	164.01	194.45
200, 800, 0.8	49.87	**73.08**	62.36	4.23	8.17
200, 800, 0.85	0.08	**0.21**	0.17	0.06	0.85

First, on the same set of randomly generated problem instances and by using the same values of the parameters for the algorithms, we have run them for a given fixed amount of time, set to 90 seconds. For each problem type, Table 2 reports the average objective function values (z) obtained by each algorithm. It is still evident that the integration of Path-relinking in the pure metaheuristics is beneficial in terms of solution quality. Moreover, in a given fixed amount of computation time the number of iterations performed by the hybrid GRASP with Path-relinking is higher than that performed by the hybrid GRASP with VNS and Path-relinking. This implies that GRASP with VNS and PR performs a smaller number of samplings of the solution space with the conclusion that in this scenario, the hybrid GRASP with PR found better quality solutions.

As further investigation, given the random component of each proposed algorithm and the great variety in their running times per iteration, we plot in Figures 7–8 the empirical distributions of the random variable *time-to-target-solution-value* considering the following four random instances:

1. $n = 100$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.70 \times n$ (Figure 7(a));
2. $n = 100$, $m = 300$, $t = 252$, and target value $\hat{z} = 0.12 \times n$ (Figure 7(b));

3. $n = 200$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.40 \times n$ (Figure 8(a));
4. $n = 300$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.28 \times n$ (Figure 8(b)).

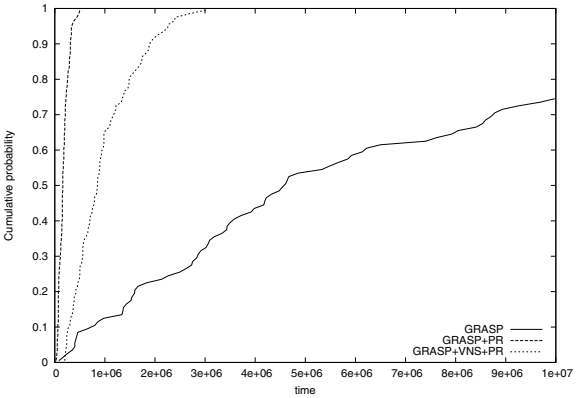
We performed 100 independent runs of each heuristic using 100 different random number generator seeds and recorded the time taken to find a solution at least as good as the target value $\hat{z}$. As in [1], to plot the empirical distribution we associate with the i^{th} sorted running time (t_i) a probability $p_i = \frac{i-1/2}{100}$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 100$. About these further experiments, looking at Figures 7 and 8, we observe that the relative position of the curves implies that, given any fixed amount of running time, the hybrid GRASP with Path-relinking has a higher probability than all competitors of finding a solution whose objective function value is at least as good as the target objective function value.

4 Concluding Remarks and Future Work

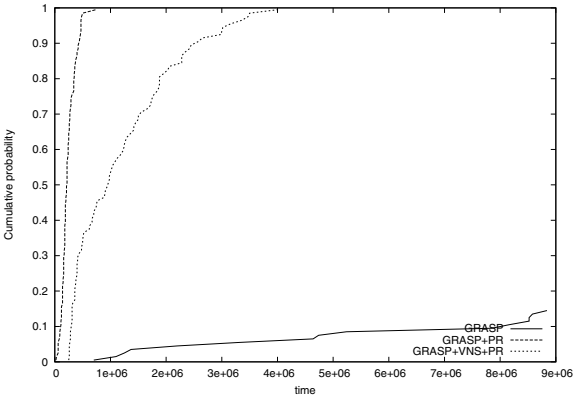
Given the computational intractability of one of the consensus problems known as the Far From Most String Problem, we have designed several hybrid metaheuristics that guarantee good quality solutions within realistic and acceptable amount of time. The algorithms were tested on several random instances and the results show that the hybrid GRASP with VNS and Path-relinking always finds much better quality solutions compared with the other competitor algorithms, but clearly with higher running times as compared to the pure GRASP and the hybrid GRASP with Path-relinking. In the following, we summarize our observations about our computational experience.

- The processing time for the pure GRASP was the smallest but the objective function values found by the algorithm were worse than those found by its hybridizations;
- Best objective function values found by GRASP and its hybrids were when the construction phase was more greedy than random.
- Overall, the hybrid GRASP with VNS and path-relinking found the best solutions, followed by GRASP with Path-relinking and the pure GRASP;
- Overall, the objective function values found by the pure VNS were the worst. This bad behavior is not surprising, given the totally random criterion adopted in the VNS construction.
- The integration of Path-relinking as intensification procedure in the pure metaheuristics was beneficial in terms of solution quality.
- We plot in Figures 7–8 the empirical distributions of the random variable *time-to-target-solution-value* considering four different random instances. Our conclusion after this further investigation is that, given any fixed amount of computing time, GRASP with Path-relinking has a higher probability than all competitors of finding a target solution.

As future work, we plan to better investigate the practical behavior of the proposed algorithms, by introducing the recently published tool designed by



(a) Random instance with $n = 100$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.70 \times n$.



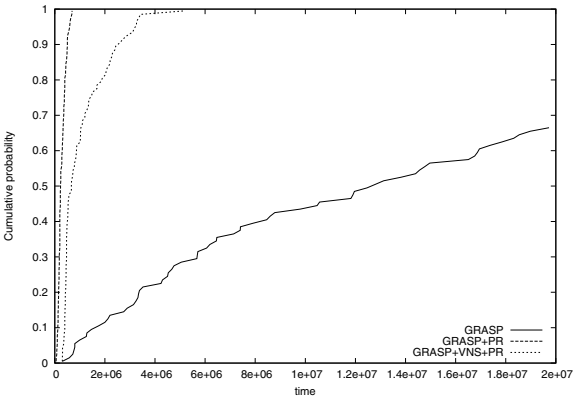
(b) Random instance with $n = 100$, $m = 300$, $t = 252$, and target value $\hat{z} = 0.12 \times n$.

Fig. 7. Time to target distributions comparing `GRASP`, `GRASP+PR`, and `GRASP+VNS+PR`

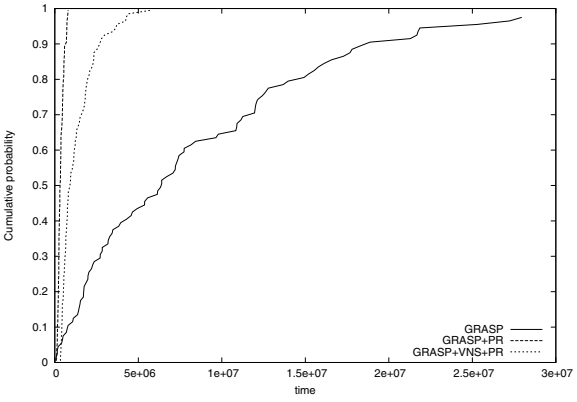
Ribeiro et al. [21] for characterizing stochastic algorithms running times under the assumption that the running times of the algorithms follow exponential (or shifted exponential) distributions, as it is the case of our hybrid heuristics.

We plan also to validate the numerical results on computational experiments with the heuristics proposed in this article, by applying them on a larger dataset of instances, both randomly generated and taken from real–world applications of the problem.

Furthermore, it would be also interesting to design some variants of the approaches proposed in this paper. Three natural extensions would be 1) to perform at the end of computation a post-optimization phase, for example by invoking Path-relinking among pairs of elite solutions; 2) to implement alternative linking strategies in Path-relinking, such as backward, mixed, and randomized Path-



(a) Random instance with $n = 200$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.40 \times n$.



(b) Random instance with $n = 300$, $m = 300$, $t = 240$, and target value $\hat{z} = 0.28 \times n$.

Fig. 8. Time to target distributions comparing `GRASP`, `GRASP+PR`, and `GRASP+VNS+PR`

relinking; 3) to integrate in the local search of the algorithms the new function devised by Mousavi et al. [20] and to be used in alternative to the objective function when evaluating neighbor solutions.

References

1. Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: Probability distribution of solution time in GRASP: an experimental investigation. Journal of Heuristics 8, 343–373 (2002)
2. Canuto, S.A., Resende, M.G.C., Ribeiro, C.C.: Local search with perturbations for the prize-collecting Steiner tree problem in graphs. Networks 38, 50–58 (2001)
3. Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. Oper. Res. Lett. 8, 67–71 (1989)

4. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. J. Global Optim. 6, 109–133 (1995)
5. Festa, P.: On some optimization problems in mulecolar biology. Mathematical Bioscience 207(2), 219–234 (2007)
6. Festa, P., Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: GRASP with path-relinking for the weighted MAXSAT problem. ACM J. of Experimental Algorithmics 11, 1–16 (2006)
7. Festa, P., Pardalos, P.M., Resende, M.G.C., Ribeiro, C.C.: Randomized heuristics for the MAX-CUT problem. Optimization Methods and Software 7, 1033–1058 (2002)
8. Festa, P., Resende, M.G.C.: GRASP: An annotated bibliography. In: Ribeiro, C.C., Hansen, P. (eds.) Essays and Surveys on Metaheuristics, pp. 325–367. Kluwer Academic Publishers (2002)
9. Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP – Part I: Algorithms. International Transactions in Operational Research 16(1), 1–24 (2009)
10. Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP – Part II: Applications. International Transactions in Operational Research 16(2), 131–172 (2009)
11. Frances, M., Litman, A.: On covering problems of codes. Theory of Computing Systems 30(2), 113–119 (1997)
12. Glover, F.: Tabu search and adaptive memory programing – Advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) Interfaces in Computer Science and Operations Research, pp. 1–75. Kluwer (1996)
13. Glover, F.: Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In: Laguna, M., Gonzáles-Velarde, J.L. (eds.) Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 1–24. Kluwer (2000)
14. Glover, F., Laguna, M.: Tabu search. Kluwer Academic Publishers (1997)
15. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics 39, 653–684 (2000)
16. Hansen, P., Mladenović, N.: Developments of variable neighborhood search. In: Ribeiro, C.C., Hansen, P. (eds.) Essays and Surveys in Metaheuristics, pp. 415–439. Kluwer Academic Publishers (2002)
17. Laguna, M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. INFORMS J. on Computing 11, 44–52 (1999)
18. Lanctot, J., Li, M., Ma, B., Wang, S., Zhang, L.: Distinguishing string selection problems. Information and Computation 185(1), 41–55 (2003)
19. Meneses, C.N., Oliveira, C.A.S., Pardalos, P.M.: Optimization techniques for string selection and comparison problems in genomics. IEEE Engineering in Medicine and Biology Magazine 24(3), 81–87 (2005)
20. Mousavi, S.R., Babaie, M., Montazerian, M.: An improved heuristic for the far from most strings problem. Journal of Heuristics 18, 239–262 (2012)
21. Ribeiro, C.C., Rosseti, I., Vallejos, R.: Exploiting run time distributions to compare sequential and parallel stochastic local search algorithms. Journal of Global Optimization 54, 405–429 (2012)
22. Sim, J.S., Park, K.: The consensus string problem for a metric is NP-complete. In: Proceedings of the Annual Australiasian Workshop on Combinatorial Algorithms (AWOCA), pp. 107–113 (1999)

On Missing Data Hybridizations for Dimensionality Reduction

Oliver Kramer

Computational Intelligence Group,
Department of Computing Science,
University of Oldenburg, Germany
oliver.kramer@uni-oldenburg.de

Abstract. Missing data is a challenge in machine learning. In this work, we compare two ways of handling patterns with missing entries in dimensionality reduction for the iterative approach unsupervised nearest neighbors (UNN). Both methods are hybridization of two heuristics as they combine UNN for dimensionality reduction with K-nearest neighbors (KNN) for repair. The first variant repair-and-embed applies nearest neighbor regression to iteratively fill the pattern gaps based on known and complete entries. The second variant embed-and-repair first embeds incomplete patterns ignoring missing entries and then completes them with nearest neighbor regression. The experimental analysis shows that both approaches allow a reasonable embedding with incomplete data, while embed-and-repair is showing better results.

1 Introduction

In practical data mining scenarios, patterns are noisy and entries may be missing. This can be caused by failures of sensors or bad environmental conditions that have an influence on sensors, CPU or memories. Besides modeling of the data mining process, selection of relevant features and tasks like normalization, the practitioner often has to preprocess the data, e.g., to complete missing entries. This preprocessing step is often neglected in machine learning research, but has an important part to play in practical scenarios.

The data mining task, we concentrate on in this work is dimensionality reduction. Although, in dimensionality reduction, the task is to reduce the number of features that is employed during learning, missing entries can render the problem even more difficult. The dimensionality reduction method we concentrate on is unsupervised nearest neighbors, an iterative solution construction algorithm based on the simple yet efficient nearest neighbor regression method. KNN regression will also play an important role for imputation, i.e., repairing of incomplete patterns. In this paper, we introduce and compare two procedure that allow to cope missing data for UNN.

– *Repair-and-embed* is the standard *imputation*-way that first repairs incomplete patterns iteratively with KNN regression and then embeds the repaired

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 189–197, 2013.

patterns employing the dimensionality method UNN. For the imputation process, the completed patterns are used to predict the missing values iteratively, i.e., they serve as part of the training set for the remaining incomplete patterns.
- *Embed-and-repair* is a novel method that first embeds incomplete patterns ignoring the features with missing data. This variant is probably more interesting as it employs the dimensionality reduction process itself for completion. After embedding, the gaps are filled with KNN regression, so that the embedded patterns optimally fit into the latent positions, and consequently minimize the error measure, called data space reconstruction error (DSRE).

With the help of experiments on simple test data sets, we compare the missing data strategies w.r.t. an increasing rate p of missing values. We consider the imputation performance and compare the DSRE with completed manifolds (results of the dimensionality reduction process).

This work is structured as follows. In Section 2, we will introduce UNN, which will be extended by missing data handling hybridizations in Section 3. Section 4 experimentally compares both approaches. We draw conclusions in Section 5.

2 Unsupervised Nearest Neighbors

UNN is an approach that we recently introduced for embedding high-dimensional patterns into discrete [5] and continuous latent space [7]. It is a manifold learning approach based on KNN regression that computes the mean of the function values of the K-nearest neighbors in a training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)$. KNN regression is defined as

$$\mathbf{f}(\mathbf{x}') = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \mathbf{y}_i \tag{1}$$

with set $\mathcal{N}_K(\mathbf{x}')$ containing the indices of the K-nearest neighbors of $\mathbf{x}'$. Recently, we enhanced the approach to evolutionary embeddings [4], a particle swarm-like approach [6], and kernel functions [7].

2.1 Unsupervised Regression

UNN is based on the framework of unsupervised regression, which is introduce in the following. Let $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\} \subset \mathbb{R}^d$ be the set of high-dimensional patterns with corresponding pattern matrix $\mathbf{Y} = [\mathbf{y}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$. Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^q$ be an arbitrary set of low-dimensional representations/latent points that define a manifold with $q < d$. Matrix $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N \in \mathbb{R}^{q \times N}$ is the corresponding latent representation. The pairs $(\mathbf{x}_i, \mathbf{y}_i)$ with $1 \leq i \leq N$ are the patterns with their latent points (positions in latent space). The low-dimensional representation should represent typical characteristics of the high-dimensional data and should loose as less information as possible, e.g., data space neighborhood relations and distances. The problem is a hard optimization problem, since the latent variables $\mathbf{X}$ are unknown. The iterative procedure of UNN allows to cope with the large number of dimensions.

2.2 Iterative Unsupervised Regression

Unsupervised nearest neighbors is an approach that constructs the manifold by iteratively adding locally optimal latent points w.r.t. a growing pattern set. For the iterative procedure, we define a notation for growing latent matrices $\overline{\mathbf{X}} \in \mathbb{R}^{q \times n}$ and pattern matrices $\overline{\mathbf{Y}} \in \mathbb{R}^{d \times n}$ for number $1 \leq n \leq N$ of currently embedded patterns. For this sake, we define the mapping $\mathbf{f}(\cdot) : \mathbb{R}^{q \times N} \to \mathbb{R}^{d \times N}$ from latent space $\mathbb{R}^q$ to data space $\mathbb{R}^d$ for matrices as follows $\mathbf{f}_{\mathbf{X}}(\mathbf{X}) = [\mathbf{f}_{\mathbf{X}}(\mathbf{x}_j)]_{j=1}^{N}$. For the optimal manifold $\mathbf{X}^*$, the DSRE

$$E(\mathbf{X}) = \|\mathbf{f}_{\mathbf{X}}(\mathbf{X}) - \mathbf{Y}\|_F^2 \tag{2}$$

is minimal, i.e., it holds $\mathbf{X}^* = \arg\min_{\mathbf{X} \in \mathbb{R}^{q \times N}} E(\mathbf{X})$. For many regression methods, the optimal solution $\mathbf{X}^*$ is not unique according to the above definition, as scaling would allow an infinite number of optimal solutions. To avoid this, a regularization term $\lambda \|\mathbf{X}\|_F^2$ is added and the optimization problem becomes $\mathbf{X}^* = \arg\min_{\mathbf{X} \in \mathbb{R}^{q \times N}} E(\mathbf{X}) + \lambda \|\mathbf{X}\|_F^2$ with penalty weight $\lambda \in \mathbb{R}^+$ that restricts latent space extension.

2.3 Latent Sorting

For generation of latent positions, we assume that latent points lie on a lattice structure in the following. For $q = 1$, finding appropriate latent positions on a line is similar to finding the best sorting of patterns. Latent sorting works as follows. For the first pattern $\mathbf{y}_1$, an arbitrary grid position can be chosen, e.g., $\mathbf{x}_1 = [1]$. The latent matrix is $\overline{\mathbf{X}} = [\mathbf{x}_1]$ and the corresponding pattern matrix is $\overline{\mathbf{Y}} = [\mathbf{y}_1]$. Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the sequence of already considered patterns with associated embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the next pattern $\mathbf{y}_i$ with $i = n + 1 \leq N$, UNN with grid sampling generates $n + 1$ candidate latent positions $0.5, 1.5, \ldots, n + 0.5$. The optimal latent position $\mathbf{x}^* = i^*$ is chosen that minimizes the DSRE. After choosing the optimal latent position i^*, all latent points get the latent position that corresponds to their rank in an increasingly sorted order $1, \ldots, n + 1$. This step is called *regularize grid* in the pseudocode. Finally, the latent matrix is extended by the novel latent position $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, i^*]$, the pattern $\mathbf{y}_i$ is added to the pattern matrix $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_i]$.

3 Missing Data

In practical applications, data sets are often incomplete. Failures of sensors, matching of databases with disjunct feature sets or conditions, where data can get lost (e.g., in outer space due to X-ray) are typical examples for practical scenarios with incomplete patterns. However, it might be desirable to compute a latent embedding of high-dimensional data. Objective of this section is to show strategies that allow UNN to cope with missing data. The question arises, if the embedding approach can exploit useful structural information to reconstruct the missing entries.

The problem of missing data in feature vectors can be treated in various kinds of ways. A simple method is to eliminate all patterns with missing entries. But this may introduce a bias, as the fact that features are missing can be caused by systematic errors. Further, the elimination of patterns from small data sets deteriorates the chance to learn good models. A class of methods to handle incomplete data are imputation methods, i.e., filling the gaps based on statistics. Incomplete data can be filled based on simple statistical parameters like the median and the mean of the available entries. Unfortunately, this method may also introduce a bias. Other imputation methods are based on regression approaches, similar to the repair step we apply in this work. For this sake, a training set consisting of the complete patterns is used, missing entries are the missing labels that have to be predicted, while the entries of the complete patterns of the corresponding dimensions are employed as labels.

In case the distribution of missingness is conditionally independent of the missing values given the observed data, the data is called *missing at random*[1]. Schafer and Graham [8] have reviewed methods to handle them. In case of sparse data sets, joint densities can be computed in a probabilistic framework [3].

If possible, the method can directly deal with missing data. Our embed-and-repair method that will be introduced in Section 3.2 belongs to this class. For support vector machine (SVM) classification, such an approach has been introduced by Chechik *et al.* [1]. It alters the SVM margin interpretation to directly deal with incomplete patterns. But the method is best suited for features that are absent than those that are MNAR. An extension has been introduced by Dick *et al.* [2]. The approach by Williams *et al.* [9] employs logistic regression for classification of incomplete data and performs an analytic integration with an estimated conditional density function instead of imputation. The approach does not only take into account the complete patterns, but also the incomplete patterns in a semi-supervised kind of way.

3.1 Repair-and-Embed

Let $\mathbf{Y}$ be the matrix of high-dimensional patterns. In the missing data scenario, we assume that some patterns in $\mathbf{Y}$ are incomplete, i.e., it holds $\exists y'_{ij}$ with $y'_{ij} = n.a.$ Let $\tilde{\mathbf{Y}}$ be the matrix of complete patterns, i.e., it holds $\nexists y_{ij}$ with $y_{ij} = n.a.$ In the following, we illustrate the imputation for one missing entry, but the approach can easily be extended to the multiple case. To complete y'_{ij}, repair-and-embed trains a regression model $\tilde{\mathbf{f}}$ based on $\tilde{\mathbf{Y}}$ and first fills the missing entries of patterns $[\mathbf{Y}]_j$ with a minimal number of missing entries. Let y_{ij} be the entry to complete. We can employ matrix $\tilde{\mathbf{Y}}^T_{-i}$ as training pattern matrix[2], while $\tilde{\mathbf{y}}_i = y_{i1}, \ldots, y_{i\tilde{N}}$ comprises the corresponding labels. Entry y'_{ij} is estimated with

[1] Missing at random (MAR) means that entries are missing randomly with uniform distribution, in contrast to *missing not at random* (MNAR), where dependencies exist, e.g., the missingness depends on the distribution of the patterns.

[2] $\tilde{\mathbf{Y}}_{-i} = [(\mathbf{y}_j)_{-i}]_{j=1}^{\tilde{N}}$ with $(\mathbf{y}_j)_{-i} = (y)_j$ and $j = 1, \ldots, d$, $j \neq i$. $\tilde{N}$ is the number of complete patterns.

$\tilde{\mathbf{f}}$ leading to the complete vector $[\mathbf{Y}]_j = \tilde{\mathbf{y}}_j$ that can be embedded as usual with UNN. As KNN regression is a non-parametric method, no training is necessary, only K has to be chosen carefully. After the pattern has been completed, the next pattern with minimal number of missing entries is chosen and the process is repeated until all patterns are complete. Then, data set $\tilde{\mathbf{Y}}$ can be embedded as usual with UNN.

3.2 Embed-and-Repair

The second variant for embedding incomplete data is to embed a vector $\mathbf{y}_j$ with a missing entry y_{ij} at dimension i ignoring the i-th component during the computation of the DSRE, i.e., minimizing

$$E_{-i}(\overline{\mathbf{X}}) = \frac{1}{N}\|\mathbf{f}_{\overline{\mathbf{X}}}(\overline{\mathbf{X}})_{-i} - \overline{\mathbf{Y}}_{-i}\|_F^2. \tag{3}$$

Algorithm 1.1 shows the pseudocode of the embed-and-repair approach. The algorithm starts iteratively with the vector $\mathbf{y}_j = [\mathbf{Y}]_j$ with increasing number of missing values. Starting the dimensionality reduction with complete patterns is reasonable to get as close as possible to the structure of the complete embedding. Embed-and-repair is a greedy approach that only considers the locally best

Algorithm 1.1. Embed-and-Repair

Require: $\mathbf{Y}$, K
1: **repeat**
2: choose $\mathbf{y}_j = [\overline{\mathbf{Y}}]_j$ with minimal number of missing entries, y_{ij} is missing
3: embed $\mathbf{y}_j$ with UNN minimizing $E_{-i}(\overline{\mathbf{X}}) \rightarrow \mathbf{x}_j$
4: complete $\mathbf{y}_j$ with KNN based on $\overline{\mathbf{X}} \rightarrow \mathbf{y}_j$
5: add $\mathbf{x}_j$ to $\overline{\mathbf{X}}$ with UNN
6: $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, \mathbf{x}_j]$, $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_j]$
7: **until** all patterns embedded

embedding w.r.t. the available information. Embedded patterns can be completed to take part in the remaining embedding process. The gaps are closed with entries that ensure that the embedding is minimal w.r.t. $e_{\overline{\mathbf{X}}}(\mathbf{x})$. This is the average of the K-nearest points for dimension i, i.e., the nearest neighbors estimation $y_{ij} = \mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}_j)_i$.

Figure 3.2 illustrates the embed-and-repair strategy for neighborhood size $K = 2$. Pattern $\mathbf{y} = (y_1, \cdot)$ is incomplete. It is embedded at the position, where it leads to the lowest DSRE w.r.t. the first dimension: between x' and x''. Then, the gap is filled with the mean of the second dimension of $\mathbf{y}'$ and $\mathbf{y}''$ yielding $\mathbf{y} = (y_1, 0.5 \cdot (y' + y''))$. The difference between KNN imputation and embed-and-repair imputation is that the embed-and-repair KNN prediction is based on neighborhoods in latent space. Hence, it is a dimensionality reduction-oriented imputation method based on characteristics introduced by UNN regression.

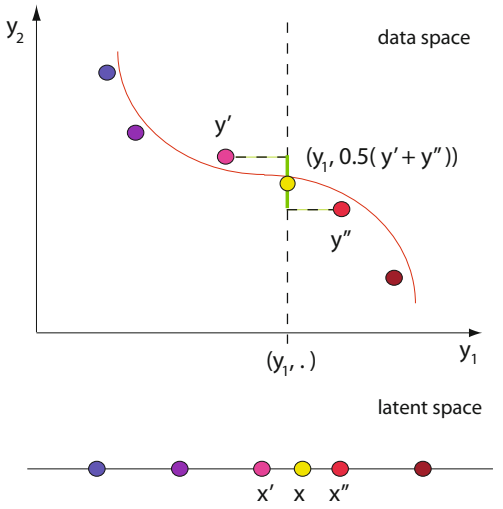


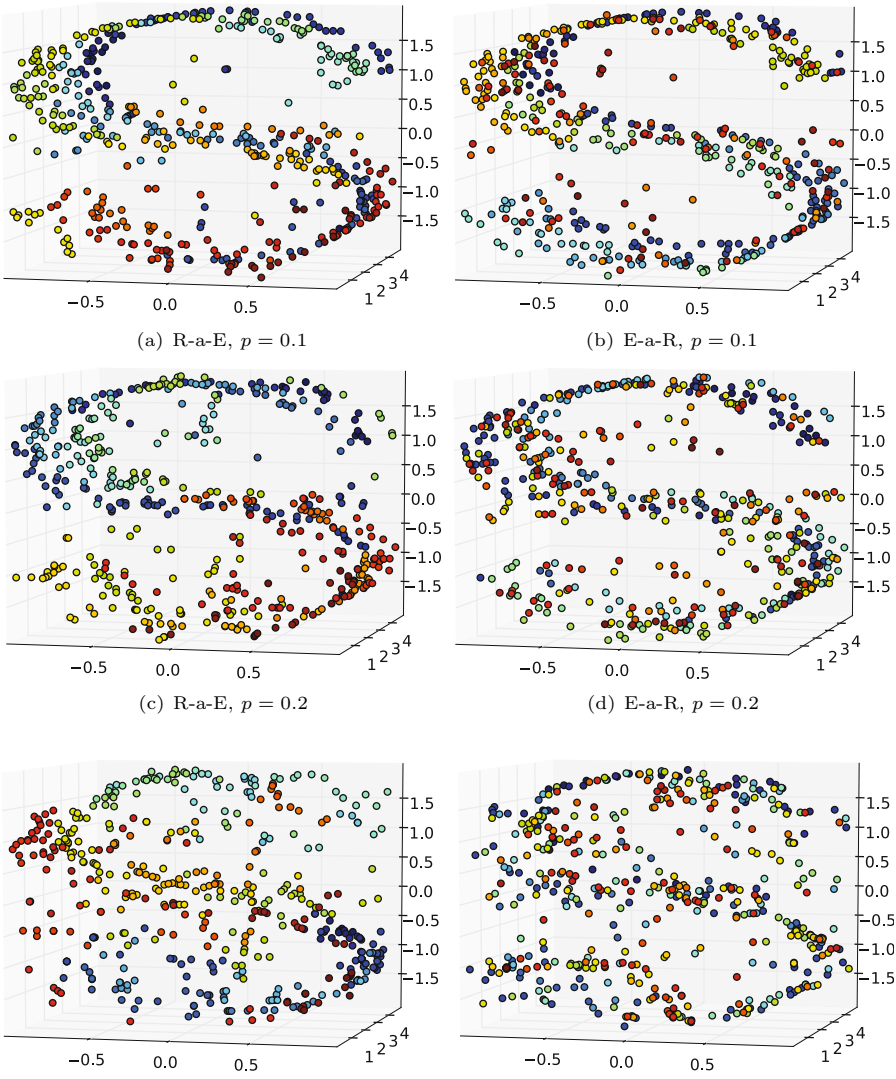
Fig. 1. Embed-and-repair. The incomplete pattern $\mathbf{y} = (y_1, \cdot)$ is embedded at position x leading to the lowest DSRE w.r.t. the first dimension, i.e., between x' and x''. Then, gap y_2 is filled with KNN and $K = 2$.

4 Experimental Comparison of Missing Data Methods

In the following, we describe the experimental setup for the comparison between both approaches. We generate a missing data matrix $\mathbf{Y}$ by removing entries y_{ij} from a complete data matrix $\mathbf{Y}^+ = [\mathbf{y}_j^+]_{j=1}^N \in \mathbb{R}^{d \times N}$ at random with uniform probability p, i.e., each entry y_{ij} is set to $n.a.$ with probability p. We experimentally compare the DSRE for embedding $\mathbf{Y}^+$ and $\mathbf{Y}$. Further, we analyze the imputation error $E_{imp} = \sum_{j=1}^N \|\mathbf{y}_j^+ - \tilde{\mathbf{y}}_j\|^2$, which is the deviation from the original complete patterns $\mathbf{y}_j^+$ and the repaired counterparts $\tilde{\mathbf{y}_j}$.

Table 1. Comparison of imputation error E_{imp} and DSRE between UNN with repair-and-embed (R-a-E) and UNN with embed-and-repair (E-a-R) on 3D-S and 3D-S$_h$ w.r.t. increasing data missing rate p

		R-a-E		E-a-R		complete
data	p	E_{imp}	DSRE	E_{imp}	DSRE	UNN
3D-S	0.01	0.0507	**147.2**	**0.0269**	165.39	142.8
	0.1	0.3129	**143.8**	**0.2884**	265.2	142.8
	0.2	0.6454	**149.0**	**0.6146**	369.2	142.8
	0.3	0.9557	**152.7**	**0.9265**	452.3	142.8
3D-S$_h$	0.01	**0.0235**	**104.2**	0.0309	119.7	105.5
	0.1	0.2671	**101.9**	**0.2595**	217.6	105.5
	0.2	0.5509	**122.2**	**0.5007**	296.8	105.5
	0.3	0.8226	**129.5**	**0.5285**	301.8	105.5



(a) R-a-E, $p = 0.1$ (b) E-a-R, $p = 0.1$

(c) R-a-E, $p = 0.2$ (d) E-a-R, $p = 0.2$

(e) R-a-E, $p = 0.4$ (f) E-a-R, $p = 0.49$

Fig. 2. UNN embeddings of 3D-S with missing data for missing rates $p = 0.1, 0.2$ and $p = 0.4$ for repair-and-embed (left column) and for embed-and-repair (right column)

Table 1 shows the experimental results for increasing data missing rates p on the data set 3D-S. The experimental results show that UNN with repair-and-embed achieves the lowest DSRE on both data sets. The results are very close to the DSRE achieved on the data set without missing values (complete). UNN with embed-and-repair achieves the lowest imputation error E_{imp} in seven of the eight cases, but much worse results for the DSRE. While the DSRE results

are still satisfactory with 1% of incomplete data, the approach fails for higher missing rates. Obviously, it is difficult to first determine the manifold structure from data sets with a high rate of missing values.

Figure 2 shows the embeddings of UNN on the 3D-S data set for increasing missing rates p. The left plots show the embeddings of repair-and-embed, the right plots the corresponding embeddings of embed-and-repair. The patterns are colorized w.r.t. their latent colors, i.e., patterns with similar colors are neighbored in latent space showing that the embedding has been successful. Figure 2(a) shows the embedding with a low missing rate of $p = 0.1$. The 3D-S is almost completely reconstructed, while the colors of the embedding show that a reasonable learning process took place. The higher the rate of incomplete data (cf. Figures (c) and (e)) the worse are the embeddings, i.e., different colors are neighbored. Higher missing rates can also be recognized by deviations from the S-structure.

UNN with embed-and-repair shows a comparatively good embedding for the low missing rate of $p = 0.1$. But the colors show that the embeddings are worse for higher missing rates. For example, Figure 2(f) shows that the embedding of UNN with repair-and-embed and data missing rate $p = 0.4$ leads to comparably bad results, which is consistent with the DSRE of Table 1.

5 Conclusions

Dimensionality reduction is an important task in practical data mining scenarios with high-dimensional data. In practical scenarios, data sets often suffer from missing entries. From the perspective of UNN, we have introduced two algorithmic variants that allow the efficient embedding of incomplete data. From the perspective of imputation, first repairing incomplete data is a straightforward approach. We introduced an iterative variant that takes into account the predicted values for completion of entries of patterns with an increasing data missing rate. First embedding patterns at locations with the lowest DSRE and then repairing the entries employing the neighbors in latent space is an approach that makes use of the intrinsic structure UNN regression assumes for imputation. This leads to comparatively good pattern reconstructions. But UNN employing repair-and-embed achieves better results, because the structure of the data (that can be reconstructed with KNN) has an important part to play for a successful embedding.

As our future work, we plan to compare both approaches in real-world scenarios with missing data, and to other hybridizations of repair approaches and dimensionality reduction algorithms.

References

1. Chechik, G., Heitz, G., Elidan, G., Abbeel, P., Koller, D.: Max-margin classification of data with absent features. Journal of Machine Learning Research 9, 1–21 (2008)
2. Dick, U., Haider, P., Scheffer, T.: Learning from incomplete data with infinite imputations. In: International Conference on Machine Learning (ICML), pp. 232–239 (2008)

3. Ghahramani, Z., Jordan, M.I.: Supervised learning from incomplete data via an EM approach. In: Advances in Neural Information Processing Systems (NIPS), pp. 120–127 (1993)
4. Kramer, O.: On evolutionary approaches to unsupervised nearest neighbor regression. In: Di Chio, C., et al. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 346–355. Springer, Heidelberg (2012)
5. Kramer, O.: On unsupervised nearest-neighbor regression and robust loss functions. In: International Conference on Agents and Artificial Intelligence (ICAART), pp. 164–170 (2012)
6. Kramer, O.: A Particle Swarm Embedding Algorithm for Nonlinear Dimensionality Reduction. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 1–12. Springer, Heidelberg (2012)
7. Kramer, O.: Unsupervised nearest neighbors with kernels. In: Glimm, B., Krüger, A. (eds.) KI 2012. LNCS, vol. 7526, pp. 97–106. Springer, Heidelberg (2012)
8. Schafer, J.L., Graham, J.W.: Missing data: Our view of the state of the art. Psychological Methods 7(2), 147–177 (2002)
9. Williams, D., Liao, X., Xue, Y., Carin, L., Krishnapuram, B.: On classification with incomplete data. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(3), 427–436 (2007)

A Hybrid ACO+CP for Balancing Bicycle Sharing Systems

Luca Di Gaspero[1], Andrea Rendl[2], and Tommaso Urli[1]

[1] Department of Electrical, Management and Mechanical Engineering,
University of Udine,
Via Delle Scienze, 206 - 33100 Udine, Italy
{luca.digaspero,tommaso.urli}@uniud.it
[2] DTS Mobility Department
Austrian Institute of Technology
Giefinggasse 2, 1210 Vienna, Austria
andrea.rendl@ait.ac.at

Abstract. Balancing bike sharing systems is an increasingly important problem, because of the rising popularity of this mean of transportation. Bike sharing systems need to be balanced so that bikes (and empty slots for returning bikes) are available to the customers, thus ensuring an adequate level of service.

In this paper, we tackle the problem of balancing a real-world bike sharing system (BBSP) by means of a hybrid metaheuristic method. Our main contributions are: *(i)* a new Constraint Programming (CP) formulation for the problem, and *(ii)* a novel hybrid approach which combines CP techniques with Ant Colony Optimization (ACO). We validate our approach against real world instances from the Vienna Citybike system.

1 Introduction

The idea of bike sharing is to provide bikes to the citizens via stations that are located all around the city. At each station, bikes are stored in special racks, such that users can easily pick up or return a bike. However, popular stations are often emptied or filled very quickly, resulting in annoyed users who cannot return or retrieve bikes. To avoid this, the stations must be balanced.

Bike sharing systems are balanced by distributing bikes from one station to another by using specific vehicles. Therefore, balancing the system corresponds to finding a tour for each vehicle, including loading and unloading instructions per station such that the resulting system is balanced. Clearly, balancing bike sharing systems is a difficult task, since it requires solving a vehicle routing problem combined with distributing single commodities (bikes) according to the target values at the stations.

In the following, we are consistent with the notation introduced in [1]. We consider balancing a bike sharing system with S stations $\mathcal{S} = \{1, \ldots, S\}$ and a set of depots $\mathcal{D} = \{S+1, \ldots, S+D\}$, where each station $s \in \mathcal{S}$ has a capacity

M.J. Blesa et al. (Eds.): HM 2013, LNCS 7919, pp. 198–212, 2013.

$C_s > 0$, a number of available bikes b_s and the number of target bikes t_s that denotes the number of bikes that should be at station s after balancing the system. We use V vehicles $\mathcal{V} = \{1, \ldots, V\}$ with capacity $c_v > 0$ and initial load $\hat{b}_v \geq 0$ that distribute the bikes within maximal $\hat{t}_v > 0$ time units. The travel times between stations (and the depots) is given by a travel time matrix $tt_{u,v}$ where $u, v \in \mathcal{S} \cup \mathcal{D}$, which includes also an estimate of the processing times needed to serve the station, if $v \in \mathcal{S}$.

We want to achieve a maximally balanced system where each vehicle travels on a minimal route. Therefore, in our cost function, we minimize the sum of the deviations from the target value for each station and include both the travel distance and the overall activity of each vehicle as a measure of the work effort.

In this paper, we first introduce a novel Constraint Programming (CP) model that is based on a vehicle routing formulation. Then we show how we can generally combine CP with Ant Colony Optimization (ACO) by utilizing ACO as search engine in the CP solving process. The combination is based on the idea of tackling the problem as a bi-level optimization problem, in which the routing variables are handled by ACO, whereas the operation variables (which model the number of bikes to load or unload) are taken care of by CP. Finally, we show with an experimental evaluation on real-world instances from Citybike Vienna, that the hybrid ACO+CP approach outperforms the pure CP formulation.

2 Related Work

A few approaches for integrating ACO and CP are available from the literature. The first attempt is due to Meyer and Ernst [2], who apply the method for solving a Job-Shop Scheduling problem. The proposed procedure employs ACO to learn the learning strategy used by CP in the tree-search. The solutions found by CP are fed back to ACO, in order to update its probabilistic model. In this approach, ACO can be conceived as a master online-learning branching heuristic aimed at enhancing the performance of a slave CP solver.

A slightly different approach has been taken by Khichane et al. [3,4]. Their algorithm works in two phases. At first CP is employed to sample the space of feasible solutions and the information collected is processed by the ACO procedure for updating the pheromone trails. In the second phase, the pheromone information is employed as the value ordering used for CP branching. Unlike the previous one, this approach uses the learning capabilities of ACO in an offline fashion. More standard approaches in which CP is used to keep track of the feasibility of the solution constructed by ACO and to reduce the domains through Constraint Propagation have been used by a number of authors. This idea has been applied to Job-Shop Scheduling [2] and Car Sequencing [5].

Our approach also shares some similarities with Large Neighborhood Search (LNS) [6] in that *(i)* we exploit constraint propagation to reduce the domains of the variables and *(ii)* we handle different subsets of variables separately. However, unlike LNS, our search process includes a learning component. Moreover, our separate treatment of variables is motivated by the good performance of ACO on routing problems, rather than by a need for a better neighborhood exploration.

Balancing of bike sharing systems has become an increasingly studied problem in the last few years. Benchimol et al. [7] consider the rebalancing as hard constraint and the objective is to minimize the travel time. They study different approximation algorithms on various instance types and derive different approximation factors for certain instance properties. Furthermore, they present a branch-and-cut approach based on an ILP including subtour elimination constraints. Contardo et al. [8] consider the dynamic variant of the problem and present a MIP model and an alternative Dantzig-Wolfe decomposition and Benders decomposition method to tackle larger instances. Raviv et al. [9] present two different MILP formulations for the static BBSP and also consider the stochastic and dynamic factors of the demand. In the approach of Chemla et al. [10], a branch-and-cut approach based on a relaxed MIP model is used in combination with a tabu search that provides upper bounds. Rainer-Harbach et al. [1] propose a heuristic approach for the BBSP in which effective routes are calculated by a variable neighbourhood search (VNS) metaheuristic and the loading instructions are computed by a helper algorithm, where they study three different alternatives (exact and heuristic) as helper algorithms.

Schuijbroek et al. [11] propose a new cluster-first route-second heuristic, in which the clustering problem simultaneously considers the service level feasibility constraints and approximate routing costs. Furthermore, they present a constraint programming model for the BBSP that is based on an scheduling formulation of the problem and therefore differs significantly from our VRP-based formulation.

3 A Constraint Model for BBSP

Our constraint model is based on the constraint model [12] of the classical Vehicle Routing Problem (VRP) that is concerned with servicing a set of customers with a fleet of vehicles with cost-optimal tours. The VRP model employs successor and precessor variables to represent the path of each vehicle on a special graph G_{VRP} that consists of three different kinds of nodes: first, the starting node for each vehicle (typically the respective depot), second, the nodes that should be visited in the tour, and third, the end nodes for each vehicle, again typically the respective depot. In summary, G_{VRP} contains $2V + S$ nodes, where V is the number of vehicles and S is the number of nodes to visit. This graph structure allows to easily define successor and predecessor variables to represent paths.

We extend the VRP model to allow unvisited stations and to capture loading instructions on a per-station basis. To achieve this we introduce a dummy vehicle that (virtually) visits all the unserviced stations. This formulation allows to treat unvisited stations as a cost component, and makes it easier to ensure that no operations are scheduled for unvisited stations by constraining the load of the dummy vehicle to be always zero. This results in an extension of G_{VRP} to graph G_{BBSP} that contains $2(V + 1) + S$ nodes, where $V + 1$ is the number of vehicles including the dummy vehicle. This encoding is illustrated in Figure 1, where the basic structure is shown on the lower layer, and the encoded G_{BBSP} and a

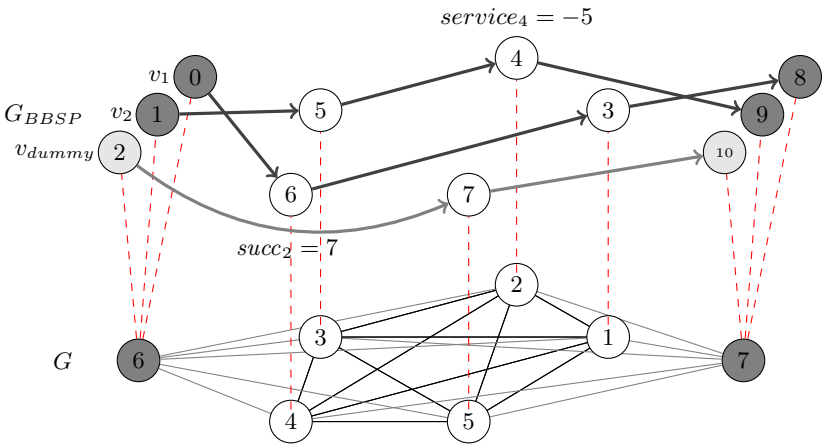


Fig. 1. Graph encoding of the BBSP employed in the routing CP model. The lower layer shows the original graph, whereas the upper layer shows the encoded graph in the case of two vehicles, and a solution. The path starting at node 2 and ending at node 10 (i.e., the dummy vehicle) corresponds to the set of unserved nodes.

possible solution is shown on the upper layer. We represent all nodes in G_{BBSP} in the ordered set $\mathcal{U}$ which is defined as follows:

$$\begin{array}{l|l} \mathcal{U} = \{\ 0, \ldots, V, & \mathcal{V}_s\text{: start nodes} \\ \quad V+1, & \text{station 1} \\ \quad V+2, & \text{station 2} \\ \quad \ldots, & \ldots \\ \quad V+S, & \text{station } S \\ \quad V+S+1, \ldots, 2V+S+2\ \} & \mathcal{V}_e\text{: end nodes} \end{array}$$

Thus, $\mathcal{U}$ first contains the starting nodes (depots) for the V vehicles and the dummy vehicle, followed by the S regular stations, and finally the end nodes (depots) for V vehicles and the dummy. Note, that we denote $\mathcal{V}_s = \{0, \ldots, V\}$ the set of start nodes of vehicles and $\mathcal{V}_e = \{V+S+1, \ldots, 2V+S+2\}$ the set of end nodes of each vehicle. Thus, $\mathcal{U} = \mathcal{V}_s \cup \mathcal{S} \cup \mathcal{V}_e$. In summary, the tour of vehicle $v \in \mathcal{V}$ starts at a depot in $\mathcal{V}_s$, continues to some station nodes in $\mathcal{S}$ and ends at a depot in $\mathcal{V}_e$. In the following, we give a detailed description of our model.

3.1 Variables

The first set of variables are the successor variables that represent the paths by defining the successor of each node in $\mathcal{U}$. Thus, we have $|\mathcal{U}|$ successor variables *succ* that range over $\mathcal{U}$, where $succ_i$ represents the node following node i. In addition, we define predecessor variables *pred* where $pred_i$ denotes the node which comes just before node i in the route. Though redundant, predecessor

Table 1. Variables in the CP Model

name[dimension]	domain	description
succ $[\mathcal{U}]$	$\mathcal{U}$	successor of node $i \in \mathcal{U}$
pred $[\mathcal{U}]$	$\mathcal{U}$	predecessor of node $i \in \mathcal{U}$
vehicle $[\mathcal{U}]$	$\mathcal{U}$	vehicle serving node $i \in \mathcal{U}$
service $[\mathcal{U}]$	$[\pm max(C_{max}, c_{max})]$	removed/added bikes at node $i \in \mathcal{U}$
load $[\mathcal{U}]$	$[0, c_v]$	load of vehicle v after serving node $i \in \mathcal{U}$
time $[\mathcal{U}]$	$[0, \hat{t}_v]$	time when vehicle v arrives at node $i \in \mathcal{U}$
loadTime $[\mathcal{U}]$	$[0, \hat{L}]$	loading time at node $i \in \mathcal{U}$
deviation $[\mathcal{S}]$	$\mathcal{S}$	deviation from target at station $s \in \mathcal{S}$
cost	$[l, u]$	overall cost of the solution

variables channelled with successor variables result in stronger propagation [12]. Second, we associate a vehicle to each node i by the variable $vehicle_i$ that ranges over $\{0, \ldots, V\}$. The loading instructions for each node are captured by operation variables *service* where $service_i$ represents the number of bikes that are added or removed at node $i \in \mathcal{U}$ and ranges over $[\pm max(C_{max}, c_{max})]$ where C_{max} and c_{max} are respectively the maximum capacities of stations and vehicles. We also introduce load variables $load_i$, which represent the load of the vehicle after visiting node $i \in \mathcal{U}$. Next come the time-related variables: $time_i$ constitutes the arrival time at which a vehicle arrives at node i. In our problem formulation, the arrival time also includes the processing time, i.e., the time for loading/unloading the vehicle at that node. Finally, we use S deviation variables *deviation* where $deviation_s$ represents the deviation from the target values at station $s \in \mathcal{S}$ after the balancing tours. Variables are summarized in Tab. 1.

3.2 Constraints

We divide the introduction of the constraints in the model by first stating the essential constraints that are required to comprehensively model the problem, and then discussing some redundant constraints that will help the solution process.

Essential Constraints. We start our description with the routing constraints for the path: all successors and predecessors take different values.

$$\texttt{alldifferent}(succ) \tag{1}$$

$$\texttt{alldifferent}(pred) \tag{2}$$

Note that, while these constraints are alone not sufficient to eliminate subtours from the solutions, according to [12] the presence of a finite time horizon for vehicles (which is the case for BBSP) ensures the absence of cycles. In the case of the dummy vehicle, which has no finite horizon, a similar task is carried out by a symmetry breaking constraint which enforces an ordering of the nodes in the route, effectively making subtours impossible to occur.

Then we set the successor-predecessor chain for each regular station

$$pred_{succ_s} = s \quad \forall s \in \mathcal{S} \tag{3}$$
$$succ_{pred_s} = s \quad \forall s \in \mathcal{S} \tag{4}$$

and the successor-predecessor chain for the start and end nodes where $\hat{s} = V+S$ represents the index of first end node in $\mathcal{U}$:

$$pred_v = \hat{s} + v \quad \forall v \in \mathcal{V}_s \tag{5}$$
$$succ_{\hat{s}+v} = v \quad \forall v \in \mathcal{V}_s \tag{6}$$

Furthermore, no loops are allowed in the paths, i.e.

$$pred_i \neq i \quad \forall i \in \mathcal{U} \tag{7}$$
$$succ_i \neq i \quad \forall i \in \mathcal{U} \tag{8}$$

We continue with constraints on the vehicle variables. First, we set the respective vehicle $v \in \mathcal{V}_s$ for each start- and end-node in the path:

$$vehicle_v = v \quad \forall v \in \mathcal{V}_s \tag{9}$$
$$vehicle_{\hat{s}+v} = v \quad \forall v \in \mathcal{V}_s \tag{10}$$

and second, we set the vehicle-chain over the path variables:

$$vehicle_{succ_i} = vehicle_i \quad \forall i \in \mathcal{U} \tag{11}$$
$$vehicle_{pred_i} = vehicle_i \quad \forall i \in \mathcal{U} \tag{12}$$

For the loading constraints, we first set the initial load $\hat{b}_v$ and constrain the dummy vehicle to be empty

$$load_v = \hat{b}_v \quad \forall v \in \mathcal{V}_s \setminus \{V\} \tag{13}$$
$$load_V = 0 \tag{14}$$

and continue with the loading restrictions along a path

$$load_{succ_i} = load_i - service_i \quad \forall i \in \mathcal{U} \tag{15}$$

Finally, every vehicle must be completely empty at the end of the route, i.e.:

$$load_v = 0 \quad \forall v \in \mathcal{V}_e \tag{16}$$

Additionally, we constrain the load for vehicles: if station s is not served by the dummy vehicle (V), then the service must not be zero, and vice versa:

$$(vehicle_s \neq V) \iff (service_s \neq 0) \quad \forall s \in \mathcal{S} \tag{17}$$

Furthermore, the load of the vehicle after visiting station $s \in \mathcal{S}$ may not exceed its capacity c:

$$load_s \leq c_{vehicle_s} \quad \forall s \in \mathcal{S} \tag{18}$$

Next come the operation constraints. At first we impose *operation monotonicity*, i.e., services at station s should either force loading or unloading bikes depending on the current number of bikes b_s and the target value of bikes t_s:

$$service_s \leq 0 \quad \forall_{s \in \mathcal{S}} : b_s > t_s \tag{19}$$
$$service_s \geq 0 \quad \forall_{s \in \mathcal{S}} : b_s < t_s \tag{20}$$

Notice that a service value of 0 is admissible in both cases since a station could remain unserved (e.g., because of the time budget constraints). The service at the start and end nodes (depots) i is zero for all vehicles:

$$service_i = 0 \quad \forall i \in \mathcal{V}_s \tag{21}$$
$$service_i = 0 \quad \forall i \in \mathcal{V}_e \tag{22}$$

Furthermore, the service is limited by the maximal number of bikes in the station, and we cannot have a negative number of bikes:

$$b_s + service_s \leq C_s \quad \forall s \in \mathcal{S} \tag{23}$$
$$b_s + service_s \geq 0 \quad \forall s \in \mathcal{S} \tag{24}$$

Finally, we state the time constraints, where we begin with setting the arrival time (and processing time) at the start depots to zero

$$time_v = 0 \quad \forall v \in \mathcal{V}_s \tag{25}$$

and set the time chain for the successor and predecessor variables:

$$time_v = time_{pred_v} + tt_{pred_v,v} \quad \forall v \in \mathcal{S} \cup \mathcal{V}_e \tag{26}$$
$$time_{succ_v} = time_v + tt_{v,succ_v} \quad \forall v \in \mathcal{V}_s \cup \mathcal{S} \tag{27}$$

At last, the overall working time for each vehicle must be within its time budget:

$$time_{\hat{s}+v} \leq \hat{t}_v \quad \forall v \in \mathcal{V} \tag{28}$$

This concludes the description of the essential of our CP model for the BBSP problem. The model can be enhanced by some redundant constraints, that will take care of some particular substructure of the problem.

Redundant Constraints. First, because of the monotonicity constraints (19–20), the stations requiring the unloading of bikes must be removed from the successors of the starting depots

$$succ_i \neq j \quad \forall i \in \mathcal{V}_s, j \in \{s \in \mathcal{S} | b_s < t_s\} \tag{29}$$

Similarly, because of constraint (16), which requires empty vehicles at the end of the path, the stations requiring the loading of bikes must be removed from the predecessors of the ending depots

$$pred_i \neq j \quad \forall i \in \mathcal{V}_e, j \in \{s \in \mathcal{S} | b_s > t_s\} \tag{30}$$

Finally, an early failure detection of the working time constraint (28) is possible. If the working time of the current partial solution plus the time to reach the final depot exceeds the total time budget, then the solution can't be feasible.

$$time_i + tt_{i,\hat{s}+vehicle_i} \leq \hat{t}_{vehicle_i} \quad \forall i \in \mathcal{S} \tag{31}$$

Cost Function. The cost function of the problem is a hierarchical one, and comprises two different major components: the level of unbalancing and the working effort.

The unbalancing component is defined in terms of the *deviation* variables, which are set to be the absolute value of the deviation from the target number of bikes at each station after service has been performed, i.e.:

$$deviation_s = |b_s + service_s - t_s| \quad \forall s \in \mathcal{S} \tag{32}$$

The working effort is the sum of the total traveling time (i.e., the sum of the times at which each vehicle reaches its ending depot) plus the overall activity performed throughout the path (i.e., the absolute value of the *service*).

The cost function is the weighted aggregation of the two components, i.e.:

$$cost = w_1 \sum_{s \in \mathcal{S}} deviation_s + w_2 (\sum_{v \in \mathcal{V}} time_{\hat{s}+v} + \sum_{s \in \mathcal{S}} |service_s|) \tag{33}$$

where $w_1 = 1$ and $w_2 = 10^{-5}$, so that the satisfaction of the first component prevails over the second one. This cost function, defined in [1], is the *scalarization* of a multi-objective problem in nature, thus some points in the Pareto optimal set are neglected by construction. The main reason for this choice was the need to compare with the current bests, moreover, to the best of our knowledge, research in multi-objective propagation techniques is still at an early stage.

4 An ACO+CP Hybrid

Ant Colony Optimization [13] is an iterative constructive metaheuristic, inspired by the ant foraging behavior. The ACO construction process is driven by a probabilistic model, based on pheromone trails, which are dynamically adjusted by a learning mechanism. Constraint Programming(CP) [14] is an exact solving approach where a constraint model is solved using a customized search strategy interleaved with strong filtering (propagation) of the variables' domains.

The hybridization of ACO and CP is described in Algorithm 1 and is, in its essence, a bi-level optimization process. The basic idea is to partition the set X of problem variables into two sets X_{Ants} and X_{CP}. The values for the X_{Ants} variables are dealt with by an ACO procedure and once they are set, a tree-search (line 12) finds the values for the remaining variables. The tree-search procedure can be either a branch-and-bound algorithm (exploiting the information of the cost function $\mathcal{F}$) or a (possibly) faster depth-first-search if we are satisfied with a good assignment of the X_{CP} variables.

Algorithm 1. ACO + CP

```
input : X = X_Ants ∪ X_CP, a set of integer variables partitioned into variables
        dealt with ACO and CP, respectively
        C, a set of constraints
        F, a cost function
1  initialize all pheromone trails to τ_start;
2  g ← 0;
3  repeat
4      for k ∈ {1, ..., n} do
5          A_k ← ∅;
6          repeat
7              select a variable x_i ∈ X_Ants, so that x_i ∉ var(A_k), and a value
               j ∈ D_j according to the pheromone trail τ_ij (and possibly the
               heuristic information η_ij);
8              add {x_i := j} to A_k;
9              if Propagate(A_k, C) = Failure then
10                 Backtrack(A_k);
11         until var(A_k) = X_Ants;
12         TreeSearch(X, C, F);
13         update pheromone trails using {A_1, ..., A_n} and F;
14         g ← g + 1;
15 until TerminateSearch(g, A_i, time);
```

Once all n ants have found a solution, the pheromone trails are updated according to the solution components and their cost value. The overall search is typically stopped at a given timeout. It is worth noticing that, in the proposed approach, the interaction between ACO and CP is two-way. The pruning capabilities of constraint propagation are employed both to restrict the number of alternatives the ants must face at each choice point (see line 7), and as a mechanism for early failure detection (see line 9). On the other hand, ACO helps CP converging faster, by avoiding search paths that lead to unfeasible solutions.

4.1 ACO+CP for BBSP

In our CP model for the BBSP problem, there is a natural partition of the decision variables into two families, i.e., routing and operation variables.

Handling of Routing Variables. The first set of variables, $succ_i$, is handled very naturally by ACO, which has been shown to be particularly effective in solving routing problems. In our approach, ACO is embodied by a two-phase branching strategy which takes care both of variable and value selection. This process is illustrated in Figure 4.1.

Variable selection. The first variable to be selected, according to the heuristic, is the *succ* of the first vehicle starting depot (Fig. 2(a)). As for the next variable to assign, we always choose the one indicated by the value of the last assigned variable, i.e., the *succ* of the last assigned node (Fig. 2(b)). By following this heuristic, we enforce the completion of existing paths first. If the successor of the last assigned node is a final depot (Fig. 2(c)), then we cannot proceed further on the current path, and we start a new one by assigning the successor of the next starting depot. Once the paths of all vehicles are set, the remaining unserved nodes will be assigned to the dummy vehicle (Fig. 2(d)).

Value selection. Once the next variable to assign is chosen, all the values in its current domain are considered as candidates. Note that, in this, we are in fact exploiting problem-specific knowledge, as the domain of a variable is, at any time, determined by the constraint propagations activated earlier in the search.

The next step is where ACO comes into play. For our approach we have chosen a popular ACO variant known as the hyper-cube framework for ACO (HC-ACO) [15]. As most other ACO approaches, HC-ACO maintains a pheromone table in which each $\langle X_i, v_j \rangle$ (variable, value) pair has a corresponding $\tau_{i,j}$ pheromone value indicating the desirability of value v_j for the variable X_i. The advantage of HC-ACO over other ACO variants, is that the update rule for pheromones involves a normalization factor which makes the approach independent of the scale of the cost function and doesn't require to enforce a $[\tau_{min}, \tau_{max}]$ interval.

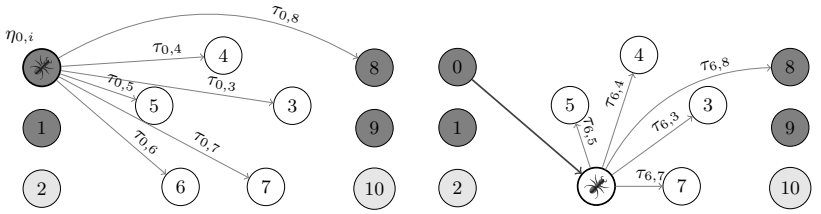
In line with the majority of ACO variants, our value selection heuristic is stochastic, with the probability of choosing a specific value being proportional to the corresponding τ-value. In particular, the probability $P(X_i, v_j)$ of choosing the value v_j for the variable X_i is

$$P(X_i, v_j) = \frac{\tau_{i,j}}{\sum_{v_k \in dom(X_i)} \tau_{i,k}} \tag{34}$$

Handling of Operation Variables. The operation variables are assigned through depth-first tree-search, based on *deviation* variables, which are the main component of our cost function. This way, employing a min value heuristic, lower cost solutions are produced before bad ones.

While other choices are possible, e.g. a full exploration of the tree by branch-and-bound, in this context we aim at finding quickly feasible solutions, so that they can be used for learning. The rationale behind this choice is that decisions taken towards the root of the search tree have a greater impact than the ones taken towards the leaves, and τ-updates are the only way to improve our ACO-based value selection heuristic.

τ Update. After all n ants have produced a feasible solution (we call this set of solutions S_{upd}), their cost function value is used to make an update to the pheromone table. The update rule is the one described in [15] (adapted to be consistent with our conventions)



(a) The ant is first placed at the starting depot of the first vehicle.

(b) The value of the $succ_i$ variable is selected according to the pheromones $\tau_{i,j}$.

(c) Once the ending depot is reached, the ant starts with the route of the next vehicle.

(d) All remaining nodes are assigned to the dummy vehicle (i.e., they are left unserved).

Fig. 2. Illustration of the graph traversal performed by one ant

$$\tau_{i,j} = (1 - \rho) \cdot \tau_{i,j} + \rho \cdot \sum_{s \in S_{upd}} \frac{F(s)}{\sum_{s' \in S_{upd}} F(s')} \tag{35}$$

where ρ is a learning rate that controls how fast the pheromones adapt to make the new solutions more likely and F is a *quality* function that in our case is defined as $F(s) = 1/cost(s)$. Note that *all* pheromones are also subject to a multiplicative evaporation of $1 - \rho$.

5 Experimental Analysis

In this section we report and discuss the experimental analysis of the algorithms. The experimental setting is as follows.

For fair comparison, both the CP and the ACO+CP algorithms were implemented in Gecode (v 3.7.3) [16], the ACO variant consisting in specialized branching and search strategies.

All pheromones were initially set to $\tau_{\max} = 1$. The ρ parameter and the number of ants have been tuned by running an *F-Race* [17] with a confidence level of 0.95 over a pool of 210 benchmark instances from Citybike Vienna. Each instance, featuring a given number of stations, was considered with different number of vehicles ($V \in \{1, 2, 3, 5\}$) and time budgets ($\hat{t} \in \{120, 240, 480\}$). Moreover, the algorithms were allowed to run for three different timeouts (30, 60, 120 seconds), totaling 7560 problems.

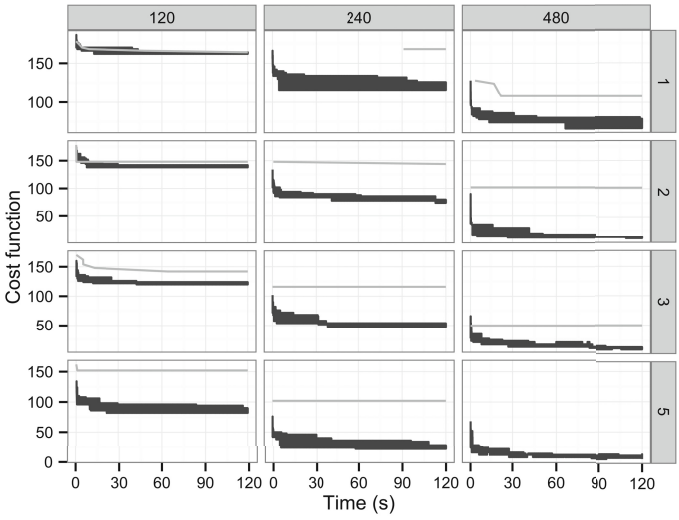


Fig. 3. Comparison between ACO+CP (dark, solid lines) and CP (light, thin lines) on a problem instance with 30 stations. The columns of the graph matrix represent the vehicle time budget and the rows represent the number of available vehicles.

We tuned the number of ants $n \in \{5, 10, 15, 20\}$ and the learning rate ρ together, as we expected an interaction between the two parameters. The 8 candidate values for ρ were instead sampled from the low-discrepancy Hammersley point set in $[0.4, 0.8]$. This interval was chosen according to a preliminary tuning of the parameters, with $\rho \in [0, 1]$ and 32 samples. The result of the tuning process is that, for the considered set of problems, the best setup involves 5 ants and $\rho = 0.65$. All the experiments were executed on an Ubuntu Linux 12.04 machine with 16 Intel® Xeon® CPU E5-2660 (2.20GHz) cores.

Comparison between CP and ACO+CP. The main goal of this comparison is to understand if a dynamic branching strategy based on ACO can indeed outperform a static branching strategy. Figure 3 shows the results on an instance from the Citybike Vienna benchmark set featuring 30 stations. The choice of this instance has been driven by the fact that a time budget of 2 minutes was too low for CP to obtain even a single solution on larger instances.

The results of the comparison show that ACO+CP clearly outperforms the pure CP approach. In fact, the CP solver is declared significantly inferior by the F-Race procedure after just 15 iterations. The superior behavior of ACO+CP is confirmed also from the analysis reported in Figure 3, for the variants of a single problem instance with 30 stations. Note that the ACO+CP data is based on 5 repetitions of the same experiment, as the process is intrinsically stochastic.

It is possible to see that the cost values achieved by ACO+CP are always lower than those of CP and in one case (namely time budget 480 and 5 vehicles) CP is even not able to find a solution within the granted timeout despite the fact that it is somehow a loosely constrained instance.

Table 2. Comparison of our CP and ACO+CP solvers with the MIP and the best VNS approach of [1]

Instance			CP			ACO+CP			MIP [1]			VNS [1]	
S	V	$\hat{t}$	$\overline{obj}_{30}$	$\overline{obj}_{60}$	$\overline{obj}_{120}$	$\overline{obj}_{30}$	$\overline{obj}_{60}$	$\overline{obj}_{120}$	$\overline{ub}$	$\overline{lb}$	$\overline{time}$	$\overline{obj}$	$\overline{time}$
10	1	120	**28.3477**	**28.3477**	**28.3477**	28.5344	28.7477	28.5478	**28.3477**	28.3477	4	**28.3477**	2
10	1	240	14.0908	11.4915	9.5589	5.2276	4.7609	4.4810	4.2942	0.0424	3600	**4.2941**	10
10	1	480	14.8247	13.2922	9.8942	0.4322	0.9120	0.6052	0.0320	0.0276	3600	**0.0317**	17
10	2	120	10.2266	10.2266	10.2266	10.4001	10.6667	10.4268	**9.8269**	9.4768	911	9.9601	3
10	2	240	5.3652	4.6987	2.7662	0.4342	0.9274	0.1009	0.0340	0.0322	856	**0.0339**	19
10	2	480	5.3637	4.8971	3.2976	0.4854	0.4586	0.8584	**0.0317**	0.0313	1245	**0.0317**	15
20	2	120	72.4942	70.4279	68.7614	64.2558	62.9492	61.4561	**5.8294**	26.9012	3600	55.3628	8
20	2	240	74.2422	72.3754	71.5087	18.3904	17.3372	15.3907	19.7884	0.0383	3600	**4.2575**	58
20	2	480	74.1093	72.7093	72.5756	3.9748	2.7743	2.8943	1.8906	0.0403	3600	**0.0615**	142
20	3	120	67.5712	64.1051	61.5053	48.4287	47.1622	45.0557	37.3759	1.4770	3600	**31.7763**	13
20	3	240	74.3813	74.1811	74.1143	7.5511	5.8310	4.7641	6.2083	0.0401	3600	**0.0650**	65
20	3	480	74.3814	74.1811	74.1144	4.5878	2.5211	2.5874	13.4191	0.0316	3600	**0.0614**	114
30	2	120	127.5604	126.4939	125.5608	122.4552	119.2022	118.0823	106.9631	56.3908	3600	**104.7633**	12
30	2	240	117.2520	116.5857	116.3188	75.7637	73.2173	70.4309	74.9886	0.0487	3600	**34.6608**	109
30	2	480	101.8650	101.8650	101.4652	13.5173	11.1311	9.3847	69.8069	0.0432	3600	**0.0925**	491
30	3	120	–	117.7058	115.6393	107.7879	105.1748	102.0554	90.4419	16.6454	3600	**78.1773**	21
30	3	240	104.6052	104.4719	104.0054	46.0564	42.7502	40.5769	61.6715	0.0461	3600	**7.1523**	191
30	3	480	100.7422	100.6089	100.6089	–	10.7450	8.5595	175.4000	0.0015	3600	**0.0925**	399
60	3	120	–	–	–	307.8148	304.4283	300.2154	274.3101	157.7350	3600	**253.8462**	45
60	3	240	–	–	–	245.3374	238.1644	233.6585	370.2000	0.0000	3600	**126.8282**	521
60	3	480	205.8871	205.8871	205.8870	127.2744	122.7286	117.6223	–	–	3600	**6.7758**	3600
60	5	120	–	–	–	283.0537	278.0540	272.8145	289.3111	34.9784	3600	**196.6749**	99
60	5	240	–	–	–	184.7371	179.0572	173.6710	370.2000	0.0000	3600	**41.6161**	1556
60	5	480	–	–	–	–	–	–	–	–	3600	**0.1902**	3600
90	3	120	–	–	–	511.8807	507.2943	504.2013	492.2319	290.8990	3600	**441.6473**	82
90	3	240	–	–	–	451.7232	445.4705	438.2044	566.2667	0.0000	3600	**294.5646**	985
90	3	480	–	–	–	334.6610	326.4350	319.5826	–	–	3600	**101.1221**	3600
90	5	120	–	–	–	490.3193	480.7739	473.9345	566.2667	0.0000	3600	**376.1432**	169
90	5	240	–	–	–	393.4433	383.3375	375.5915	–	–	3600	**174.3566**	3304
90	5	480	–	–	–	213.3140	202.3017	192.3832	–	–	3600	**1.6855**	3600

Comparison with Other Methods. In this second experiment, we compare ACO+CP and CP, with state-of-the-art results of [1], who solved the same set of instances by a Mixed Integer Linear Programming solver (MILP) and a Variable Neighborhood Search (VNS) strategy. The results of the comparison are reported in Table 2, where we compare against the best of the three different VNS strategies described in [1]. The results reported are averages across instances with the same number of stations.

In this respect, the results are still unsatisfactory, since the best VNS approach of [1] is outperforming our ACO+CP on almost all instances. Nevertheless, our ACO+CP is able to do better than the MIP approach for mid- and large-sized instances.

6 Conclusions and Future Work

In this paper we tackle the problem of balancing the Citybike Vienna bike sharing system by means of a hybrid ACO+CP approach. The contributions of the paper are twofold.

First, we devise a novel CP formulation for the problem based on an extension of the classical CP vehicle routing model [12]. Up to the best of our knowledge, this is, together with [11], one of the two available CP formulations. Second, we propose a novel hybrid ACO+CP approach with the aim of improving the results of the pure CP solver. The proposed hybridization approach is quite general and can be applied also to other problems having a similar bi-level optimization structure. Moreover, the hybrid approach is implemented as an extension of the Gecode CP system and requires a small customization for handling different problem models.

From our experiments, it is clear that the ACO+CP approach outperforms the standard branch-and-bound CP solution method. However, despite these promising initial results, the performances of ACO+CP are still not as good as those achieved by the state-of-the-art metaheuristic approaches for this problem.

Among the alternatives we want to explore, there is the validation of the proposed ACO+CP approach on other bi-level optimization problems such as the integrated vehicle routing and packing problem. Moreover, we plan to investigate other methods for combining metaheuristics and CP, e.g., LNS.

Acknowledgements. This work is part of the project BBSP, partially funded by the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT) within the strategic programme I2VSplus under grant 831740 and it has been also supported by Google Inc. under the Google Focused Grant Program on "Mathematical Optimization and Combinatorial Optimization in Europe". We thank Marian Rainer-Harbach, Petrina Papazek, Bin Hu and Günther R. Raidl from the Vienna University of Technology, and Matthias Prandtstetter and Markus Straub from the Austrian Institute of Technology, and City Bike Vienna for the collaboration in this project, constructive comments and for providing the test instances and the results of the methods they developed.

References

1. Rainer-Harbach, M., Papazek, P., Hu, B., Raidl, G.R.: Balancing bicycle sharing systems: A variable neighborhood search approach. In: Middendorf, M., Blum, C. (eds.) EvoCOP 2013. LNCS, vol. 7832, pp. 121–132. Springer, Heidelberg (2013)
2. Meyer, B., Ernst, A.: Integrating ACO and constraint propagation. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) ANTS 2004. LNCS, vol. 3172, pp. 166–177. Springer, Heidelberg (2004)
3. Khichane, M., Albert, P., Solnon, C.: CP with ACO. In: Perron, L., Trick, M. (eds.) CPAIOR 2008. LNCS, vol. 5015, pp. 328–332. Springer, Heidelberg (2008)
4. Khichane, M., Albert, P., Solnon, C.: Strong combination of ant colony optimization with constraint programming optimization. In: Lodi, A., Milano, M., Toth, P. (eds.) CPAIOR 2010. LNCS, vol. 6140, pp. 232–245. Springer, Heidelberg (2010)
5. Khichane, M., Albert, P., Solnon, C.: Integration of ACO in a constraint programming language. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) ANTS 2008. LNCS, vol. 5217, pp. 84–95. Springer, Heidelberg (2008)

6. Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics. International Series in Operations Research & Management Science, vol. 146, pp. 399–419. Springer, US (2010)
7. Benchimol, M., Benchimol, P., Chappert, B., De la Taille, A., Laroche, F., Meunier, F., Robinet, L.: Balancing the stations of a self service bike hire system. RAIRO – Operations Research 45(1), 37–61 (2011)
8. Contardo, C., Morency, C., Rousseau, L.M.: Balancing a Dynamic Public Bike-Sharing System. Technical Report CIRRELT-2012-09, CIRRELT, Montreal, Canada, submitted to Transportation Science (2012)
9. Raviv, T., Tzur, M., Forma, I.A.: Static Repositioning in a Bike-Sharing System: Models and Solution Approaches. EURO Journal on Transportation and Logistics (2012), doi:10.1007/s13676-012-0017-6
10. Chemla, D., Meunier, F., Calvo, R.W.: Bike sharing systems: Solving the static rebalancing problem. Discrete Optimization 10(2), 120–146 (2013)
11. Schuijbroek, J., Hampshire, R., van Hoeve, W.J.: Inventory Rebalancing and Vehicle Routing in Bike Sharing Systems. Technical Report 2013-E1, Tepper School of Business, Carnegie Mellon University (2013)
12. Kilby, P., Shaw, P.: Vehicle routing. In: Rossi, F., van Beek, P., Walsh, T. (eds.) Handbook of Constraint Programming, pp. 799–834. Elsevier, New York (2006)
13. Dorigo, M., Birattari, M.: Ant colony optimization. In: Encyclopedia of Machine Learning, pp. 36–39 (2010)
14. Rossi, F., van Beek, P., Walsh, T.: Handbook of Constraint Programming (Foundations of Artificial Intelligence). Elsevier Science Inc., New York (2006)
15. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34(2), 1161–1172
16. Gecode: Generic constraint development environment (2006), `http://www.gecode.org`
17. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-race and iterated f-race: An overview. In: Experimental Methods for the Analysis of Optimization Algorithms, pp. 311–336 (2010)

Author Index