A Variable Neighborhood Search algorithm for the Multimode Set Covering Problem.

Fabio Colombo* Roberto Cordone† Guglielmo Lulli‡

June 29, 2013

Abstract

This paper introduces the *Multi-Mode Set Covering Problem*, which consists of a plurality of set covering problems linked by cardinality constraints. We propose a *Variable Neighbourhood Search* algorithm and a *Greedy Randomized Adaptive Search Procedure* based on a common local search routine. This routine applies a penalized relaxation of the covering constraints, tuned by self-adapting parameters, and visits a sequence of neighborhoods in a nested strategy. We compare the two heuristics with each other and with a time-limited run of a general-purpose *Integer Linear Programming* solver, on a benchmark set of instances with heterogeneous structure. Both heuristics outperform the solver, though with interesting differences with respect to the various classes of instances. In particular, the *Variable Neighbourhood Search* algorithm proves more effective and less dependent on the specific features of the instances.

1 Introduction.

Given a ground set I and a family of subsets, the classical *Set Covering Problem* (*SCP*) associates a cost to each subset and requires to find a collection of subsets "covering" the ground set, i. e. whose union coincides with I, such that the total cost of the selected subsets is minimum. The elements of the ground set are usually denoted as *rows* and the subsets as *columns*, because the problem can be represented on a binary matrix as the search for a group of columns with at least one nonzero entry overall in each row.

The *Multi-Mode Set Covering Problem* (*MM-SCP*) introduces a set of *modes* and considers for each mode a *SCP* instance defined on the same ground set I. Wlog, we may assume that the collections of subsets associated to the *SCP*

*University of Milano, Department of Computer Science, via Comelico 39, 20135 Milano, Italy, fabio.colombo2@unimi.it

†University of Milano, Department of Computer Science, via Comelico 39, 20135 Milano, Italy, roberto.cordone@unimi.it

‡University of Milano "Bicocca", Department of Informatics, Systems and Communication, viale Sarca 336, 20122 Milano, Italy, lulli@disco.unimib.it

instances have the same cardinality. In view of this assumption, each *SCP* instance has the same number of rows and columns. In the following, we denote corresponding rows (columns) in the different subproblems as a single row (column) considered in different modes. In general, a column has different costs and covers different subsets of rows in each mode. All rows should be covered in all modes. The set covering subproblems associated to each mode are linked by cardinality constraints, which impose a limit on the number of modes in which the same column can be included in the solution. We will denote as I the set of rows, as J the set of columns and as M the set of modes.

We can now formulate the *MM-SCP* problem as follows:

$$\min f = \sum_{j \in J, m \in M} c_{jm} x_{jm} \tag{1a}$$

$$\sum_{j \in J} a_{ijm} x_{jm} \geq 1 \qquad i \in I, m \in M \tag{1b}$$

$$\sum_{m \in M} x_{jm} \leq b_j \qquad j \in J \tag{1c}$$

$$x_{jm} \in \{0, 1\} \qquad j \in J, m \in M \tag{1d}$$

where $x_{jm} \in \{0, 1\}$ is the binary decision variable indicating if column j is used in mode m, or not; a_{ijm} indicates if row i is covered by column j in mode m (indicator function); c_{jm} is the cost of selecting column j in mode m; b_j is the maximum number of modes in which column j can be used.

In the *MM-SCP* formulation, the covering constraints (1b) state that each element of the ground set belongs to at least one of the selected subsets for each mode. The cardinality constraints (1c) limit the number of modes for each column.

The *MM-SCP* is a generalization of the *SCP*. In the last decades, several studies have been reported on the *SCP* and its extensions. We will briefly review these developments and their relations with the *MM-SCP*.

Taxonomy

The *MM-SCP* trivially reduces to a *SCP* when only one mode is considered, i.e., $|M| = 1$. An alternative reduction can be obtained setting $b_j \geq |M|$ for all $j \in J$. Under this assumption, the cardinality constraints are redundant and they are inactive in correspondence to any optimal solution. As well, the k-set-covering problem, which consists in computing k disjoint coverings of minimum total cost [2], is a special case of the *MM-SCP* in which the number of modes is equal to k, each column covers the same rows in all modes and $b_j = 1$ for all $j \in J$.

In addition to the two cited problems, there are several problems of the set cover family which are related to the multimode version herein studied. For instance, we can mention the *multiple-choice set cover problem*, which consists in finding a cover of all the elements of the ground set I selecting one subset (a

choice) from each family (set of possible choices) [1]. With respect to the *MM-SCP*, the multiple-choice set covering problem requires to find a single covering, instead of one for each mode, and has disjunction constraints ($b_j = 1$) instead of general cardinality constraints.

The *set multi-cover problem* expects each element of the ground set to be covered several times [4], but it does not require the solution to consist of $|M|$ full coverings and it does not impose cardinality constraints. It is therefore a relaxation of the *MM-SCP*: a feasible solution of the *MM-SCP* is also feasible for the set multi-cover problem. A similar relation holds with the multi-set version of the multi-cover problem, in which each column can be used several times to provide multiple coverings [14].

Applications

Sensor Networks. Multiple wireless sensor networks under the administration of different authorities, but located physically on the same area or close to each other, have to service a set of clients distributed on the considered region. Each network must service all customers. Sensors belonging to different networks cannot be located on the same site [5]. This problem is equivalent to the *MM-SCP* with $b_j = 1$ for all $j \in J$.

Computational Biology. The scope of reverse engineering methods is to reconstruct gene regulatory networks from DNA microarray gene expression data, i.e., to produce a high-fidelity representation of the cellular network topology as a graph, where nodes represents genes and arcs represent direct regulatory interactions (i.e., influences of gene products upon the expression of other genes), thus explaining gene expression data [6, 17]. Both the activation and the inhibition of each gene must be explained as the effect of a minimum set of regulator genes, each of which must be labelled either as an activator or as an inhibitor.

Binate Covering. A fundamental problem in logic synthesis requires to find the minimum cost solution of a Boolean equation $f(x) = true$, where f is a formula in conjunctive normal form and the assignment of value *true* to each Boolean variable is associated to a cost. This problem is known as Binate Covering Problem [20], and is a generalization of the Satisfiability Problem (SAT). Each variable should be set either in the *true* or in the *false* mode, so as to satisfy all Boolean clauses at minimum cost.

Structure of the paper

Section 2 discusses the computational complexity and the approximability status of the *MM-SCP*; it also reviews some basic reduction procedures commonly adopted for the standard *SCP*, distinguishing those which can be extended to the *MM-SCP* from those which cannot. The following section describes in detail the two algorithms proposed, which follow, respectively, the *Variable Neighbourhood Search* (*VNS*) and the *Greedy Randomized Adaptive Search procedure* (*GRASP*) approach. The two algorithms share a common local search pro-

cedure, but restart the search in different ways. Our choice of these two approaches is motivated by the fact that, according to our previous investigations, they perform effectively on special cases of the *MM-SCP* [8, 9]. Moreover, *VNS* is characterized by a very small number of parameters, and the *reactive* form of *GRASP* allows an automatic tuning of the parameters. Both algorithms, therefore, portend a robust implementation and an off-the-shelf application. Section 4 discusses the computational results, and conclusions close the paper.

2 Computational complexity and problem properties

The decision version of the *MM-SCP* is $\mathcal{NP}$-complete, because it includes, as a special case, the well-known *SCP*, obtained by simply restricting the set of modes to a singleton, i.e., $|M| = 1$. However, the introduction of additional modes makes the general *MM-SCP* much more complex than its single-mode counterpart. Indeed, it is trivial to determine whether a *SCP* instance is feasible or not, and the *SCP* admits a logarithmic approximation guarantee, though not a constant one [15, 7]. Approximation results are also available for more general problems of the set cover family, such as the set multi-cover problem and its multi-set version as provided in [19]. By contrast, in the following we prove that, if there are at least two modes, even the feasibility of the *MM-SCP* is $\mathcal{NP}$-complete, and no approximation guarantee can be provided. This result is a consequence of the limit on the number of modes in which the same column can be included in the solution of *MM-SCP*.

Theorem 1 *It is $\mathcal{NP}$-complete to determine whether a given instance of the* MM-SCP *is feasible or not, even if the mode set M has only two elements.*

Proof. The proof is by reduction from the satisfiability problem (*SAT*). Given n Boolean variables ξ_j and r Boolean clauses C_i, an instance of *SAT* requires to determine whether there exists a truth assignment to the Boolean variables such that all Boolean clauses are true. In the following, we show that for any instance of *SAT* it is possible to build an instance of *MM-SCP* which is feasible if and only if the instance of *SAT* is feasible.

Let the mode set M consist of two modes, representing the affirmation and negation of the Boolean variables. Since the rows of the *MM-SCP* need to be covered in each of the two modes only by columns used in that mode, we first build an auxiliary *SAT* instance to bridge the transformation. To that purpose, we replace each Boolean clause with two clauses obtained as follows: in the former, we replace each negative literal $\bar{\xi}_j$ with an auxiliary affirmative one ξ'_j, in the latter we replace each affirmative literal ξ_j with an auxiliary negative one $\bar{\xi}'_j$. By construction, any satisfying truth assignment of the original instance identifies one of the new instance, in which the auxiliary variable ξ'_j has the opposite value of ξ_j. Conversely, any satisfying truth assignment of the auxiliary instance such that ξ_j and ξ'_j assume complementary values identifies a satisfying truth assignment for the original instance.

The set J of the *MM-SCP* instance includes one column for each ξ_j Boolean variable (namely, columns $j = 1, \ldots, n$) and one for each ξ'_j Boolean variable (namely, columns $j = n+1, \ldots, 2n$). The capacity values b_j are set equal to 1 for all columns, so that every column can appear in the solution in at most one mode, corresponding to the fact that the truth assignment must be consistent. The rest of the construction introduces suitable constraints to impose that each column is used in one of the two modes, that the columns associated to the ξ_j and ξ'_j Boolean variables assume complementary values and that each Boolean clause is satisfied. The first two conditions are guaranteed introducing in set I one row for each Boolean variable (namely, rows $i = 1, \ldots, n$), and setting $a_{i,i,m}$ and $a_{i,n+i,m}$ to 1 and all other coefficients to zero for both modes $m = 1$ and 2. The resulting covering constraints:

$$\begin{cases} x_{j,1} + x_{n+j,1} \geq 1 \\ x_{j,2} + x_{n+j,2} \geq 1 \end{cases} \qquad j = 1, \ldots, n$$

can be combined with the cardinality constraints

$$\begin{cases} x_{j,1} + x_{j,2} \leq 1 \\ x_{n+j,1} + x_{n+j,2} \leq 1 \end{cases} \qquad j = 1, \ldots, n$$

to obtain the desired result. In fact, if $x_{j,1} = 1$, necessarily $x_{j,2} = 0$, which implies that $x_{n+j,2} = 1$, and $x_{n+j,1} = 0$. Conversely, if $x_{j,1} = 0$, necessarily $x_{n+j,1} = 1$, which implies that $x_{n+j,2} = 0$, and $x_{n+j,1} = 1$. Consequently, $x_{j,1} = x_{n+j,2} = 1 - x_{j,2} = 1 - x_{n+j,1}$: in every feasible solution, columns j and $n+j$ are complementary in both modes, and the two modes of each column are complementary.

The satisfaction of the Boolean clauses is imposed introducing in set I one row for each original Boolean clause C_{i-n} (namely, rows $i = n+1, n+r$). Row i in mode 1 corresponds to the auxiliary clause with only affirmative literals: if literal ξ_j occurs in the clause, coefficient $a_{i,j,1}$ is set to 1; if literal ξ'_j occurs in the clause, coefficient $a_{i,n+j,1}$ is set to 1; all other coefficients are set to zero. Similarly, row i in mode 2 corresponds to the auxiliary clause with only negative literals: if literal $\bar{\xi}_j$ occurs in the clause, coefficient $a_{i,j,2}$ is set to 1; if literal $\bar{\xi}'_j$ occurs in the clause, coefficient $a_{i,n+j,2}$ is set to 1; all other coefficients are set to zero. The resulting covering constraints guarantee that each Boolean clause includes at least one satisfying literal.

So, the x_{jm} variables behave like a consistent truth assignment to Boolean variables, and the second family of constraints in mode $m = 1$ requires the truth assignment to be satisfying. Therefore, if the *MM-SCP* instance is feasible, the *SAT* instance is also feasible.

Conversely, assume that *SAT* admits a feasible solution: the corresponding truth assignment can be converted into a *MM-SCP* solution setting $x_{j,1} = x_{n+j,2} = 1$ and $x_{j,2} = x_{n+j,1} = 0$ when the variable ξ_j is true and $x_{j,1} = x_{n+j,2} = 0$ and $x_{j,2} = x_{n+j,1} = 1$ when it is false. This solution is feasible, since the capacity constraints are respected, as well as the covering constraints

expressing the relation between variables $x_{j,m}$ and $x_{n+j,m}$ and those expressing the satisfaction of the Boolean clauses. ■

This $\mathcal{NP}$-completeness proof also implies that no polynomial algorithm can have an approximation guarantee, unless in the unlikely case that $\mathcal{P} = \mathcal{NP}$. We remind that, given a suitable function $\alpha(\cdot)$, a minimization problem is $\alpha(\cdot)$-*approximable* when it admits a polynomial time algorithm which, applied to any instance of size d and minimum cost f^*, provides a solution costing at most $\alpha(d) f^*$.

Corollary 1 *The* MM-SCP *does not admit any polynomial algorithm with an approximation guarantee, unless* $\mathcal{P} = \mathcal{NP}$.

Proof. Given any *SAT* instance, consider the corresponding *MM-SCP* instance built as in Theorem 1. Introduce two additional columns $2n+1$ and $2n+2$. Set the costs $c_{2n+1,1}$ and $c_{2n+2,2}$ to 1 and all other costs to 0. Set the coefficients $a_{i,2n+1,1}$ and $a_{i,2n+2,2}$ to 1, $a_{i,2n+1,2}$ and $a_{i,2n+2,1}$ to 0, for all rows $i \in I$. The additional pair of variables $x_{2n+1,m}$ and $x_{2n+2,m}$ now behave like an additional Boolean variable ξ_{n+1}, which, when affirmed, satisfies all Boolean clauses at unitary cost.

The original *SAT* instance is feasible if and only if the *MM-SCP* instance admits a solution of zero cost. A polynomial algorithm solving the *MM-SCP* with an approximation guarantee would provide a solution costing at most $\alpha(d) f^* = 0$, that is an optimal solution. Consequently, it would solve *SAT* in polynomial time, implying that $\mathcal{P} = \mathcal{NP}$. ■

A number of reduction procedures are commonly adopted to manipulate *SCP* instances so as to produce smaller equivalent instances prior to attacking them with heuristic or exact algorithms [3]. We briefly extend two of these procedures to the *MM-SCP* and provide a simple counterexample for a third one.

Proposition 1 *(Row domination) Let* $i_1, i_2 \in I$ *be two rows and* $m \in M$ *a mode such that all columns covering* i_1 *in mode* m *also cover* i_2 *in the same mode:* $a_{i_1 j m} \leq a_{i_2 j m}$ *for all* $j \in J$. *Then, the constraint on row* i_2 *in mode* m *is redundant.*

Proof. If $a_{i_1 j m} \leq a_{i_2 j m}$ for all $j \in J$, then $\sum_{j \in J} a_{i_1 j m} x_{j m} \leq \sum_{j \in J} a_{i_2 j m} x_{j m}$ for all $x_{j m} \in \{0, 1\}$, which trivially implies that the constraint on row i_1 and mode m is tighter than that on row i_2 and mode m. ■

Proposition 2 *(Column essentiality) Let* $i \in I$ *be a row and* $m \in M$ *a mode such that a single column* $j' \in J$ *covers row* i *in mode* m: $a_{ij'm} = 1$ *and* $a_{ijm} = 0$ *for all* $j \neq j'$. *In all feasible solutions column* j *is used in mode* m.

Proof. If $a_{ij'm} = 1$ and $a_{ijm} = 0$ for all $j \neq j'$, then $\sum_{j \in J} a_{ijm} x_{jm} = x_{j'm} \geq 1$, which implies that $x_{j'm} = 1$. ■

The *column domination* property for the *SCP* states that, given two columns j_1 and j_2, such that $c_{j_1} \leq c_{j_2}$ and all rows covered by j_2 are also covered by j_1,

column j_2 can be removed from the instance. This property does not hold for the *MM-SCP*, even if the two conditions are imposed on all modes, due to the cardinality constraint.

Remark 1 *The* column domination *property cannot be extended from the* SCP *to the* MM-SCP*.*

Proof. Consider a *MM-SCP* instance with a column of unitary capacity and zero cost, covering all rows in all modes: $b_j = 1$, $c_{jm} = 0$ for all $m \in M$ and $a_{ijm} = 1$ for all $i \in I$, $m \in M$. All other columns are clearly dominated by column j. However, if the number of modes is larger than one, some of the other columns are required to obtain a feasible solution. ■

3 Heuristics for the Multimode Set Covering Problem

This section describes two local search metaheuristics for the *MM-SCP*, respectively following the *VNS* and the *GRASP* frameworks. The former periodically restarts the search from solutions generated perturbating the currently best known one (see Section 3.2); the latter restarts from solutions generated by a greedy randomized constructive heuristic (see Section 3.3). Both algorithms apply the same basic local search procedure, in which the cardinality constraints (1c) are strictly enforced, while the covering constraints (1b) can be freely violated. In order to compare solutions with different degrees of unfeasibility, we define the subset of row-mode pairs whose covering constraints are violated in a solution x as

$$V(x) = \left\{ (i,m) \in I \times M : \sum_{j \in J} a_{ijm} x_{jm} = 0 \right\}$$

and, given two solutions x' and x'', we consider x' better than x'' when $|V(x')| < |V(x'')|$, or $|V(x')| = |V(x'')|$ and $f(x') < f(x'')$.

3.1 The local search procedure

We here describe the local search procedure shared by the two algorithms in a strictly top-down approach because it is rather sophisticated. We first introduce the several neighborhoods explored and the general structure of the exploration procedure, then we describe into details how moves are evaluated and its technicalities.

Neighbourhood hierarchy The use of different neighborhoods allows to enrich and strengthen the search. In fact, when the current solution is a local optimum with respect to one of them, it is generally not so for others, and switching to an alternative neighborhood allows to proceed the search, further improving the solution [13]. In this work, we consider the following neighborhoods, all of

which include only solutions respecting the cardinality constraint (1c), while the covering constraints (1b) can be freely violated:

- $\mathcal{N}_1$: flip the value of one of the x_{jm} variables, thus adding (removing) column j in mode m to (from) the solution;
- $\mathcal{N}_X$: given a column j and two modes m and m' such that $x_{jm} = 1$ and $x_{jm'} = 0$, exchange the values of the two variables, thus changing one of the modes in which the column is used;
- $\mathcal{N}_2^h$: given a mode m and two columns j and j' whose intersection in mode m consists of h rows ($|\{i \in I : a_{ijm} = a_{ij'm} = 1\}| = h$) and such that $x_{jm} = 1$ and $x_{j'm} = 0$, exchange the values of the two variables, thus replacing column j with j' in mode m.

The size of neighborhoods $\mathcal{N}_1$ and $\mathcal{N}_X$ is linear in the number of columns $|J|$ and modes $|M|$; the union of all neighborhoods $\mathcal{N}_2^h$ has a linear size with respect to $|M|$ and a quadratic size with respect to $|J|$.

The exploration of different neighborhoods is the basis of the *Variable Neighbourhood Descent* (*VND*) approach [13]. Adhering to this framework, we explore the neighborhoods listed above in the following sequence:

$$\mathcal{N}_1 \to \mathcal{N}_X \to \mathcal{N}_2^{h_{\max}} \to \mathcal{N}_2^{h_{\max}-1}, \ldots, \mathcal{N}_2^1 \tag{2}$$

where $h_{\max}$ is the cardinality of the largest intersection between two columns in the same mode. Given a current solution $\tilde{x}$, the procedure considers its neigborhood $\mathcal{N}_1$, evaluates all the neighbor solutions, chooses one of them, denoted by x', and decides whether to replace $\tilde{x}$ with x' or not. If x' replaces $\tilde{x}$, the search resumes from the first neighborhood $\mathcal{N}_1$ of the new current solution. Otherwise, the procedure remains centered on $\tilde{x}$ and considers the following neighborhood in the sequence. This nested strategy is graphically represented in Figure 1.

Notice that, since the aim of the local search is to oscillate between the feasible and the unfeasible region, in general solution x' is not chosen as the best in the current neighborhood and, consequently, solution $\tilde{x}$ is not the best one found so far, denoted as x^*. The specific strategy used to choose x' and to decide whether x' should replace $\tilde{x}$ or not is described in detail in the following paragraph.

Move evaluation The solutions of a given neighborhood are estimated by a weighted sum of their cost $f(x)$ and an auxiliary objective function $\pi(x)$, which measures the violation of the covering constraints:

$$\pi(x) = \sum_{(i,m) \in V(x)} \pi_{im} \tag{3}$$

where the coefficients π_{im} are defined in the following.

Since $f(x)$ and $\pi(x)$ are dimensionally different, and they could have a different importance during the exploration, we:

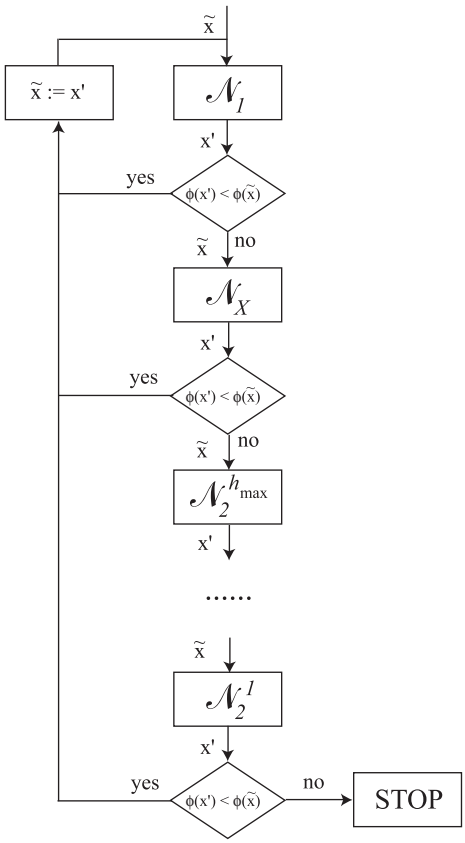

Figure 1: Graphical representation of the basic local search procedure

- normalize the two functions before combining them, and
- update their relative weights according to the current solution.

The normalization is performed with respect to the best and worst alternatives available in the current neighborhood:

$$\bar{f}(x) = \frac{f(x) - f_{\min}}{f_{\max} - f_{\min}} \qquad \bar{\pi}(x) = \frac{\pi(x) - \pi_{\min}}{\pi_{\max} - \pi_{\min}}$$

where $f_{\min}$ and $f_{\max}$ are the smallest and largest costs of the neighbor solutions, while $\pi_{\min}$ and $\pi_{\max}$ are the smallest and largest violations of the neighbor solutions. In this way, both functions assume values in $[0; 1]$, and can be combined meaningfully.

The combined function used to evaluate the neighbor solutions is

$$\phi(x) = \gamma \bar{f}(x) + (1 - \gamma)\,\bar{\pi}(x) \tag{4}$$

where $\gamma \in (0, 1)$ is a suitable coefficient. All solutions in the current neighborhood are evaluated and the one which minimizes function ϕ, denoted as x', is compared to the current solution $\tilde{x}$. If $\phi(x') < \phi(\tilde{x})$, then x' replaces $\tilde{x}$ as the new current solution. Otherwise, the exploration of the neighborhood terminates, and the search proceeds with the next neighborhood in sequence (2).

Tie-breaking rules For some classes of instances, function $\phi(x)$ often assumes the same value for different solutions (in particular, this occurs when the cost of all columns in all modes is the same). In those cases, the use of a tie-breaking rule is important to improve the search diversification. In the experimental result section, we will compare two alternative tie-breaking rules:

- the *random* selection rule chooses one of the best moves at random using a uniform distribution;
- the *memory*-based selection rule chooses the best move which involves the variables whose value has changed longest ago.

The memory-based rule is inspired by the similar mechanism applied in algorithm *HSAT* [12] for the *SAT* problem. In order to apply this rule, we simply store the iteration in which each variable x_{jm} assumed its current value, and use it to compare moves yielding the same $\phi(x)$.

Self-adaptive tuning of parameter γ The coefficient γ which combines the normalized values of the total cost and the total violation according to Equation (4) is obviously a very influential parameter. If γ is too close to 1, the search evolves towards cheaper, but unfeasible, solutions; if γ is too close to 0, feasibility is regained, but the solution cost increases. The best value of γ is hard to estimate, and probably varies during the search. We therefore introduced a self-adaptive mechanism to tune γ automatically based on the feasibility of the

current solution $\tilde{x}$. The procedure sets $\gamma = 0.5$ at the beginning of the local search, and updates it at each iteration according to the following rule:

$$\gamma^{(t+1)} = \begin{cases} \rho\,\gamma^{(t)} + (1-\rho) & \text{if } \tilde{x} \text{ is feasible} \\ \rho\, z\gamma^{(t)} & \text{if } \tilde{x} \text{ is unfeasible} \end{cases}$$

The rationale of such a rule is to perform a convex combination of the current value of γ with 1 when the solution is feasible (and therefore the weight of the cost function should increase), and with 0 when the solution is unfeasible (and therefore, the weight of the violation function is underestimated).

Parameter $\rho \in (0; 1)$ measures the speed of the update: the closer it is to 1, the slower is the update. Its value must be specified by the user. Section 4.1 is devoted to discuss the effect of its value on the quality of the obtained results.

Self-adaptive tuning of coefficients π_{im} According to definition (3), the value of the violation function $\pi(\tilde{x})$ in the current solution $\tilde{x}$ is the sum of suitable coefficients π_{im} over all the covering constraints violated in $\tilde{x}$. The values of these coefficients should reflect the hardness to satisfy each single covering constraint; they are hard to estimate *a priori*, and probably their correct value varies during the search. For these reasons, we introduce a self-adaptive tuning mechanism similar to the one used for parameter γ. Specifically, each coefficient π_{im} varies within a given range $[0; c_{im}^{\max}]$, where $c_{im}^{\max} = \max_j \{a_{ijm} c_{ij}\}$ is the largest cost of the columns covering row i in mode m. The procedure sets $\pi_{im} := c_{im}^{\max}$ at the beginning of the local search, and updates it at each iteration according to the following rule:

$$\pi_{im}^{(t+1)} := \begin{cases} \rho\pi_{im}^{(t)} + (1-\rho)\, c_{im}^{\max} & \text{if } \tilde{x} \text{ does not cover row } i \text{ in mode } m \\ \rho\pi_{im}^{(t)} & \text{if } \tilde{x} \text{ covers row } i \text{ in mode } m \end{cases}$$

The rationale is to perform a convex combination of the current value of the parameter with its largest possible value when the associated constraint is violated, and with its smallest possible value when it is respected. When π_{im} assumes its maximum value $c_{im}^{\max}$, it is always profitable to cover row i in mode m by adding a new column. For the sake of simplicity, the coefficient ρ used to tune the speed of the update is the same used for parameter γ.

Termination conditions The exploration of each neighborhood terminates as soon as the best neighbor solution x' proves worse than the current solution $\tilde{x}$ with respect to the auxiliary function ϕ. However, since the weight γ and the coefficients π_{im} used to define the objective function are updated at each iteration, there is no guarantee that the local search procedure will converge in a finite number of iterations to a local optimum. Consequently, we also terminate the search after a given number $\ell_{\max}$ of iterations.

Efficient exploration of the neighborhoods Suitable conditions allow to improve the efficiency of the exploration, reducing the neighborhoods with little

impact on the quality of the best solution found. In particular, neighborhood $\mathcal{N}_2^0$ is not explored, because any of its moves can be seen equivalently as the sequence of two moves in $\mathcal{N}_1$, respectively setting x_{jm} to 0 and $x_{j'm}$ to 1, that is dropping column j and adding column j' in mode m. The authors of [22] investigate the exploration of $\mathcal{N}_2^0$ for the *SCP*. Just as we do with auxiliary function (3), they define a violation function given by the sum of suitable coefficients over all violated constraints. Their investigation leads to a formal proof that, if a move in $\mathcal{N}_2^0$ improves the sum of the total cost plus the violation function, then one of the two elementary moves improves it as well. Hence, if one always explores neighborhood $\mathcal{N}_1$ in the first place, neighborhood $\mathcal{N}_2^0$ can be ignored.

We apply the same restriction to our local search procedure. However, since the cost and the violation function, instead of being simply summed, are normalized and combined with a self-adapting coefficient, the formal proof of [22] no longer holds, and the restriction is heuristic, though extensively justified by the experimental results.

For the same reason, neighborhood $\mathcal{N}_X$ is explored only in part. In fact, any of its moves can be seen equivalently as the sequence of two moves in $\mathcal{N}_1$, respectively setting $x_{jm'}$ to 1 and x_{jm} to 0, that is using a column in a new mode and quit using it in an old one. If the cardinality constraint of column j is non-active, that is the column is not used in the maximum feasible number of modes ($\sum_{m \in M} x_{jm} < b_j$), both moves are feasible, and an improving move in $\mathcal{N}_X$ probably implies that one of the two elementary moves in $\mathcal{N}_1$ is also improving. Since we always first explore neighborhood $\mathcal{N}_1$, the exploration of neighborhood $\mathcal{N}_X$ is restricted to the columns for which the cardinality constraint is tight.

3.2 A Variable Neighbourhood Search algorithm

The *Variable Neighborhood Search* (*VNS*) approach extends the concept of systematic neighborhood change typical of *VND* from the improvement phase to the restart phase [13] used by local search based metaheuristics. It involves a hierarchy of size-increasing neighborhoods and a basic local search procedure operating on a relatively small neighborhood.

A *VNS* algorithm alternatively runs the basic local search procedure and a *shaking* procedure, which generates a new starting solution for the following application of local search, extracting it at random from one of the neighborhoods of the best known solution x^*. At the first run, the new solution is extracted from the smallest neighborhood; in the following ones, the choice of the neighborhood depends adaptively on the result of the previous local search: if this has not improved the best known result, the shaking procedure switches to the immediately larger neighborhood; otherwise, it goes back to the smallest one. The rationale of the approach is to first generate new starting solutions close to the best known result, so as to intensify the search in a promising region of the solution space. If such a restart fails, intensification is deemed less profitable, and starting solutions are generated farther and farther away from the current best known one, so that diversification replaces intensification. As soon as a new best result is found, the approach switches back to intensification, and solutions

are generated again near the best known one.

Usually, the basic local search explores the smallest neighborhood in the hierarchy. Our *VNS* algorithm for the *MM-SCP*, on the contrary, adopts different neighborhoods for the basic local search and the shaking procedures. The former uses the refined *VND* approach described in Section 3.1, while the latter uses another family of neighborhoods which has been proposed in the literature on set covering problems (see for example [16]). In particular, neighborhood $\mathcal{N}'_k$ includes all the solutions which can be generated by selecting k column-mode pairs (j, m) used in the current solution ($x_{jm} = 1$), removing them (i. e. setting $x_{jm} = 0$) and regaining feasibility with respect to the covering constraints (as much as possible) with a deterministic greedy heuristic. The size of neighborhood $\mathcal{N}'_k$ is therefore in $O\left(|J|^k |M|^k\right)$.

As a greedy heuristic, we apply an adaptation of the classical heuristic proposed by Chvátal [7] for the *SCP*. Each step of the greedy heuristic considers all column-mode pairs (j, m) such that: a) they are not used in the current partial solution ($x_{jm} = 0$), b) they have not been set to zero by the removal procedure, c) they can be added without violating the cardinality constraint ($\sum_{m \in M} x_{jm} < b_j$), d) they decrease the number $\nu_{jm}(x)$ of violated covering constraints, if added to x. For each of these column-mode pairs, the heuristic evaluates the ratio $c_{jm}/\nu_{jm}(x)$, finds the minimum one, and sets to 1 the corresponding variable x_{jm}. When no column-mode pair satisfies the four conditions, the procedure terminates. In some classes of instances, in particular those where the cost of all columns is the same in all modes, we adopt the same tie-breaking rule described in Section 3.1 for the choice of the best solution in the current neighborhood: among the column-mode pairs with the minimum ratio, either we choose one at random (*random* strategy), or we choose that which has been added to the solution longest ago (*memory* strategy).

The $\mathcal{N}'_k$ neighborhoods exhibit features somehow complementary with respect to those used in the *VND* procedure, besides being much larger. In fact, the covering constraints are respected as much as possible, instead of freely violated; several variables can be flipped by a single move, instead of one or two; the selection of the column-mode pairs to be added to the solution is greedy, instead of exhaustive.

Figure 2 reports a pseudocode of the *VNS* algorithm. The procedure builds an initial solution x with the greedy heuristic, and saves it as the best one found so far, x^*. The neighborhood index k is set to 1, which is its minimum possible value. Each run of the *VND* procedure provides an improved solution x^o. If this is better than x^*, index k is set back to 1 and x^o replaces x^*. Otherwise, k increases. When k exceeds the number of column-mode pairs contained in x^*, we set k back to 1, given that further increases are useless. Finally, procedure *Shaking* extracts from $\mathcal{N}'_k$ the solution x which will be used for the following restart. The algorithm terminates after $R_{\max}$ restarts.

Algorithm *VNS_MMSCP*$(a, b, c, R_{\max}, \rho, \ell_{\max})$
$x :=$ *Greedy*(a, b, c);
$x^* := x$;
$k := 1$;
for $r := 1$ to $R_{\max}$ **do**
  $x^o :=$ *VND*$(x, a, b, c, \rho, \ell_{\max})$;
  {Update the starting solution and possibly the best known one}
  if $(|V(x^o)| < |V(x^*)|)$ or $(|V(x^o)| = |V(x^*)|$ and $f(x^o) < f(x^*))$ **then**
    $k := 1$;
    $x^* := x^o$;
  else
    $k := k + 1$;
    if $k >$ *Card*(x^*) **then** $k := 1$;
  end if
  $x :=$ *Shaking*(x^*, k, a, b, c);
end for
return x^*;

Figure 2: Pseudocode of the *VNS* algorithm

3.3 A Greedy Randomized Adaptive Search procedure

GRASP is a multi-start metaheuristic which alternatively builds starting solutions and improves them with local search [10]. The construction phase of our implementation for the *MM-SCP* uses a randomized version of the greedy heuristic described above (see Section 3.2). The improvement phase uses the *VND* procedure described in Section 3.1.

The pseudocode of the construction phase is reported in Figure 3. It starts from an empty solution ($x_{jm} = 0$ for all $j \in J$ and $m \in M$). At each step, it builds a candidate list CL containing all the column-mode pairs such that: a) they are not used in the current partial solution ($x_{jm} = 0$), b) they can be added without violating the cardinality constraint ($\sum_{m \in M} x_{jm} < b_j$), c) they decrease the number $\nu_{jm}(x)$ of violated covering constraints, if added to x. Then, procedure *Restrict* shrinks CL to a restricted candidate list RCL, keeping only the elements with the smallest values of $\delta = c_{jm}/\nu_{jm}(x)$. At the end of each step, the heuristic selects from RCL one of the candidate moves, and sets the corresponding variable to 1.

We have considered two alternative implementations of procedure *Restrict*:

1. *cardinality-based*: the cardinality of the RCL is fixed to $\lceil \alpha \cdot |CL| \rceil$, removing all the elements with the largest values of $\delta := c_{jm}/\nu_{jm}(x)$;

2. *quality-based*: the RCL contains all the (j, m, δ) triplets such that δ falls within $[\delta_{min}; \delta_{min} + \alpha(\delta_{max} - \delta_{min})]$, where δ_{min} and δ_{max} are the minimum and maximum value of δ.

In both implementations, parameter $\alpha \in (0; 1]$ is tuned by a reactive mecha-

Algorithm *GreedyRandomized_MMSCP*(a, b, c, α)
for each $(j, m) \in J \times M$ **do** $x_{jm} := 0$;
repeat
  $CL := \emptyset$;
  for all $(j, m) \in J \times M$ **do**
    if $x_{jm} = 0$ **and** $\sum_{m \in M} x_{jm} < b_j$ **and** $\nu_{jm}(x) > 0$ **then**
      $\delta := c_{jm} / \nu_{jm}(x)$;
      $CL := CL \cup \{(j, m, \delta)\}$;
    end if
  end for
  $RCL := Restrict(CL, \alpha)$;
  $(j^*, m^*, \delta^*) := RandomlyExtract(RCL)$;
  $x_{j^* m^*} := 1$;
until $CL = \emptyset$;
return x;

Figure 3: Pseudocode of the greedy randomized heuristic.

nism [18].

The overall *GRASP* algorithm is summarized in the pseudocode of Figure 4. At each restart, it calls procedure *GreedyRandomized_MMSCP*, choosing α at random from a user-defined set $\{\alpha_i\}$, with suitable probabilities $\{p_i\}$. Then, it improves the resulting solution with the *VND* procedure, and possibly updates the best solution found so far. The probability p_i of each value α_i is uniform for the first R_α restarts. After each sequence of R_α restarts, it is replaced by $p_i = (f^*/f_i) / \sum_i (f^*/f_i)$, where f^* is the value of the best known solution and f_i the average value of the solutions found with $\alpha = \alpha_i$. Accordingly, the values of α that lead to better solutions will be used more frequently in the following restarts. As in [18], we define set $\{\alpha_i\} = \{0.1, 0.2, \ldots, 1.0\}$ and $R_\alpha = 100$. The *GRASP* algorithm terminates after a total number of $R_{\max}$ iterations.

4 Computational Experience

In this section we present the computational results of the proposed algorithms, which have all been coded in C++, compiled by `gcc 4.4.3` and run on a PC equipped with an Intel Core2 Quad-core 2.66 GHz and 4 GB of RAM. We have considered three benchmark sets of random instances: the first one is composed of small instances, that have been used exclusively for the fine tuning of the algorithm parameters; the other two, composed of instances of medium and large size, that proved to be challenging for the commercial solver CPLEX 12.4, have been used to evaluate the computational performance of the proposed algorithms.

Guided by the applications described in Section 1, we have generated several types of instances. A first classification is into *square* and *rectangular* instances. For the former type, the number of columns $|J|$ is equal to the number of rows

Algorithm *GRASP_MMSCP*$(a, b, c, R_{\max}, R_\alpha, \{\alpha_i\}, \rho, \ell_{\max})$
InitializeProbabilities$(\{p_i\})$;
for $r := 1$ to $R_{\max}$ **do**
  $\alpha :=$ *Extract*$(\{\alpha_i\}, \{p_i\})$;
  $x :=$ *GreedyRandomized_MMSCP*(a, b, c, α);
  $x^o :=$ *VND*$(x, a, b, c, \rho, \ell_{\max})$;
  {Possibly update the best known solution}
  if $(|V(x^o)| < |V(x^*)|)$ or $(|V(x^o)| = |V(x^*)|$ and $f(x^o) < f(x^*))$ **then**
    $x^* := x^o$;
  end if
  if $(r \mod R_\alpha) = 0$ **then** *UpdateProbabilities*$(\{p_i\})$;
end for
return x^*;

Figure 4: Pseudocode of the *GRASP* algorithm

$|I|$: 100 rows and columns for the small instances, 500 for the medium ones and 1 000 for the large ones. This type of instances arises in computational biology and k-set-covering problems. For the rectangular type, the number of columns per mode is larger than the number of rows. More specifically, in this computational analysis, the number of columns $|J|$ has been set to ten times the number of rows $|I|$: 50 rows and 500 columns for the small instances, 100 rows and 1 000 columns for the medium ones, 500 rows and 5 000 columns for the large ones.

In addition to the column-row ratio and to the number of rows, each instance is also characterized by the following parameters: the number of modes, the right hand side of the cardinality constraints (capacity), the cost and the covering pattern of each column. As for the number of modes, we generated instances with $|M| = 2$ modes and instances with $|M| = 3$ modes. All of the former have capacity $b_j = 1$ for all $j \in J$, whereas for the latter we generated tighter instances with $b_j = 1$, and looser ones with $b_j = 2$. As for the cost column cost, we generated *unweighted* instances, with unitary costs for all columns in all modes, and *weighted* instances, with random costs drawn from a uniform distribution in the set $\{1, \ldots, 10\}$. Finally, the covering pattern of a column in different modes can be the same, i. e. the column covers the same rows in all the modes, or different. The first class of instances are named *uniform mode* instances, as opposed to their counterpart that we name *assorted mode* instances. For each combination of the parameters listed above, we have generated a pool of 5 randomly generated instances: the set of rows covered by each column is extracted from a uniform distribution imposing a 5% density[1]. In order to avoid "easy instances", we imposed in each mode that each column should cover at least one row and that each row should be covered by at least two columns. Overall, we generated 120 instances for each benchmark set.

[1]The density or coverage degree of a subset is the number (%) of elements of the ground set included in the subset, i.e., the number (%) of rows covered by the column.

4.1 Parameter tuning

A first computational campaign has been devoted to tuning the parameter ρ that controls the updating mechanisms of the local search procedure, which is the core routine of both the *VNS* and the *GRASP* algorithms. In addition, we compare two different implementations of the *tie-breaking rule* (*random* and *memory*) for both the metaheuristics, and two different reactive implementations of the *Restricted Candidate List* (*RCL*) for the *GRASP* (*value*-based and *cardinality*-based).

This campaign has been executed on small size instances for which the optimal solution has been computed by CPLEX solver, in a computational time ranging from one second to twenty seconds.

For all the implementations considered, we adopt as termination criteria the number of shakings/restarts that has been limited to $R_{\max} = 1\,000$ (with a maximum number of local search iterations per shaking/restart set to $\ell_{\max} = 200$). These two values guarantee to obtain stable results on most of the benchmark instances in a reasonable computational time. Moreover, the stopping criterium allows to evaluate the quality of each implementation neutralizing the possible effect of computing better solutions due to a larger number of shaking/restarts.

The diagrams reported in Figure 5 and Figure 6 summarize the tuning phase for the *VNS* and the *GRASP* metaheuristics respectively. Both the figures display the gap of the metaheuristic's solutions with respect to the corresponding optimal ones for each value of the local search parameter ρ (reported on the abscissa). The gap is the average value computed over all the 120 benchmark instances considered in this phase. Figure 5 displays the results of the *random* and the *memory* implementation of *VNS*. More specifically, the results of the random implementation are reported with a solid line while those of the memory implementation are displayed with a dashed line. By and large, the memory implementation provides better results than the random implementation, especially in correspondence to the optimal value of ρ that is equal to 0.98. In order to estimate whether the difference among the performance of these settings may be deemed significant, we apply the *Wilcoxon matched-pairs signed-ranks test* [21]. Referring to the tuning of the local search parameter ρ, the test suggests that the differences among the results obtained setting ρ in the range $[0.96; 0.98]$ are not statistically relevant, whereas the results obtained out of this range are significantly worse. As well, comparing the random and the memory implementation with $\rho = 0.98$, though the second one provides the best average gap, the difference proves not statistically relevant. In the following experiments, however, we will adopt the parameter setting yielding the best average performance, that is $\rho = 0.98$ and the memory-based tie-breaking rule.

Figure 6 reports the computational results of GRASP. We consider four implementations of the algorithm, obtained combining the *tie-breaking rule* and the *RCL* implementations, here listed: 1) cardinality-memory displayed with a dotted line; 2) cardinality-random represented with a dashed-dotted line; 3) value-memory represented with a solid line; 4) value-random represented with a dashed line. The diagrams reported in Figure 6 clearly show that the cardinality-

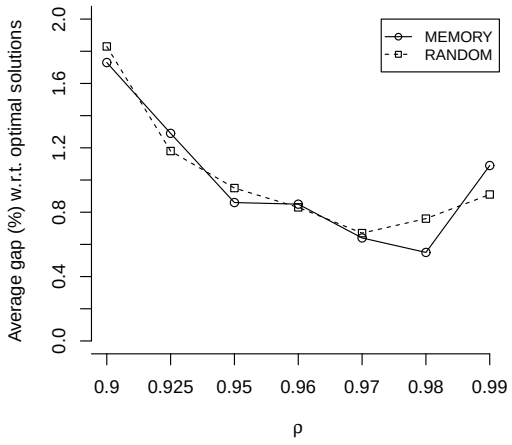

Figure 5: Comparison of the random and memory implementation of the VNS.

based *RCL* outperforms the value-based *RCL* for each of the two tie-breaking rules and for all the values of ρ tested. Moreover, the cardinality-based implementation of the *RCL* is more robust with respect to the variations of parameter ρ, meaning that the performances of this implementation of *GRASP* deteriorates less strongly by varying the parameter ρ. The dominance of the cardinality-based implementation over the value-based one is statistically significant according to Wilcoxon's test. In fact, the probability that the discrepancies observed between the two series of results obtained are due to a random fluctuation is smaller than $6 \cdot 10^{-23}$.

As for the implementation of the *tie-breaking rule*, the memory implementation is better than the random one and this difference is statistically significant, too. According to Wilcoxon's test, the probability that the discrepancies observed between the two series of results are due to a random fluctuation is smaller than 10^{-10}.

Finally, it is important to note the congruence of the optimal setting of the local search parameter ρ in the two tuning phases. The best performances of both the metaheuristics are obtained for $\rho = 0.98$, and the only alternative setting which is statistically not dominated is $\rho = 0.97$. In the following experiments, we will adopt the parameter setting yielding the best average performance, that is $\rho = 0.98$ and the memory-based tie-breaking rule.

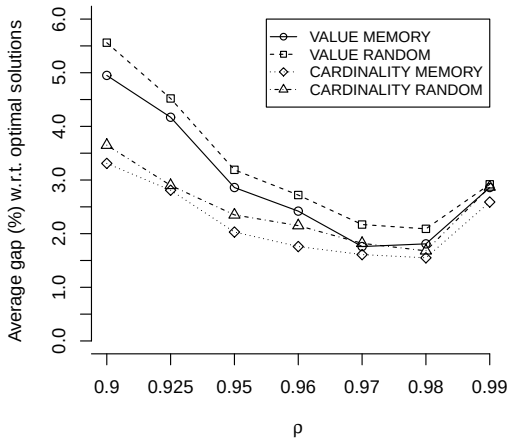

Figure 6: Comparison of the implementations of the GRASP.

4.2 Comparison

We here compare the computational performance of the *VNS* with the memory-based tie-breaking rule, the reactive *GRASP* with the cardinality-based *RCL* and the memory-based tie-breaking rule, and version 12.4 of the commercial solver CPLEX. Table 1 and Table 2 summarize the computational results obtained by the proposed algorithms within a time limit of 10 minutes. More specifically, the tables report on each row the average gap computed on a pool of 5 instances with the same features. The gap refers to the best known solution obtained with one of the three proposed methods; for the smaller instances, it refers to the optimal solution, computed by CPLEX. The first five columns of both tables report the settings of the instance parameters. In particular, the fifth column (A/U) distinguishes the assorted mode instances (denoted with A) from the uniform mode instances (denoted with U).

Table 1 reports the results for the weighted version of the *MM-SCP*. By and large, it appears that *VNS* provides the best computational performance, with an average gap computed on all the instances (last row of the table) that is equal to 0.60%. Moreover, *VNS* is more reliable than *GRASP* and CPLEX, meaning that its performance does not deteriorate strongly on any specific group of instances. Indeed, the largest average gap on all groups of instances is 2.65%. By contrast, *GRASP* and CPLEX show a larger variability across the group of instances: both hit worse results on the larger instances. This trend is particularly evident for *GRASP*. Finally, it should be noted that CPLEX computed the optimal solution on the medium size rectangular instances (gaps reported

$\mid I \mid$	$\mid J \mid$	$\mid M \mid$	b	A/U	CPLEX	*VNS*	GRASP
500	500	2	1	A	0.00%	1.06%	3.18%
500	500	2	1	U	0.31%	0.63%	2.79%
500	500	3	1	A	0.19%	0.75%	4.77%
500	500	3	1	U	0.00%	1.16%	4.04%
500	500	3	2	A	0.21%	0.90%	4.43%
500	500	3	2	U	0.00%	0.60%	4.68%
100	1000	2	1	A	**0.00%**	0.00%	0.83%
100	1000	2	1	U	**0.00%**	0.43%	3.04%
100	1000	3	1	A	**0.00%**	0.81%	5.08%
100	1000	3	1	U	**0.00%**	2.65%	7.01%
100	1000	3	2	A	**0.00%**	0.86%	4.57%
100	1000	3	2	U	**0.00%**	1.76%	7.46%
1000	1000	2	1	A	0.00%	0.17%	6.18%
1000	1000	2	1	U	0.68%	0.50%	8.34%
1000	1000	3	1	A	2.52%	0.00%	9.77%
1000	1000	3	1	U	0.77%	0.21%	8.60%
1000	1000	3	2	A	1.37%	0.54%	10.62%
1000	1000	3	2	U	0.79%	0.55%	10.91%
500	5000	2	1	A	1.45%	0.29%	16.57%
500	5000	2	1	U	3.83%	0.00%	16.41%
500	5000	3	1	A	1.75%	0.00%	21.10%
500	5000	3	1	U	3.12%	0.19%	21.01%
500	5000	3	2	A	1.75%	0.19%	22.05%
500	5000	3	2	U	2.93%	0.19%	21.36%
			TOTAL AVG		0.90%	0.60%	9.37%
			TOTAL VAR		0.027%	0.009%	0.470%

Table 1: Computational results on medium and large size instances for the weighted version of *MM-SCP*.

in bold in Table 1).

$\mid I \mid$	$\mid J \mid$	$\mid M \mid$	b	A/U	CPLEX	VNS	GRASP
500	500	2	1	A	3.59%	0.00%	0.30%
500	500	2	1	U	5.94%	0.30%	0.89%
500	500	3	1	A	4.90%	0.20%	0.00%
500	500	3	1	U	4.82%	0.19%	0.97%
500	500	3	2	A	4.56%	0.20%	0.80%
500	500	3	2	U	4.94%	0.78%	0.60%
100	1000	2	1	A	1.38%	0.00%	1.38%
100	1000	2	1	U	0.69%	0.00%	0.69%
100	1000	3	1	A	1.81%	0.00%	0.91%
100	1000	3	1	U	2.22%	0.43%	1.76%
100	1000	3	2	A	1.36%	0.45%	1.36%
100	1000	3	2	U	1.82%	0.00%	1.82%
1000	1000	2	1	A	3.93%	0.25%	0.73%
1000	1000	2	1	U	5.10%	0.49%	0.00%
1000	1000	3	1	A	4.85%	0.32%	0.16%
1000	1000	3	1	U	22.46%	0.16%	0.48%
1000	1000	3	2	A	3.40%	0.32%	0.16%
1000	1000	3	2	U	3.41%	0.97%	0.00%
500	5000	2	1	A	1.46%	0.36%	0.00%
500	5000	2	1	U	1.81%	0.36%	0.36%
500	5000	3	1	A	0.95%	0.00%	0.24%
500	5000	3	1	U	41.12%	0.00%	0.48%
500	5000	3	2	A	0.96%	0.72%	0.24%
500	5000	3	2	U	1.19%	0.00%	0.00%
			TOTAL AVG		5.36%	0.27%	0.60%
			TOTAL VAR		1.458%	0.003%	0.010%

Table 2: Computational results on medium and large size instances for the unweighted version of *MM-SCP*.

Table 2 reports the computational results for the unweighted version of the *MM-SCP*. The performance of CPLEX on this class of instances is poor. Indeed, CPLEX is unable to compute the best known solution for any group of instances and the largest average gap rises as high as 41.12%. *VNS* provides the best computational results also on this class of instances, hitting the lowest average gap (see last row of Table 2), with a value below 1% for each group of instances. The variance of the gap distribution is also the smallest one (0.003). On the unweighted instances, also *GRASP* provides good quality solutions, outperforming CPLEX on most of the groups of instances. Moreover, its computational performance is very competitive on the larger instances.

So far, in Table 1 and Table 2 we have focused on the comparison of the proposed algorithms with respect to weighted and unweighted instances of the

	CPLEX	VNS	GRASP
Rectangular	2.98%	0.41%	6.49%
Square	3.28%	0.47%	3.48%
Unweighted	5.36%	0.27%	0.60%
Weighted	0.90%	0.60%	9.37%
$b = 1, \mid M \mid = 2$	1.89%	0.30%	3.86%
$b = 1, \mid M \mid = 3$	5.72%	0.44%	5.40%
$b = 2, \mid M \mid = 3$	1.79%	0.57%	5.69%
Assorted modes	1.77%	0.35%	4.81%
Uniform modes	4.50%	0.52%	5.15%

Table 3: Summary of the computational results for specific features of the *MM-SCP* instances.

MM-SCP. In Table 3, we extend the comparison of the algorithms with respect to different features of the instances. The results of Table 3 strengthen our comments and observations reported above. *VNS* is quite stable in terms of computational results across all the groups of instances, showing a modest variability of the gap, whose values are in the range $[0.27\%; 0.60\%]$. By contrast, the computational performances of CPLEX, besides being strongly affected by the cost function, is also sensitive to the ratio $b/|M|$ and to the covering pattern (assorted versus uniform). Its sensitivity to the ratio $b/|M|$ is most likely due to the combinatorial structure of the problem: a tighter capacity constraint increases the number of variables with fractional values in correspondence to the optimal solution of the continuous relaxation, thus increasing the number of branch-and-bound nodes required to solve the instance. The sensitivity to the covering pattern might be justified by the intrinsic symmetry characterizing uniform mode instances. Finally, *GRASP*, besides being affected by the cost function, seems to be also sensitive to the shape of the instances, i.e., rectangular and square.

5 Conclusion

In this paper, we have presented the *Multi-Mode Set Covering Problem MM-SCP*. In addition to complexity and nonapproximability results, we have presented two metaheuristics: a *VNS* and a *GRASP* algorithm. The *VNS* algorithm provides the best computational performance and does not show a relevant downturn for any class of instances. Moreover, it does not exhibit statistically significant differences between the memory-based and the random-based implementation of the tie-breaking rule used in the local search procedure, and it is robust with respect to the tuning of the local search parameters. We believe that this has an important implication for the practical use of the algorithm. By contrast, *GRASP*, though it adopts a more complex and finer tuning phase,

has a computational behavior more strongly dependent on the class of instances considered. Both heuristics, anyway, outperform a time-limited run of an *ILP* solver.

References

[1] U. Aickelin, K. A. Dowsland (2000) Exploiting problem structure in a genetic algorithm approach to a nurse rostering problem. *Journal of Scheduling*, Vol. 3, Iss. 3, pp. 139-153.

[2] M. Ashouri, Z. Zali, S. R. Mousavi, M. R. Hashemi (2012) New optimal solution to disjoint set K-coverage for lifetime extension in wireless sensor networks. *IET Wirel. Sens. Syst.*, Vol. 2, Iss. 1, pp. 31-39.

[3] J. E. Beasley (1987) An algorithm for Set Covering problems. *European Journal of Operational Research*, Vol. 31, pp. 85–93,

[4] P. Berman, B. DasGupta, E. Sontag (2007) Randomized approximation algorithms for set multicover problems with applications to reverse engineering of protein and gene network. *Discrete Applied Mathematics*, Vol. 155, Iss. 6-7, pp. 733-749.

[5] K. Bicakci, I. E. Bagci, B. Tavli, Z. Pala (2013) Neighbor Sensor Networks: Increasing Lifetime and Eliminating Partitioning Through Cooperation. *Computer Standards & Interfaces*, Vol. 35, Iss. 4, pp 396 - 402.

[6] T. Chen, V. Filkov, S. S. Skiena (2001) Identifying gene regulatory networks from experimental data. *Parallel Computing*, Vol. 27, pp 141–162.

[7] V. Chvátal (1979) A greedy heuristic for the set covering problem. *Mathematics of Operations Research*, Vol. 4, pp. 233-235.

[8] R. Cordone, G. Lulli (2013) An integer optimization approach for reverse engineering of gene regulatory networks. *Discrete Applied Mathematics*, Vol. 61, Iss. 4- 5, pp. 580- 592.

[9] R. Cordone, G. Lulli (2012) A *GRASP* metaheuristic for microarray data analysis *Computers & Operations Research*, [in press: DOI: 10.1016/j.cor.2012.10.008]

[10] T. A. Feo, M. G. C. Resende (1989) A probabilistic heuristic for a computationally difficult set covering problem *Operations Research Letters*, Vol. 8, pp. 67-71.

[11] M. R. Garey, D. S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-completeness.* W. H. Freeman and Company, New York.

[12] I. P. Gent, T. Walsh, (1993) Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI'93)* pp. 28-33.

[13] P. Hansen, N. Mladenoviç, J. A. Moreno Pérez (2010) Variable Neighbourhood Search: Methods and Applications. *Annals of Operations Research*, Vol. 175, pp. 367–407.

[14] Q.-S. Hua, D. Yu, F. C. M. Lau, Y. Wang (2009) Exact Algorithms for Set Multicover and Multiset Multicover Problems. In *Proceedings of ISAAC Conference 2009*, Y. Dong, D.-Z. Du, and O. Ibarra (Eds.), LNCS 5878, pp. 34-44.

[15] D. S. Johnson (1974) Approximation Algorithms for Combinatorial Problems. *Journal of Computer and Systems Sciences*, Vol. 9, pp. 256-278.

[16] G. Lan, G. W. DePuy, G. E. Whitehouse (2007) An effective and simple heuristic for the set covering problem. *European Journal of Operational Research*, Vol. 176, pp. 1387-1403.

[17] G. Lulli, M. Romauch (2009) A Mathematical Program to Refine Gene Regulatory Networks. *Discrete Applied Mathematics*, Vol. 157, pp. 2469–2482.

[18] M. Prais, C. C. Ribeiro (2000) Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, Vol. 12, Iss. 3, pp. 164-176

[19] A. Srinivasan (2006) An Extension of the Lovász Local Lemma, and its Applications to Integer Programming. *SIAM Journal on Computing*, Vol. 36, Iss. 3, pp. 609-634.

[20] T. Villa, T. Kam, R. K. Brayton, A. L. Sangiovanni-Vincentelli (1997) Explicit and implicit algorithms for binate covering problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, Iss. 7, pp. 677-691.

[21] F. Wilcoxon (1945) Individual Comparisons by Ranking Methods. *Biometrics*, Vol. 1, pp. 80–83.

[22] M. Yagiura, M. Kishida, T. Ibaraki (2006) A 3-Flip Neighborhood Local Search for the Set Covering Problem. *European Journal of Operational Research*, Vol. 172, Iss. 2, pp. 472-499.