

RANDOMIZED HEURISTICS FOR THE REGENERATOR LOCATION PROBLEM

A. DUARTE, R. MARTÍ, M.G.C. RESENDE, AND R.M.A. SILVA

ABSTRACT. Telecommunication systems make use of optical signals to transmit information. The strength of a signal in an optical network deteriorates and loses power as it gets farther from the source, mainly due to attenuation. Therefore, to enable the signal to arrive at its intended destination with good quality, it is necessary to regenerate it periodically using regenerators. These components are relatively expensive and therefore it is desirable to deploy in the network as few of them as possible. In the regenerator location problem, we are given an undirected graph, positive edge lengths, and a parameter specifying the maximum length that a signal can travel before its quality deteriorates and regeneration is needed. The problem consists in determining paths that connect all pairs of nodes in the graph and, if necessary, locating single regenerators in some of those nodes such that the signal never travels more than the maximum allowed distance without traversing a regenerator node. In this paper we present a GRASP and a biased random-key genetic algorithm for the regenerator location problem. Computational experiments comparing the proposed solution procedure with other heuristics described in the literature illustrate the efficiency and effectiveness of our methods.

1. INTRODUCTION

Telecommunication systems make use of optical signals to transmit information. The strength of an optical signal deteriorates and loses power as it gets farther from the source, mainly due to attenuation. Therefore, to enable the signal to arrive at its intended destination with good quality, it is necessary to regenerate it periodically using regenerators. These components are relatively expensive and therefore it is desirable to deploy in the network as few of them as possible.

In the *regenerator location problem* (RLP), we are given an undirected graph $G = (V, E)$, where V is the node set, E is the set of edges, with each edge $(i, j) \in E$ having a real-valued length $d_{i,j} \in \mathbb{R}^+$. Furthermore, a parameter $D \in \mathbb{R}^+$ is given to specify the maximum length that a signal can travel before its quality deteriorates and regeneration is needed. The problem consists in determining paths that connect all pairs of nodes in the graph and, if necessary, locating single regenerators in some of those nodes. Between each pair of nodes $\{s, t\} \in V \times V$, a *path* $\{(s, v_1), (v_1, v_2), \dots, (v_k, t)\}$ connecting these nodes is formed by one or more path segments. In this context, a *path segment* consists of a sequence of consecutive edges $\{(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \dots, (v_{q-1}, v_q)\}$ in the path, satisfying the condition

Date: July 13, 2010.

Key words and phrases. Metaheuristics, network design, regenerator, location, GRASP, genetic algorithms.

AT&T Labs Research Technical Report.

that the *total length* of the segment

$$d_{v_i, v_{i+1}} + d_{v_{i+1}, v_{i+2}} + \dots + d_{v_{q-1}, v_q} \leq D.$$

If the total length of the path is no more than D , then the path consists of a single path segment. Otherwise, it will consist of two or more path segments. In this case, one or more regenerators will be located in the internal nodes of the path, separating consecutive path segments. The objective of the RLP is to determine paths that connect all pairs of nodes in the graph utilizing a minimum number of regenerators.

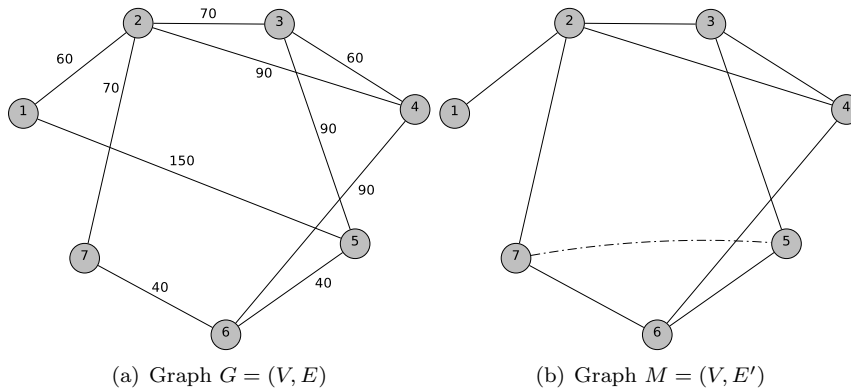


FIGURE 1. Example of a seven node graph G with edges lengths and corresponding communication graph M .

Figure 1(a) shows an example of a network with seven nodes and a maximum distance parameter $D = 100$. The numbers near the edges represent their lengths. Note that the length $d_{1,5} = 150$ of edge $(1, 5)$ is greater than D and therefore it cannot be part of any path. In this figure we can see that the shortest path from node 1 to node 3 is $\{(1, 2), (2, 3)\}$ with a total length of $60 + 70 = 130 > 100$. Therefore, it must be decomposed into two path segments $\{(1, 2)\}$ and $\{(2, 3)\}$ and a regenerator must be placed in node 2 to connect node 1 with node 3 using this path. The shortest path connecting nodes 1 and 5 is the single edge path $\{(1, 5)\}$ with length 150. Since edge $(1, 5)$ is longer than D , then it cannot be used in any path. Therefore, to connect node 1 with node 5 we identify the shortest feasible path to be $\{(1, 2), (2, 3), (3, 5)\}$ with total length $60 + 70 + 90 = 220$. This path must be decomposed into three path segments which requires two regenerators, one at node 2 and one at node 3. On the other hand, we can connect node 5 with node 7 using the shortest path $\{(5, 6), (6, 7)\}$ with a total length of $40 + 40 = 80$ and therefore we do not need to allocate any regenerator in this case. Finally, note that placing regenerators in nodes 2 and 3 allows for communication between all pairs of nodes in the graph.

Regenerator placement in the context of traffic engineering with restoration can be traced back to Yetginer and Karasan (2003). Gouveia et al. (2003) address an MPLS over WDM network design problem in which one type of constraint (called WDM path constraint) forbids path segments between two components that are longer than a given maximum length. Chen and Raghavan (2007) and Chen et al.

(2010) introduce the regenerator location problem and present a branch and cut procedure for the Steiner arborescence problem with a unit degree constraint on the root node, which they show to be equivalent to the RLP. Furthermore, they introduce three heuristic procedures for this problem. Computational experiment using 740 test problems show that the heuristics obtain the optimal solutions in 454 instances while the branch and cut finds the optimal solution in 536 instances. The same authors generalize the problem defining a set of nodes where regenerators can be placed and a set of nodes that need to be connected (Chen et al., 2009). The regenerator location problem was shown to be NP-hard by Flammini et al. (2009) and Chen et al. (2010).

Chen et al. (2010) describe the *communication graph* which they use in their algorithms for the RLP. Given the weighted graph $G = (V, E)$, we first delete all of the edges that have length greater than D . For all non-adjacent pairs of nodes, we add an edge between them of length equal to the length of the corresponding shortest path if the edge length in G is smaller than D . Finally, we disregard all length information. The resulting unweighted graph is referred to as $M = (V, E')$. If the resulting graph is complete, then there is no need for any regenerator. If the resulting graph is not connected, then the problem is infeasible. Alternatively, if the resulting graph is connected but not complete, then one or more regenerators will be required.

Figure 1(b) represents the communication graph M that results from the graph in the example of Figure 1(a). The dashed link in the graph between nodes 5 and 7 is the only link added during the procedure to create M from G .

The remainder of this paper is organized as follows. In Section 2, we review the greedy algorithm and local search procedure proposed in Chen et al. (2010). Section 3 combines a randomized variant of the greedy algorithm with the local search in a multi-start GRASP heuristic. A biased random-key genetic algorithm (BRKGA) for the RLP is proposed in Section 4. Computational experiments comparing the proposed GRASP and BRKGA with the heuristics of Chen et al. (2010) are described in Section 5. Concluding remarks are made in Section 6.

2. PREVIOUS CONSTRUCTIVE HEURISTICS

In this section, we review several constructive heuristics proposed in Chen et al. (2010) for the RLP, a greedy algorithm, and two heuristics called H1 and H2.

2.1. Greedy algorithm. The greedy algorithm takes as input the set of pairs of nodes that are not directly connected (NDC) in the communication graph and builds a set R of regenerator nodes, one node at a time. At each iteration, the procedure determines a node u^* whose inclusion in R enables the connection of the largest number of yet unconnected pairs in the communication graph. Node u^* is added to R and the communication graph is updated by adding to it the edges connecting those yet unconnected pairs that become connected by placing a regenerator in u^* .

Algorithm 1 shows pseudo-code of our adaptation of the greedy algorithm. It takes as input the communication graph $M = (V, E')$ and outputs the set of regenerator nodes R . In line 1 the set R of regenerator nodes is initialized empty and the set C of candidate regenerator nodes is initialized with all of the nodes in the graph. In line 2, the set E' of not directly connected (NDC) pairs is initialized with the complement of the set E' of edges in M . The main loop of the greedy algorithm

goes from line 3 to 23. It is repeated until M is complete, i.e. while $\bar{E}' \neq \emptyset$. For each candidate node $u \in C$, lines 4 to 17 compute the set $\mathcal{X}(u)$ of yet unconnected pairs in M that would become connected if node u where to house a regenerator. Let $g(u)$ denote the cardinality of set $\mathcal{X}(u)$. In line 5, $\mathcal{X}(u)$ is initialized empty and in line 6, $g(u)$ is initialized to zero. Line 7 computes the set $\mathcal{N}(u)$ of nodes adjacent to u in M . In this step we also initialize the set A of nodes accessible to u . In lines 8 to 10 we add to A all vertices adjacent to those in A that have a regenerator. For all pairs of nodes, both of which are in A , lines 11 to 16 verify if they are connected in M and if they are not connected we add them to $\mathcal{X}(u)$. In line 18, the node u^* that enables the connection of the largest number of yet unconnected pairs in M is selected. Ties are broken at random. In lines 18 to 21, sets R , C , E' , and \bar{E}' are updated to reflect the inclusion of node u^* in set R . In line 24, the set R of regenerator nodes is returned.

```

procedure Greedy
  Data: Communication graph:  $M = (V, E')$ 
  Result: Set of regenerator nodes:  $R \subseteq V$ 
  1  $R \leftarrow \emptyset; C \leftarrow V;$ 
  2  $\bar{E}' = \{(i, j) \in V \times V : (i, j) \notin E'\};$ 
  3 while  $\bar{E}' \neq \emptyset$  do
  4   for  $u \in C$  do
  5      $\mathcal{X}(u) \leftarrow \emptyset;$ 
  6      $g(u) \leftarrow 0;$ 
  7      $A \leftarrow \mathcal{N}(u) = \{v \in V : (u, v) \in E'\};$ 
  8     for  $v \in A \cap R$  do
  9        $A \leftarrow A \cup \mathcal{N}(v);$ 
 10    end
 11    for  $(i, j) \in A \times A$  do
 12      if  $(i, j) \notin E'$  then
 13         $\mathcal{X}(u) \leftarrow \mathcal{X}(u) \cup \{(i, j)\};$ 
 14         $g(u) \leftarrow g(u) + 1;$ 
 15      end
 16    end
 17  end
 18   $u^* \leftarrow \operatorname{argmax}\{g(u) : u \in C\};$ 
 19   $R \leftarrow R \cup \{u^*\};$ 
 20   $C \leftarrow C \setminus \{u^*\};$ 
 21   $E' \leftarrow E' \cup \mathcal{X}(u^*);$ 
 22   $\bar{E}' \leftarrow \bar{E}' \setminus \mathcal{X}(u^*);$ 
 23 end
 24 return  $R;$ 

```

Algorithm 1: Pseudo-code for GREEDY: Greedy algorithm for the RLP.

Consider as an example the communication graph depicted in Figure 1. For each candidate regeneration node u , Table 1 shows $\mathcal{X}(u)$, the set of pairs of nodes not

TABLE 1. $\mathcal{X}(u)$ and $g(u)$ for all candidate nodes u in first iteration of greedy algorithm on example of Figure 1(b).

u	$\mathcal{X}(u)$	$g(u)$
1	\emptyset	0
2	$\{(1, 3), (1, 4), (1, 7), (3, 7), (4, 7)\}$	5
3	$\{(4, 5), (2, 5)\}$	2
4	$\{(2, 6), (3, 6)\}$	2
5	$\{(3, 7), (3, 6)\}$	2
6	$\{(4, 7), (4, 5)\}$	2
7	$\{(2, 6), (2, 5)\}$	2

connected that become connected if a regenerator is placed in node u and $g(u)$, the cardinality of $\mathcal{X}(u)$.

In iteration 1 of the greedy algorithm, node $u^* = 2$ is selected, since $g(2) = 5$ is the maximum $g(u)$ for all $u \in C$. Edges $(1, 3)$, $(1, 4)$, $(1, 7)$, $(3, 7)$, and $(4, 7)$ are added to graph M resulting in the graph shown in Figure 2(a). Similarly, in iteration 2, node $u^* = 7$ is selected and edges $(1, 5)$, $(1, 6)$, $(2, 5)$, $(3, 6)$, $(4, 5)$, and $(2, 6)$ are added to graph M making it complete, as can be seen in Figure 2(b). The set R of regeneration nodes obtained with this greedy algorithm is therefore $\{2, 7\}$.

2.2. H1 heuristic. The aim of the H1 heuristic is to find a spanning tree on the communication graph M with the maximum number of leaf nodes. In this way, the heuristic aims to minimize the number of internal nodes in the spanning tree. Regenerators are assigned to these internal nodes. The algorithm starts by computing $u^* \leftarrow \mathbf{argmin}\{\deg(u) : u \in V\}$, a node of minimum degree (ties are broken by favoring the node with smallest index) and initializes the set of nodes in the spanning tree, $S \leftarrow \{u^*\}$, and its complement, $\bar{S} = V \setminus \{u^*\}$. The set R of regenerators is initialized empty, i.e. $R \leftarrow \emptyset$. H1 then calls the recursive function, $\text{Tree}(u^*, S, \bar{S}, R)$, described in Algorithm 2.

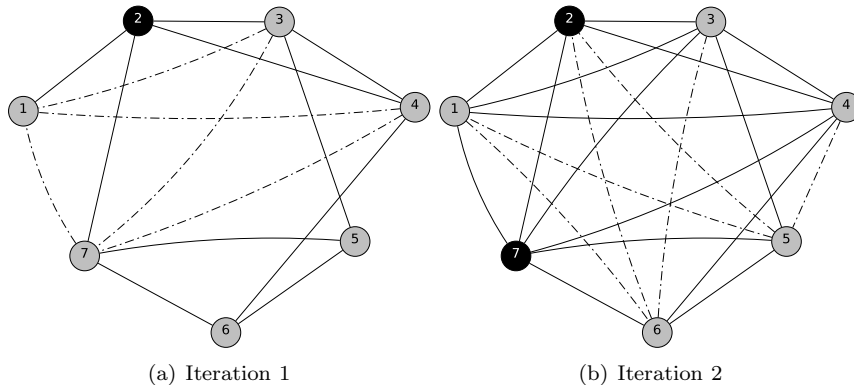


FIGURE 2. Communication graph on two iterations of the greedy algorithm.

```

procedure Tree
  Data:  $u, S, \bar{S}, R$ 
  Result:  $R$ 
  1  $\mathcal{U}(u) \leftarrow \mathcal{N}(u) \cap \bar{S}$ ;
  2 if  $|S| > 1$  then
  3    $S \leftarrow S \cup \mathcal{U}(u)$ ;
  4    $\bar{S} \leftarrow V \setminus S$ ;
  5 end
  6 while  $\mathcal{U}(u) \neq \emptyset$  do
  7   Compute  $\deg_{\bar{S}}(v)$  for all  $v \in \mathcal{U}(u)$ ;
  8    $u^* \leftarrow \mathbf{argmax}\{\deg_{\bar{S}}(v) : v \in \mathcal{U}(u)\}$ ;
  9   if  $\deg_{\bar{S}}(u^*) > 0$  then
 10     $R \leftarrow R \cup \{u^*\}$ ;
 11    Tree( $u^*, S, \bar{S}, R$ );
 12   end
 13    $\mathcal{U}(u) \leftarrow \mathcal{U}(u) \setminus \{u^*\}$ ;
 14 end
 15 return

```

Algorithm 2: Pseudo-code for **Tree** used in H1.

Tree takes as input the lowest degree vertex u of the graph and constructs a spanning tree recursively. In step 3 of Algorithm 2 the spanning tree is augmented with the unspanned nodes that are neighbors of node u . Note that in the previous call to the procedure a regenerator is placed in node u^* in step 10. Applying this algorithm to the example in Figure 1, we obtain the same solution as the one obtained by **Greedy**.

2.3. H2 heuristic. Heuristic H2 in some sense resembles the two previous methods. In each step, it first identifies the node of lowest degree and then examines the neighbors of these nodes and selects the one with the highest degree to assign a regenerator. Once a regenerator is assigned to a node, the communication graph is updated by adding edges with endpoints in newly connected pairs of nodes. H2 resembles **Greedy** in the sense that it updates the communication graph at each step and it resembles H1 in the sense that it is based on the degree of the nodes.

3. GRASP FOR THE RLP

A GRASP, or greedy randomized adaptive search procedure, is a multi-start or iterative process in which each iteration consists of two phases: construction and local search (Feo and Resende, 1989; 1995). The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found after the application of the local search phase. At each iteration of the construction phase, GRASP maintains a set of candidate elements CL that can be feasibly added to the partial solution under construction. Every candidate element is evaluated according to a greedy function in order to select the next element to be added to the construction. A restricted candidate list (RCL) is created with the best

elements in CL . This is the greedy aspect of the method. The element to be added to the partial solution is randomly selected from those in the RCL . This is the probabilistic aspect of the heuristic. Once the selected element is added to the partial solution, the candidate list CL is updated and its elements evaluated. This is the adaptive aspect of the heuristic. We refer the reader to Resende and Ribeiro (2003; 2010) for two recent reviews of GRASP.

3.1. Construction procedures. We implemented the three constructive algorithms CG , $C1$, and $C2$ based on Greedy, H1, and H2 (Chen et al., 2010), respectively, described in Section 2.

Given the communication graph M , the CG constructive method builds a set R of regenerator nodes, one node at a time. At each iteration, CG first determines the set CL of candidate nodes to house a regenerator, which is simply the set of nodes in M in which no regenerator has been previously placed. Then, for each candidate node $u \in CL$, it computes the set $\mathcal{X}(u)$ of yet unconnected pairs in M that would become connected if node u where to house a regenerator. Let $g(u)$ denote the cardinality of set $\mathcal{X}(u)$.

The restricted candidate list, RCL , is formed with those nodes with a relatively good g -value. To this end, the method computes g_{min} and g_{max} as, respectively, the minimum and maximum values of g in CL . The algorithm next constructs a RCL with all the nodes in CL having a g -value greater than or equal to a specified cutoff value

$$(1) \quad g_{min} + \alpha \cdot (g_{max} - g_{min}),$$

where

$$(2) \quad g_{min} = \min_{u \in CL} g(u) \quad \text{and} \quad g_{max} = \max_{u \in CL} g(u).$$

CG randomly selects an element u^* in RCL , adds it to the set R and updates the communication graph (by adding to it the edges connecting those yet unconnected pairs that become connected by placing a regenerator in u^*). The construction ends when the graph M is complete. We test in the computational experiments different values for α .

The H1 heuristic tries to find a spanning tree S on the communication graph M with the maximum number of leaf nodes, minimizing in this way the number of internal nodes in S in which regenerators are assigned. At each iteration, the set CL of candidates nodes on which to place a regenerator, referred to as $\mathcal{U}(u)$ in Algorithm 2, is computed as $\mathcal{N}(u) \cap \bar{S}$, where \bar{S} is the complement of the spanning tree S under construction, and u is the node in which a regenerator was placed in the previous iteration. H1 selects the node $u^* \in CL$ with maximum degree $\deg_{\bar{S}}(u^*)$ with respect to \bar{S} and includes it in S . In the randomized construction $C1$, instead of selecting the node $u^* \in CL$ with maximum degree, we compute the restricted candidate list, RCL , with all the nodes in CL having a degree in \bar{S} greater than or equal to a specified threshold value

$$(3) \quad deg_{min} + \alpha \cdot (deg_{max} - deg_{min}),$$

where

$$(4) \quad deg_{min} = \min_{u \in CL} deg_S(u) \quad \text{and} \quad deg_{max} = \max_{u \in CL} deg_S(u).$$

Then, *C1* randomly selects a node in *RCL* and adds it, together with its adjacent vertices, to the spanning tree *S* under construction.

Finally, constructive method *C2* determines at each iteration the set *CL* of candidate nodes to house a regenerator as the set of nodes in *M* in which no regenerator has been previously placed. Then, for each candidate node $u \in CL$, it computes its degree $d(u)$ in *M*. Instead of selecting the node with the largest degree (as H2), *C2* computes a restricted candidate list with those nodes with a degree greater than or equal to a specified threshold value, as in (3), and randomly selects a node u^* in *RCL*. The method adds u^* to the set *R* and updates the communication graph by adding to it the edges connecting those yet unconnected pairs that become connected by placing a regenerator in u^* .

procedure Move

Data: Communication graph $M = (V, E')$, regenerator set *R*, nodes *i* and *j*

Result: New regenerator set *R'*

```

1  $R' \leftarrow R \setminus \{i, j\}$ ;
2  $C \leftarrow V \setminus R'$ ;
3 for  $u \in (\mathcal{N}(i) \cup \mathcal{N}(j)) \setminus R'$  do
4   if  $\mathcal{N}(u) \cap R' = \emptyset$  then
5      $C \leftarrow C \cap \mathcal{N}(u)$ ;
6   end
7   if  $C = \emptyset$  then
8     return nil;
9   end
10 end
11 Select  $v \in C$ ;
12  $R' \leftarrow R' \cup \{v\}$ ;
13 return  $R'$ ;

```

Algorithm 3: Pseudo-code for MOVE: a 2-1 or 1-0 move in local search for the RLP.

3.2. Local search. Given two nodes *i* and *j* in the set *R* of regenerator nodes, procedure `Move(i, j)` shown in Algorithm 3 attempts to remove both regenerators from nodes *i* and *j* and replace them with a single regenerator (that eventually could be in placed in *i* or *j*). A set *C* of potential candidates to host a regenerator is initialized in step 2. In steps 3 to 10, each non-regenerator node *u* adjacent to either *i* or *j* is analyzed. If *u* is not adjacent to an existing regenerator node (distinct from *i* and *j*), then in step 5 the set of potential candidate nodes is updated, eliminating from *C* the nodes not adjacent to *u*. If *C* is empty, then nodes *i* and *j* cannot be replaced in *R* with fewer than two nodes and the procedure returns a `nil` indicator.

Otherwise, any element in the final set C can be selected to replace i and j . Note that i , j , or both i and j can be in C . In step 11 a node v is selected from C and is added to R' in step 12. A 1-0 move results if either i or j is selected. Otherwise, a 2-1 move is applied. The node in C with the largest degree in the communication graph is selected. Ties are broken by node index.

The local search procedure scans the regenerator nodes in the order given by their degree in the original graph (where the node with the lowest degree is examined first). Nodes with a low degree are expected to be good candidates to move their regenerator to an alternative node. The procedure then applies a first improvement strategy, performing the first associated move in its exploration. The local search finishes when there is no further improvement and returns the local optimum found.

4. BIASED RANDOM-KEY GENETIC ALGORITHM

Genetic algorithms with random keys, or *random-key genetic algorithms* (RKGA), were first introduced by Bean (1994) for solving combinatorial optimization problems involving sequencing. In a RKGA, chromosomes are represented as vectors of randomly generated real numbers in the interval $[0, 1]$. A deterministic algorithm, called a *decoder*, takes as input a solution vector and associates with it a solution of the combinatorial optimization problem for which an objective value or fitness can be computed.

A RKGA evolves a population of random-key vectors over a number of iterations, called *generations*. The initial population is made up of p vectors of random-keys. Each component of the solution vector is generated independently at random in the real interval $[0, 1]$. After the fitness of each individual is computed by the decoder in generation k , the population is partitioned into two groups of individuals: a small group of p_e *elite* individuals, i.e. those with the best fitness values, and the remaining set of $p - p_e$ *non-elite* individuals. To evolve the population, a new generation of individuals must be produced. All elite individual of the population of generation k are copied without modification to the population of generation $k + 1$. RKGAs implement mutation by introducing *mutants* into the population. A mutant is simply a vector of random keys generated in the same way that an element of the initial population is generated. At each generation, a small number (p_m) of mutants is introduced into the population. With the p_e elite individuals and the p_m mutants accounted for in population $k + 1$, $p - p_e - p_m$ additional individuals need to be produced to complete the p individuals that make up the new population. This is done by producing $p - p_e - p_m$ offspring through the process of mating or crossover.

Bean (1994) selects two parents at random from the entire population to implement mating in a RKGA. A *biased random-key genetic algorithm*, or BRKGA (Gonçalves and Resende, 2009), differs from a RKGA in the way parents are selected for mating. In a BRKGA, each element is generated combining one element selected at random from the elite partition in the current population and one from the non-elite partition. Repetition in the selection of a mate is allowed and therefore an individual can produce more than one offspring in the same generation. *Parametrized uniform crossover* (Spears and DeJong, 1991) is used to implement mating in BRKGAs. Let $\rho_e > 0.5$ be the probability that an offspring inherits the vector component of its elite parent. Let n denote the number of components in

the solution vector of an individual. For $i = 1, \dots, n$, the i -th component $c(i)$ of the offspring vector c takes on the value of the i -th component $e(i)$ of the elite parent e with probability ρ_e and the value of the i -th component $\bar{e}(i)$ of the non-elite parent \bar{e} with probability $1 - \rho_e$.

When the next population is complete, i.e. when it has p individuals, fitness values are computed for all of the newly created random-key vectors and the population is partitioned into elite and non-elite individuals to start a new generation.

A BRKGA searches the solution space of the combinatorial optimization problem indirectly by searching the continuous n -dimensional hypercube, using the decoder to map solutions in the hypercube to solutions in the solution space of the combinatorial optimization problem where the fitness is evaluated.

We propose a BRKGA heuristic for the regenerator location problem. As in the previous algorithms, we denote vertex and edge sets of the communication graph M as V and E' , respectively. The heuristic encodes the solution as a $|V|$ -vector x of random keys. The i -th element of x corresponds to node $i \in V$. To decode x , we apply the greedy algorithm `Decoder` shown in Algorithm 4.

procedure Decoder

Data: Communication graph $M = (V, E')$ and vector x of random keys

Result: Set of regenerator nodes: $R \subseteq V$

```

1  $R \leftarrow \emptyset; C \leftarrow V;$ 
2  $\bar{E}' = \{(i, j) \in V \times V : (i, j) \notin E'\};$ 
3 Order the nodes in  $C$  w.r.t. their associated  $x$  value;
4 while  $\bar{E}' \neq \emptyset$  do
5   | Select the next node  $k \in C$  following the  $x$ -order;
6   |  $\mathcal{X}(k) \leftarrow \emptyset;$ 
7   |  $A \leftarrow \mathcal{N}(k) = \{v \in V : (k, v) \in E'\};$ 
8   | for  $v \in A \cap R$  do
9   |   |  $A \leftarrow A \cup \mathcal{N}(v);$ 
10  | end
11  | for  $(i, j) \in A \times A$  do
12  |   | if  $(i, j) \notin E'$  then
13  |     |  $\mathcal{X}(k) \leftarrow \mathcal{X}(k) \cup \{(i, j)\};$ 
14  |   | end
15  | end
16  |  $R \leftarrow R \cup \{k\};$ 
17  |  $C \leftarrow C \setminus \{k\};$ 
18  |  $E' \leftarrow E' \cup \mathcal{X}(k);$ 
19  |  $\bar{E}' \leftarrow \bar{E}' \setminus \mathcal{X}(k);$ 
20 end
21 return  $R;$ 

```

Algorithm 4: Pseudo-code for `DECODER`.

In the initialization of the `Decoder` algorithm (see Algorithm 4) we order the nodes in C , initially with all the nodes in V , according to the random key vector x . In this way, nodes associated with components in x with a relative large value (close to 1) come first. The main loop of the `Decoder` algorithm goes from line 4 to 20. It is repeated until M is complete, i.e. while $\bar{E}' \neq \emptyset$. In this loop nodes are selected following the order induced by x . For each candidate node $k \in C$, lines 6 to 15 compute the set $\mathcal{X}(k)$ of yet unconnected pairs in M that would become connected if node k were to house a regenerator. In lines 16 to 19, sets R , C , E' , and \bar{E}' are updated to reflect the inclusion of node k in set R . In line 21, the set R of regenerator nodes is returned.

5. EXPERIMENTAL RESULTS

This section describes the computational experiments that we performed to first test the efficiency of our different procedures and then compare them with a number of methods in Chen et al. (2010). We have implemented the methods in Java SE 6 and all the experiments were conducted on a Pentium 4 computer at 3 GHz with 2 GB of RAM.

In our computational experiments we used the instances from Chen et al. (2010) that the authors kindly made available to us. Specifically, these graphs come from two different sources, the M -graph instances, in which they directly generated the communication graph, and random instances, in which they first generate a graph with random edge lengths and then compute the communication graph. The authors sent us a total of 280 instances, 200 from the first set and 80 from the second one, with $n = 40, 60, 80$, and 100.

In each experiment, we compute for each instance the overall best solution value, *BestValue*, obtained by all executions of the methods considered. Then, for each method, we compute the relative percentage deviation between the best solution value obtained with that method and *BestValue* for that instance. We report the average of this relative percentage deviation (Dev.) across all the instances considered in each particular experiment. We finally report, for each method, the number of instances (#Best) in which the value of the best solution obtained with this method matches *BestValue*.

In our preliminary experimentation we consider a set of 20 instances randomly selected from those instances in our set with $n = 80, 100$. In the first preliminary experiment, we study the parameter α in constructive methods CG, C1, and C2. Specifically, we test three values of this parameter on each method: 0.3, 0.6, and 0.9. We run CG, C1, and C2 100 times, thus obtaining 100 solutions for each method and instance pair. Table 2 reports, for this set of 20 instances and each value of α tested, the values of #Best and Dev. We also report the average objective function value (number of regenerators), Value, and the CPU time in seconds.

The results in Table 2 show that the best outcomes are obtained when the constructive method CG is run with a value of $\alpha = 0.9$. However it consumes longer running times than its competing GRASP constructions C1 and C2.

In the second preliminary experiment, we compare the constructive with the local search methods for the RLP. Specifically we target the previous constructive and improvement methods Greedy+LS, H1+LS and H2+LS by Chen et al. (2010), and our corresponding GRASP methods CG+LS, C1+LS and C2+LS. We use the best value of α for each method according to the results of the first experiment.

TABLE 2. GRASP Constructive methods

	Value	#Best	Dev(%)	CPU(sec.)
CG				
0.3	7.00	7	25.77	12935.50
0.6	5.65	13	5.92	10001.00
0.9	5.35	19	0.45	7958.05
C1				
0.3	6.80	8	20.42	767.75
0.6	6.10	10	10.79	747.25
0.9	6.25	11	12.10	761.55
C2				
0.3	6.30	7	16.59	573.85
0.6	6.10	11	10.64	562.40
0.9	5.95	12	8.51	543.65

TABLE 3. Constructive with local search methods

	Value	#Best	Dev(%)	CPU(sec.)
Greedy+LS	5.65	11	8.96	176.20
H1+LS	5.70	11	9.96	48.95
H2+LS	5.55	14	7.50	14.05
GC(0.9)+LS	5.25	18	1.45	8193.05
C1(0.6)+LS	5.15	20	0.00	2238.25
C2(0.9)+LS	5.25	18	1.70	1093.55

In particular we set α to 0.9, 0.6, and 0.9 in CG, C1 and C2 respectively. As in the previous experiment we consider the set of 20 instances and report, in Table 3, Value, #Best, Dev., and CPU time.

Table 3 shows that our three new approaches based on the GRASP methodology are able to improve upon previous methods also based on construction plus local search. Specifically, CG, C1, and C2 present an average percent deviation from the best solutions of 1.45, 0.00, and 1.70, respectively, while Greedy+LS, H1+LS, and H2+LS obtain 8.96, 9.96, and 7.50, respectively. It must be noted that these three previous methods are deterministic procedures that can only be run once. The randomized nature of the GRASP construction allows multiple runs, which provide a high-quality best solution over these runs. As expected, the GRASP based constructions present much longer running times than the deterministic methods.

In our final experiment, we compare our best method, C1+LS, with the parameter α set to 0.6 with the best previous method, H2+LS. We also include in this experiment the biased random key genetic algorithm, BRKGA, described in the previous section. Table 4 reports the Value, #Best, Dev., and CPU time over the entire set of 280 instances divided according to their size.

TABLE 4. Comparison of best methods

	Value	#Best	Dev(%)	CPU(sec.)
<i>n</i> = 40				
H2+LS	3.96	59	4.46	3.61
C1(0.6)+LS	3.77	70	0.00	253.69
BRKGA	3.83	66	1.71	1445.57
<i>n</i> = 60				
H2+LS	4.37	62	1.74	6.43
C1(0.6)+LS	4.26	70	0.00	646.46
BRKGA	4.34	64	1.48	4462.64
<i>n</i> = 80				
H2+LS	4.90	46	8.25	10.26
C1(0.6)+LS	4.50	70	0.00	1356.20
BRKGA	4.67	58	4.39	9742.37
<i>n</i> = 100				
H2+LS	5.27	50	5.69	16.20
C1(0.6)+LS	4.91	70	0.00	2393.73
BRKGA	5.00	62	2.13	20169.06

Table 4 shows the merit of the proposed GRASP procedure. Our C1+LS implementation consistently produces the best solutions with percent deviations smaller than those of the competing methods (and with number of best solutions found larger than the others). On the other hand, the H2+LS algorithm is able to obtain relatively good solutions in short computational time. The BRKGA method performs well, better than H2+LS, although it consumes significantly longer running times than the other methods.

We ran C1+LS 100 times on a representative instance, stopping when a solution with objective value equal to the best known for this instance was found. For each run we recorded the running time. Each run was independent of the other, using a different initial seed for the random number generator. With these 100 running times, we plot the time-to-target plots (run time distributions), shown in Figure 3. This experiment confirms the expected exponential runtime distribution for our GRASP. Therefore, linear speed is expected if the algorithm is implemented in parallel.

6. CONCLUDING REMARKS

In this paper we propose several heuristics based on GRASP and Genetic Algorithms for the regenerator location problem. The proposed GRASP heuristics consist of three randomized greedy construction procedures and a local search procedure. The genetic algorithm implements a random key design. Overall, experiments with 280 previously reported instances were performed to compare our procedures with previous heuristics for this problem. The comparison favors our GRASP algorithm.

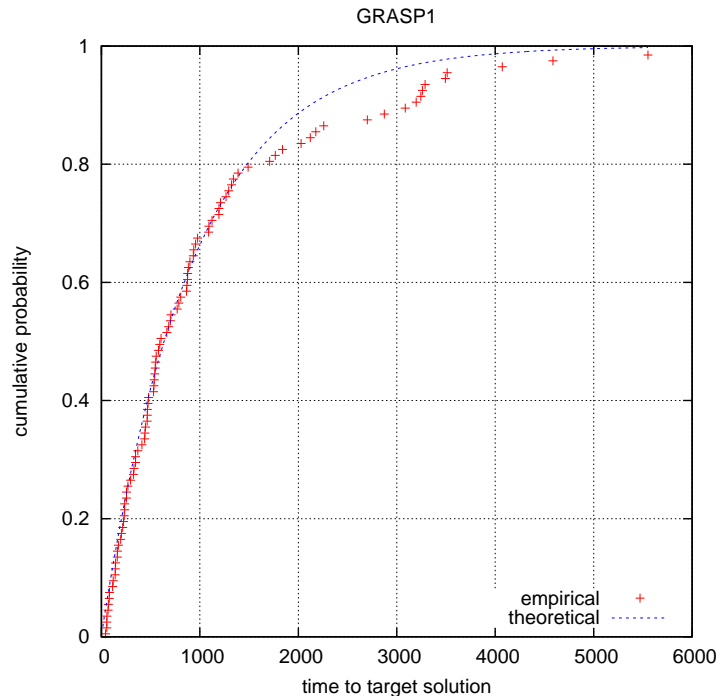


FIGURE 3. Time to target plots (run time distributions) for C1+LS.

ACKNOWLEDGMENT

The research of A. Duarte and R. Martí was partially supported by the Ministerio de Ciencia e Innovación of Spain (TIN2009-07516). The research of R.M.A. Silva was partially supported by the Brazilian National Council for Scientific and Technological Development (CNPq) and the Foundation for Support of Research of the State of Minas Gerais, Brazil (FAPEMIG). The authors thank Profs. Chen, Ljubić, and Raghavan for making their instances available to us.

REFERENCES

- J.C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA J. on Computing*, 6:154–160, 1994.
- Si Chen and S. Raghavan. The regenerator location problem. In *Proceedings of the 2007 International Network Optimization Conference (INOC 2007)*, 2007.
- Si Chen, I. Ljubić, and S. Raghavan. The generalized regenerator location problem. In M.G. Scutellà, editor, *Proceedings of the 2009 International Network Optimization Conference (INOC 2009)*, 2009.
- Si Chen, I. Ljubić, and S. Raghavan. The regenerator location problem. *Networks*, 55:205–220, 2010.
- 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(2):109–133, 1995.

- M. Flammini, A. Machetti, G. Monaco, L. Moscardelli, and S. Zaks. On the complexity of the regenerator placement problem in optical networks. In *Proceedings of SPAA 2009*, pages 154–162, Calgary, 2009.
- J.F. Gonçalves and M.G.C. Resende. Biased random-key genetic algorithms for combinatorial optimization. Technical report, AT&T Labs Research, Florham Park, NJ 07932, 2009. (<http://www.research.att.com/~mgcr/doc/srkgg.pdf>).
- L. Gouveia, P. Patrício, A. Sousa, and R. Valadas. MPLS over WDM network design with packet level QoS constraints based on ILP models. In *Proceedings of the IEEE Infocom*, volume 1, pages 576–586, 2003.
- M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 219–250. Kluwer Academic Publishers, 2003.
- 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 281–317. Springer Science+Business Media, 2nd edition, 2010.
- 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.
- E. Yetginer and E. Karasan. Regenerator placement and traffic engineering with restoration in GMPLS networks. *Photonic Network Commun.*, 6:139–149, 2003.
- (A. Duarte) DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN, UNIVERSIDAD REY JUAN CARLOS, SPAIN.
E-mail address: `abraham.duarte@urjc.es`
- (R. Martí) DEPARTAMENTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA, FACULTAD DE MATEMÁTICAS, UNIVERSIDAD DE VALENCIA, DR. MOLINER 50, 46100 BURJASSOT (VALENCIA), SPAIN
E-mail address: `rafael.marti@uv.es`
- (M.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`
- (R.M.A. Silva) COMPUTATIONAL INTELLIGENCE AND OPTIMIZATION GROUP, DEPT. OF COMPUTER SCIENCE, FEDERAL UNIVERSITY OF LAVRAS, C.P. 3037, CEP 37200-000, LAVRAS, MG, BRAZIL AND CENTRO DE INFORMÁTICA (CIN), FEDERAL UNIVERSITY OF PERNAMBUCO, AV. PROFESSOR LUÍS FREIRE S/N, CIDADE UNIVERSITÁRIA, RECIFE, PE, BRAZIL.
E-mail address, R.M.A. Silva: `rmas@dcc.ufla.br`