# AUTOMATIC TUNING OF GRASP WITH EVOLUTIONARY PATH-RELINKING

L.F. MORÁN-MIRABAL, J.L. GONZÁLEZ-VELARDE, AND M.G.C. RESENDE

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

## 1. Introduction

Combinatorial optimization problems can often be "hard" to solve optimally using exact methods. Heuristics have been proved to find optimal or good suboptimal solutions in less time than required by exact methods. An example of this kind of heuristic is GRASP (Feo and Resende, 1989; 1995), which iteratively builds feasible solutions, and improves them applying a local search procedure. GRASP can be further hybridized with other intensification procedures, such as path-relinking (Glover, 1996) and evolutionary path-relinking (Festa et al., 2002; Resende and Werneck, 2004).

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

Automatic tuning procedures have been proposed in the literature, and have been shown to improve the performance of optimization algorithms when compared with variants using manually-tuned settings (see, e.g. Adenso-Diaz and Laguna (2006) and Hutter et al. (2007)). This paper proposes an automatic tuning procedure for parameters in a GRASP with evolutionary path-relinking by using a biased random-key genetic algorithm (Gonçalves and Resende, 2011).

*Date*: May 2013.

*Key words and phrases.* Randomized heuristics, GRASP, biased random-key genetic algorithm, automatic tuning.

AT&T Labs Research Technical Report. Published in "Proceedings of Hybrid Metaheuristics 2013 (HM 2013)," Ischia, M.J. Blesa et al., (Eds.), *Lecture Notes in Computer Science*, vol. 7919, pp. 62-77, 2013.

A BRKGA is an evolutionary algorithm based on the random-key genetic algorithm of Bean (1994). It evolves a population of solutions encoded as vectors of *random keys* applying genetic operators such as crossover and mutation. As a result, the algorithm returns the *fittest* individual of an evolved population (i.e. a best-valued solution). To apply a BRKGA for automatic tuning, we assume each individual encodes a set of parameters of the algorithm being tuned, and its fitness is a measure of the algorithm's performance using the encoded settings.

## 2. GRASP with evolutionary path-relinking

A *greedy randomized adaptive search procedure* (GRASP) (Feo and Resende, 1989; 1995) is a multi-start heuristic for combinatorial optimization. It applies local search to a series of solutions generated with a greedy randomized algorithm. As initially proposed, GRASP did not have any memory mechanism.Laguna and Martí (1999) introduced a memory mechanism in GRASP, hybridizing it with path-relinking (Glover, 1996). In the resulting heuristic, a pool of the best solutions found during the search is maintained by the algorithm. After each GRASP local minimum is produced, a solution is selected at random from the pool, and the solution space spanned by the two solutions is explored by path-relinking. Evolutionary path-relinking (Aiex et al., 2005; Festa et al., 2002; Resende and Werneck, 2004) uses the path-relinking operator in an attempt to improve the pool of elite solutions. Given a pool, evolutionary path-relinking applies path-relinking between pairs of pool solutions, updating the pool if better solutions are found.

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

**Algorithm 1:** GRASP with evolutionary path-relinking

The pseudo-code in Algorithm 1 illustrates a GRASP+evPR for a minimization problem. The algorithm begins in line 1 by initializing the incumbent solution

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

## 3. Automatic tuning using a BRKGA

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

Adenso-Diaz and Laguna (2006) and Hutter et al. (2007) were among the first to consider automatic tuning procedures. Adenso-Diaz and Laguna proposed CALIBRA, a framework which combines two different Design of Experiments approaches along with a local search procedure to tune up to five parameters. Hutter et al. proposed PARAMILS, a tuning methodology that combines stochastic local search procedures and mechanisms that tackle properties found in algorithm configuration problems. Both CALIBRA and PARAMILS have been shown to improve academic solvers and heuristics.

Festa et al. (2010) proposed an automatic tuning procedure using a biased random-key genetic algorithm (BRKGA). They propose the tuning procedure for an implementation of a GRASP with path-relinking heuristic for the generalized quadratic assignment problem (GQAP). They consider 30 parameters and show that their tuning improves algorithm performance with respect to manually tuned parameters on four out of five instances. Pedrola et al. (2012) recently applied the approach of Festa et al. (2010) to automatically tune a GRASP heuristic for the multilayer IP/MPLS-over-Flexgrid optimization problem. They use five small traffic instances to tune a simple GRASP with three parameters.

BRKGAs evolve population of vectors of random keys (or individuals) applying Darwin's principle of survival of the fittest (Gonçalves and Resende, 2011). A

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

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

Let $a$ and $b$ denote the elite and non-elite individuals to be mated, and let $c$ denote the resulting offspring. Mating is done with parameterized uniform crossover (Spears and DeJong, 1991), where a biased coin is tossed $n$ times, to determine from which parent the offspring will inherit each key. The coin has probability $p_h > 0.5$ to result in heads. For $i = 1, \ldots, n$, the $i$-th component of $c$ receives the $i$-th component of $a$ if the coin toss results in heads or the $i$-th component of $b$ otherwise. This way, $c$ has a greater chance to inherit the keys of its elite parent. Also, since such parent is selected from the smaller set $\mathcal{P}_e$, it has a greater chance of mating than a non-elite parent. Bean (1994) proposed a similar algorithm, except that parents are selected at random from the entire population and the coin flip does not necessarily favor the more fit parent.

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

3.1. **Encoding and decoding.** Let $n$ denote the number of algorithm configurations and parameters. These are encoded as a vector $\chi$ of $n$ real-valued random keys, each in the range $(0, 1]$. Suppose an algorithm configuration consists of a finite set of components, each of which can take on a single state. For example, *path-relinking type* is a component which can take on a single state from the set { *forward*, *backward*, *back&forth*, *mixed* }. Each state from these finite sets correspond to a different interval in the range $(0, 1]$. A set with $s$ states would associate state 1 with interval $(0, 1/s]$, state 2 with interval $(1/s, 2/s]$, and so on. To decide which state the $i$-th

**Data**: $|\mathcal{P}|, |\mathcal{P}_e|, |\mathcal{P}_m|, n, p_h$
Generate population $\mathcal{P}$ with individuals having $n$ random-keys;
**while** *stopping criterion is not satisfied* **do**
  Evaluate fitness of each new individual in $\mathcal{P}$;
  Partition $\mathcal{P}$ into sets $\mathcal{P}_e$ and $\mathcal{P}_{\bar{e}}$;
  Initialize next population: $\mathcal{P}^+ \leftarrow \mathcal{P}_e$;
  Generate mutants $\mathcal{P}_m$ each having $n$ random-keys $\in (0, 1]$;
  $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \mathcal{P}_m$;
  **for** $i \leftarrow 1$ **to** $|\mathcal{P}| - |\mathcal{P}_e| - |\mathcal{P}_m|$ **do**
    Select parent $a$ at random from $\mathcal{P}_e$;
    Select parent $b$ at random from $\mathcal{P}_{\bar{e}}$;
    **for** $j \leftarrow 1$ **to** $n$ **do**
      Toss biased coin having probability $p_h > 0.5$ of heads;
      **if** *Toss is heads* **then** $c[j] \leftarrow a[j]$;
      **else** $c[j] \leftarrow b[j]$;
    **end**
    $\mathcal{P}^+ \leftarrow \mathcal{P}^+ \cup \{c\}$;
  **end**
  $\mathcal{P} \leftarrow \mathcal{P}^+$;
**end**
**return argmin**$\{f(x) \mid x \in \mathcal{P}\}$

**Algorithm 2:** Biased random-key genetic algorithm.

component takes, the decoder identifies which interval contains the random key $\chi[i]$ and assigns the state corresponding to that interval to the component. A real-value parameter in the range $(l, u]$ and encoded as $\chi[i]$ is decoded as $l + (u - l) \times \chi[i]$. For example the $i$-th parameter *path-relinking truncation* can take on any value in the real interval $(0.2, 0.7]$. Therefore, the random key $\chi[i] = 0.44$ is decoded as $0.2 + (0.7 - 0.2) \times 0.44 = 0.42$. Note that each decoding of a vector of random keys is independent of the other, and hence can be parallelized to speed up the automatic tuning procedure.

## 4. GRASP+evPR for three optimization problems

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

4.1. **Set covering.** Let $\mathcal{U} = \{e_1, e_2, \ldots, e_n\}$ be a set of $n$ elements (i.e. the universe) and let $J = \{\mathcal{J}_1, \mathcal{J}_2, \ldots, \mathcal{J}_m\}$ be a collection of subsets of $\mathcal{U}$ with associated costs $c_1, c_2, \ldots, c_m$, respectively. The *set covering problem* (SCP) consists in finding a minimum cost collection of sets $\mathcal{S}$ from $J$, such that the union of the sets in $\mathcal{S}$ is $\mathcal{U}$. The cost of the cover is defined as $\sum_{\mathcal{J}_j \in \mathcal{S}} c_j$. Set covering is NP-hard (Karp, 1972).

Feo and Resende (1989) introduced a GRASP for set covering. Their construction procedure is based on the greedy algorithm of Johnson (1974). This greedy algorithm starts with an empty cover $\mathcal{S} = \emptyset$ and among unselected subsets, selects a subset $\mathcal{J}_j^*$ that maximizes the ratio $\kappa_j/c_j$, where $\kappa_j$ is the number of uncovered elements $e_i \in \mathcal{U}$ that become covered if $\mathcal{J}_j^*$ is added to the solution.

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

Three variants of GRASP construction are considered. The first uses a *fixed* value for $\alpha$, while the second selects at *random* a value of $\alpha$ from the uniform interval $[\alpha_{\min}, \alpha_{\max}]$ each time construction is called. The third variant uses a value of $\alpha$ selected at random from a finite set of $\alpha$ values with probabilities favoring those values that produced better solutions in previous

constructions. The number $n_\alpha$ of $\alpha$ values varies from 2 to 10 and the values go from 0 to 0.8. This approach, called *Reactive GRASP*, was proposed by Prais and Ribeiro (2000).

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

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

Path-relinking comes in several flavors. In *forward* path-relinking, $\mathcal{S}_l$ is the local optimum found by local search and $\mathcal{S}_t$ is a solution selected at random from the elite pool. In *backward* path-relinking the roles of $\mathcal{S}_l$ and $\mathcal{S}_t$ are reversed. In *back&forth* path-relinking, *backward* is applied first and then *forward* is applied next. In *mixed* path-relinking a path is started from $\mathcal{S}_l$ and another from $\mathcal{S}_t$, and both meet in the middle. Finally the entire path need not be explored and a truncated variant is possible where $\gamma = 20\%$ to 80% of the path is explored.

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

4.2. **Maximum cut.** Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph, where $\mathcal{V} = \{1, \ldots, n\}$ is the set of vertices and $\mathcal{E}$ is the set of edges, and let $w_{i,j}$ be the weights associated with edges $(i, j) \in \mathcal{E}$. The *maximum cut problem* (max-cut) consists in finding a cut $\mathcal{C} \subseteq \mathcal{E}$ of maximum weight which partitions the vertices in $\mathcal{V}$ in two non-empty sets. The weight of the cut $\mathcal{C}$ is defined as $\sum_{(i,j)\in\mathcal{C}} w_{i,j}$. The decision version of the max-cut problem was proven to be NP-Complete by Karp (1972).

Festa et al. (2002) introduced a GRASP with path-relinking heuristic for the max-cut problem. The construction phase of their heuristic uses a greedy function that takes into account the contribution to the objective function achieved by assigning a particular vertex into one of the subsets that define the cut, i.e. $\mathcal{C}$ and $\bar{\mathcal{C}}$. The greedy function is related to the sum of the weights of its outgoing arcs. Their local search procedure works by starting from the first elements of $\mathcal{C}$ and $\bar{\mathcal{C}}$ and in turn checks whether moving the elements from one set to the other leads to an improvement of the objective function.

In the path-relinking phase of Festa et al. (2002), a solution $y$ is represented as an $n$-dimensional binary vector, such that $y_i = 1$ if vertex $v_i \in \mathcal{C}$, and $y_i = 0$ if vertex $v_i \in \bar{\mathcal{C}}$. The procedure starts by computing the symmetric difference between the initial solution $x$ and the target solution $t$, defined to be $\Delta_{x,t} = \{i \mid x_i \neq t_i,\ i = 1, \ldots, n\}$. At each path-relinking step, an index from $\Delta_{y^k,t}$ is selected and used obtain the next solution in the path, i.e. $y^{k+1}$. This is done by evaluating, for each $i \in \Delta_{y^k,t}$, the cost change $g_i$ resulting from flipping the value of $y_i^k$. The greedy move selects from $\Delta_{x,t}$, the index corresponding to the largest $g_i$ value. Two path-relinking flavors, *forward* and *backward*, are considered. An evolutionary strategy is used as a post-processing phase.

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

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

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

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

4.3. **Node capacitated graph partitioning.** Given a node- and edge-weighted directed graph $G = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of nodes, and $\mathcal{E}$ is the set of arcs. For each node $v \in \mathcal{V}$, let $p_v \in \mathbb{Z}^+$ denote a non-negative integer node weight and for each arc $(u, v) \in \mathcal{E}$, let $q_{u,v} \in \mathbb{Z}^+$ denote a non-negative integer arc weight. In the *node capacitated graph partitioning* problem (NCGPP) we wish to partition the set of nodes into $n$ clusters $\{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_n\}$ such that, for $i = 1, 2, \ldots, n$, the weight sum of the nodes assigned to cluster $\mathcal{C}_i$ is no greater than the capacity $c_i \in \mathbb{Z}^+$ of the cluster. Furthermore, we seek the assignment that minimizes the edge weight sum $Q$ for all edges having endpoints assigned to different clusters. Mehrotra and Trick (1997) and Ferreira et al. (1998) were among the first to study this problem, proposing a branch and price algorithm and a branch and cut algorithm, respectively. Deng and Bard (2011) proposed a GRASP with path-relinking heuristic for capacitated clustering, essentially the same problem.

The GRASP+evPR heuristic we consider for the NCGPP was proposed by Morán-Mirabal et al. (2012). The construction phase builds a solution one node to cluster assignment at a time. Assignments are made as long as the capacity of the cluster is not exceeded. Let $\bar{\mathcal{V}}$ be the set of all yet-unassigned nodes. When a cluster $k$ being scanned is empty, a node $i \in \bar{\mathcal{V}}$ is assigned to $k$ with a probability proportional to the sum of edge weights between $i$ and all the nodes in $\bar{\mathcal{V}}$ (e.g. the greedy choice would select the node in $\bar{\mathcal{V}}$ with the maximum edge-weight sum). Once an assignment is made, the available capacity of the cluster $c_k$ is updated.

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

Once the construction phase produces a solution, local search attempts to reduce the sum of edge weights $Q$ by making changes in the assignment. Morán-Mirabal et al. (2012) propose three local search move types that scan the nodes in increasing order of their total node weight: *move-1*, *move-max*, and *swap-2*. In *move-1*, the procedure is restarted at the first node in the permutation, whereas in *move-max* it proceeds to the next node in the permutation. In *swap-2*, pairs of node assignments are considered for swapping. The number of pairs considered for swapping is limited by the tunable parameter $\beta$, where $0.01 \leq \beta \leq 0.3$. Three local search options are considered: *check-once*, *check-until*, and *variable*. In *check-once*, each node is scanned only once. In *check-until*, nodes are scanned until there is no further improving moves. Finally, in *variable*, at each invocation of local search a move type and an option are chosen at random.

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

An RCL is defined such that $h(i) \geq h^* - \alpha_p(h^* - h_*)$, where $h_* = \min\{h(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, $h^* = \max\{h(i) \mid i \in \Gamma(\Pi_s, \Pi_g)\}$, and the tunable parameter $\alpha_p \in \mathbb{R}$ such that $0 \leq \alpha_p \leq 1$. A move is selected from the RCL uniformly at random. The path-relinking operator of Morán-Mirabal et al. (2012) generates a sequence of neighboring solutions, each of which may be feasible or infeasible, and selects among the feasible solutions, a solution with the lowest value of $Q$. If all solutions explored are infeasible, then the path-relinking phase is deemed unsuccessful and GRASP+evPR continues to the next phase.

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

## 5. Experimental results

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

5.1. **Instances.** For the SCP we consider 20 benchmark instances from Beasley (1987). These instances are from three different problem sets, ten from set 4, eight from set 5, and two from set 6. All instances have 200 rows, 1000 or 2000 columns, and a graph density of 2% or 6%.

For max-cut we consider 21 benchmark instances from Helmberg and Rendl (1997). These instances divide into seven subsets of three which share size and structure. The subsets range from 800 to 2000 nodes, 1600 to 19900 edges, and a graph density of 0.2% to 6.0%.

For the NCGPP we use a 20 of the synthetic instances from Morán-Mirabal et al. (2012). The instances divide into four subsets of five which share size and structure. Each subset is created combining number of nodes $(200, 400)$ and number of clusters $(15, 25)$ in the set.

5.2. **The Experiments.** For each problem, two types of experiments are done, automatic tuning experiments and comparison of algorithm performance using automatically-tuned and manually-tuned settings. All implementations of GRASP+evPR and BRKGA use an implementation of the Mersenne Twister algorithm (Matsumoto and Nishimura, 1998) as their random-number generator. BRKGA was implemented using the API described in Toso and Resende (2012). All experiments were done using the Condor job control system (University of Wisconsin, 2012) which submitted jobs to either a cluster running Intel Xeon X5650 processors at 2.67 GHz, or a cluster running Intel Xeon E5530 processors at 2.4 GHz.

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

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

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

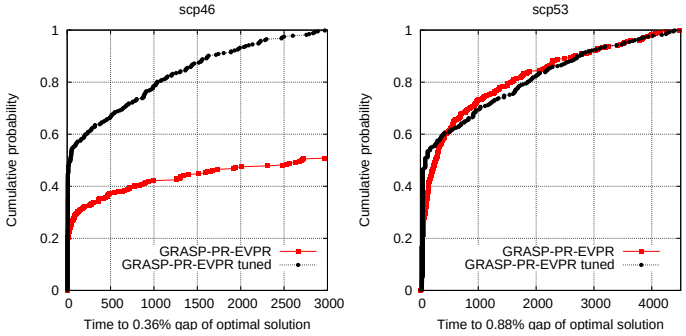
Most instances are automatically tuned to either *fixed* or *random* construction. Only two instances were tuned to *reactive*. All instances are tuned to *remove-all* as local search move and

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

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

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

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




most of them are tuned to be *best improving*. All instances are tuned to use *back&forth* path-relinking and most of them are *greedy*. Also the majority are tuned to a truncated path with length $\leq 70\%$. Finally all instances are tuned to use evolutionary path-relinking with $\alpha_q$ values that are at most 0.57.

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

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

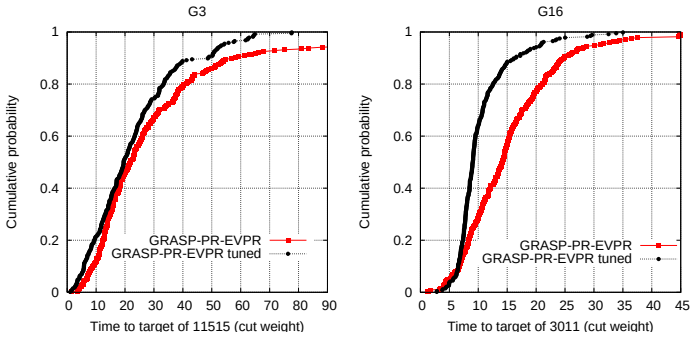
| | **Instance** | | **GRASP+evPR (manual)** | | **GRASP+evPR (tuned)** | |
|---|---|---|---|---|---|---|
| Name | Target | Max Time | Avg. Weight | Avg. TTT | Avg. Weight | Avg. TTT |
| G1 | 11511 | 1000 | 11514.92 | 27.409 | 11535.12 | 36.629 |
| G2 | 11498 | | 11503.72 | 11.035 | 11507.74 | 7.724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

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

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

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

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



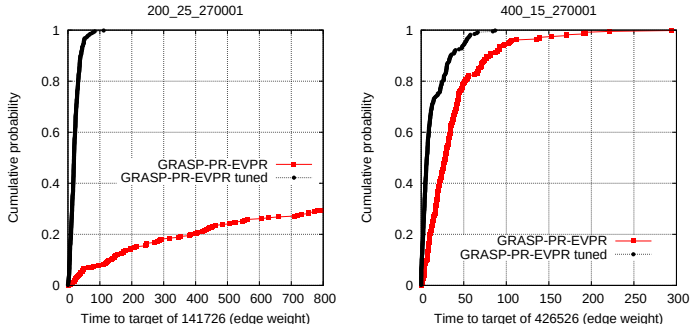

For the comparison experiments, a target cut weight was selected and a maximum runtime of 1000 seconds was set. Next, a total of 300 independent runs were made and the best solution value and time to target solution were saved. Table 2 shows the results. In all but four instances the average cut weight with *tuned* is greater or equal than that with *manual*. Note, however, that all four instances have an average cut weight found by *tuned* that is greater that the target and a difference of less than 1% from the average cut weight found by *manual*. Only two instances show no significant statistical difference when applying a Wilcoxon-Mann-Whitney text. In such two

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

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

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

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




cases, either *tuned* or *manual* performed faster, hence no dominance is noticed. Figure 2 shows examples of empirical runtime distributions for two of the instances considered. Each distribution shows how tuned GRASP+evPR performs similarly to manual GRASP+evPR, but tends to find the target in shorter times.

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

The majority of instances are tuned to *fixed* construction, and only 5 to *random* construction. Most $\alpha$ values are tuned to less than 0.10, which correspond to less randomized constructions. Most instances were tuned to use *check-until* local search with a *move-max* option, except for two instances that were tuned to use *variable*, and three that were tuned to *move*-1. Most instances chose *mixed* and *forward* path-relinking. More than half of the instances are tuned to use *greedy randomized* path-relinking with a complete path. Almost half of the instances are tuned not to use evolutionary path-relinking.

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

## 6. Concluding remarks

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

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

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

## Acknowledgment

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

## References

B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54:99–114, 2006.

R.M. Aiex, P.M. Pardalos, M.G.C. Resende, and G. Toraldo. GRASP with path-relinking for three-index assignment. *INFORMS J. on Computing*, 17:224–247, 2005.

J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal On Computing*, 2:154–160, 1994.

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

Y. Deng and J.F. Bard. A reactive GRASP with path relinking for capacitated clustering. *J. of Heuristics*, 17:119–152, 2011.

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. *J. of Global Optimization*, 6:109–133, 1995.

C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81:229–256, 1998.

P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized heuristics for the MAX-CUT problem. *Optimization Methods and Software*, 7:1033–1058, 2002.

P. Festa, J.F. Gonçalves, M.G.C. Resende, and R.M.A. Silva. Automatic tuning of GRASP with path-relinking heuristics with a biased random-key genetic algorithm. In *Experimental Algorithms, 9th International Symposium (SEA 2010)*, Lecture Notes in Computer Science, pages 338–349, Berlin, Heidelberg, 2010. Springer-Verlag.

F. Glover. Tabu search and adaptive memory programming – 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, 1996.

J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. *J. of Heuristics*, 17:487–525, 2011.

C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 1997.

F. Hutter, H.H. Hoos, and T. Sttzle. Automatic algorithm configuration based on local search. In *Proceedings of the Twenty-second Conference on Artificial Intelligence (AAAI07)*, pages 1152–1157, 2007.

D.S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9:256–278, 1974.

R.M. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, NY, 1972.

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

M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

A. Mehrotra and M.A. Trick. Cliques and clustering: A combinatorial approach. *Operations Research Letters*, 22:1–12, 1997.

L.F. Morán-Mirabal, J.L. Gonzalez-Velarde, and M.G.C. Resende. Randomized heuristics for handover minimization in mobility networks. Technical report, AT&T Labs Research, Florham Park, New Jersey, August 2012.

O. Pedrola, A. Castro, L. Velasco, M. Ruiz, J. P. Fernández-Palacios, and D. Careglio. CAPEX study for a multilayer IP/MPLS-over-flexgrid optical network. *J. of Optical Communications and Networking*, 4:639–650, 2012.

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

M.G.C. Resende and R.F. Werneck. A hybrid heuristic for the $p$-median problem. *J. of Heuristics*, 10:59–88, 2004.

W.M. Spears and K.A. DeJong. On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 230–236, 1991.

R.F. Toso and M.G.C. Resende. A C++ application programming interface for biased random-key genetic algorithms. Technical report, AT&T Labs Research, Florham Park, NJ, 2012.

University of Wisconsin. Condor high throughput computing, 2012. `research.cs.wisc.edu/condor`, last visited on June 25, 2012.

(Luis F. Morán-Mirabal) Tecnológico de Monterrey, Monterrey, Mexico.
*E-mail address*: `luismoranm@gmail.com`

(José Luis González-Velarde) Tecnológico de Monterrey, Monterrey, Mexico.
*E-mail address*: `gonzalez.velarde@itesm.mx`

(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*, M.G.C. Resende: `mgcr@research.att.com`