

Variable and large neighborhood search to solve the multiobjective set covering problem

T. Lust · D. Tuyttens

Received: 18 May 2011 / Revised: 17 September 2013 / Accepted: 21 December 2013 /
Published online: 8 January 2014
© Springer Science+Business Media New York 2014

Abstract In this paper, we study the multiobjective version of the set covering problem. To our knowledge, this problem has only been addressed in two papers before, and with two objectives and heuristic methods. We propose a new heuristic, based on the two-phase Pareto local search, with the aim of generating a good approximation of the Pareto efficient solutions. In the first phase of this method, the supported efficient solutions or a good approximation of these solutions is generated. Then, a neighborhood embedded in the Pareto local search is applied to generate non-supported efficient solutions. In order to get high quality results, two elaborate local search techniques are considered: a large neighborhood search and a variable neighborhood search. We intensively study the parameters of these two techniques. We compare our results with state-of-the-art results and we show that with our method, better results are obtained for different indicators.

Keywords Multiobjective optimization · Combinatorial optimization · Metaheuristics · Set covering problem · Large neighborhood search · Variable neighborhood search

T. Lust (✉)
LIP6, Université Pierre et Marie Curie,
4, Place Jussieu, 75252 Paris Cedex 05, France
e-mail: thibaut.lust@lip6.fr

D. Tuyttens
Laboratory of Mathematics and Operational Research,
Faculté Polytechnique de Mons, University of Mons,
20, Place du Parc, 7000 Mons, Belgium
e-mail: daniel.tuyttens@umons.ac.be

1 Introduction

Since the 70s, *multiobjective optimization* (MO) became an important field of operations research. In many real applications, there exist effectively more than one objective to be taken into account to evaluate the quality of the feasible solutions.

It is first the problems with continuous variables that called the attention of the researchers [see the book of Steuer (1986) for multiobjective *linear programming* (MOLP) problems and of Miettinen (1999) for multiobjective *non-linear programming* (MONLP) problems]. However it is well-known that discrete variables are often unavoidable in the modeling of many applications and multiobjective combinatorial optimization (MOCO) problems have been considered afterwards (Ehrgott and Gandibleux 2002). MOCO problems are generally defined as follows:

$$\begin{aligned} \text{“min”} \quad & z(x) = (z_1(x), \dots, z_l(x), \dots, z_p(x)) \quad (\text{MOCO}) \\ \text{s.t.} \quad & x \in \mathcal{X} = D \cap \{0, 1\}^n \end{aligned}$$

where D is a specific polytope characterizing the particular CO problem, n is the number of variables, $z_k(x) : \mathcal{X} \rightarrow \mathbb{R}$ represents the k^{th} objective function and \mathcal{X} is the set of feasible solutions. We will denote by $\mathcal{Z} = \{z(x) : x \in \mathcal{X}\} \subset \mathbb{R}^p$ the image of \mathcal{X} in the objective space.

Due to the typically conflictive objectives, the notion of optimal solution does not exist anymore for MO problems. However, based on the *dominance relation of Pareto* (see Definition 1), the notion of optimal solution can be replaced by the notion of *efficient* (or Pareto optimal) *solution* (see Definition 2).

Definition 1 A vector $z \in \mathcal{Z}$ dominates a vector $z' \in \mathcal{Z}$ if, and only if, $z_l \leq z'_l, \forall l \in \{1, \dots, p\}$, with at least one index l for which the inequality is strict. We denote this dominance relation by $z \prec z'$.

Definition 2 A feasible solution $x \in \mathcal{X}$ is efficient if there does not exist any other solution $x' \in \mathcal{X}$ such that $z(x') \prec z(x)$. The image of an efficient solution in the objective space is called a non-dominated point.

In the following, we will denote by \mathcal{X}_E the set of all efficient solutions (the efficient set) and by \mathcal{Z}_N the image of \mathcal{X}_E in the objective space (the Pareto front).

In the case of MOCO problems, we can distinguish two types of efficient solutions:

- The supported efficient solutions. These solutions are optimal solutions of weighted single-objective problems

$$\begin{aligned} \min \quad & \sum_{l=1}^p \lambda_l z_l(x) \\ \text{s.t.} \quad & x \in \mathcal{X} \end{aligned}$$

where $\lambda \in \mathbb{R}_+^p$ is a weight vector with all positive components $\lambda_l, \forall l = 1, \dots, p$. We denote by \mathcal{X}_{SE} and \mathcal{Z}_{SN} respectively the set of supported efficient solutions and the set of the corresponding non-dominated points in \mathbb{R}^p . The points of \mathcal{Z}_{SN} are located on the frontier of the convex hull of \mathcal{Z} .

- The non-supported efficient solutions. Contrary to a MO linear programming problem, \mathcal{Z}_{SN} is generally a proper subset of \mathcal{Z}_N due to the non-convex character of \mathcal{Z} , and there exist efficient solutions that are not optimal solutions for any weighted single-objective problems. These solutions are called non-supported efficient solutions. We denote by $\mathcal{X}_{NE} = \mathcal{X}_E \setminus \mathcal{X}_{SE}$ and $\mathcal{Z}_{NN} = \mathcal{Z}_N \setminus \mathcal{Z}_{SN}$ respectively the set of non-supported efficient solutions and the set of the corresponding non-dominated points in \mathbb{R}^p .

Even if other approaches exist to tackle a MO problem (aggregation of the objectives with a utility function, hierarchy of the objectives, goal programming, interactive method to generate a “good compromise”; see [Teghem 2009](#)), in this paper, we are only interested in the determination, or the approximation, of \mathcal{X}_E and \mathcal{Z}_N .

It is quite difficult to determine exactly the whole set of efficient solutions \mathcal{X}_E and the set of non-dominated points \mathcal{Z}_N for MOCO problems. This is a \mathcal{NP} -Hard problem even for CO problems for which a polynomial algorithm exists for the single-objective version, such as the linear assignment problem. Moreover, many MOCO problems have been proved to be intractable ([Ehrgott and Gandibleux 2002](#)).¹ Therefore, we can only expect to apply exact methods to determine the sets \mathcal{X}_E and \mathcal{Z}_N for small instances and for few number of objectives. For this reason, many methods are heuristic methods which produce approximations $\tilde{\mathcal{X}}_E$ and $\tilde{\mathcal{Z}}_N$ to the sets \mathcal{X}_E and \mathcal{Z}_N .

During the last 20 years, many heuristic methods for solving MOCO problems have been proposed. From the first survey [Ulungu and Teghem in 1994](#) till [Ehrgott and Gandibleux in 2002](#), a lot of papers have been published and this flow is still increasing. Due to the success of metaheuristics for single-objective CO ([Glover and Kochenberger 2003](#)), most of the heuristics developed are based on metaheuristics adapted to MO [called multiobjective metaheuristics (MOMHs)]. Presently, it is a real challenge for the researchers to improve the results previously obtained for some classic MOCO problems.

The two main difficulties related to the development of a MOMH are related to the basic needs of any metaheuristics [Glover and Kochenberger \(2003\)](#):

- To assure sufficient convergence, that is to assure to get an approximation $\tilde{\mathcal{Z}}_N$ as close as possible to \mathcal{Z}_N
- To assure sufficient diversity, that is to cover with $\tilde{\mathcal{Z}}_N$ all the parts of \mathcal{Z}_N

If evolutionary methods have intensively been applied to solve MOCO problems ([Coello Coello et al. 2002](#); [Deb 2001](#)), few results are known about the use of elaborate local search (LS) techniques, such as large neighborhood search (LNS) ([Pisinger and Ropke 2010](#); [Shaw 1998](#)) and variable neighborhood search (VNS) ([Hansen and Mladenovic 2001](#)). It is mainly because it is more natural to use a method based on a population, as we are looking for an approximation to a set. However, we show in this paper that by embedding these evolved LS techniques into the two-phase Pareto local search (2PPLS) ([Lust and Teghem 2010](#)), which uses as population the set of potentially efficient solutions, we can produce a very effective heuristic. As indicated by its

¹ A MOCO problem is intractable if the number of non-dominated points is exponential in the size of the instance.

name, the method is composed of two phases. In the first phase, an initial population of potentially efficient solutions is produced. In the second phase, the Pareto local search (PLS) (Angel et al. 2004; Paquete et al. 2004) is run from this population. PLS is a straightforward adaptation of LS to MO and only needs a neighborhood function $\mathcal{N}(x)$, which is applied to every new potentially efficient solution generated. At the end, a local optimum, defined in a MO context, is obtained (Paquete et al. 2004) (called a Pareto local optimum set). Therefore, to adapt 2PPLS to a MOCO problem, we only have to define an initial population and a neighborhood function.

Adaptations of 2PPLS were enabled to produce state-of-the-art results for the MOTSP (Lust and Teghem 2010) and the MOMKP (Lust and Teghem 2012). In both adaptations, only one neighborhood function is used and the method stops when a Pareto local optimum set is obtained. Strategies to escape from Pareto local optimal set have scarcely been studied in multiobjective optimization. We mention only one recent work of Drugan and Thierens (2012). In this work, they present two perturbation strategies to escape from Pareto local optimal set, based on mutation and path-guided mutation.

We present in this work a new strategy to escape from a Pareto local optimal set, based on the VNS technique: once a Pareto local optimum set has been found according to a neighborhood, we increase the size of the neighborhood in order to generate new potentially efficient solutions and to escape from the Pareto local optimum set.

The contributions of this paper are two folds. First, we present a metaheuristic method in order to solve MOCO problems, that includes at the same time LNS and VNS techniques (Sects. 3 and 4). Secondly, we demonstrate the efficiency of the method through the multiobjective set covering problem (MOSCP) (Sect. 2), for which we present a new heuristic (Sect. 5) and new state-of-the-art results (Sect. 6).

2 The multiobjective set covering problem

In the MOSCP, we have a set of m rows (or items), and each row can be covered by a subset of n columns (or sets). Each column j has p costs c_l^j ($l = 1, 2, \dots, p$). The MOSCP consists in determining a subset of columns, among the n columns ($j = 1, \dots, n$) such that all the rows are covered by at least one column and that the total costs are minimized.

More precisely, the MOSCP is defined as follows:

$$(\text{MOSCP}) \begin{cases} \text{“min” } z_l(x) = \sum_{j=1}^n c_l^j x_j & l = 1, 2, \dots, p \\ \text{s.t. } \sum_{j=1}^n t_{ij} x_j \geq 1 & i = 1, \dots, m \\ x_j \in \{0, 1\} & j = 1, \dots, n \end{cases}$$

with n the number of columns, m the number of rows, x the decision vector, formed of the binary variables x_j ($x_j = 1$ means that the column j is in the solution), c_l^j the

cost l of the column j and the binary data t_{ij} equal to 1 if the column j covers the row i and equal to 0 otherwise.

It is assumed that all coefficients c_l^j are nonnegative integer.

The data associated to the MOSCP are thus a cost matrix of size $(n \times p)$ and a covering matrix of size $(m \times n)$.

We denote by \mathcal{X} a feasible set in decision space, defined by $\mathcal{X} = \{x \in \{0, 1\}^n \mid \{\sum_{j=1}^n t_{ij}x_j \geq 1, \forall i = 1, \dots, m\}\}$. The corresponding feasible set in the objective space is called \mathcal{Z} and is defined by $\mathcal{Z} = z(\mathcal{X}) = \{(z_1(x), z_2(x), \dots, z_p(x)), \forall x \in \mathcal{X}\} \subset \mathbb{N}^2$.

As the single-objective version of the SCP is \mathcal{NP} -Hard, the MOSCP is \mathcal{NP} -Hard too. Therefore, our aim is to generate a good approximation of the efficient set. It is important to say that, with our method, if two solutions in the decision space give the same non-dominated point, only one solution is retained (only a good approximation of a minimal complete set Hansen 1979 is generated). The MOSCP did not receive as much attention as other classic MOCO problems, like the MO multidimensional knapsack problem (Lust and Teghem 2012) (MOMKP) or the MO traveling salesman problem (Lust and Teghem 2010) (MOTSP). To our knowledge, only two groups of authors have tackled this problem:

- Jaskiewicz was the first one, in 2003, with the adaptation of the Pareto memetic algorithm (PMA). In this method, two populations are managed: a population P and an elitist set, that is the approximation set $\tilde{\mathcal{X}}_E$ containing the potentially efficient solutions found so far. Two solutions from P are selected (the parents) and crossed to form a new solution (the offspring). The selection of the parents is based on a tournament. The two parents selected are the winners of a tournament between solutions coming from a sample of size T randomly drawn from the population P . Once the offspring has been generated, a LS is applied from it and if the new solution obtained is better than the second best solution in T (according to a scalarizing function), the offspring is added to P and to $\tilde{\mathcal{X}}_E$ (if the offspring is potentially efficient).
- In 2006, Prins and Prodhon (2006) have also tackled this problem, by using a two-phase heuristic method (called TPM) using primal-dual Lagrangian relaxations to solve different single-objective SCPs. In the first phase, potentially supported efficient solutions are generated by solving linear weighted sum problems.² In the second phase, they try to produce non-supported efficient solutions. With this aim, they use the information contained in the set of potentially supported efficient solutions to construct with a heuristic potentially non-supported efficient solutions. The procedure is successively applied between each pair of adjacent potentially supported non-dominated points.

The MOSCP has also been considered in two applications. The first is in public transport where the MOSCP is used to model the crew-scheduling problem (the crew is represented by the bus drivers) (Lourenço et al. 2001). Two objectives are considered:

² However, the authors claim that they generate all the supported efficient solutions, even if they use a heuristic to solve the linear weighted sum problems.

the cost and the quality of the service. The authors have developed a hybrid method, based on an own adaptation of GRASP, tabu search and genetic algorithms to solve this problem. At the end, a set of potentially efficient solutions is proposed to the decision makers (which may be several since different companies of bus are involved in the decision process).

The second application is in group technology where the MOSCP is used to model the cell formation problems (Hertz et al. 1994). Different objectives are considered: the number of bottleneck operations, the number of machines or bottleneck parts, the intercellular flow, the intercellular load balancing, the subcontracting costs, etc. The authors propose for this problem a multiobjective adaptation of tabu search.

3 Two-phase Pareto local search

The new heuristic, in order to solve the MOSCP, is based on the two-phase Pareto local search (2PPLS), developed by Lust and Teghem (2010), mainly composed of two phases: generation of an initial population and application of PLS (Angel et al. 2004; Paquete et al. 2004) from this population. We have however integrated the VNS technique in order to escape from a Pareto local optimum set.

The pseudo-code of 2PPLS with VNS is given by the Algorithm 1.

The method needs four entries: an initial population P_0 , the sizes of the different neighborhood structures to order them from the smallest (k_{\min}) to the largest (k_{\max}) one, and the different neighborhood functions $\mathcal{N}_k(x)$ for each $k \in \mathbb{Z} : k_{\min} \leq k \leq k_{\max}$.

The method starts with the population P composed of potentially efficient solutions given by the initial population P_0 . The neighborhood structure initially used is the smallest ($k = k_{\min}$). To each solution $x \in \tilde{\mathcal{X}}_E$, we also associate the value $k(x)$. This function gives for a solution x the maximal size of the neighborhood that has been explored from the solution (for example, if $k(x) = 3$, that means that the neighborhoods of size $k = 1, 2$ and 3 have been explored from the solution). This function will help us to avoid to explore the neighborhood of a solution if it has already been explored before.

Then, PLS is started and considering $\mathcal{N}_k(x)$, all the neighbors p' of each solution p of P are generated. If a neighbor p' is not weakly dominated by the current solution p (that is $z(p) \not\prec z(p')$ and $z(p) \neq z(p')$), we try to add the solution p' to the approximation $\tilde{\mathcal{X}}_E$ of the efficient set, which is updated with the procedure `AddSolution`. This procedure is not described in this paper but simply consists in updating an approximation $\tilde{\mathcal{X}}_E$ of the efficient set when a new solution p' is added to $\tilde{\mathcal{X}}_E$. This procedure has five parameters: the set $\tilde{\mathcal{X}}_E$ to actualize, the new solution p' , its evaluation $z(p')$, an optional boolean variable called *Added* that returns *True* if the new solution has been added and *False* otherwise, and another optional variable k . This last variable allows to update the value of the function $k(p')$ associated to the solution p' . Then, if the solution p' has been added to $\tilde{\mathcal{X}}_E$, *Added* is true and the solution p' is added to an auxiliary population P_a , which is also updated with the procedure `AddSolution`. Therefore, P_a is only composed of new potentially efficient solutions. Once all the neighbors from \mathcal{N}_k of each solution of P have been generated, the algorithm starts again, with P equal to P_a , until $P = P_a = \emptyset$. The auxiliary population P_a is used

Algorithm 1 2PPLS with VNS

Parameters \downarrow : an initial population P_0 , neighborhood functions $\mathcal{N}_k(x)$, k_{\min} , k_{\max} .
Parameters \uparrow : an approximation $\tilde{\mathcal{X}}_E$ of the efficient set \mathcal{X}_E .

-| Initialization of $\tilde{\mathcal{X}}_E$ and a population P with the initial population P_0
 $\tilde{\mathcal{X}}_E \leftarrow P_0$
 $P \leftarrow P_0$

-| Initialization of an auxiliary population P_a
 $P_a \leftarrow \emptyset$

-| Initialization of the neighborhood structure
 $k \leftarrow k_{\min}$

-| Initialization of $k(x)$: the maximal size of the neighborhood that has been explored from a solution x
for all $x \in \tilde{\mathcal{X}}_E$ **do**
 $k(x) = k_{\min}$
-| PLS with VNS

repeat
while $P \neq \emptyset$ **do**
-| Generation of all neighbors p' of each solution $p \in P$
for all $p \in P$ **do**
for all $p' \in \mathcal{N}_k(p)$ **do**
if $z(p) \not\leq z(p')$ **then**
AddSolution($\tilde{\mathcal{X}}_E \uparrow, p' \downarrow, z(p') \downarrow, Added \uparrow, k \uparrow$)
if $Added = true$ **then**
AddSolution($P_a \uparrow, p' \downarrow, z(p') \downarrow$)
if $P_a \neq \emptyset$ **then**
-| P is composed of the new potentially efficient solutions
 $P \leftarrow P_a$
-| Reinitialization of P_a
 $P_a \leftarrow \emptyset$
-| We start again with the smallest neighborhood structure
 $k \leftarrow k_{\min}$
else
-| We use a larger neighborhood structure
 $k \leftarrow k + 1$
-| We use as population the solutions of $\tilde{\mathcal{X}}_E$ that are not already Pareto local optimum for $\mathcal{N}_k(x)$
 $P \leftarrow \{x \in \tilde{\mathcal{X}}_E \mid k(x) < k\}$
until $k > k_{\max}$

such that the neighborhood of each solution of P is explored, even if some solutions of P become dominated following the addition of a new solution to P_a . Thus, sometimes, neighbors are generated from a dominated solution.

In the case of $P = P_a = \emptyset$, the set $\tilde{\mathcal{X}}_E$ obtained is a Pareto local optimum set according to \mathcal{N}_k , and cannot be improved with \mathcal{N}_k . We thus increase the size of the neighborhood ($k \leftarrow k + 1$), and apply again PLS with this larger neighborhood.

Please note that, in general, a solution Pareto local optimum for the neighborhood k is not necessary Pareto local optimum for the neighborhood $(k - 1)$. That is why, after considering a larger neighborhood, we always restart the search with the smallest neighborhood structure.

After the generation of a Pareto local optimum set according to \mathcal{N}_k , the population P used with the neighborhood \mathcal{N}_{k+1} is $\tilde{\mathcal{X}}_E$, without considering the solutions of $\tilde{\mathcal{X}}_E$

that could already be Pareto local optimal for \mathcal{N}_{k+1} , that is the solutions of $\tilde{\mathcal{X}}_E$ such that $k(x) < k$.

The method stops when a Pareto local optimum set has been found, according to all the neighborhood structures considered.

4 Large neighborhood search

With LS (and therefore PLS), the larger the neighborhood, the better the quality of the local optimum obtained is. However, by increasing the size of the neighborhood, the time to explore the neighborhood becomes higher. Therefore, using a larger neighborhood does not necessary give rise to a more effective method. If we want to keep reasonable running times while using a large neighborhood, an efficient strategy has to be implemented in order to explore the neighborhood.

As neighborhood in PLS, we will use a large neighborhood search (LNS) (Pisinger and Ropke 2010; Shaw 1998). LNS was introduced by Shaw in 1998 to solve the vehicle routing problem. With LNS, the neighborhood consists of two procedures: a destroy method and a repair method. The destroy method destructs some parts of the current solution while the repair method rebuilds the destroyed solution. The LNS belongs to the class of neighborhood search known as very large scale neighborhood search (VLSNS) (Ahuja et al. 2002). These two neighborhood often cause confusion. Contrary to LNS, in VLSNS, the neighborhood is usually restricted to a neighborhood that can be searched efficiently. Ahuja et al. (2002) defines three methods used in VLSNS to explore the neighborhood: variable depth methods, network flow based improvement methods and methods based on restriction to subclasses solvable in polynomial time. Therefore, VLSNS is not only a large neighborhood (furthermore the definition of large is imprecise), but essentially a neighborhood that uses a heuristic to explore efficiently a large neighborhood.

In this paper, the size of the neighborhood will be kept relatively small, and a VLSNS will not be needed. However, we will keep the destroy and repair techniques of the LNS.

VLSNS and LNS are very popular in single-objective optimization (Ahuja et al. 2002; Pisinger and Ropke 2010). For example, the Lin and Kernighan (1973) heuristic, one of the best heuristics for solving the single-objective traveling salesman problem (TSP), is based on VLSNS. On the other hand, there is almost no study of LNS/VLSNS for solving MOCO problems. The only known result is the LS of Angel et al. (2004), which integrates a dynasearch neighborhood (the neighborhood is solved with dynamic programming) to solve the biobjective TSP.

Starting from a current solution, called x^c , the aim of LNS is to produce a set of neighbors of high quality, in a reasonable time. The general technique that we use for solving the MOSCP is the following:

1. Destroy method: Identification of a set of variables candidates to be removed from x^c (set \mathcal{O})
2. Repair method:
 - Identification of a set of variables, not in x^c , candidates to be added (set \mathcal{I})

- Creation of a residual multiobjective problem formed by the variables belonging to $\{\mathcal{O} \cup \mathcal{I}\}$, and the constraints not anymore fulfilled (that is the items not anymore covered, in the case of the MOSCP).
- Resolution of the residual problem: a set of potentially efficient solutions of this problem is produced. The potentially efficient solutions of the residual problem are then merged with the unmodified variables of x^c to produce the neighbors.

This scheme will be presented with more details in Sect. 5.2.

5 Adaptation of 2PPLS to the MOSCP

In this section, we present how 2PPLS has been adapted to the MOSCP. We first present how the initial population has been generated. We expose then the LNS considered in PLS, method used as second phase in 2PPLS, that only needs a neighborhood (see Sect. 3).

5.1 Initial population

The initial population is composed of a good approximation of the supported efficient solutions. These solutions can be generated by solving single-objective problems obtained by applying a weighted sum of the objectives. To generate the different weight sets, we have used the dichotomic method of [Aneja and Nair \(1979\)](#). This method consists in generating all the weight sets necessary to get a minimal complete set of extreme supported efficient solutions of a biobjective problem ([Przybylski et al. 2008](#)) (extreme supported efficient solutions are the supported efficient solutions that are located on the vertex set of the convex hull of \mathcal{Z}). The weight sets are based on the computation of normal vectors to two consecutive supported non-dominated points.

For solving the single-objective SCP obtained, we have considered two different methods:

- The free mixed integer linear programming (MILP) solver `lp_solve`.
- A metaheuristic, called `Meta-Raps` (for metaheuristic for randomized priority search), developed by [Moraga \(2002\)](#). This metaheuristic has been adapted to the SCP by [Lan et al. \(2007\)](#). This has resulted in an efficient heuristic, whose the code has been published.

With the exact `lp_solve` solver, an exact minimal set of the extreme supported efficient solutions can be generated. However, for the biggest instances of the MOSCP, we will see that using this method can be time-consuming. That is why the heuristic method based on the `Meta-Raps` metaheuristic has also been considered.

5.2 Large neighborhood search

We present now how we have adapted the LNS to the MOSCP. We have followed the general scheme presented at the end of Sect. 4. Before giving more details about the LNS, we point out two particularities related to the MOSCP:

1. When we remove columns from a solution, some rows are not anymore covered. The set \mathcal{I} must thus be composed of columns that can cover these rows.
2. We have noted that removing a small number k of columns (less than 4, as it will be shown in Fig. 1) from x^c was enough. Indeed, when we work with a larger set \mathcal{O} , it becomes more difficult to define the set \mathcal{I} since many rows become uncovered. Moreover, the LNS always starts from a solution of good quality, and only some small modifications of these solutions are needed to produce new potentially efficient solutions. Therefore, as we will keep k small, for each value of k , we will create more than one residual problem. If $k = 1$, the number of residual problems will be equal to the number of columns present in the current solution x . For each residual problem, the set \mathcal{O} will be thus composed of one of the columns present in x^c . If $k > 1$, we create a set \mathcal{L} of size $L (L \geq k)$, that includes the columns that are candidates to be removed from x^c (these columns are identified by the ratio R_1^s defined in the following). Then, all the combinations of k columns from \mathcal{L} will be considered to form the set \mathcal{O} . The number of residual problems will be thus equal to the number of combinations of k elements into a set of size L , that is C_k^L . The size of the set \mathcal{I} will also be limited to L .

More precisely, the LNS works as follows:

Destroy method:

If $k = 1$, the set \mathcal{O} is composed of one of the columns present in x^c .

If $k > 1$, the set \mathcal{O} will be composed of k columns chosen among the list \mathcal{L} (all combinations are tested) containing the L worst columns (present in x^c) for the ratio R_1^s , defined by

$$R_1^s = \frac{\sum_{l=1}^p \lambda_l c_l^s}{\sum_{i=1}^m t_{is}}$$

for a column s . This ratio is equal to the weighted aggregation of the costs on the number of rows that the column s covers (the smaller the better). The weight set λ is randomly generated.

Repair method:

1. The set \mathcal{I} of columns candidates to be added will be composed of the L best columns (not in x^c) for the ratio R_2^s , defined by

$$R_2^s = \frac{\sum_{l=1}^p \lambda_l c_l^s}{n_s}$$

for a column s , where n_s represent the number of rows that the column s covers among the rows that are not anymore covered when the k columns have been

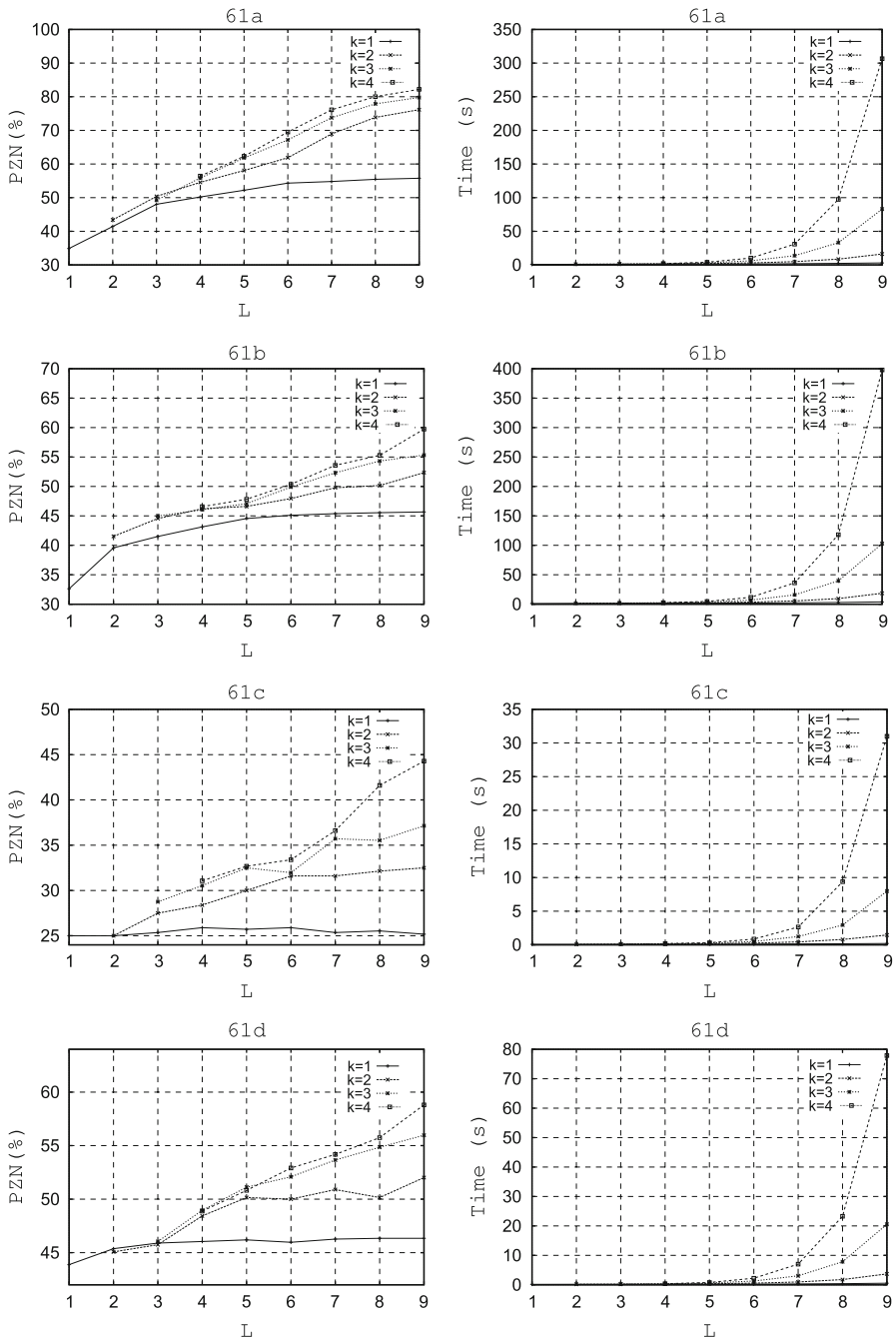


Fig. 1 Evolution of the P_{ZN} indicator and the running time according to L and k for the 61a, 61b, 61c and 61d instances

removed. This ratio is only computed for the columns that cover at least one of the rows not anymore covered ($n_s > 0$). The weight set λ is randomly generated.

2. A residual problem is defined, of size $k + L$, composed of the columns belonging to the set $\{\mathcal{O} \cup \mathcal{I}\}$ and of the items not anymore covered.
3. To solve the residual problem, we simply generate all the efficient solutions of the problem, with an enumeration algorithm. We then merge the efficient solutions of the residual problem with the unmodified variables of x^c , to obtain the neighbors.

6 Results

We first present results for biobjective instances.

6.1 Data and reference sets

We use the same instances as [Jaszkiewicz \(2003\)](#) and [Prins and Prodhon \(2006\)](#) considered, from the size 100×10 (100 columns, 10 rows) to the size 1000×200 (1000 columns, 200 rows). Those instances have however been generated by [Gandibleux et al. \(1998\)](#). For each size instance, four different kinds of objectives A, B, C and D are defined. In the case of instances of type A, the costs of each objective are randomly generated. For the type B, the costs of the first objective are randomly generated and the ones of the second objective are made dependent in the following way:

$$c_2^j = c_1^{n-j+1} \quad \forall j = 1, \dots, n$$

For the type C, a subdivision of the columns into several subsets is done for each objective (for each objective, the subdivision is different). Then, all the columns from one subset have the same cost for the objective corresponding to the subdivision. Finally, the instances of type D combine characteristics of the sets B and C.

6.2 Measuring the quality of the approximations

Before describing the results obtained, it is necessary to define how we have measured the quality of the approximations generated. In MO, measuring the quality of an approximation or comparing the approximations obtained by various methods remains a difficult task: the problem of the quality assessment of the results of a heuristic is in fact also a multicriteria problem. Consequently, several indicators have been introduced in the literature to measure the quality of an approximation (see [Zitzler et al. 2002](#) for instance).

We have used the following unary indicators:

- The hypervolume \mathcal{H} (to be maximized) ([Zitzler 1999](#)): the volume of the dominated space defined by $\tilde{\mathcal{Z}}_N$, limited by a reference point.
- The T measure (normalized between 0 and 1, to be maximized) ([Jaszkiewicz 2002](#)): evaluation of $\tilde{\mathcal{Z}}_N$ by the expected value of the weighted Tchebycheff utility function over a set of normalized weight vectors, computed as follows:

$$T(\tilde{Z}_N, z^0, \Lambda) = \frac{1}{|\Lambda|} \sum_{\lambda \in \Lambda} \min_{z \in \tilde{Z}_N} \|z - z^0\|_\lambda$$

where z^0 is a reference point and Λ is a set of normalized weight sets ($\sum_{k=1}^p \lambda_k = 1, \lambda_k \geq 0$).

- The average distance D_1 and maximal distance D_2 (to be minimized) (Czyzak and Jaskiewicz 1998; Ulungu et al. 1999) between the points of a reference set Z_R and the points of \tilde{Z}_N , by using the Euclidean distance. If we consider the Euclidean distance $d(z^1, z^2)$ between two points z^1 and z^2 in the objective space, we can define the distance $d'(\tilde{Z}_N, z^1)$ between a point $z^1 \in Z_R$ and the points $z^2 \in \tilde{Z}_N$ as follows:

$$d'(\tilde{Z}_N, z^1) = \min_{z^2 \in \tilde{Z}_N} d(z^2, z^1)$$

This distance is thus equal to the minimal distance between a point z^1 of Z_R and the points of the approximation \tilde{Z}_N . The average distance D_1 and maximal distance D_2 are then computed as follows:

$$D_1(\tilde{Z}_N, Z_R) = \frac{1}{|Z_R|} \sum_{z^1 \in Z_R} d'(\tilde{Z}_N, z^1)$$

$$D_2(\tilde{Z}_N, Z_R) = \max_{z^1 \in Z_R} d'(\tilde{Z}_N, z^1)$$

- The ϵ factor $I_{\epsilon 1}$ (to be minimized) by which the approximation \tilde{Z}_N is worse than a reference set Z_R with respect to all the objectives:

$$I_{\epsilon 1}(\tilde{Z}_N, Z_R) = \inf_{\epsilon \in \mathbb{R}^+} \{ \forall z \in Z_R, \exists z' \in \tilde{Z}_N : z'_k \leq \epsilon \cdot z_k, k = 1, \dots, p \}$$

- The proportion P_{Z_N} (to be maximized) of non-dominated points generated.

Unfortunately, none of these indicators allows to conclude that an approximation is better than another one (see Zitzler et al. 2003 for details). Nevertheless, an approximation that finds better values for these indicators is generally preferred to the others.

The reference set used is the Pareto front, that we have generated with an exact method based on the ϵ -constraint method (Laumanns et al. 2004). The reference point used in the hypervolume is the Nadir point, multiplied by 1.1 for both objectives, in order to avoid that a point of an approximation does not dominate the reference point of the hypervolume. The reference point necessary to the computation of the weighted Tchebycheff utility function for the indicator T is equal to the ideal point and the number of weight set used is equal to 100.

However, we were not able to generate the Pareto front for some of the biggest and most difficult instances tackled (the 82c, 82d, 101c, 101d, 102c, 102d, 201a and 201b instances). Therefore, for these instances, we have considered outperformance relations (Hansen and Jaskiewicz 1998).

With outperformance relations, a pairwise comparison is realized between the points of two approximations A and B . Four cases can occur: a point of A is dominated by at least one point of B , a point of A dominates at least one point of B , a point of A is equal to another point of B , or the result of the comparison belongs to none of these three

possibilities (the point of A is incomparable to all the points of B). For each of the four cases, the percentage of points of the approximation A that check the case is computed. As our algorithm will be run twenty times, the outperformance relations will be represented with box-plot graphs, that allow to represent the variations in the different values (a pairwise comparison is made between the runs of two distinct algorithms). In the tables, the best values achieved by the different indicators are indicated in bold.

The computer used for the experiments is a Pentium IV with 3 GHz CPUs and 512 MB of RAM.

6.3 Phase 1

In this section, we study the results obtained after the first phase of 2PPLS. We recall that the aim of the first phase is to generate a good approximation of the supported efficient solutions.

In Table 1, we compare the results obtained with `lp_solve` or `Meta-Raps` as solver for optimizing the different weighted sum single-objective problems. For `Meta-Raps`, we have used the default parameters defined by Lan et al. (2007). The quality indicators considered are the proportion of supported non-dominated points generated [$P_{Z_{SN}}(\%)$] and the proportion of non-dominated points generated [$P_{Z_N}(\%)$]. We also indicate the number of potentially efficient solutions generated ($|PE|$) and the running time in seconds. The results are exposed for the instances for which we have the Pareto front. Please note that for the method which uses the heuristic `Meta-Raps`, the solutions obtained after solving the weighted sum single-objective problems are not necessary supported efficient solutions. However they can be non-supported efficient solutions.

The results of `Meta-Raps` are rather disparate: we can generate from 18.33 % of Z_{SN} (instance 81c) to 95 % of Z_{SN} (instance 62a). The running times are however small and always less than 35 s.

With `lp_solve`, we can of course generate 100 % of Z_{SN} for most of the instances (except for the instances 61d and 62d, which means that for these instances, non-extreme supported efficient solutions exist). But the running time can be very high: for example, for the instance 62d, `lp_solve` needs more than 1,000 s for generating 91 % of Z_{SN} , while `Meta-Raps` only needs 5 s to generate 82 % of Z_{SN} . We see thus through this example that it is worthwhile to use a heuristic instead of an exact method in the first phase. However, for certain instances, `lp_solve` enables to find better results in less running times than `Meta-Raps`. Therefore, in these cases, it is preferable to use `lp_solve`.

We have adopted the following rule to define if `lp_solve` or `Meta-Raps` has to be used in the first phase:

- If the instance has less than 600 columns and 60 rows, `lp_solve` is used.
- If the instance has not more than 800 columns and 80 rows, and it is of type A or B, `lp_solve` is used.
- Otherwise, `Meta-Raps` is used.

In Table 1, the solver used in the sequel is indicated in italic. For the other instances (the ones for which we do not have the Pareto front), `Meta-Raps` is used.

Table 1 Comparison between the two solvers used in phase 1 (*lp_solve* and *Meta-Raps*)

Instance	Algorithm	$P_{Z_{SN}}(\%)$	$P_{Z_N}(\%)$	$ PE $	Time(s)
61a	<i>lp_solve</i>	100.00	12.45	32.00	8.66
	<i>Meta-Raps</i>	49.53	9.14	29.50	18.73
61b	<i>lp_solve</i>	100.00	10.65	36.00	12.32
	<i>Meta-Raps</i>	24.72	5.58	38.00	21.45
61c	<i>lp_solve</i>	100.00	25.00	7.00	7.87
	<i>Meta-Raps</i>	79.29	21.96	6.95	5.05
61d	<i>lp_solve</i>	70.37	28.36	19.00	234.36
	<i>Meta-Raps</i>	36.30	21.64	21.30	15.13
62a	<i>lp_solve</i>	100.00	13.27	13.00	21.38
	<i>Meta-Raps</i>	95.00	13.42	13.20	7.04
62b	<i>lp_solve</i>	100.00	20.20	20.00	23.72
	<i>Meta-Raps</i>	86.25	18.13	17.95	9.47
62c	<i>lp_solve</i>	100.00	33.33	2.00	1470.11
	<i>Meta-Raps</i>	67.50	23.33	2.05	0.92
62d	<i>lp_solve</i>	90.91	26.32	10.00	1088.55
	<i>Meta-Raps</i>	82.27	24.61	9.75	5.32
81a	<i>lp_solve</i>	100.00	8.96	38.00	7.13
	<i>Meta-Raps</i>	56.45	6.82	34.40	32.36
81b	<i>lp_solve</i>	100.00	10.17	36.00	9.22
	<i>Meta-Raps</i>	41.53	7.05	34.85	32.07
81c	<i>lp_solve</i>	100.00	64.29	9.00	18.34
	<i>Meta-Raps</i>	18.33	17.14	7.25	7.14
81d	<i>lp_solve</i>	100.00	75.00	9.00	39.50
	<i>Meta-Raps</i>	30.56	34.17	7.10	6.87
82a	<i>lp_solve</i>	100.00	17.42	23.00	61.83
	<i>Meta-Raps</i>	83.26	15.91	21.00	19.62
82b	<i>lp_solve</i>	100.00	20.45	18.00	8.14
	<i>Meta-Raps</i>	83.33	18.58	16.35	13.41
101a	<i>lp_solve</i>	100.00	15.29	24.00	241.71
	<i>Meta-Raps</i>	83.33	15.32	24.55	29.37
101b	<i>lp_solve</i>	100.00	15.60	22.00	159.27
	<i>Meta-Raps</i>	75.45	15.32	22.20	30.87
102a	<i>lp_solve</i>	100.00	25.30	21.00	76.37
	<i>Meta-Raps</i>	83.33	22.29	19.25	33.04
102b	<i>lp_solve</i>	100.00	22.09	19.00	132.56
	<i>Meta-Raps</i>	85.79	18.95	16.30	29.06

We also see in this table, through the P_{Z_N} indicator, that the supported efficient solutions represent only a small part of the efficient set (at most 33 %, except for the 81c and 81d instances).

6.4 Phase 2

We study here the influence of the size of the neighborhood k and the length L of the lists \mathcal{L} and \mathcal{I} .

In Fig. 1, we show the evolution of $P_{\mathcal{Z}_N}$ and the running time according to L for the 61a, 61b, 61c and 61d instances, for different values of k . We vary the values of L from 1 to 9 and k from 1 to 4. We see that, of course, increasing the values of L or k allows to obtain better quality results. The best improvement is when k is moved from 1 to 2 for values of L superior to 5. On the other hand, using $k = 3$ or $k = 4$ instead of $k = 2$ does not give impressive improvements, whatever the value of L . We have obtained similar results for the other instances.

Concerning the running time, as we use an exact enumeration algorithm to solve the residual problems, we see that its evolution is exponential. Using $k = 3$ or $k = 4$ with $L \geq 8$ becomes very time-consuming.

Therefore, for the comparison to state-of-the-art results, we will use $k = 2$ and $L = 9$.

6.5 Comparison to state-of-the-art results

We first compare the results obtained with 2PPLS to the results of TPM and PMA, for the small instances (less than 600 columns and 60 rows). These instances are not very interesting to experiment with a heuristic since we were able to generate the efficient sets in few seconds with an exact method based on the ϵ -constraint method (Laumanns et al. 2004). However, Jaskiewicz and Prins et al. have considered these instances and it remains interesting to attest the quality of our heuristic on these small instances. We indicate in Table 2 the mean values obtained for one indicator, the proportion $P_{\mathcal{Z}_N}$ of non-dominated points generated. We see that for each group of instances considered (in a group of instances, the four types A, B, C and D are considered, the indicators are averaged over a group), we obtain more than 50 % of \mathcal{Z}_N . This proportion is largely better than TPM and PMA.

In Tables 3 and 4, we compare the results obtained with 2PPLS to the results of TPM and PMA for bigger instances. We use the quality indicators described in Sect. 6.2, that is the hypervolume \mathcal{H} , the ϵ factor $I_{\epsilon 1}$, the T indicator, the average distance D_1 , the maximal distance D_2 and the proportion $P_{\mathcal{Z}_N}$ of non-dominated points generated. We also indicate the number of potentially efficient solutions generated ($|PE|$) and the running time in seconds (separated in two parts: first phase and second phase). We do not have the running times of TPM and PMA for each instance, but we will compare

Table 2 Comparison of $P_{\mathcal{Z}_N}$ (%) of 2PPLS, TPM and PMA for small instances

Groups of instances	Dimension	2PPLS	TPM	PMA
11	10 × 100	77.30	36.42	52.85
41	40 × 200	63.17	27.50	10.77
42	40 × 400	57.93	20.62	2.47
43	40 × 200	87.12	61.64	41.69

Table 3 Comparison between 2PPLS, TPM and PMA based on the indicators (1)

Instance	Algorithm	$\mathcal{H}(10^5)$	I_{ϵ_1}	T	D_1	D_2	$P_{Z_N}(\%)$	$ PE $	Time(s)
61a	2PPLS	650.4639	1.013633	0.929305	0.090	2.287	76.15	221.10	8.66 + 16.28
	TPM	646.9003	1.067160	0.927248	1.349	5.949	15.56	55.00	–
	PMA	639.0582	1.062786	0.925659	1.233	7.015	2.61	110.10	–
61b	2PPLS	806.8344	1.011334	0.918550	0.148	2.510	52.35	267.40	12.32 + 18.59
	TPM	801.0863	1.063717	0.916390	1.224	6.449	13.31	64.00	–
	PMA	784.6685	1.085539	0.911936	1.798	4.815	0.09	136.20	–
61c	2PPLS	76.9568	1.046435	0.879243	1.895	7.465	30.36	15.60	5.05 + 1.37
	TPM	72.8138	1.161202	0.863144	7.073	17.823	21.43	7.00	–
	PMA	66.1876	1.139919	0.840228	5.598	14.767	0.00	9.60	–
61d	2PPLS	585.2976	1.057449	0.931428	1.061	6.246	34.63	50.75	15.13 + 4.23
	TPM	581.5946	1.118967	0.930271	2.403	10.061	28.36	24.00	–
	PMA	549.4480	1.191023	0.912641	4.819	15.118	0.00	46.20	–
62a	2PPLS	139.4577	1.054538	0.925718	0.308	2.746	57.35	75.80	21.38 + 6.44
	TPM	136.7701	1.245484	0.921079	2.448	9.112	16.33	20.00	–
	PMA	138.5127	1.111875	0.924904	0.583	3.652	34.80	70.10	–
62b	2PPLS	196.6011	1.042553	0.936334	0.187	2.085	64.49	79.35	23.72 + 7.29
	TPM	195.4355	1.113794	0.933906	1.171	5.363	24.24	31.00	–
	PMA	195.5629	1.096406	0.935249	0.671	4.317	27.58	63.80	–
62c	2PPLS	1.7594	1.034191	0.778159	2.859	6.905	44.17	3.40	0.92 + 0.30
	TPM	1.7849	1.022750	0.780545	2.866	6.886	33.33	3.00	–
	PMA	.1029	1.195617	0.497462	27.260	40.446	0.00	1.30	–
62d	2PPLS	7.6825	1.080195	0.887484	0.652	3.292	46.71	25.30	5.32 + 1.76
	TPM	7.6778	1.096886	0.887863	1.056	3.717	34.21	14.00	–
	PMA	7.2203	1.203339	0.877871	1.248	3.713	16.58	26.00	–
81a	2PPLS	1804.7490	1.015879	0.918999	0.094	1.164	58.87	351.65	7.13 + 24.66
	TPM	1787.6098	1.103112	0.915902	1.957	9.729	10.61	64.00	–
	PMA	1777.7066	1.064281	0.915603	1.253	3.792	0.61	173.80	–
81b	2PPLS	2295.7315	1.021002	0.919304	0.141	1.176	56.20	272.25	9.22 + 21.75
	TPM	2277.4616	1.090868	0.916750	1.690	6.398	11.30	55.00	–
	PMA	2267.9781	1.066492	0.916470	1.197	6.756	3.64	138.10	–
81c	2PPLS	113.9085	1.074723	0.983918	2.998	12.301	21.79	12.00	7.14 + 1.49
	TPM	114.0509	1.037037	0.984317	3.596	15.592	35.71	9.00	–
	PMA	110.0607	1.251852	0.975104	21.113	61.595	0.00	1.50	–
81d	2PPLS	168.5435	1.030773	0.983111	4.098	16.119	42.50	10.25	6.87 + 1.14
	TPM	169.3999	1.014968	0.983678	2.827	23.631	75.00	10.00	–
	PMA	160.0092	1.160129	0.964133	10.640	42.617	0.00	6.60	–

later in this section, the average running times obtained by each algorithm, for each group of instances.

We see that we obtain better results for the indicators considered, except for the instances 62c and 62d for which TPM obtains slightly better values for some indicators,

Table 4 Comparison between 2PPLS, TPM and PMA based on the indicators (2)

Instance	Algorithm	$\mathcal{H}(10^5)$	$I_{\epsilon 1}$	T	D_1	D_2	$P_{Z_N}(\%)$	$ PE $	Time(s)
82a	2PPLS	433.5862	1.052256	0.937949	0.201	2.970	62.84	108.15	19.62 + 11.06
	TPM	427.6693	1.279070	0.932029	2.190	7.907	17.42	27.00	–
	PMA	431.1933	1.099454	0.936085	0.774	4.617	20.45	87.30	–
82b	2PPLS	233.6640	1.049694	0.952172	0.191	5.143	72.16	73.50	8.14 + 7.92
	TPM	232.8426	1.151741	0.950443	1.179	6.040	23.86	28.00	–
	PMA	232.8318	1.130456	0.949976	0.612	6.534	35.91	58.60	–
101a	2PPLS	1287.8008	1.051087	0.954936	0.494	8.865	45.83	107.80	29.37 + 14.51
	TPM	1284.6645	1.119284	0.953671	1.340	13.648	19.11	44.00	–
	PMA	1282.0624	1.123602	0.953413	1.020	15.550	10.00	90.50	–
101b	2PPLS	944.1663	1.042192	0.949094	0.444	6.274	48.16	103.75	30.87 + 13.01
	TPM	937.8272	1.251705	0.945765	2.324	10.751	19.15	32.00	–
	PMA	939.9055	1.104710	0.947423	1.264	17.493	14.96	91.00	–
102a	2PPLS	305.8453	1.090276	0.956545	0.395	3.450	56.51	60.75	33.04 + 6.59
	TPM	286.7343	2.256186	0.941059	9.920	24.121	0.00	6.00	–
	PMA	305.1965	1.144873	0.955852	0.675	6.340	33.01	55.80	–
102b	2PPLS	340.0914	1.074174	0.948418	0.419	6.688	64.77	66.70	29.06 + 8.80
	TPM	329.3906	2.012500	0.936739	3.032	26.430	0.00	31.00	–
	PMA	338.8972	1.146556	0.946918	0.644	3.737	28.95	61.50	–

and for the instances 81c and 81d for which TPM obtains better values for most of the indicators.

The proportion $P_{Z_N}(\%)$ of non-dominated points generated is included between 21 and 92 % and the running time is always less than 45 s.

For the other instances (the 82c, 82d, 101c, 101d, 102c, 102d, 201a and 201b instances), we have used the outperformance relations to compare the results of 2PPLS with PMA (and not with TPM since we did not get all the results of TPM for these instances). The results are given in Fig. 2. We see that the proportion of solutions of 2PPLS that are dominated by PMA is very weak. On the other hand, the proportion of solutions of 2PPLS that dominate PMA is superior to 50 %.

Finally, the comparison of the average running times for solving the different groups of instances is given in Table 5. We see that the running times of 2PPLS are less or equal than the running times of TPM and PMA (TPM has been executed on a Pentium IV with 1.8 GHz CPUs and 512 MB of RAM, and PMA on a computer with 750 MHz). Those computers are slower than the one that we used (Pentium IV with 3 GHz CPUs), but we have obtained less computation times for most of the instances, and our results are considerably better on the different indicators considered.

6.6 Three-objective instances

We present in this section results for three-objective instances. We use different instances of each type (A, B, C and D, see Sect. 6.1). The costs have been generated

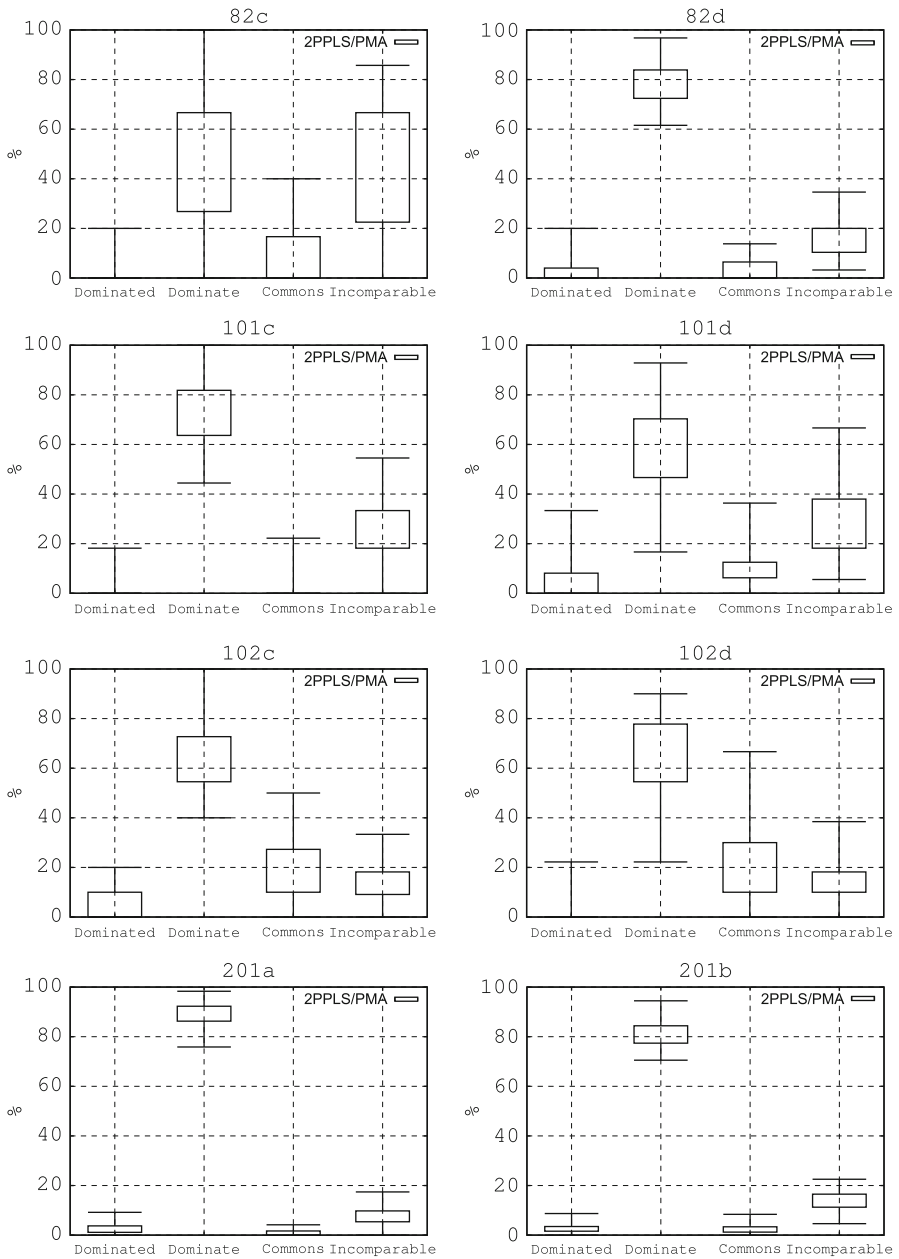


Fig. 2 Outperformance relations

following the scheme presented in Sect. 6.1. For the type B, the costs of the third objective is randomly generated. We do not have state-of-the-art results for these instances, therefore we will be the first to present results for the three-objective set covering problem.

Table 5 Comparison of the running times (in seconds) of 2PPLS, TPM and PMA

Groups of instances	Dimension	2PPLS	TPM	PMA
11	10 × 100	0.56	2.05	4.2
41	40 × 200	7.62	7.69	20.3
42	40 × 400	8.56	8.56	45.7
43	40 × 200	2.34	7.34	14.8
61	60 × 600	20.41	20.27	132.4
62	60 × 600	16.78	20.35	98.8
81	80 × 800	19.85	30.22	165.9
82	80 × 800	16.44	30.27	148.4
101	100 × 1000	26.48	50.10	311.7
102	100 × 1000	23.75	50.41	282.1
201	200 × 1000	60.36	70.81	686.8

We have first tried to solve an instance with 60 rows and 600 columns of type A. For the neighborhood, we have used the same parameters than the instances with two objectives, that is $k = 2$ and $L = 9$. For the initial population, we generate randomly 1000 weighted-sum problems, that are solved with the exact method based on `lp_solve`. We have obtained the following result: the method needs about 30 min, to generate 11039 potentially efficient solutions, while for the instance of the same size, with two objectives, only 25 s were needed, to generate 221 solutions (on average). We see that increasing the number of objectives considerably increases the number of potentially efficient solutions to generate, and therefore the computational time.

One way to limit the computational time is to limit the number of potentially efficient solutions generated. In order to do so, we have proceeded in the following way: we maintain a hypergrid in the objective space, dynamically updated according the minimum and maximum values of the solutions for each objective, and we measure the “mean density of the hypergrid”. The mean density of the hypergrid is equal to the number of potentially non-dominated generated divided by the number of hypercubes of the hypergrid that contain at least one potentially non-dominated points.

The hypergrid allows thus to obtain information concerning the repartition of the solutions in the objective space and to stop the search once *enough* solutions have been generated. Here, we will simply stop the method when the mean density of the hypergrid attains a certain threshold.

We show the results that we have obtained with this technique in Fig. 3. The figures represent the evolution of the T indicator and the evolution of the number of potentially efficient solutions $|PE|$ according to the running time. The running time is controlled with the density threshold parameter. We start with a value of density equal to 2 with 2 divisions of the objective space for each objective and we end with a value of density equal to 20 with 20 divisions. Between these two values, we increase both the mean density threshold and the number of divisions with a step equal to 2. The method will stop when the mean density threshold will be attained or if a Pareto local optimum set has been reached. We have used for the neighborhood the same parameters than with the instances with two objectives, that is $k = 2$ and $L = 9$. The results of 2PPLS are compared with the method used in Phase 1 (P1): we randomly generate weight sets and

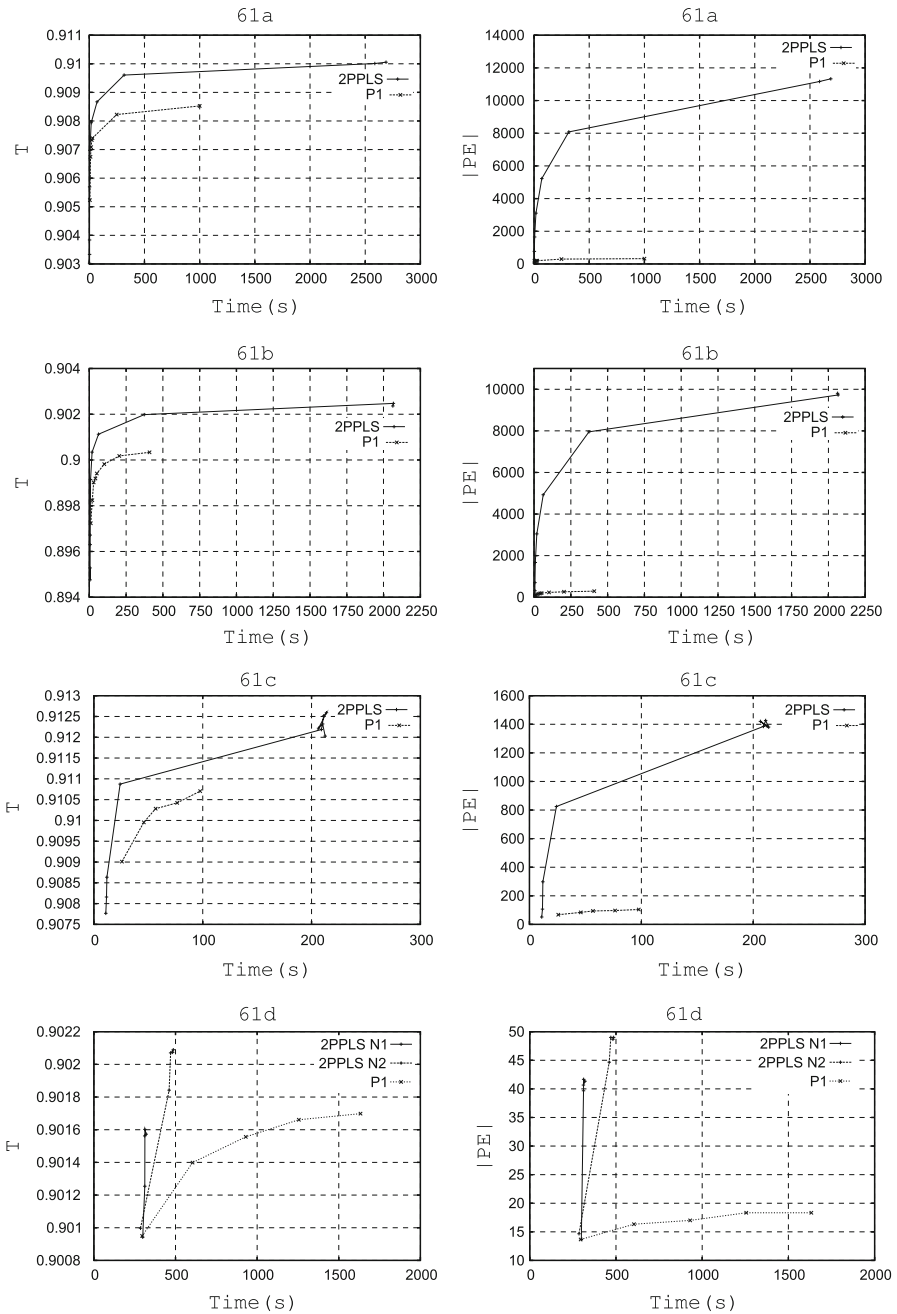


Fig. 3 Evolution of the T indicator and the number of potentially efficient solutions ($|PE|$) according to the running time for the 61a, 61b, 61c and 61d instances

the weighted-sum problems are solved with the exact method based on `lp_solve`. The running time of P1 is controlled with the number of weight sets.

For the instance 61a, from the Fig. 3, we see that the density threshold allows to control the running time of 2PPLS. The last point of the graph has been obtained with a density threshold of 20, with a number of divisions also equal to 20. This point has been reached in more than 2,500 s. However, with a density threshold of 16, we can attain a point with a similar quality, in less than 500 s. We also see that 2PPLS always gives better results than P1.

We come to the same conclusion with the instance 61b.

For the instance 61c, from a density threshold equal to 10, we see that a Pareto local optimum set has been obtained since increasing the density threshold does not allow to improve the results. We also see that, for this instance, using a hypergrid is not essential: as the number of potentially efficient solutions is rather small comparing to the instances 61a and 61b, a Pareto local optimum set is reached in about 200 s.

We get the same conclusion for the instance 61d, except that the Pareto local optimum set has been reached from a density threshold equal to 6. Also, for this instance, we have considered two sets of parameters for the neighborhood: the initial set ($k = 2$ and $L = 9$, under the name N1) and a larger one, with $k = 4$ and $L = 10$ (under the name N2). We see that this larger neighborhood allows to get better final results. However, if the running time was limited to around 300 s, it is preferable to use the neighborhood N1 since better results are faster reached.

We have thus seen through these experiments the difficulty to tune the parameters of 2PPLS for solving three-objective instances. For some instances, a Pareto local optimum set is rapidly reached and it is not necessary to use a hypergrid to control the search. On the other hand, for some of the instances, the number of potentially efficient solutions generated is so large that it is preferable to use a tool (such as the hypergrid) to control the number of potentially efficient solutions generated by the algorithm.

A step further will be to dynamically updating the parameters of the algorithm according to the instances or to the behavior of the algorithm during the search, and to attain an “any-time” behavior (that is, whatever the time allowed to the algorithm, the best set of parameters is used in order to get the best approximation that is possible to reach for the time considered), as it has already been studied in biobjective optimization (Dubois-Lacoste et al. 2012).

7 Conclusion and perspectives

We have shown through this paper the effectiveness of the approach based on 2PPLS, VNS and LNS to solve the MOSCP. The integration of VNS and LNS into 2PPLS allowed to obtain new state-of-the-art results for the MOSCP (biobjective and three-objective instances). LNS is particularly suited to multiobjective optimization, since with this approach a set of new potentially efficient solutions can be easily produced from one solution. VNS, for his part, helps to escape from a local Pareto optimum set.

Among the perspectives, more experiments could be conducted with the LNS: use of other ratios, considering only a subset of rows to define the neighborhood or

using a heuristic to solve the residual problems instead of using an exact enumeration algorithm. A dynamic control of the parameters, according to the instances solved, would also be worth to be studied, in order to reach an “any-time” behavior of the algorithm.

For the three-objective instances, we could use techniques based on the integration of the preferences of the decision maker (Thiele et al. 2009), so that the search of potentially efficient solutions could only be realized in a specific part of the search space. With this aim, refinement of the Pareto dominance (by integration of some preferences of the decision maker) may be useful.

Finally, the approach is general and could be used to develop new heuristics for solving other MOCO problems and some applications.

References

- Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123**(1–3), 75–102 (2002)
- Aneja, Y.P., Nair, K.P.K.: Bicriteria transportation problem. *Manag. Sci.* **25**, 73–78 (1979)
- Angel, E., Bampis, E., Gourvès, L.: A dynasearch neighborhood for the bicriteria traveling salesman problem. In: Gandibleux, X., Sevaux, M., Sörensen, K., T’kindt, V. (eds.) *Metaheuristics for Multiobjective Optimisation. Lecture Notes in Economics and Mathematical Systems*, pp. 153–176. Springer, Berlin (2004)
- Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*. Kluwer Academic Publishers, New York (2002)
- Czyzak, P., Jaszkiwicz, A.: Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *J. Multi-Criteria Decis. Anal.* **7**, 34–47 (1998)
- Deb, K.: *Multi-objective Optimization Using Evolutionary Algorithms*. Wiley, New-York (2001)
- Drugan, M.M., Thierens, D.: Stochastic pareto local search: Pareto neighbourhood exploration and perturbation strategies. *J. Heuristics* **18**(5), 727–766 (2012)
- Dubois-Lacoste, J., López-Ibáñez, M., Stützle, T.: Pareto local search algorithms for anytime bi-objective optimization. In: Hao, J.-K., Middendorf, M. (eds.) *EvoCOP of Lecture Notes in Computer Science*, vol. 7245, pp. 206–217. Springer, Berlin (2012)
- Ehrgott, M., Gandibleux, X.: Multiobjective combinatorial optimization. In: Ehrgott, M., Gandibleux, X. (eds.) *Multiple Criteria Optimization—State of the Art Annotated Bibliographic Surveys*, vol. 52, pp. 369–444. Kluwer Academic Publishers, Boston (2002)
- Gandibleux, X., Vancoppenolle, D., Tuytens, D.: A first making use of GRASP for solving MOCO problems. In *14th International Conference in Multiple Criteria Decision-Making*, Charlottesville (1998)
- Glover, F., Kochenberger, G.: *Handbook of Metaheuristics*. Kluwer, Boston (2003)
- Hansen, P.: Bicriterion path problems. In: Fandel, G., Gal, T. (eds.) *Lecture Notes in Economics and Mathematical Systems*, vol. 177, pp. 109–127. Springer, Berlin (1979)
- Hansen, M.P., Jaszkiwicz, A.: Evaluating the quality of approximations of the nondominated set. Technical Report, Technical University of Denmark, Lyngby, Denmark (1998)
- Hansen, P., Mladenovic, N.: Variable neighborhood search: principles and applications. *Eur. J. Oper. Res.* **130**(3), 449–467 (2001)
- Hertz, A., Jaumard, B., Ribeiro, C.: A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *RAIRO* **28**(3), 303–328 (1994)
- Jaszkiwicz, A.: On the performance of multiple-objective genetic local search on the 0/1 knapsack problem—a comparative experiment. *IEEE Trans. Evol. Comput.* **6**(4), 402–412 (2002)
- Jaszkiwicz, A.: Do multiple-objective metaheuristics deliver on their promises? A computational experiment on the set-covering problem. *IEEE Trans. Evol. Comput.* **7**(2), 133–143 (2003)
- Lan, G., DePuy, G.W., Whitehouse, G.E.: An effective and simple heuristic for the set covering problem. *Eur. J. Oper. Res.* **176**(3), 1387–1403 (2007)

- Laumanns, M., Thiele, L., Zitzler, E.: An adaptative scheme to generate the Pareto front based on the epsilon-constraint method. Technical Report 199, Technischer Bericht, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) (2004)
- Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. *Oper. Res.* **21**, 498–516 (1973)
- Lourenço, H.R., Paixão, J.P., Portugal, R.: The crew-scheduling module in the gist system. Economics Working Papers 547, Department of Economics and Business, Universitat Pompeu Fabra (2001)
- Lust, T., Teghem, J.: The multiobjective traveling salesman problem a survey and a new approach. In: Coello Coello, C., Dhaenens, C., Jourdan, L. (eds.) *Advances in Multi-Objective Nature Inspired Computing of Studies in Computational Intelligence*, vol. 272, pp. 119–141. Springer, Berlin (2010)
- Lust, T., Teghem, J.: The multiobjective multidimensional knapsack problem: a survey and a new approach. *Int. Trans. Oper. Res.* **19**(4), 495–520 (2012)
- Lust, T., Teghem, J.: Two-phase Pareto local search for the biobjective traveling salesman problem. *J. Heuristics* **16**(3), 475–510 (2010)
- Miettinen, K.: *Nonlinear Multiobjective Optimization*. Kluwer, Boston (1999)
- Moraga, R.: *Meta-RaPS: an effective solution approach for combinatorial problems*. PhD thesis, University of Central Florida, Orlando (FL), US (2002)
- Paquete, L., Chiarandini, M., Stützle, T.: Pareto local optimum sets in the biobjective traveling salesman problem: an experimental study. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) *Metaheuristics for Multiobjective Optimisation*. Lecture Notes in Economics and Mathematical Systems, vol. 535, pp. 177–199. Springer, Berlin (2004)
- Pisinger, D., Ropke, S.: Large neighborhood search. In: Gendreau, M., Potvin, J.-Y. (eds.) *Handbook of Metaheuristics of International Series in Operations Research & Management Science*, vol. 146, pp. 399–419. Springer, USA (2010)
- Prins, C., Prodhon, C.: Two-phase method and lagrangian relaxation to solve the bi-objective set covering problem. *Ann. Oper. Res.* **147**(1), 23–41 (2006)
- Przybylski, A., Gandibleux, X., Ehrgott, M.: Two-phase algorithms for the biobjective assignment problem. *Eur. J. Oper. Res.* **185**(2), 509–533 (2008)
- Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In *CP '98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pp. 417–431, London, UK (1998)
- Steuer, R.: *Multiple Criteria Optimization: Theory, Computation and Applications*. Wiley, New York (1986)
- Teghem, J.: Multiple objective linear programming. In: Bouyysou, D., Dubois, D., Prade, H., Pirlot, M. (eds.) *Decision-Making Process (Concepts and Methods)*, pp. 199–264. Wiley, Hoboken (2009)
- Thiele, L., Miettinen, K., Korhonen, P.J., Molina, J.: A preference-based evolutionary algorithm for multi-objective optimization. *Evol. Comput.* **17**(3), 411–436 (2009)
- Ulungu, E.L., Teghem, J.: Multiobjective combinatorial optimization problems: a survey. *J. Multi-Criteria Decis. Anal.* **3**, 83–104 (1994)
- Ulungu, E.L., Teghem, J., Fortemps, Ph, Tuytens, D.: MOSA method: a tool for solving multiobjective combinatorial optimization problems. *J. Multi-Criteria Decis. Anal.* **8**(4), 221–236 (1999)
- Zitzler, E.: *Evolutionary algorithms for multiobjective optimization: methods and applications*. PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (1999)
- Zitzler, E., Laumanns, M., Thiele, L., Fonseca, C.M., Grunert da Fonseca, V.: Why quality assessment of multiobjective optimizers is difficult. In: Langdon, W.B., Cantú-Paz, E., Mathias, K., Roy, R., Davis, D., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E., Jonoska, N. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2002)*, pp. 666–673. Morgan Kaufmann Publishers, San Francisco (2002)
- Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M.: Performance assessment of multiobjective optimizers: an analysis and review. *IEEE Trans. Evol. Comput.* **7**(2), 117–132 (2003)