A HYBRID LAGRANGEAN HEURISTIC WITH GRASP AND PATH-RELINKING FOR SET K-COVERING

LUCIANA S. PESSOA, MAURICIO G. C. RESENDE, AND CELSO C. RIBEIRO

Abstract. The set multicovering or set k-covering problem is an extension of the classical set covering problem, in which each object is required to be covered at least k times. The problem finds applications in the design of communication networks and in computational biology. We describe a GRASP with path-relinking heuristic for the set k-covering problem, as well as the template of a family of Lagrangean heuristics. The hybrid GRASP Lagrangean heuristic employs the GRASP with path-relinking heuristic using modified costs to obtain approximate solutions for the original problem. Computational experiments carried out on 135 test instances show experimentally that the Lagrangean heuristics performed consistently better than GRASP. By properly tuning the parameters of the GRASP Lagrangean heuristic, it is possible to obtain a good trade-off between solution quality and running times. Furthermore, the GRASP Lagrangean heuristic makes better use of the dual information provided by subgradient optimization and is able to discover better solutions and to escape from locally optimal solutions even after the stabilization of the lower bounds, when other strategies fail to find new improving solutions.

1. Introduction

Given a set $I = \{1, \ldots, m\}$ of objects, let $\{P_1, \ldots, P_n\}$ be a collection of subsets of I, with a non-negative cost c_j associated with each subset P_j, for $j = 1, \ldots, n$. A subset $\hat{J} \subseteq J = \{1, \ldots, n\}$ is a *cover* of I if $\cup_{j \in \hat{J}} P_j = I$. The cost of a cover $\hat{J}$ is $\sum_{j \in \hat{J}} c_j$. The *set covering problem* consists of finding a minimum cost cover J^*.

The *set multi-covering problem* is a generalization of the set covering problem, in which each object $i \in I$ must be covered by at least $\ell_i \in \mathbb{Z}_+$ elements of $\{P_1, \ldots, P_n\}$. A special case of the set multi-covering problem arises when $\ell_i = k$, for all $i \in I$. Following Vazirani (2004), we refer to this problem as the *set k-covering problem* (SC$_k$P).

Let the $m \times n$ binary matrix $A = [a_{ij}]$ be such that for all $i \in I$ and $j \in J$, $a_{ij} = 1$ if and only if $i \in P_j$; $a_{ij} = 0$, otherwise. Let a solution $\hat{J}$ of SC$_k$P be represented by a binary n-vector x, where $x_j = 1$ if and only if $j \in \hat{J}$. An integer

Date: December 1st, 2010.
Key words and phrases. GRASP, hybrid heuristics, metaheuristics, path-relinking, Lagrangean relaxation, Lagrangean heuristics, local search, set covering, set multicovering, set k-covering.

programming formulation for the set k-covering problem is

$$z(x) = \min \sum_{j=1}^{n} c_j x_j \tag{1}$$

s.t.

$$\sum_{j=1}^{n} a_{ij} x_j \geq k, \quad i = 1, \ldots, m, \tag{2}$$

$$x_j \in \{0, 1\}, \quad j = 1, \ldots, n. \tag{3}$$

Lucena (2009) noticed that few branch-and-cut algorithms exist for set covering problems and its variants, because the separation problem associated with some families of strong inequalities (Avella et al. (2009); Balas and Ng (1989a;b)) are very hard to be solved even for heuristics. Therefore, the use of heuristics for large problems with large duality gaps is a need.

In this paper, we propose a template of Lagrangean heuristics for the set k-covering problem, based on the hybridization of subgradient algorithms to solve its Lagrangean relaxation with greedy and GRASP heuristics. Applications of the set k-covering problem and related work are reviewed in the next section. A GRASP with path-relinking heuristic for the set k-covering problem is customized in Section 3. A template for Lagrangean heuristics for SC$_k$P based on basic constructive heuristics and subgradient optimization is proposed in Section 4. Different implementation strategies for the basic constructive heuristics and a hybridization of GRASP with a Lagrangean heuristic are discussed in Section 5. Computational results are reported in Section 6. Concluding remarks are made in Section 7.

2. Applications and related work

Applications of the set multicovering problem arise in a variety of fields, such as marketing, logistics, security, telecommunications, and computational biology. Though some of these applications can be modeled as set covering problems, for reliability purposes they are treated as multicovering problems. Some applications are described in Hall and Hochbaum (1992). We describe here two applications that served as motivations for this paper.

In the context of computational biology, k-covers have an important application in the *minimum robust tagging SNP problem* (Bafna et al., 2003). Single nucleotide polymorphisms (SNPs) are the most abundant form of genetic variation in the human genome. According to Brookes (1999), SNPs are single base pair positions in DNA at which different sequence alternatives (alleles) exist in the individuals of a population. SNPs can be bi-, tri-, or tetra-allelic, assuming one of two, three, or four variants among {A,C,T,G}. Tri-allelic and tetra-allelic SNPs are very rare in humans. Figure 1 (a) shows five SNP sites in four chromosomes. The first SNP takes the allele C for chromosomes 1, 2, and 4, and the allele A for chromosome 3.

The specific set of alleles at nearby SNPs on a single chromosome is called a haplotype (Brown, 2002; The International HapMap Consortium, 2003). Figure 1 (b) shows the haplotypes relative to the chromosomes in Figure 1 (a). Due to the proximity of the SNPs in a haplotype, they are inherited together through the generations and this causes a strong association among them. In this way, a small subset of these SNPs, called *tag SNPs*, can provide sufficient information about the remaining SNPs (The International HapMap Consortium, 2003).

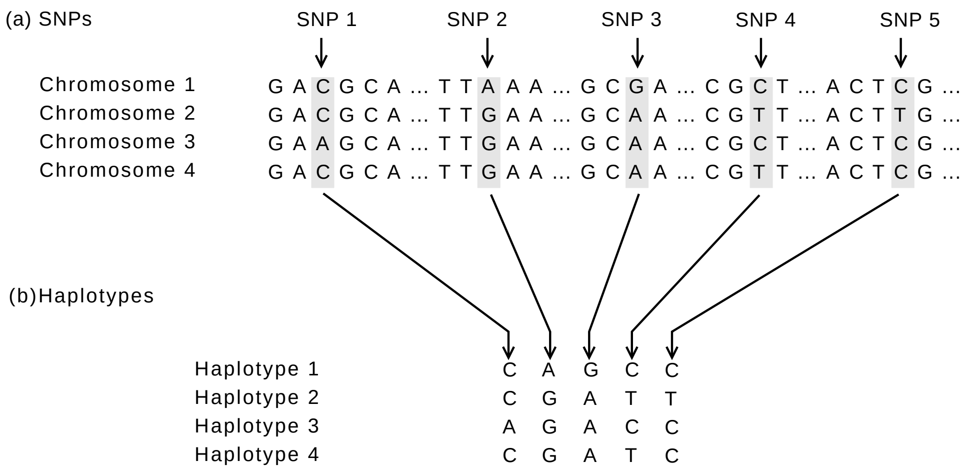
(a) SNPs
SNP 1
SNP 2
SNP 3
SNP 4
SNP 5
Chromosome 1 G A C G C A ... T T A A A ... G C G A ... C G C T ... A C T C G ...
Chromosome 2 G A C G C A ... T T G A A ... G C A A ... C G T T ... A C T T G ...
Chromosome 3 G A A G C A ... T T G A A ... G C A A ... C G C T ... A C T C G ...
Chromosome 4 G A C G C A ... T T G A A ... G C A A ... C G T T ... A C T C G ...
(b)Haplotypes
Haplotype 1 C A G C C
Haplotype 2 C G A T T
Haplotype 3 A G A C C
Haplotype 4 C G A T C

FIGURE 1. SNPs and haplotypes.

Given a group of p haplotypes with n SNPs each, the *minimum tagging SNP problem* (MTS) consists in finding the smallest subset of SNPs capable of distinguishing each haplotype from the others. However, it may be the case that some tag SNPs are missing in one or more haplotypes. In this situation, there exists a subgroup of SNPs, called *robust tag SNPs*, which is able to distinguish each pair of haplotypes unambiguously when at most a given number of SNPs is missing. This problem was formulated by Huang et al. (2005) as a set k-covering problem and was shown to be NP-hard. To find robust tag SNPs, they proposed two greedy algorithms, an exhaustive enumeration algorithm, and a linear programming relaxation. Later, Chang et al. (2006) developed a hybrid method that combines the ideas of a branch-and-bound method and one of the greedy algorithms of Huang et al. (2005). Pessoa and Ribeiro (2007) proposed a GRASP heuristic and reported results on simulated and biological datasets.

Another application of the set multicovering problem arises in telecommunications (Resende, 2007). Suppose customers are serviced by equipment placed in points-of-presence (PoPs). For example, a PoP could host a modem pool to which a customer dials up for Internet access, or it could host an antenna which connects the customer to the network. In the *PoP placement problem*, we are given a set of customers, a set of potential PoP locations, and the set of PoPs that can provide service to each customer. We wish to determine in which PoPs to place the equipment such that each customer can be serviced by at least one PoP. Since PoPs may have different costs associated with them, we wish to select the least-cost set of PoPs. Clearly, if a customer is covered by exactly one PoP and that PoP fails, the customer will lose service. To improve the reliability of the service, we may want to require that each customer be covered by at least k PoPs. This problem is also known as the *redundant PoP placement problem*.

Hall and Hochbaum (1983; 1992) developed and tested ten primal heuristics for the set multicovering problem. They used these heuristics as well as Lagrangean relaxation in a branch-and-bound algorithm. Computational experiments on instances with up to 200 variables show that solutions within 0.5% of the optimal value were found. Gonsalvez et al. (1987) examined these and other primal heuristics to construct confidence intervals for the unknown optimal values.

3. GRASP WITH PATH-RELINKING

GRASP is short for greedy randomized adaptive search procedures. It was introduced by Feo and Resende (1989) for solving a set covering problem with unit costs. GRASP is a multi-start metaheuristic which consists of applying local search to feasible starting solutions generated with a greedy randomized construction heuristic. Tutorials on GRASP can be found, for example, in Feo and Resende (1995), Resende and Ribeiro (2003), Resende (2008), and Resende and Ribeiro (2010). Annotated bibliographies of GRASP are presented by Festa and Resende (2002; 2009a;b).

Path-relinking (Glover, 1996) is an intensification scheme that explores paths in the solution space connecting good-quality solutions. Memory structures may be introduced in GRASP through its hybridization with path-relinking (Laguna and Martí, 1999; Resende and Ribeiro, 2005; 2010; Resende et al., 2010).

In this section, we specialize GRASP and path-relinking into a heuristic for the set k-covering problem.

3.1. **Construction phase.**

A greedy algorithm for set k-covering builds a solution from scratch, adding one of the sets $P_1, \dots, P_n$ at a time to a partial solution, until all objects are k-covered, i.e. each object is covered by at least k sets. Given a partial solution, at each step of the construction, let the *covering cardinality* τ_j be the number of objects not yet k-covered by the partial solution that become covered if P_j is introduced in partial solution. A candidate list L is formed by the indices of all sets P_j not in the partial solution for which $\tau_j > 0$. Each set P_j, with $j \in L$, is evaluated according to a greedy function defined as the ratio $\rho_j = c_j/\tau_j$ between its cost and its covering cardinality. The greedy algorithm adds to the partial solution a minimum ratio candidate set.

Algorithm 1 shows the pseudo-code of the randomized variant of the above greedy algorithm, which is used to construct the initial solutions for the GRASP heuristic. The solution x and the candidate list L are initialized in lines 2 and 3, respectively. The covering cardinality and the greedy function value are computed in line 4 for all candidate elements. The loop in lines 5 to 14 adds one set at a time to the cover, until all objects are k-covered. The minimum (ρ^-) and maximum (ρ^+) greedy function values of the candidate elements are computed in lines 6 and 7, respectively. The restricted candidate list (RCL), formed by all candidate elements whose greedy function value is less than or equal to $\rho^- + \alpha(\rho^+ - \rho^-)$, is built in line 8, where α is a real-valued parameter in the interval $[0, 1]$. An element e is chosen at random from the RCL in line 9 and the set P_e is added to the solution in line 10. The covering cardinalities are recomputed in line 11 to account for the inclusion of set e in the solution. The candidate list is updated in line 12 by removing set P_e and all those sets having null covering cardinalities. Finally, in line 13, the greedy function value is updated for all candidate sets.

```
1  GreedyRandomizedConstruction
2  $x_j \leftarrow 0$, for $j = 1, \dots, n$;
3  $L \leftarrow \{1, \dots, n\}$;
4  Compute $\tau_j$ and $\rho_j$, for $j = 1, \dots, n$;
5  while there exists some object that is not $k$-covered do
6      $\rho^- \leftarrow \min\{\rho_j \ : \ j \in L\}$;
7      $\rho^+ \leftarrow \max\{\rho_j \ : \ j \in L\}$;
8      RCL $\leftarrow \{j \in L \ : \ \rho_j \leq \rho^- + \alpha(\rho^+ - \rho^-) \}$;
9      Select, at random, an element $e$ from the RCL;
10     $x_e \leftarrow 1$;
11     Recompute $\tau_j$, $\forall j \in L : j \neq e$;
12     $L \leftarrow L \setminus (\{e\} \cup \{j \in L : \tau_j = 0\})$;
13     Recompute $\rho_j$, $\forall j \in L$
14 end
```

Algorithm 1: Greedy randomized construction procedure.

3.2. **Local search.** Solutions built with the randomized greedy algorithm are not guaranteed to be locally optimal, even with respect to simple neighborhood structures. Therefore, the application of local search to such a solution usually results in an improved locally optimal solution. We next describe a local search procedure for the set k-covering problem.

Starting from an initial solution, local search explores its neighborhood for a cost-improving solution. If none is found, then the search returns the initial solution as a local minimum. Otherwise, if an improving solution is found, it is made the new initial solution, and the procedure repeats itself.

The local search proposed in this paper makes use of two simple neighborhoods. The first neighborhood is a $(1, 0)$-exchange in which we attempt to remove superfluous sets from the multicover. The second neighborhood is a $(1, 1)$-exchange in which we attempt to replace a more expensive set in the multicover by a less expensive unused one.

The local search procedure is illustrated by the pseudo-code in Algorithm 2. In the following we refer to sets in the multicover by their indices in the original set J. The loop in lines 2 to 23 is repeated while a locally optimal solution is not found. In line 3, all sets in the multicover are made candidates to leave the solution and their indices are placed in S. The loop in lines 4 to 22 attempts to remove each set in S, examining them in decreasing order of their costs. The next candidate j^+ for removal is determined in line 5 and the corresponding variable x_{j^+} is tentatively set to 0 in line 6. If the new solution obtained is feasible, then j^+ is removed from S in line 21 and a new set will be tested for removal from the cover. Otherwise, if the test in line 7 determines that the new solution is infeasible, then we build in line 8 a set $\bar{S}$ of candidates to replace j^+ in the cover. In line 9, we select the least-cost candidate j^- from $\bar{S}$. The loop in lines 10 to 17 examines all profitable elements in $\bar{S}$ in an attempt to make a feasible cost-improving exchange. In line 11, we tentatively insert the set j^- into the solution and test in line 12 if the resulting solution is feasible. If this is not the case, we undo the tentative insertion in line 13 and remove set j^- from $\bar{S}$ in line 14. If there are still candidates available for insertion in set $\bar{S}$, then in line 16, we determine the least-cost candidate as the next one to be examined for insertion. After there are no more cost-improving candidates in $\bar{S}$, we test in line 18 if a feasible solution was obtained. If this is true, then j^- is inserted in S; otherwise, we undo the assignment made in line 6.

```
1  LocalSearch
2  while $x$ is not locally optimal do
3      Initialize the solution index set:
       $S \leftarrow \{j = 1, \ldots, n \ : \ x_j = 1\}$;
4      while $S \neq \emptyset$ do
5          $j^+ \leftarrow \texttt{argmax}\{c_j \ : \ j \in S\}$;
6          $x_{j^+} \leftarrow 0$;
7          if $x$ is not feasible then
8              $\bar{S} \leftarrow \{j = 1, \ldots, n \ : \ x_j = 0$ and $j \neq j^+\}$;
9              $j^- \leftarrow \texttt{argmin}\{c_j \ : \ j \in \bar{S}\}$;
10             while $\bar{S} \neq \emptyset$ and $x$ is not feasible and $c_{j^-} < c_{j^+}$ do
11                 $x_{j^-} \leftarrow 1$;
12                 if $x$ is not feasible then
13                     $x_{j^-} \leftarrow 0$;
14                     $\bar{S} \leftarrow \bar{S} \setminus \{j^-\}$;
15                 end
16                 if $\bar{S} \neq \emptyset$ then $j^- \leftarrow \texttt{argmin}\{c_j \ : \ j \in \bar{S}\}$;
17             end
18             if $x$ is feasible then $S \leftarrow S \cup \{j^-\}$ else
19             $x_{j^+} \leftarrow 1$;
20         end
21         $S \leftarrow S \setminus \{j^+\}$;
22     end
23 end
```

Algorithm 2: Local search procedure.

3.3. **Path-relinking.** The basic implementation of GRASP is memoryless, since computations in a GRASP iteration do not make use of information collected in previous iterations. Path-relinking is an intensification strategy that can be applied to introduce memory structures in GRASP (Resende and Ribeiro, 2005; 2010). Path-relinking explores paths in the solution space connecting good-quality solutions. The procedure maintains a pool P formed by a limited number of elite solutions (i.e., a diverse set of good-quality solutions found during the search). Path-relinking is carried out between each solution x obtained by local search and a local minimum x^p, randomly selected from the pool. Depending on the strategy that will be used by the path-relinking procedure, one of x or x^p will be considered as the initial solution x^s and the other will be the target solution x^t.

Algorithm 3 describes the path-relinking procedure for the set k-covering problem, where x^s is the binary vector representing an initial solution obtained after the

```
1  PathRelinking
2  $\Delta \leftarrow \{j = 1, \ldots, n \ : \ x^s_j \neq x^t_j\}$;
3  $x^* \leftarrow \mathrm{argmin}\{z(x^s), z(x^t)\}$;
4  $z^* \leftarrow \min\{z(x^s), z(x^t)\}$;
5  $y \leftarrow x^s$;
6  while $|\Delta| > 1$ do
7      $\ell^* \leftarrow \mathrm{argmin}\{z(y \oplus \ell) \ : \ \ell \in \Delta$ and $(y \oplus \ell)$ is feasible$\}$;
8      $\Delta \leftarrow \Delta \setminus \{\ell^*\}$;
9      $y_{\ell^*} \leftarrow 1 - y_{\ell^*}$;
10     if $z(y) < z^*$ then
11         $x^* \leftarrow y$;
12         $z^* \leftarrow z(y)$;
13     end
14 end
```

Algorithm 3: Path-relinking procedure.

```
1  GRASP+PR
2  Initialize elite set $P \leftarrow \emptyset$;
3  Initialize best solution value $z^* \leftarrow \infty$;
4  for $i = 1, \ldots, N$ do
5      $x \leftarrow$ GreedyRandomizedConstruction();
6      $x \leftarrow$ LocalSearch($x$);
7      if $i = 1$ then insert $x$ into the elite set $P$;
8      else
9          Choose, at random, a pool solution $x^p \in P$;
10         Determine which solution (between $x$ and $x^p$) is the
           initial solution $x^s$ and the target solution $x^t$;
11         $x \leftarrow$ PathRelinking($x^s, x^t$);
12         $x \leftarrow$ LocalSearch($x$);
13         Update the elite set $P$ with $x$;
14     end
15     if $z(x) < z^*$ then
16         $x^* \leftarrow x$;
17         $z^* \leftarrow z(x)$;
18     end
19 end
```

Algorithm 4: GRASP with path-relinking procedure.

local search phase and x^t is the binary vector representing a target solution. The set $\Delta = \{j = 1, \ldots, n \ : \ x^s_j \neq x^t_j\}$ of positions in which x^s and x^t differ is computed in line 2. The best solution x^* among x^t and x^s and its cost $z(x^*)$ are determined in lines 3 and 4, respectively. The current path-relinking solution y is initialized to x^s is line 5. The loop in lines 6 to 14 progressively determines the next solution in the path connecting x^s and x^t until the entire path is traversed. For every position $\ell \in \Delta$, we define $y \oplus \ell$ to be the solution obtained from y by complementing the current value of y_ℓ. Line 7 determines the component ℓ^* of Δ for which $y \oplus \ell$ results in the least-cost feasible solution. This component is removed from Δ in line 8 and the current solution is updated in line 9 by complementing the value of its ℓ^*-th position. If the test in line 10 detects that the new current solution y improves the best solution x^* in the path, then the latter and its cost are updated in lines 11 and 12, respectively.

To see that there always exists a path connecting x^s and x^t, observe that by first setting to 1 all components of x^s that are equal to 0 in x^s and to 1 in x^t will result in a series of feasible multicovers leading from x^s to some feasible solution y. Next, by removing each of the superfluous components of y (i.e. setting to 0 the components equal to 1 in y and to 0 in x^t) will result again in a series of feasible multicovers leading from y to x^t.

Algorithm 4 shows the pseudo-code for the complete GRASP with path-relinking procedure. Lines 2 and 3 initialize the elite set P and the value z^* of the best known solution. The loop from line 4 to 19 corresponds to the GRASP with path-relinking iterations. At each iteration, an initial solution is built by the greedy randomized procedure in line 5. A locally optimal solution x with respect to (1,0)- and (1,1)-exchanges is computed by local search in line 6. The elite set P is initialized in line 7 with the local optimum x obtained in the first iteration. For all other iterations, lines 8 to 14 perform the application of path-relinking and the elite set management.

A pool solution x^p is chosen, at random, from the elite set in line 9. To favor longer paths, x^p is chosen with probability proportional to its Hamming distance to the current solution x, i.e. $|\{j = 1, \ldots, n \ : \ x^p_j \neq x_j\}|$. We do not consider a pool solution if its Hamming distance with respect to x is less than four, since any path between them cannot contain solutions simultaneously better than both of them. Line 10 determines whether x or x^p is the starting solution x^s. The other one is defined as the target solution, x^t. Path-relinking is applied to the pair x^s and x^t of solutions in line 11 resulting in a solution x, which is reoptimized by local search in line 12. The elite set P is update in line 13. If the pool is not full and the new solution is different from all others in the pool, then it is automatically inserted in the elite set. Otherwise, if the new solution x is better than the worst solution in the elite set, it replaces the highest cost solution in the pool. If x does not improve upon the worst solution in the elite set, then it is discarded. The best solution x^* and its cost z^* are updated in lines 15 to 18.

The attribution of x and x^p to the initial solution x^s or to the target solution x^t depends on the path-relinking strategy. Different approaches have been considered in the implementation of this procedure (Resende and Ribeiro, 2005; 2010; Resende et al., 2010). In this paper we considered three strategies:

- Forward: when the initial solution is the highest cost solution between x^s and x^t.
- Backward: when the initial solution is the lowest cost solution between x^s and x_t.
- Mixed: when two paths are simultaneously explored by interchanging the roles of initial and target solution after each move. In this case, the attribution of either x or x^p to x^s or to x^t is indifferent.

4. A template for Lagrangean heuristics

Lagrangean relaxation (Beasley, 1993; Fisher, 2004) is a mathematical programming technique that can be used to provide lower bounds for combinatorial optimization problems. However, the primal solutions produced by the algorithms used to solve the Lagrangean dual problem are not necessarily feasible. Held and Karp (1970; 1971) were among the first to explore the use of the dual multipliers produced by Lagrangean relaxation to derive lower bounds, applying this idea in the context of the traveling salesman problem.

Lagrangean heuristics exploit the dual multipliers to generate primal feasible solutions. Beasley (1987; 1990b) described a Lagrangean heuristic for set covering which can be extended to the set k-covering problem.

A Lagrangean relaxation of the set k-covering problem can be defined by associating dual multipliers $\lambda_i \in \mathbb{R}_+$, for $i = 1, \ldots, m$, to each inequality (2). This results in the following *Lagrangean relaxation problem* LRP(λ):

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} \lambda_i (k - \sum_{j=1}^{n} a_{ij} x_j) \\ &\text{s.t.} \\ &\quad x_j \in \{0,1\}, \quad j = 1, \ldots, n. \end{aligned}$$

By letting $c'_j = c_j - \sum_{i=1}^{m} \lambda_i a_{ij}$, formulation LRP($\lambda$) simplifies to

$$\begin{aligned} z'(\lambda) = \min \; &\sum_{j=1}^{n} c'_j x_j + \sum_{i=1}^{m} \lambda_i k \\ \text{s.t.} \\ &x_j \in \{0,1\}, \quad j = 1, \ldots, n, \end{aligned}$$

whose optimal solution $x'(\lambda)$ is given by

$$x'_j(\lambda) = \begin{cases} 1, & \text{if } c'_j \le 0 \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

for $j = 1, \ldots, n$, where the objective function value given by

$$z'(\lambda) = \sum_{j=1}^{n} c'_j x'_j(\lambda) + k \sum_{i=1}^{m} \lambda_i$$

is a lower bound to the optimal value of the original problem (1)–(3). The best lower bound $z'(\lambda^*)$ is the solution of the *Lagrangean dual problem* LDP:

$$z_D = \max_{\lambda \in \mathbb{R}_+^m} z'(\lambda). \tag{5}$$

Subgradient optimization may be used to solve (5). Subgradient algorithms may start from any feasible set of dual multipliers, such as $\lambda_i = 0$, for $i = 1, \ldots, m$, and iteratively generate further multipliers. We use the same strategy described in Held et al. (1974) for updating the dual multipliers from one iteration to the next.

At any iteration q, let λ^q be the current vector of multipliers and let $x'(\lambda^q)$ be an optimal solution to problem $LRP(\lambda^q)$, whose optimal value is $z'(\lambda^q)$. Furthermore, let $\bar{z}$ be a known upper bound to the optimal value of problem (1)–(3). Additionally, let $g^q \in \mathbb{R}^m$ be a subgradient of $z'(\lambda)$ for $\lambda = \lambda^q$, with

$$g_i^q = k - \sum_{j=1}^{n} a_{ij} x'_j(\lambda^q), \quad i = 1, 2, \ldots, m. \tag{6}$$

To update the Lagrangean multipliers, the algorithm makes use of a step size

$$d^q = \frac{\eta\,(\bar{z} - z'(\lambda^q))}{\sum_{i=1}^{m} (g_i^q)^2}, \tag{7}$$

where $\eta \in (0, 2]$. Multipliers are then updated according to

$$\lambda_i^{q+1} = \max\{0; \lambda_i^q + d^q g_i^q\}, \quad i = 1, \ldots, m, \tag{8}$$

and the subgradient algorithm proceeds to iteration $q + 1$.

Beasley (1990b) reports as computationally useful to adjust the components of the subgradients to zero whenever they do not effectively contribute to the update of the multipliers, i.e. arbitrarily set $g_i^q = 0$ whenever $g_i^q > 0$ and $\lambda_i^q = 0$, for $i = 1, \ldots, m$.

The Lagrangean heuristic proposed in this section makes use of the dual multipliers λ^q and of the optimal solution $x'(\lambda^q)$ to each problem $LRP(\lambda^q)$ to build feasible solutions to the original problem (1)–(3). To do this, let $\mathcal{H}$ be a heuristic that builds a feasible solution x from an initial solution x^0. Two approaches are considered to define x^0: Beasley (1990b) sets $x^0 = x(\lambda^q)$, while Caprara et al. (1999) simply initialize $x_j^0 = 0$, for $j = 1, \ldots, n$. In other words, the first approach repairs the initial solution $x'(\lambda^q)$ to make it feasible, while the second builds a feasible solution from scratch.

Heuristic $\mathcal{H}$ is initially applied from scratch using the original cost vector c. In any subsequent iteration q of the subgradient algorithm, $\mathcal{H}$ either uses Lagrangean reduced costs $c'_j = c_j - \sum_{i=1}^{m} \lambda_i^q a_{ij}$ or complementary costs $\bar{c}_j = (1 - x'_j(\lambda^q))c_j$. Let $x^{\mathcal{H},\gamma}$ be the solution obtained by $\mathcal{H}$, using a generic cost vector γ corresponding to either one of the above modified cost schemes or to the original cost vector. Its cost is given by $\sum_{j=1}^{n} c_j x_j^{\mathcal{H},\gamma}$ and may be used to update the upper bound $\bar{z}$ to the optimal value of the original problem (1)–(3). This upper bound may be further improved by local search and is used to adjust the step size in (7).

Algorithm 5 describes the pseudo-code of the Lagrangean heuristic. Lines 2 to 4 initialize the upper and lower bounds, the iteration counter, and the dual multipliers. The iterations of the subgradient algorithm are performed along the loop in lines 5 to 22. The reduced costs are computed in line 6 and the Lagrangean relaxation problem is solved by inspection in line 7. In the first iteration of the Lagrangean heuristic, the original cost vector is assigned to γ in line 8, while in subsequent iterations a modified cost vector is assigned in line 9. Lines 10 to 16 determine that a basic heuristic is used to produce a primal feasible solution to problem (1)–(3) whenever the iteration counter q is a multiple of an input parameter H. A heuristic $\mathcal{H}$ is applied in line 11 to produce the feasible solution $x^{\mathcal{H},\gamma}$. If the cost of this solution is lower than the current upper bound, the best solution so far and its cost are updated in lines 13 and 14, respectively. If the lower bound $z'(\lambda^q)$ computed in iteration q is greater than the best lower bound z_D, then in line 17 the lower bound z_D is updated. Line 18 computes the subgradient and line 19 computes

```
1  **LagrangeanHeuristic**
2  Initialize bounds: $\bar{z} \leftarrow \sum_{j=1}^{n} c_j$ and $z_D \leftarrow 0$;
3  Initialize iteration counter: $q \leftarrow 0$;
4  Initialize dual multipliers: $\lambda_i^q \leftarrow 0,\ i = 1, \ldots, m$;
5  **repeat**
6      Compute reduced costs $c'_j \leftarrow c_j - \sum_{i=1}^{m} \lambda_i^q a_{ij},\ j = 1, \ldots, n$;
7      Solve $LRP(\lambda^q)$ by inspection to obtain $x'(\lambda^q)$;
8      **if** $q = 0$ **then** set $\gamma \leftarrow c$;
9      **else** set $\gamma$ to the modified cost vector;
10     **if** *q is a multiple of H* **then**
11         Apply a basic heuristic $\mathcal{H}$ with cost vector $\gamma$ to obtain $x^{\mathcal{H},\gamma}$;
12         **if** $\sum_{j=1}^{n} c_j x_j^{\mathcal{H},\gamma} < \bar{z}$ **then**
13             $x^* \leftarrow x^{\mathcal{H},\gamma}$;
14             $\bar{z} \leftarrow \sum_{j=1}^{n} c_j x_j^{\mathcal{H},\gamma}$;
15         **end**
16     **end**
17     **if** $z'(\lambda^q) > z_D$ **then** $z_D \leftarrow z'(\lambda^q)$;
18     Compute subgradient: $g_i^q = k - \sum_{j=1}^{n} a_{ij} x'_j(\lambda^q),\ \ i = 1, 2, \ldots, m$;
19     Compute step size: $d^q \leftarrow \eta\,(\bar{z} - z'(\lambda^q)) / \sum_{i=1}^{m} (g_i^q)^2$;
20     Update dual multipliers: $\lambda_i^{q+1} \leftarrow \max\{0, \lambda_i^q - d^q g_i^q\}, i = 1, \ldots, m$;
21     Increment iteration counters: $q \leftarrow q + 1$;
22 **until** *stopping criterion satisfied* ;
```

Algorithm 5: Pseudo-code of the template for a Lagrangean heuristic.

the step size. The dual multipliers are updated in line 20 and the iteration counter is incremented in line 21.

Different choices for the initial solution x^0 and for the modified costs γ, as well as for the heuristic $\mathcal{H}$ itself, lead to different Lagrangean heuristics.

5. Basic heuristics and Lagrangean GRASP

Different implementation strategies of the heuristic $\mathcal{H}$ in the template of Algorithm 5 lead to distinct Lagrangean heuristics. We considered two variants: the first makes use of a greedy algorithm (as presented in Section 3.1) with local search (as presented in Section 3.2), while the second is a GRASP with path-relinking (as presented in Section 3.3).

5.1. **Greedy heuristic.** This heuristic either builds a feasible solution x from scratch, or repairs the solution $x'(\lambda^q)$ produced in line 7 of the Lagrangean heuristic described in Algorithm 5 to make it feasible for problem (1)–(3). It corresponds to the greedy randomized construction described in Algorithm 1 using parameter $\alpha = 0$ and modified costs (c' or $\tilde{c}$). The local search described in Algorithm 2 is applied to the resulting solution, using the original cost vector c. We shall refer to the Lagrangean heuristic that uses the greedy heuristic as the *greedy Lagrangean heuristic* or simply GLH.

5.2. **GRASP heuristic.** Instead of simply performing one construction step followed by local search as in the greedy heuristic, this variant applies the GRASP

Table 1. Characteristics of the test problems: for each class, the table lists its name, dimension (rows $\times$ columns), density, and the number of instances making up the class.

Classes	Dimension	Density	Number of instances
scp4	200×1000	2%	10
scp5	200×2000	2%	10
scp6	200×1000	5%	5
scpa	300×3000	2%	5
scpb	300×3000	5%	5
scpc	400×4000	2%	5
scpd	400×4000	5%	5

with path-relinking heuristic of Algorithm 4 either to build a feasible solution x from scratch, or to repair the solution $x'(\lambda^q)$ produced in line 7 of the Lagrangean heuristic described in Algorithm 5 to make it feasible for problem (1)–(3). We shall refer to the Lagrangean heuristic that uses the GRASP heuristic as the *GRASP Lagrangean heuristic* or simply LAGRASP.

Although the GRASP heuristic produces better solutions than the greedy heuristic, the latter is much faster. To appropriately address this trade-off, we choose in line 11 of Algorithm 5 to use the GRASP heuristic with probability β and the greedy heuristic with probability $1 - \beta$, where β is a parameter of the algorithm.

We note that this strategy involves three main parameters: the number H of iterations after which the basic heuristic is always applied, the number Q of iterations performed by the GRASP with path-relinking heuristic when it is chosen as the basic heuristic, and the probability β of choosing the GRASP heuristic as $\mathcal{H}$. We shall refer to the Lagrangean heuristic that uses this hybrid strategy as LAGRASP(β, H, Q).

To implement path-relinking in the GRASP basic heuristic, LAGRASP maintains a global pool P which is empty at the start of the subgradient method. Each solution obtained in the GRASP local search phase is a candidate to be inserted in the elite set, following the pool management policy presented in Section 3.3.

6. Computational experiments

The computational experiments were performed on a 2.33 GHz Intel Xeon E5410 Quadcore computer running Linux Ubuntu 8.04. Each run was limited to a single processor. All codes were implemented in C and compiled with `gcc` 4.1.2. We generated 135 test instances for the set k-covering problem from 45 set covering instances of the OR-Library (Beasley, 1990a). For each original instance, three different coverage factors k are considered:

- k_{min}: $k = 2$;
- k_{max}: $k = \min_{i=1,\ldots,m} \sum_{j=1}^{n} a_{ij}$;
- k_{med}: $k = \lceil (k_{min} + k_{max})/2 \rceil$

The characteristics of the seven classes of test problems are shown in Table 1.

6.1. **Comparative metrics.** We used the following metrics to compare the heuristics:

- *BestValue*: for each instance, *BestValue* is the best solution value obtained over all executions of the methods considered.
- *Dev*: for each run of a method, *Dev* is the relative deviation in percentage between *BestValue* and the solution value obtained in that run.
- *AvgDev*: average value of *Dev* over all instances and runs of a method in a particular experiment.
- *#Best*: for each method, this metric gives the number of runs whose solution value matched *BestValue*.
- *NScore*: for each method and instance, this metric gives the number of methods that found better solutions than this specific method for this instance. In case of ties, all methods receive the same score, equal to the number of methods strictly better than all of them.
- *Score*: for each method, this metric gives the sum of the *NScore* values over all instances in the experiment. Thus, lower values of *Score* correspond to better methods.
- *TTime*: for each method, this metric gives the sum over all instances of the average time taken by this method over all runs of the same instance.

6.2. **GRASP with path-relinking.** The experiments reported in this section aim to evaluate the quality of the solutions returned by different variants of the GRASP with path-relinking heuristic GRASP+PR.

The RCL parameter α in the construction phase is automatically adjusted according to the Reactive GRASP strategy, as suggested by Prais and Ribeiro (2000). For each value of α in a set of discrete values $\{\alpha_1, \alpha_2, \ldots, \alpha_r\}$, we associate probabilities $p_i, i = 1, \ldots, r$. Before starting the GRASP+PR iterations, we set $p_i = 1/r$, for $i = 1, \ldots, r$. These probabilities are periodically updated according to

$$p_i = q_i / \sum_{j=1}^{r} q_j, \quad i = 1, \ldots, r,$$

with

$$q_i = \left(\frac{f^*}{M_i} \right)^{\delta}, \quad i = 1, \ldots, r,$$

where f^* is the value of the best solution found among all previous GRASP+PR iterations and M_i is the average value of the solutions found using the RCL parameter α set to α_i. In doing so, values of α leading to better results will have a higher probability of being selected. The factor $\delta = 100$ is used to attenuate low value probabilities and to intensify high value probabilities. In this experiment, r is fixed to 20 and $\alpha_i = i/20$, for $i = 1, \ldots, r$. The probabilities p_i are updated every 100 iterations.

Path-relinking was implemented according to the forward, backward, and mixed strategies. The pool of elite solutions was set to have at most 100 elements.

To evaluate each variant of GRASP+PR, eight runs were carried out for each instance, varying the initial seed given to the random number generator. The algorithm stops whenever a maximum time limit is reached. The time limit given to the instances in each class is approximately that needed by a pure, memoryless GRASP variant to perform 1000 iterations on the first instance of the class. Table

Table 2. Time limits (in seconds) given to the instances in each class and for each coverage factor in the experiments with GRASP with path-relinking.

Classes	k_{min}	k_{med}	k_{max}
scp4	5	15	27
scp5	10	45	90
scp6	5	20	38
scpa	21	141	265
scpb	17	235	288
scpc	39	329	580
scpd	26	489	544

2 shows, for each coverage factor, the time limits (in seconds) given to the instances in each class of test problems.

Table 3 shows comparisons over all 135 test instances for a pure GRASP heuristic (Gpure) and three variants of GRASP with path-relinking: backward (GPRb), forward (GPRf), and mixed (GPRm). The table shows that all variants with path-relinking performed similarly and were better than the pure GRASP, since their average percentage deviations from the best value ranged from 0.87% to 0.94%, while for pure GRASP this value amounted to 2.52%. Although the pure GRASP obtained the worst average percentage deviation, it found the best solutions in a greater number of runs than the variants using path-relinking. Furthermore, we observed that Gpure obtained the best solutions for 52 instances, while GPRb, GPRf, and GPRm found the best solutions for 38, 30, and 37 instances, respectively. These results of Gpure, however, did not lead to the best AvgDev metric value since, for the most of the instances, the relative deviation from the best value is greater than those obtained by the other variants of GRASP. Among the variants using path-relinking, the best results were obtained by the backward strategy, which presented the best results for the three metrics reported in the table.

Table 3. Summary of the numerical results obtained with four variants of GRASP.

	Gpure	GPRb	GPRf	GPRm
AvgDev	2.52 %	0.87 %	0.89 %	0.94 %
#Best	80	67	50	61
Score	189	169	184	185

6.3. **Greedy Lagrangean heuristic.** This section reports on the computational experiments performed to evaluate the efficiency of different variants of the greedy Lagrangean heuristic.

By combining the two different approaches to build the initial solution x^0 and the two modified cost schemes used in the heuristic $\mathcal{H}$, four different variants of greedy Lagrangean heuristics were devised:

- GLH1_LL: Lagrangean modified costs are used to build a feasible solution from the one provided by the Lagrangean relaxation.
- GLH2_CL: complementary modified costs are used to build a feasible solution from the one provided by the Lagrangean relaxation.
- GLH3_LS: Lagrangean modified costs are used to build a feasible solution from scratch.
- GLH4_CS: complementary modified costs are used to build a feasible solution from scratch.

For all variants, the step size parameter η is initially set to 2 and halved after every 50 consecutive iterations of the subgradient algorithm without improvement in the best lower bound. The greedy heuristic is run at every subgradient iteration. Following Beasley (1990b), the greedy Lagrangean heuristic stops whenever the lower bound z_D matches the upper bound $\bar{z}$ or the step size parameter η becomes too small ($\eta \leq 10^{-4}$ in our implementation).

Table 4 displays a summary of the results obtained over all 135 test instances with the four variants of the greedy Lagrangean heuristic. The four heuristics were able to find good solutions of similar quality, as demonstrated by their average deviations from the best value, which ranged from 0.09 to 0.15%. However, the two variants based on building feasible solutions from scratch consumed much more running time (about twice the times observed for the other variants). With respect to the variants that start from the solutions provided by the Lagrangean relaxation, the one using Lagrangean modified costs (GLH1_LL) obtained best results for the three quality metrics, finding 384 best solutions over the eight executions for each of the 135 instances at the cost of a small additional running time.

Table 4. Summary of the numerical results obtained with four variants of the greedy Lagrangean heuristic. Total time (TTime) is given in seconds.

	GLH1_LL	GLH2_CL	GLH3_LS	GLH4_CS
AvgDev	0.09 %	0.15 %	0.09 %	0.13 %
#Best	384	231	364	298
Score	83	216	98	153
TTime	24,274.71	22,677.02	37,547.50	41,804.25

6.4. **GRASP Lagrangean heuristic.** In this section, we report the computational experiments involving the LAGRASP hybridization of the best variant of GRASP with path-relinking (as presented in Section 6.2) with the best variant of the greedy Lagrangean heuristic (as presented in Section 6.3). Instead of building an initial solution from scratch, the GRASP construction phase receives the solution x^0 provided by the Lagrangean relaxation. Furthermore, instead of the original costs, Lagrangean reduced costs are used to evaluate the candidate elements. The RCL parameter α used in the GRASP construction phase was set to 0.3 to reduce the computational burden with respect to the reactive variant used in the GRASP implementations.

The aim of the first experiment with the GRASP Lagrangean heuristic is to evaluate the relationship between running times and solution quality for different parameter settings. Parameter H, the number of iterations between successive calls to the heuristic $\mathcal{H}$, was set to 1, 5, 10, and 50. Parameter β, the probability of GRASP being applied as the heuristic $\mathcal{H}$, was set to 0, 0.25, 0.50, 0.75, and 1. Parameter Q, the number of iterations carried out by the GRASP heuristic was set to 1, 5, 10, and 50. By combining these parameter values, 68 variants of the hybrid LAGRASP heuristic were created. Each variant was applied eight times to each instance, with different initial seeds given to the random number generator. The set of 21 instances considered in this experiment was formed by the first instance of each class described in Table 1.

The plot in Figure 2 summarizes the results for all evaluated variants, displaying points whose coordinates are the values of the AvgDev and TTime metrics for each combination of parameter values.

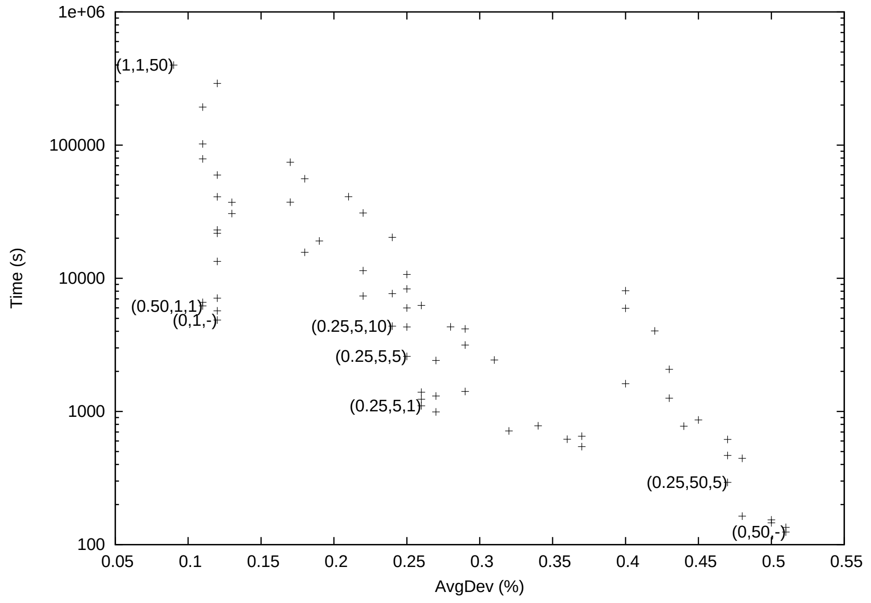

FIGURE 2. Average deviation from the best value and total running time for 68 different variants of LAGRASP: each point represents a unique combination of parameters β, H, and Q.

Eight variants of special interest are identified and labeled with the corresponding parameters β, H, and Q, in this order. These variants correspond to some selected Pareto points in the plot in Figure 2, i.e, for a given AvgDev value there is no other variant which reaches the same value in less CPU time. Additionally, for a given CPU time, there is no other variant which shows a better result for the AvgDev metric in at most this CPU time. Setting $\beta = 0$ and $H = 1$ corresponds to the greedy Lagrangean heuristic (GLH) or, equivalently, to LAGRASP(0,1,-), whose average deviation from the best value amounted to 0.12% in 4,859.16 seconds of total running time. Table 5 shows the values of AvgDev and TTime for each variant.

In the following experiment, all 135 test instances were considered in the comparison of the eight variants of LAGRASP selected above. Table 6 summarizes the results obtained by the eight variants. It shows that LAGRASP(1,1,50) found the

Table 5. Summary of the numerical results obtained with the selected variants of the GRASP Lagrangean heuristic. Total time (TTime) is given in seconds.

Heuristic	AvgDev	TTime
LAGRASP(1,1,50)	0.09 %	399,101.14
LAGRASP(0.50,1,1)	0.11 %	6,198.46
LAGRASP(0,1,-)	0.12 %	4,859.16
LAGRASP(0.25,5,10)	0.24 %	4,373.56
LAGRASP(0.25,5,5)	0.25 %	2,589.79
LAGRASP(0.25,5,1)	0.26 %	1,101.64
LAGRASP(0.25,50,5)	0.47 %	292.95
LAGRASP(0,50,-)	0.51 %	124.26

best solutions, with their average deviation from the best values being as small as 0.079%. It also found the best known solutions in 365 executions, again with the best performance when the eight variants are evaluated side by side, although at the cost of the longest running times. On the other hand, the smallest running times were observed for LAGRASP(0,50,-), which was over 3000 times faster than LAGRASP(1,1,50) but found the worst-quality solutions.

Table 6. Summary of the numerical results obtained with the best variants of the GRASP Lagrangean heuristic. Total time (TTime) is given in seconds.

Heuristic	AvgDev	#Best	Score	TTime
LAGRASP(1,1,50)	0.079 %	365	74	1,803,283.64
LAGRASP(0.50,1,1)	0.134 %	242	168	30,489.17
LAGRASP(0,1,-)	0.135 %	238	169	24,274.72
LAGRASP(0.25,5,10)	0.235 %	168	320	22,475.54
LAGRASP(0.25,5,5)	0.247 %	163	350	11,263.80
LAGRASP(0.25,5,1)	0.249 %	164	405	5,347.78
LAGRASP(0.25,50,5)	0.442 %	100	625	1,553.35
LAGRASP(0,50,-)	0.439 %	97	666	569.30

Figures 3 and 4 illustrate the merit of the proposed approach for instances scp43-k$_{max}$ and scpd3-k$_{min}$. We first observe that all variants reach the same lower bounds, which is expected since they depend exclusively on the common subgradient algorithm. However, as the dual information (i.e., the lower bound) seems to stabilize, the upper bound obtained by LAGRASP(0,1,-) (or GLH) also seems to freeze. On the other hand, the Lagrangean heuristics based on GRASP continue to make improvements in discovering better upper bounds, since the randomized GRASP construction makes it possible to escape from locally optimal solutions and to find new, improved upper bounds.

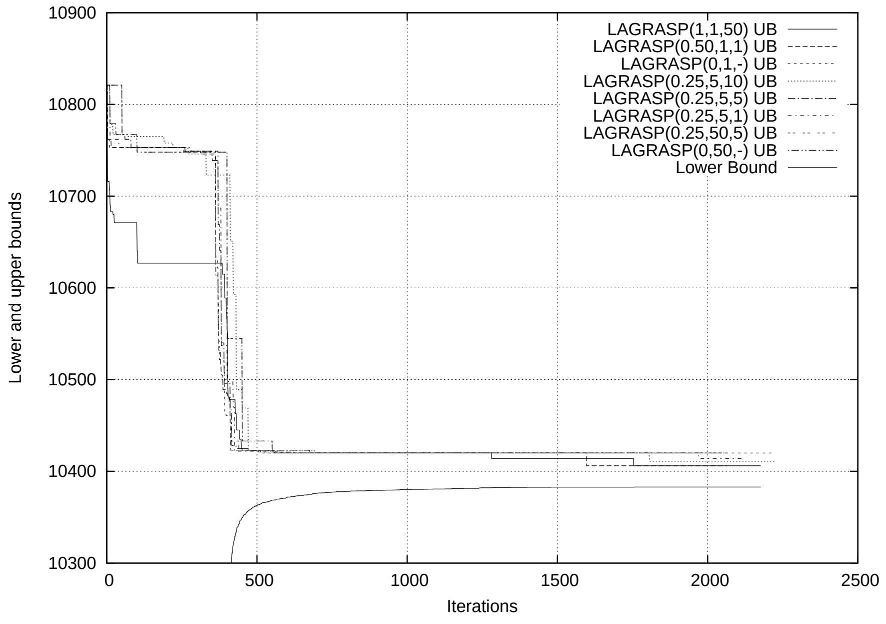

FIGURE 3. Evolution of lower and upper bounds over iterations for different variants of LAGRASP (scp43-k$_{max}$ instance).

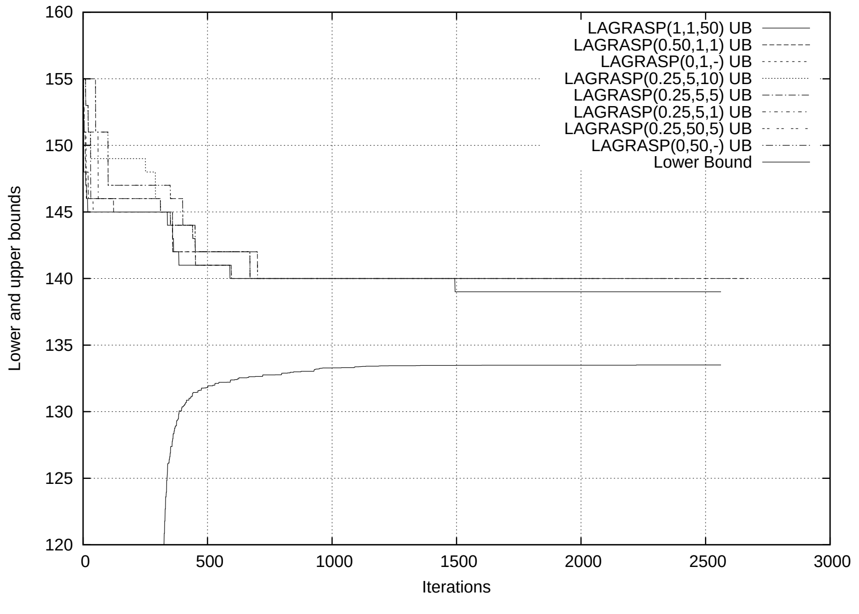

FIGURE 4. Evolution of lower and upper bounds over iterations for different variants of LAGRASP (scpd3-k$_{min}$ instance).

6.5. **Comparative results between LAGRASP and GRASP.** Finally, we compare in this section the performances of GRASP and LAGRASP when the same time limits are used as the stopping criterion for both heuristics. We consider the best variant of GRASP presented in Section 6.2, in which GRASP is combined with backward path-relinking (GPRb). GPRb is compared with the eight variants of the Lagrangean heuristics selected in Section 6.4. Results of GPRb and LAGRASP heuristics are also compared with the best solutions found by the commercial integer programming solver CPLEX 11. The stopping criterion for CPLEX was either the convergence of lower and upper bounds (proved optimality) or a maximum time limit set at 86,400 seconds (24 hours).

Tables 7-9 report, for each group of instances, the best solution values obtained. These solutions may be useful for future benchmarking studies. For each instance, the tables list the value of the best solution found by CPLEX 11 in at most 24 hours of CPU time, the number of branch and bound nodes explored by CPLEX, and the value of the best solution found by LAGRASP. Each LAGRASP solution corresponds to the best solution found by the eight LAGRASP Pareto configurations shown in the plot in Figure 2 and in Table 5. CPLEX solution values displayed in boldface correspond to those for which CPLEX was able to prove optimality.

The Lagrangean heuristics and GPRb were run on the 135 test instances with the time limits defined in Table 2. Eight runs were performed for each heuristic and each instance, using different initial seeds input to the random number generator. The results in Table 10 show that all variants of LAGRASP outperformed GPRb and were able to find solutions whose costs are very close to or as good as those obtained by CPLEX, while GPRb found solutions whose costs are on average 4.05% larger than the best values obtained by the commercial solver.

Figures 5 and 6 display the typical behavior of the two methods compared in this section for instances scp43-k$_{max}$ and scpd3-k$_{min}$, respectively. As opposed to the GRASP with path-relinking heuristic, the Lagrangean heuristics are able to escape from local optima and keep improving the solutions to obtain the best results.

6.6. **Numerical results for the original set covering instances.** In this section, the GRASP Lagrangean heuristic is applied to the 45 original instances of the set covering problem described in Table 1, which correspond to set k-covering instances with $k = 1$.

Table 11 reports the results obtained by the eight selected variants of LAGRASP, comparing the best solutions found over eight runs for each instance with the optimal values presented in Caprara et al. (1999). This table shows that variant LAGRASP(1,1,50) obtained the best results over the eight variants, at the cost of longer running times. The average deviation from the optimal value was only 0.11% and this heuristic found the optimal solutions for the largest number of runs (305 out of 360 runs). For each algorithm, we compute the number of instances for which at least one run of the eight attempts found the optimal solution. These values are shown in the fourth column of Table 11. These results are promising given that they were obtained without any special tuning of the parameters for this particular case of $k = 1$.

7. Concluding remarks

The main goal of this paper was to advance the current state-of-the-art of hybrid heuristics combining metaheuristics with Lagrangean relaxations. To the best of our knowledge, the paper reports on the first proposal of hybridization between GRASP and Lagrangean heuristics based on subgradient optimization.

TABLE 7. For each of the 135 test instances, the table lists the best solution value found by CPLEX in at most 24 hours (boldface indicates CPLEX proved optimality), the number of branch and bound nodes explored by CPLEX, and the best solution value found by LAGRASP for k$_{min}$ instances.

Instance	Best value CPLEX	B&B Nodes	Best value LAGRASP
scp41-k$_{min}$	**1148**	4	1150
scp42-k$_{min}$	**1205**	0	1205
scp43-k$_{min}$	**1213**	0	1214
scp44-k$_{min}$	**1185**	0	1185
scp45-k$_{min}$	**1266**	1	1266
scp46-k$_{min}$	**1349**	1	1349
scp47-k$_{min}$	**1115**	0	1115
scp48-k$_{min}$	**1225**	34	1225
scp49-k$_{min}$	**1485**	0	1485
scp410-k$_{min}$	**1356**	0	1356
scp51-k$_{min}$	**579**	0	579
scp52-k$_{min}$	**677**	69	679
scp53-k$_{min}$	**574**	9	574
scp54-k$_{min}$	**582**	15	587
scp55-k$_{min}$	**550**	0	550
scp56-k$_{min}$	**560**	1	560
scp57-k$_{min}$	**695**	0	695
scp58-k$_{min}$	**662**	0	662
scp59-k$_{min}$	**687**	34	687
scp510-k$_{min}$	**672**	0	672
scp61-k$_{min}$	**283**	27	283
scp62-k$_{min}$	**302**	18	302
scp63-k$_{min}$	**313**	0	313
scp64-k$_{min}$	**292**	31	292
scp65-k$_{min}$	**353**	32	353
scpa1-k$_{min}$	**562**	157	563
scpa2-k$_{min}$	**560**	83	560
scpa3-k$_{min}$	**524**	73	524
scpa4-k$_{min}$	**527**	22	527
scpa5-k$_{min}$	**557**	31	559
scpb1-k$_{min}$	**149**	2076	149
scpb2-k$_{min}$	**150**	184	151
scpb3-k$_{min}$	**165**	216	165
scpb4-k$_{min}$	**157**	853	157
scpb5-k$_{min}$	**151**	135	152
scpc1-k$_{min}$	**514**	445	515
scpc2-k$_{min}$	**483**	286	486
scpc3-k$_{min}$	**544**	6026	544
scpc4-k$_{min}$	**484**	109	485
scpc5-k$_{min}$	**488**	569	490
scpd1-k$_{min}$	**122**	1044	122
scpd2-k$_{min}$	**127**	211	127
scpd3-k$_{min}$	**138**	241	138
scpd4-k$_{min}$	**122**	517	123
scpd5-k$_{min}$	**130**	358	130

Table 8. For each of the 135 test instances, the table lists the best solution value found by CPLEX in at most 24 hours (boldface indicates CPLEX proved optimality), the number of branch and bound nodes explored by CPLEX, and the best solution value found by LAGRASP for k_{med} instances.

Instance	Best value CPLEX	B&B Nodes	Best value LAGRASP
scp41-k_{med}	**8350**	2640	8366
scp42-k_{med}	**6111**	443	6117
scp43-k_{med}	**4676**	114	4690
scp44-k_{med}	**4670**	162	4679
scp45-k_{med}	**8389**	92	8409
scp46-k_{med}	**6416**	1473	6432
scp47-k_{med}	**6281**	43	6284
scp48-k_{med}	**8421**	287	8439
scp49-k_{med}	**7101**	827	7121
scp410-k_{med}	**5355**	41	5364
scp51-k_{med}	**11205**	38720	11239
scp52-k_{med}	**14418**	18400	14473
scp53-k_{med}	**11476**	15892	11513
scp54-k_{med}	**9944**	25618	9965
scp55-k_{med}	**10880**	12030	10918
scp56-k_{med}	**10581**	54459	10629
scp57-k_{med}	**14919**	351673	14984
scp58-k_{med}	**10622**	275662	10687
scp59-k_{med}	**11042**	22482	11081
scp510-k_{med}	**12436**	52775	12475
scp61-k_{med}	**7653**	9916749	7692
scp62-k_{med}	**6739**	5412524	6773
scp63-k_{med}	**8309**	548135	8365
scp64-k_{med}	**8546**	7220253	8585
scp65-k_{med}	**9038**	1519470	9070
scpa1-k_{med}	21227	2678041	21324
scpa2-k_{med}	21739	3183837	21820
scpa3-k_{med}	20095	3878035	20155
scpa4-k_{med}	22865	3219403	22985
scpa5-k_{med}	18643	3346013	18706
scpb1-k_{med}	29222	1767524	29234
scpb2-k_{med}	28112	2134303	28187
scpb3-k_{med}	27872	2301677	27944
scpb4-k_{med}	25678	2272003	25742
scpb5-k_{med}	28203	2313903	28297
scpc1-k_{med}	32659	1285550	32763
scpc2-k_{med}	32765	1373093	32871
scpc3-k_{med}	34492	1256093	34610
scpc4-k_{med}	31366	1354393	31495
scpc5-k_{med}	30060	1401693	30196
scpd1-k_{med}	38991	1123793	39132
scpd2-k_{med}	39030	1167593	39098
scpd3-k_{med}	39198	1108293	39271
scpd4-k_{med}	38781	1241341	38879
scpd5-k_{med}	40321	1158993	40409

Table 9. For each of the 135 test instances, the table lists the best solution value found by CPLEX in at most 24 hours (boldface indicates CPLEX proved optimality), the number of branch and bound nodes explored by CPLEX, and the best solution value found by LAGRASP for k$_{max}$ instances.

Instance	Best value CPLEX	B&B Nodes	Best value LAGRASP
scp41-k$_{max}$	**18265**	0	18290
scp42-k$_{max}$	**12360**	2160	12405
scp43-k$_{max}$	**10396**	49	10398
scp44-k$_{max}$	**10393**	5713	10427
scp45-k$_{max}$	**18856**	0	18856
scp46-k$_{max}$	**15394**	1210	15419
scp47-k$_{max}$	**15233**	1241	15280
scp48-k$_{max}$	**18602**	792	18628
scp49-k$_{max}$	**16558**	392	16591
scp410-k$_{max}$	**11607**	58	11618
scp51-k$_{max}$	**35663**	994835	35749
scp52-k$_{max}$	**45396**	4802	45433
scp53-k$_{max}$	**36329**	340559	36388
scp54-k$_{max}$	**28017**	9508	28051
scp55-k$_{max}$	**32779**	57608	32878
scp56-k$_{max}$	**29608**	312752	29653
scp57-k$_{max}$	**41930**	111582	41954
scp58-k$_{max}$	**32320**	32718	32405
scp59-k$_{max}$	**33584**	67633	33655
scp510-k$_{max}$	**38709**	106627	38807
scp61-k$_{max}$	**23516**	10620461	23534
scp62-k$_{max}$	**19934**	6490122	20025
scp63-k$_{max}$	**27983**	106240	28027
scp64-k$_{max}$	**26442**	10067517	26530
scp65-k$_{max}$	**27069**	1678983	27124
scpa1-k$_{max}$	68522	2803421	68669
scpa2-k$_{max}$	65842	3619480	65922
scpa3-k$_{max}$	66829	2609228	67016
scpa4-k$_{max}$	72334	3776330	72465
scpa5-k$_{max}$	60491	2500372	60625
scpb1-k$_{max}$	105506	3076095	105636
scpb2-k$_{max}$	102922	2644498	103046
scpb3-k$_{max}$	98280	2590497	98445
scpb4-k$_{max}$	93777	2582402	93836
scpb5-k$_{max}$	102810	2750998	102905
scpc1-k$_{max}$	112471	1479684	112667
scpc2-k$_{max}$	113916	1688788	114145
scpc3-k$_{max}$	117416	1705592	117680
scpc4-k$_{max}$	110823	1653941	111091
scpc5-k$_{max}$	104439	1596493	104591
scpd1-k$_{max}$	144887	1367853	145060
scpd2-k$_{max}$	144096	1656196	144218
scpd3-k$_{max}$	140474	1326320	140685
scpd4-k$_{max}$	143513	1495597	143582
scpd5-k$_{max}$	146307	1610496	146452

TABLE 10. Comparative results for the best variants of LAGRASP and GRASP.

Heuristic	AvgDev	#Best	Score
LAGRASP(1,1,50)	3.30 %	0	949
LAGRASP(0.50,1,1)	0.35 %	171	152
LAGRASP(0,1,-)	0.35 %	173	120
LAGRASP(0.25,5,10)	0.45 %	138	229
LAGRASP(0.25,5,5)	0.45 %	143	236
LAGRASP(0.25,5,1)	0.46 %	137	288
LAGRASP(0.25,50,5)	0.65 %	97	491
LAGRASP(0,50,-)	0.65 %	93	534
GPRb	4.05 %	0	1043

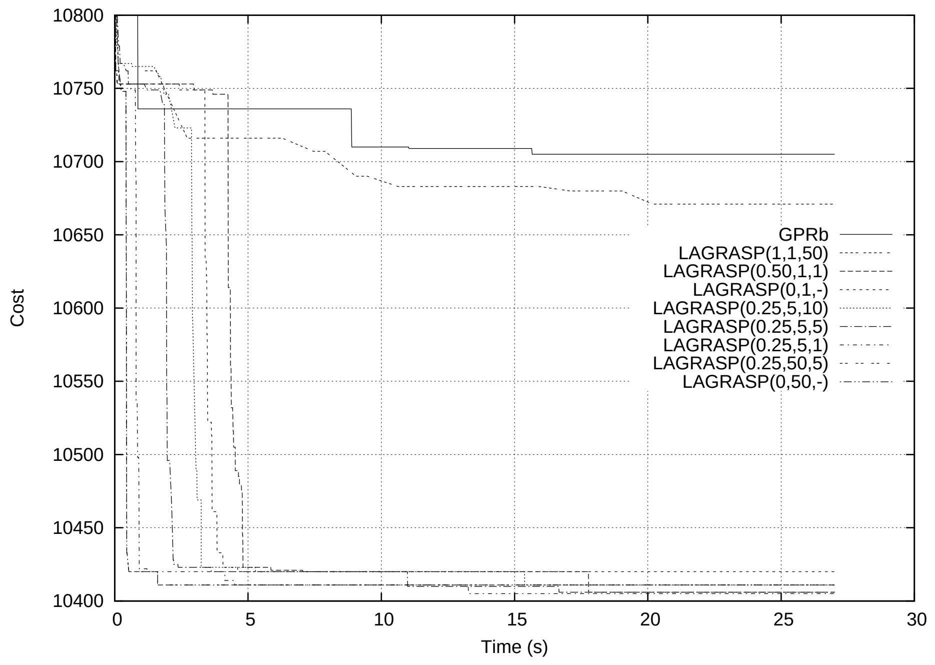

FIGURE 5. Evolution of solution costs with time for the best variants of LAGRASP and GRASP+PR (scp43-k$_{max}$ instance).

The set k-covering problem was used as the test bed for the algorithmic developments and computational experiments. Few heuristics are available in the literature for this problem. The need for good approximate algorithms for this problem is well established, due to the hardness of solving the separation problem associated with branch-and-cut algorithms for its exact solution. Two applications of the set k-covering problem were described and 135 test instances were derived from set covering instances in the OR-Library.

TABLE 11. Summary of the numerical results obtained with the best variants of the GRASP Lagrangean heuristic for the original set covering instances. Total time (TTime) is given in seconds.

Heuristic	AvgDev	#Best	Optimal	Score	TTime
LAGRASP(1,1,50)	0.11 %	305	40/45	5	23,285.11
LAGRASP(0.50,1,1)	0.35 %	219	31/45	35	257.54
LAGRASP(0,1,-)	0.40 %	209	27/45	62	210.83
LAGRASP(0.25,5,10)	0.41 %	214	34/45	25	265.07
LAGRASP(0.25,5,5)	0.46 %	202	30/45	42	160.35
LAGRASP(0.25,5,1)	0.47 %	199	28/45	58	80.06
LAGRASP(0.25,50,5)	0.81 %	161	25/45	105	52.80
LAGRASP(0,50,-)	0.90 %	148	20/45	150	43.33

We first described a GRASP with path-relinking heuristic for the set k-covering problem, followed by the template of a family of Lagrangean heuristics. The greedy Lagrangean heuristic makes use of a greedy algorithm to obtain solutions for the Lagrangean relaxation, while the hybrid GRASP Lagrangean heuristic LAGRASP employs the best variant of GRASP with path-relinking for this purpose.

Extensive computational experiments were carried out on 135 test instances, comparing running times and different metrics of solution quality for pure GRASP,

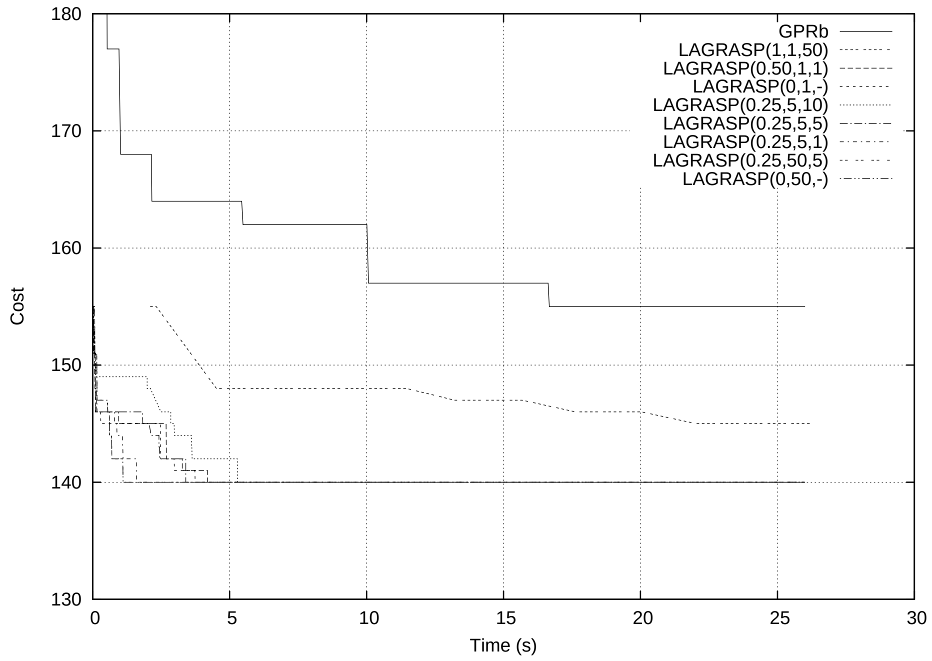

FIGURE 6. Evolution of solution costs with time for the best variants of LAGRASP and GRASP+PR (scpd3-k$_{min}$ instance).

GRASP with path-relinking, greedy Lagrangean, and GRASP Lagrangean heuristics. The numerical results show that, for the set k-covering problem, the Lagrangean heuristics performed consistently better than GRASP.

The comparison of different variants of LAGRASP showed that, by properly tuning its parameters, it is possible to obtain a good trade-off between solution quality and running time. Despite consuming longer running times, LAGRASP was able to find better solutions than the greedy Lagrangean heuristics for a larger number of instances.

Furthermore, it is important to observe that LAGRASP makes better use of dual information provided by subgradient optimization and is able to discover better solutions and to escape from locally optimal solutions after the stabilization of the lower bounds, when the greedy Lagrangean heuristic fails to find new improving solutions.

References

P. Avella, M. Boccia, and I. Vasilyev. Computational experience with general cutting planes for the Set Covering problem. *Operations Research Letters*, 37: 16–20, 2009.

V. Bafna, B.V. Halldorsson, R. Schwartz, A.G. Clark, and S. Istrail. Haplotypes and informative SNP selection algorithms: Don't block out information. In *Proceedings of the 7th Annual International Conference on Research in Computational Molecular Biology*, pages 19–27, Berlin, 2003. ACM.

E. Balas and S.M. Ng. On the set covering polytope: I. All the facets with coefficients in 0,1,2. *Mathematical Programming, Series A*, 43:57–69, 1989a.

E. Balas and S.M. Ng. On the set covering polytope: II. Lifting the facets with coefficients in 0,1,2. *Mathematical Programming, Series A*, 45:1–20, 1989b.

J.E. Beasley. An algorithm for set-covering problem. *European Journal of Operational Research*, 31:85–93, 1987.

J.E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990a.

J.E. Beasley. A Lagrangian heuristic for set-covering problems. *Naval Research Logistics*, 37:151–164, 1990b.

J.E. Beasley. Lagrangean relaxation. In C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, pages 243–303. Blackwell Scientific Publications, Oxford, 1993.

A.J. Brookes. The essence of SNPs. *Gene*, 234:177–186, 1999.

T.A. Brown. *Genomes*. BIOS Scientific Publishers Ltd, New York, 2002.

A. Caprara, P. Toth, and M. Fischetti. A heuristic method for the set covering problem. *Operations Research*, 47:730–743, 1999.

C.J. Chang, Y. Huang, and K.M. Chao. A greedier approach for finding tag SNPs. *Bioinformatics*, 22:685–691, 2006.

T.A. Feo and M.G.C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

P. Festa and M. G. C. Resende. An annotated bibliography of GRASP, Part I: Algorithms. *International Transactions in Operational Research*, 16:1–24, 2009a.

P. Festa and M. G. C. Resende. An annotated bibliography of GRASP, Part II: Applications. *International Transactions in Operational Research*, 16:131–172, 2009b.

P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 325–367. Kluwer Academic Publishers, Boston, 2002.

M.L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 50:1861–1871, 2004.

F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer Academic Publishers, Boston, 1996.

D.J. Gonsalvez, N.G. Hall, W.T. Rhee, and S.P. Siferd. Heuristic solutions and confidence intervals for the multicovering problem. *European Journal of Operational Research*, 31:94–101, 1987.

N.G. Hall and D.S. Hochbaum. The multicovering problem: The use of heuristics, cutting planes, and subgradient optimization for a class of integer programs. Technical report, U.C. Berkeley, 1983.

N.G. Hall and D.S. Hochbaum. The multicovering problem. *European Journal of Operational Research*, 62:323–339, 1992.

M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18:1138–1162, 1970.

M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

M. Held, P. Wolfe, and H.P. Crowder. Validation of subgradient optimization. *Mathematical Programming*, 6:62–88, 1974.

Y. Huang, K. Zhang, T. Chen, and K.M. Chao. Selecting additional tag SNPs for tolerating missing data in genotyping. *BMC Bioinformatics*, 6:263–278, 2005.

M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

A. Lucena. Private communication, 2009.

L.S. Pessoa and C.C. Ribeiro. A GRASP for the minimum informative subset problem. In *Abstracts Book of the 7th Metaheuristics International Conference*, page 44, Montréal, 2007.

M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

M.G.C. Resende. An optimizer in the telecommunications industry. *SIAM SIAG/Optimization Views-and-News*, 18(2):8–19, 2007.

M.G.C. Resende. Metaheuristic hybridization with Greedy Randomized Adaptive Search Procedures. In Zhi-Long Chen and S. Raghavan, editors, *TutORials in Operations Research*, pages 295–319. INFORMS, 2008.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–249. Kluwer Academic Publishers, 2003.

M.G.C. Resende and C.C. Ribeiro. GRASP with path-relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, and M. Yagiura, editors, *Metaheuristics: Progress as Real Problem Solvers*, pages 29–63. Springer-Verlag, 2005.

M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures: Advances and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 293–319. Springer, 2nd edition, 2010.

M.G.C. Resende, C.C. Ribeiro, R. Martí, and F. Glover. Scatter search and path-relinking: Fundamentals, advances, and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, pages 87–107. Springer, 2nd edition, 2010.

The International HapMap Consortium. The International HapMap Project. *Nature*, 426:789–796, 2003.

V.V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, 2004.

(Luciana S. Pessoa) Department of Informatics and Applied Mathematics, Universidade Federal do Rio Grande do Norte, Campus Universitário - Lagoa Nova, Natal, RN 59072-970 Brazil.

E-mail address: `luciana@dimap.ufrn.br`

(Mauricio G. C. Resende) Algorithms and Optimization Research Department, AT&T Labs Research, 180 Park Avenue, Room C241, Florham Park, NJ 07932 USA.

E-mail address: `mgcr@research.att.com`

(Celso C. Ribeiro) Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria, 156, Niterói, RJ 24210-240 Brazil.

E-mail address: `celso@ic.uff.br`