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 k-covering problem is an extension of the set covering problem, in which each object has to be covered at least k times. We describe a GRASP with path-relinking heuristic for its solution, 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 solutions for the Lagrangean relaxation problem. Numerical experiments have shown that the Lagrangean heuristics performed consistently better than GRASP. The GRASP Lagrangean heuristic makes better use of the dual information provided by subgradient optimization and is able to discover better solutions even after the stabilization of the lower bounds.

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 $i \in I$. We refer to this problem as the *set k-covering problem* ($\mathrm{SC}_k\mathrm{P}$).

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 $\mathrm{SC}_k\mathrm{P}$ be represented by a binary n-vector x, where $x_j = 1$ if and only if $j \in \hat{J}$. An integer programming formulation for the set k-covering problem is

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

s.t.

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

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

Applications of the set multicovering problem arise in a variety of fields, such as marketing, logistics, security, telecommunications (Resende, 2007), and computational biology (Bafna et al., 2003). Though some of these applications can

Date: February 18, 2010.

Key words and phrases. GRASP, hybrid heuristics, Lagrangean relaxation, Lagrangean heuristics, set k-covering.

be modeled as set covering problems, for reliability purposes they are treated as multicovering. Other applications are described by Hall and Hochbaum (1992).

We propose a template of Lagrangean heuristics for the set k-covering problem, based on the hybridization of greedy and GRASP heuristics with subgradient optimization. A GRASP with path-relinking heuristic for the set k-covering problem is customized in Section 2. A template for Lagrangean heuristics for $\mathrm{SC}_k\mathrm{P}$ based on constructive heuristics and subgradient optimization is proposed in Section 3. Different implementation strategies for the constructive heuristics and a hybridization of GRASP with a Lagrangean heuristic are discussed in Section 4. Numerical results are reported in Section 5. Concluding remarks are made in the last section.

2. GRASP with path-relinking

GRASP is the short for greedy randomized adaptive search procedures, a multi-start metaheuristic which consists of applying local search to feasible starting solutions generated with a greedy randomized construction heuristic (Resende and Ribeiro, 2003). It was introduced in Feo and Resende (1989) for solving a set covering problem with unit costs.

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 (Resende and Ribeiro, 2005). In this section, we specialize GRASP and path-relinking into a heuristic for the set k-covering problem.

2.1. **Construction phase.** A greedy algorithm for set k-covering builds a solution from scratch, adding one of the sets $P_1, \ldots, P_n$ at a time to a partial solution, until each object is covered by at least k sets. 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 the 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 ρ_j between its cost c_j and its covering cardinality τ_j. The greedy algorithm adds to the partial solution a minimum ratio candidate set.

In a greedy randomized construction, a candidate set is randomly selected to be added to the partial solution from a restricted list, formed by all elements $j \in L$ whose ratio ρ_j is less than or equal to $\rho^- + \alpha(\rho^+ - \rho^-)$, with $\rho^- = \min\{\rho_j \ : \ j \in L\}$, and $\rho^+ = \max\{\rho_j \ : \ j \in L\}$, and α a real-valued parameter in the interval $[0, 1]$.

2.2. **Local search.** Solutions built with the randomized greedy algorithm are not guaranteed to be locally optimal, even with respect to simple neighborhoods. Therefore, the application of local search to such a solution usually results in an improved local optimum. Starting from an initial solution, local search explores its neighborhood for a cost-improving solution. If none is found, the search returns the initial solution as a local minimum. Otherwise, the cost-improving solution is made the new initial solution, and the procedure repeats itself.

The local search proposed in this paper makes use of two neighborhoods. The first is a $(1, 0)$-exchange, in which we attempt to remove superfluous sets from the cover. The second neighborhood is a $(1, 1)$-exchange, in which we attempt to replace a more expensive set in the cover by a less expensive one not in the cover.

2.3. **Path-relinking.** The basic implementation of GRASP is memoryless, since any iteration does 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). It explores paths in the solution space connecting good-quality solutions, one of them being an initial solution x^s and the other a target solution x^t. The procedure maintains a pool P of diverse elite solutions found during the search. Path-relinking is carried out after local search, between the local minimum and a randomly selected pool solution.

We take x^s as the binary vector representing the solution obtained after the local search phase and x^t as the binary vector representing a pool solution. To favor longer paths, the pool solution x^t is chosen at random with probability proportional to its Hamming distance to x^s, i.e. $|\{j = 1, \ldots, n \ : \ x^s_j \neq x^t_j\}|$. We do not consider a pool solution if its Hamming distance to x^s is less than four, since any path between them cannot contain solutions simultaneously better than both x^s and x^t.

```
1  GRASP+PR
2  Initialize elite set P ← ∅;
3  Initialize best solution value z* ← ∞;
4  for i = 1, ..., N do
5  |  x ← GreedyRandomizedConstruction();
6  |  x ← 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 ∈ P;
10 |  |  Determine which solution (between x and x^p) is the
   |  |  initial solution x^s and the target solution x^t;
11 |  |  x ← PathRelinking(x^s, x^t);
12 |  |  x ← LocalSearch(x);
13 |  |  Update the elite set P with x;
14 |  end
15 |  if z(x) < z* then
16 |  |  x* ← x;
17 |  |  z* ← z(x);
18 |  end
19 end
```

Algorithm 1: GRASP with path-relinking procedure.

Algorithm 1 shows the pseudo-code for the GRASP with path-relinking procedure. Lines 2 and 3 initialize the elite set P and the value of the best solution z^*. 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 local optimum x with respect to (0,0)- and (1,0)-exchanges is computed in line 6. The elite set P is initialized in line 7. 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. The candidates for selection are all solutions in P whose Hamming distance to x^s is

greater than three. As already observed, candidates are selected with probabilities proportional to their Hamming distances to x^s. Line 10 determines, between x and x^p, which one is the starting solution x^s. The other is the target solution, x^t. Path-relinking is applied in line 11, resulting in a solution x which is reoptimized by local search in line 12. The elite set P is updated in line 13. If it is not full and the new solution is different from all others in the pool, then it is inserted in the elite set. Otherwise, if x is better than the worst solution in the pool, it replaces the latter. 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. Distinct approaches have been considered in the implementation of this procedure (Resende and Ribeiro, 2005). We used the backward strategy, in which the initial solution is the best between x^s and x^t.

3. A template for Lagrangean heuristics

The primal solutions obtained along the resolution of the Lagrangean dual of a combinatorial optimization problem are not necessarily feasible (Beasley, 1993; Fisher, 2004). Lagrangean heuristics exploit the dual multipliers to generate primal feasible solutions. Beasley (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):

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

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

$$z'(\lambda) = \min \sum_{j=1}^{n} c'_j x_j + \sum_{i=1}^{m} \lambda_i k$$
$$x_j \in \{0,1\}, \quad j = 1, \ldots, n,$$

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

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

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

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

being 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*:

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

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

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

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 proposed Lagrangean heuristic 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). 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 2 describes the pseudo-code of the Lagrangean heuristic. Lines 2 to 4 initialize the 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 use a basic heuristic to produce primal feasible solutions to problem (1)–(3) whenever the iteration counter q is a multiple of H. In line 11, a heuristic $\mathcal{H}$ is applied 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 and its cost are updated in lines 13 and 14, respectively. If the lower bound $z'(\lambda^q)$ is greater than the best lower bound z_D, then z_D is updated in line 17.

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 γ to the modified cost vector;
10 **if** *q is a multiple of H* **then**
11 Apply a basic heuristic $\mathcal{H}$ with cost vector γ 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 not satisfied* ;

Algorithm 2: Pseudo-code of the template for a Lagrangean heuristic.

Lines 18 and 19 compute the subgradient and 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.

4. Basic heuristics and Lagrangean GRASP

Different implementation strategies of the basic heuristic in the template of Algorithm 2 lead to distinct Lagrangean heuristics. We considered two variants of the basic heuristic $\mathcal{H}$. The first is a greedy algorithm with local search, while the second is a GRASP with path-relinking.

4.1. Greedy basic 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 2 to make it feasible for problem (1)–(3). It corresponds to the greedy randomized construction using parameter $\alpha = 0$ and modified costs (c' or $\tilde{c}$). Local search is applied to the resulting solution, using the original cost vector c. We shall refer to the Lagrangean heuristic that uses the greedy basic heuristic as the *greedy Lagrangean heuristic* or simply GLH.

4.2. GRASP basic heuristic.

Instead of performing one construction step followed by local search as in the greedy basic heuristic, this heuristic applies the

GRASP with path-relinking heuristic of Algorithm 1 either to build a feasible solution from scratch, or to repair the solution $x'(\lambda^q)$ produced in line 7 of the Lagrangean heuristic described in Algorithm 2 to make it feasible for problem (1)–(3). We shall refer to the Lagrangean heuristic that uses the GRASP basic heuristic as the *GRASP Lagrangean heuristic* or simply LAGRASP.

Although the GRASP basic heuristic produces better solutions than the greedy basic heuristic, the latter is much faster. To address this, in line 11 of Algorithm 2, we choose to use the GRASP basic heuristic with probability β and the greedy basic heuristic with probability $1-\beta$. 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, and the probability β of choosing the GRASP heuristic. We shall refer to the Lagrangean heuristic that uses this hybrid strategy as LAGRASP(β, H, Q).

5. Computational experiments

The numerical experiments were performed on an 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). Three coverage factors k were used:

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

5.1. Comparative metrics.

We used the metrics below 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 heuristic, *Dev* is the relative deviation in percent between *BestValue* and the best solution value obtained with that method.
- *AvgDev*: this value is the average of *Dev* over all instances considered in a particular experiment.
- *#Best*: for each heuristic, this metric gives the number of instances for which the best solution obtained with this heuristic matches *BestValue*.
- *NScore*: for each instance and heuristic, it gives the number of algorithms that found a better solution. In case of ties, all algorithms receive the same score, equal to the number of heuristics strictly better than all of them.
- *Score*: for each heuristic, it gives the sum of the *NScore* values over all instances in the experiment. Thus, the lower *Score* the better the heuristic.
- *TTime*: for each heuristic, it gives the sum over all instances of the average time taken by this heuristic over all runs of the same instance.

5.2. Greedy Lagrangean heuristic.

This section reports on the computational experiments performed to evaluate the efficiency of different versions 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 basic heuristic $\mathcal{H}$, four different versions of greedy Lagrangean heuristics have been devised:

- GLH1_LL: Lagrangean modified costs are used to build a feasible solution from that provided by the Lagrangean relaxation.
- GLH2_CL: complementary modified costs are used to build a feasible solution from that 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 versions, 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 basic 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 1. Summary of the numerical results obtained with four versions of the greedy Lagrangean heuristic.

	GLH1_LL	GLH2_CL	GLH3_LS	GLH4_CS
AvgDev	0.05 %	0.07 %	0.05 %	0.04 %
#Best	79	47	78	65
Score	83	216	98	153
TTime	24,274.71	22,677.02	37,547.50	41,804.25

Table 1 displays a summary of the results obtained over all 135 test instances with the four versions of the greedy Lagrangean heuristic. The four heuristics are able to find good solutions of similar quality, as demonstrated by their average deviations from the best value, which range from 0.04 to 0.07%. However, the two versions based on building feasible solutions from scratch consumed much more running time. With respect to the versions 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, being able to find 79 best solutions over the 135 instances at the cost of a small additional running time.

5.3. **GRASP Lagrangean heuristic.** In this section, we report the computational experiments involving the hybridization of GRASP with path-relinking with the best version of the greedy Lagrangean heuristic. The GRASP construction phase receives the solution x_0 provided by the Lagrangean relaxation. Lagrangean reduced costs are used to evaluate the candidate elements. Parameter α used in the GRASP construction phase was set to 0.3 to reduce the computational burden with respect to the reactive version.

The aim of the first experiment 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 basic heuristic, was set to 1, 5, 10, and 50. Parameter β, the probability of GRASP being applied as the basic heuristic, was set to 0, 0.25, 0.50, 0.75, and 1. Parameter Q, the number of iterations carried out by the GRASP basic heuristic was set to 1, 5, 10, and 50. By combining these parameter values, 68 versions of LAGRASP were created. Each version was applied eight times to each instance, with different initial seeds for the random number generator. A subset of 21 instances was considered in this experiment.

The plot in Figure 1 summarizes the results for all versions, displaying points whose coordinates are the values of the AvgDev and TTime metrics for each combination of parameter values. Points closer to the origin correspond to the best balance between solution quality and running time. Three versions, of special interest, are identified and labeled with the corresponding parameters β, H, and Q, in this order. Setting $\beta = 0$ and $H = 1$ corresponds to the greedy Lagrangean heuristic (GLH) or, equivalently, to LAGRASP(0,1,-): the average deviation from the best value amounted to 0.09% in 4,859.16 seconds of running time. Another particularly interesting version is that with $\beta = 0.25$, $H = 1$, and $Q = 5$: LAGRASP(0.25,1,5) obtained better solutions than LAGRASP(0,1,-) at the cost of an increase in running time, since its average deviation from the best value was only 0.07% and the running time attained 13,394.27 seconds. Finally, setting $\beta = 0.50$, $H = 10$, and $Q = 5$ lead to LAGRASP(0.50,10,5), which took less running time than LAGRASP(0,1,-), but at the cost of a deterioration in solution quality: the average deviation from the best value was 0.10% and the running time only 2,414.78 seconds.

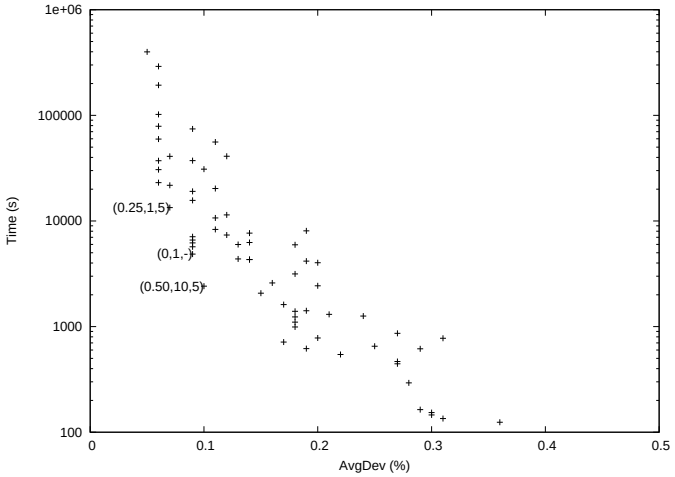

FIGURE 1. Average deviation from the best value and total running time for 68 different versions of LAGRASP: each point represents a unique combination of parameters β, H, and Q.

Next, all 135 test instances were considered in the comparison of the three versions of LAGRASP selected as above. Table 2 summarizes the results obtained by the three versions. It shows that LAGRASP(0.25,1,5) found the best solutions, with their average deviation from the best values being as low as 0.01%. It also found 111 among the best known solutions, again with the best performance when the three versions are evaluated side by side, although at the cost of the highest running times. On the other hand, the smallest running time are observed for LAGRASP(0.50,10,5), which spent about one sixth of the time consumed by LAGRASP(0.25,1,5) but found the lowest-quality solutions. LAGRASP(0,1,-) (or GLH) has offered the best trade-off between running time and solution quality.

Figure 2 illustrates the merit of the proposed method for instance scp510-k$_{max}$. We notice that, 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.

TABLE 2. Summary of the numerical results obtained with the best versions of the GRASP Lagrangean heuristic.

	LAGRASP (0,1,-)	LAGRASP (0.25,1,5)	LAGRASP (0.50,10,5)
AvgDev	0,03 %	0,01 %	0,07 %
#Best	73	111	47
Score	80	25	160
Time	24,274.71	63,603.06	11,401.26

However, both LAGRASP(0.25,1,5) and LAGRASP(0.50,10,5) continue to make improvements in discovering better upper bounds, since the randomized aspect of the GRASP constructive procedure makes it possible to escape from locally optimal solutions.

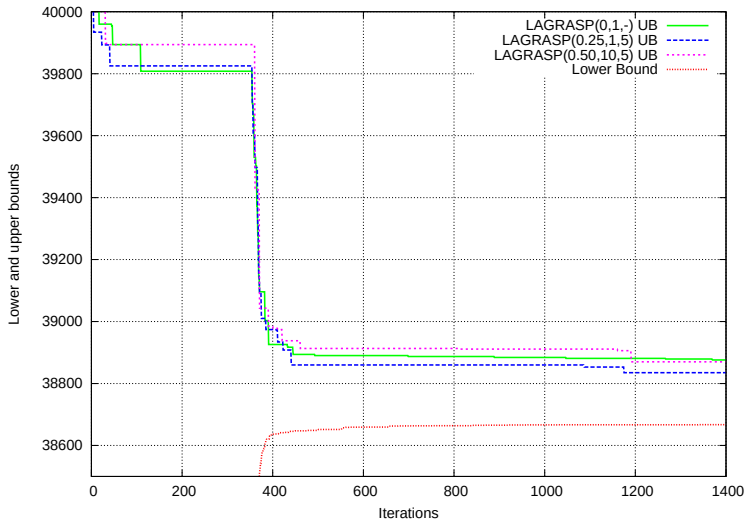

FIGURE 2. Evolution of lower and upper bounds over time for different versions of LAGRASP.

5.4. **Comparative results.** In this section, we compare the performance of GRASP and LAGRASP when the same time limits are given as the stopping criterion for both heuristics. GRASP with backward path-relinking (GPRb) is compared to LAGRASP(0.25,1,5), which obtained the best solutions among the three versions of the Lagrangean heuristics compared in Section 5.3.

LAGRASP(0.25,1,5) and GPRb run for the same time needed by a pure, memoryless GRASP variant to perform 1000 iterations for the same instance. Eight runs have been performed for each heuristic and each instance, with different initial seeds given to the random number generator. The results in Table 3 show that LAGRASP(0.25,1,5) beat GPRb and found the best known solutions for all instances, while GPRb found solutions whose costs are larger by 3.60% on average than the best values obtained by LAGRASP(0.25,1,5).

Figure 3 displays for the same instance scp510-k$_{max}$ the typical behavior of the two methods compared in this section: LAGRASP(0.25,1,5) consistently finds better solutions than GPRb along all the execution time.

TABLE 3. Comparative results for LAGRASP and GRASP.

	LAGRASP(0.25,1,5)	GPRb
AvgDev	0.00 %	3.60 %
#Best	135	0
Score	0	135

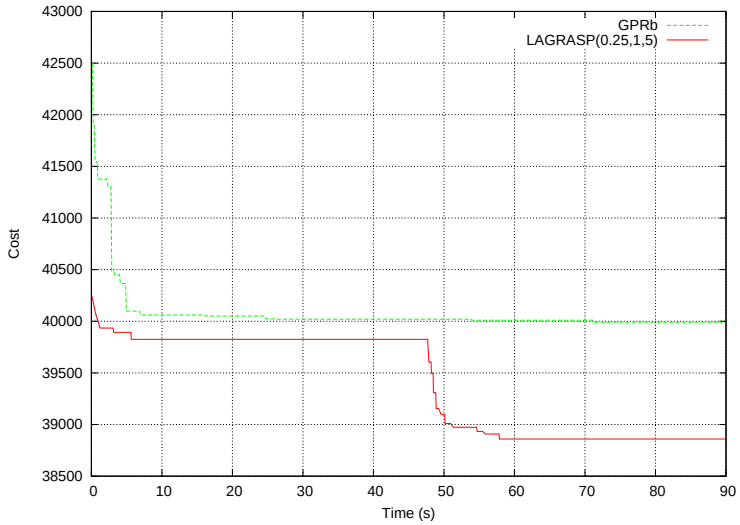

FIGURE 3. Evolution of solution costs with time for LAGRASP and GRASP.

6. CONCLUDING REMARKS

This paper advances the current state-of-the-art of hybrid heuristics combining metaheuristics with Lagrangean relaxations, reporting on the hybridization between GRASP and subgradient optimization. The set k-covering problem was used as the test bed for the algorithmic developments and computational experiments.

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 GRASP Lagrangean heuristic LAGRASP employs the best variant of GRASP with path-relinking for this purpose.

Computational experiments have been carried out comparing running times and solution quality for GRASP with path-relinking, greedy Lagrangean, and GRASP Lagrangean heuristics. The results have shown that the Lagrangean heuristics performed consistently better than GRASP for the set k-covering problems.

LAGRASP found better solutions for a greater number of instances than the greedy Lagrangean heuristic. It 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 after the stabilization of the lower bounds, when the greedy Lagrangean heuristic fails to find new improving solutions.

References

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.

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

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.

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.

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

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

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 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.

(Luciana S. Pessoa) Department of Computer Science, Universidade Federal Fluminense, Rua Passo da Pátria 156, Niterói, RJ 24210-240, Brazil.
E-mail address: lpessoa@ic.uff.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