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 new implementations of previous heuristics and two new heuristics, a GRASP and a biased random-key genetic algorithm, for the regenerator location problem. Computational experiments comparing the proposed solution procedures with previous 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* $\{(s, v_1), (v_1, v_2), \ldots, (v_k, t)\}$ connecting these nodes is formed by one or more path segments. In this context, a *path segment* consists of a sequence of consecutive edges $\{(v_i, v_{i+1}), (v_{i+1}, v_{i+2}), \ldots, (v_{q-1}, v_q)\}$ in the path, satisfying the condition

Date: November 18, 2011.

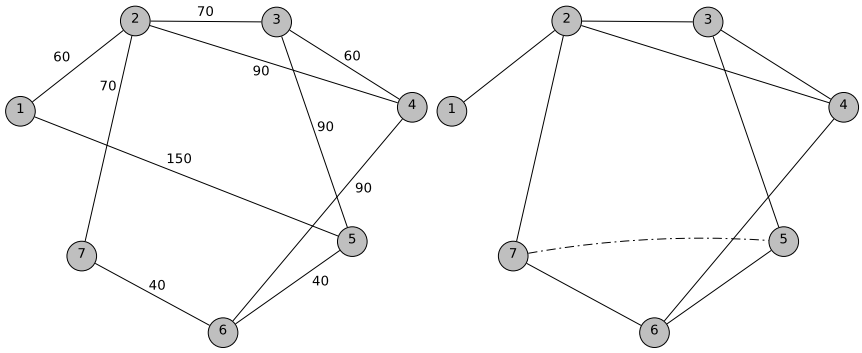
Key words and phrases. Metaheuristics, network design, regenerator, location, GRASP, genetic algorithms.

AT&T Labs Research Technical Report.

that the *total length* of the segment

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

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.



(a) Graph $G = (V, E)$ (b) Graph $M = (V, E')$

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 7 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 a multi-protocol label switching (MPLS) over wave division multiplexing (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 experiments 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. We add an edge between all non-adjacent pairs of nodes having 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 propose new and efficient implementations of the three constructive algorithms originally proposed in Chen et al. (2010). In Section 3 we point out an error in the description of a local search proposed in Chen et al. (2010). We then combine randomized variants of three greedy algorithms proposed in Chen et al. (2010) with a correct version of their local search, resulting in three multi-start GRASP heuristics. 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. Constructive heuristics

In this section, we adapt several constructive heuristics proposed in Chen et al. (2010) for the RLP: a greedy algorithm, and two heuristics called `H1` and `H2`. We propose efficient implementations of these procedures that, as will be shown in our computational experience, outperform their original implementations by up to three orders of magnitude.

2.1. **Greedy algorithm.**

The greedy algorithm takes as input the set of pairs of nodes in the communication graph that are not directly connected (NDC) and builds a set R of regenerator nodes, one node at a time. At each iteration, the procedure computes, for each node u, $g(u)$, the number of yet-unconnected pairs in the communication graph that become connected with its inclusion in R. The method then determines a node u^* with maximum g-value. Node u^* is added to

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

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^*.

In our implementation of this method, we store for each vertex u, the set of its neighbors, $N(u)$, in a hash-table. Therefore, basic operations, such as adding, removing, or checking the membership of an element in $N(u)$ can be practically performed in constant time. For example, to calculate $g(u)$ we need to compute the Cartesian product of $N(u)$ in the communication graph, which is extremely fast with the hash table. Moreover, at the same time that we compute $g(u)$ for each vertex u, we store the best one so far, and we do not need to go through them again to determine u^*, as apparently the original implementation of Chen et al. (2010) does.

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 $\bar{E}'$ 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 20. It is repeated until M is complete, i.e. while $\bar{E}' \neq \emptyset$. For each candidate node $u \in C$, lines 4 to 14 compute the set $\mathcal{X}(u)$ of yet-unconnected pairs in M that would become connected if node u were 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. For all pairs of nodes, both of which are in $\mathcal{N}(u)$, lines 8 to 13 verify if they are connected in M and if they are not connected we add them to $\mathcal{X}(u)$. In line 15, 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 15 to 18, 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.

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 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 $(4,7)$

procedure `Greedy`
  Data: Communication graph: $M = (V, E')$
  Result: Set of regenerator nodes: $R \subseteq V$
$R \leftarrow \emptyset; C \leftarrow V;$
$\bar{E}' = \{(i,j) \in V \times V \ : \ (i,j) \notin E'\};$
while $\bar{E}' \neq \emptyset$ **do**
  for $u \in C$ **do**
    $\mathcal{X}(u) \leftarrow \emptyset;$
    $g(u) \leftarrow 0;$
    $\mathcal{N}(u) = \{v \in V \ : \ (u,v) \in E'\};$
    for $(i,j) \in \mathcal{N}(u) \times \mathcal{N}(u)$ **do**
      if $(i,j) \notin E'$ **then**
        $\mathcal{X}(u) \leftarrow \mathcal{X}(u) \cup \{(i,j)\};$
        $g(u) \leftarrow g(u) + 1;$
      end
    end
  end
  $u^* \leftarrow$ **argmax**$\{g(u) \ : \ u \in C\};$
  $R \leftarrow R \cup \{u^*\};$
  $C \leftarrow C \setminus \{u^*\};$
  $E' \leftarrow E' \cup \mathcal{X}(u^*);$
  $\bar{E}' \leftarrow \bar{E}' \setminus \mathcal{X}(u^*);$
end
return R;

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

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 heuristic `H1` is to find a spanning tree on the communication graph M with the maximum number of leaf nodes. 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 $u^* \leftarrow$ **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, `Tree`$(u^*, S, \bar{S}, R)$, described in Algorithm 2.

`Tree` takes as input the lowest degree vertex u of the graph and recursively constructs a spanning tree . 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

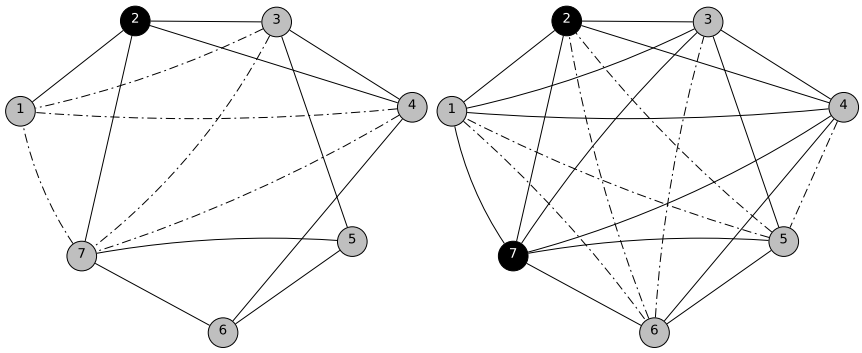
```
procedure Tree
  Data: u, S, S̄, R
  Result: R
1 U(u) ← N(u) ∩ S̄;
2 if |S| > 1 then
3 |   S ← S ∪ U(u);
4 |   S̄ ← V \ S;
5 end
6 while U(u) ≠ ∅ do
7 |   Compute deg_S̄(v) for all v ∈ U(u);
8 |   u* ← argmax{deg_S̄(v) : v ∈ U(u)};
9 |   if deg_S̄(u*) > 0 then
10|   |   R ← R ∪ {u*};
11|   |   Tree(u*, S, S̄, R);
12|   end
13|   U(u) ← U(u) \ {u*};
14 end
15 return
```

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

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`.

In our implementation of `H1` we use a hash set to store the set of unvisited vertices instead of using the stack of the system. Additionally, our algorithm is iterative instead of recursive, as seems to be the intent of the original implementation of Chen et al. (2010). Finally, as we did in `Greedy`, the $N(v)$ set is indexed with a hash table and therefore its associated operations can be performed in O(1).

2.3. **H2 heuristic.** Heuristic `H2` in some sense resembles the two previous methods. In each step, it first identifies the nodes of lowest degree and then examines the



(a) Iteration 1 (b) Iteration 2

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

neighbors of theses 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 is based on the degree of the nodes.

Contrasting our implementation of `H2` with the original implementation in Chen et al. (2010), it is worth mentioning that our Java code employs hash-sets for an efficient update of the communication graph and the computation of the lowest and highest degree nodes. In each iteration, we compute the set of newly connected nodes by calculating the Cartesian product of the set of nodes adjacent to the node just made to house a regenerator. As in the previous methods, the use of hash-sets significantly accelerates this computation. In our experiments (see Section 5) we empirically compare both implementations.

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 partial solution. 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 three constructive algorithms, *CG*, *C1*, and *C2* based on, respectively, the heuristics `Greedy`, `H1`, and `H2` of Chen et al. (2010), 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 having 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

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

where

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

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 different values of α in the computational experiments in Section 5.

Heuristic `H1` tries to find a spanning tree S on the communication graph M having the maximum number of leaf nodes, thus minimizing the number of internal nodes in S to 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

$$deg_{min} + \alpha \cdot (deg_{max} - deg_{min}), \tag{1}$$

where

$$deg_{min} = \min_{u \in CL} \deg_{\bar{S}}(u) \text{ and } deg_{max} = \max_{u \in CL} \deg_{\bar{S}}(u).$$

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` does), *C2* computes a restricted candidate list with those nodes having degree greater than or equal to a specified threshold value, as in (1), 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^*.

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 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,

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 $v \in C$ producing a feasible solution;
12 $R' \leftarrow R' \cup \{v\}$;
13 **return** R';

Algorithm 3: Pseudo-code for `MOVE`: a 2-1 or 1-0 move.

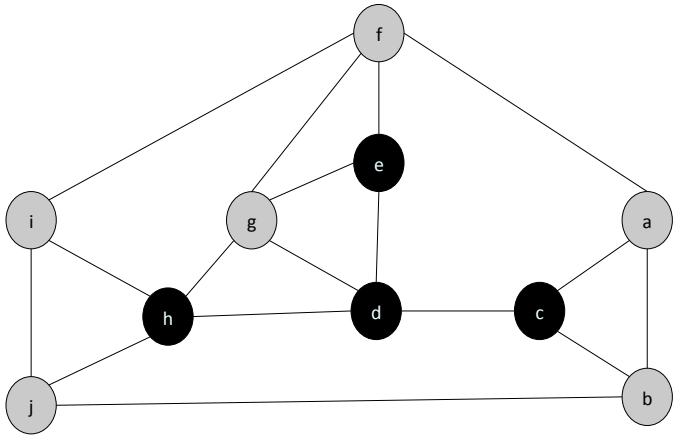


FIGURE 3. Example where original `MOVE` fails.

a 2-1 move is applied. Ties are broken by node index. The node in C with the largest degree in the communication graph is selected.

Consider the example in Figure 3 with the solution $R = \{c, d, e, h\}$ (depicted in black in the figure) and apply the `MOVE` procedure in Algorithm 3 to nodes d and e. In steps 3 to 9 we take each vertex $u \in (\mathcal{N}(d) \cup \mathcal{N}(e)) \setminus R' = \{f, g\}$ and compute C, obtaining $C = \{a, g, i\}$. According to the original design of the `MOVE` procedure in Chen et al. (2010), we could replace the regenerators in d and e with a single regenerator allocated in any node in C. However, we can see that, in this case,

this would produce an infeasible solution. For example, if we select a, we would obtain $R' = \{c, h, a\}$ leaving node e disconnected. To remedy this problem, we have modified this method to include a condition on the feasibility of the resulting solutions (step 11 in Algorithm 3). Specifically, we check if the subgraph induced by the set of regenerators is connected. If it is, then the solution is feasible; otherwise, we discard this selection and resort to the next node in C. In the example in Figure 3, none of the candidates in C produces a feasible solution.

Our local search procedure scans the regenerator nodes in the order given by their degrees in the original graph (where the node with the lowest degree is examined first). Nodes with 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, 2011), 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. *Parameterized 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 $i = 1, \ldots, n$, 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: $R \subseteq V$
$R \leftarrow \emptyset; C \leftarrow V$;
$\bar{E}' = \{(i,j) \in V \times V \ : \ (i,j) \notin E'\}$;
Order the nodes in C w.r.t. their associated x value;
while $\bar{E}' \neq \emptyset$ **do**
  Select the next node $k \in C$ following the x-order;
  $\mathcal{X}(k) \leftarrow \emptyset$;
  $A \leftarrow \mathcal{N}(k) = \{v \in V \ : \ (k,v) \in E'\}$;
  for $v \in A \cap R$ **do**
    $A \leftarrow A \cup \mathcal{N}(v)$;
  end
  for $(i,j) \in A \times A$ **do**
    if $(i,j) \notin E'$ **then**
      $\mathcal{X}(k) \leftarrow \mathcal{X}(k) \cup \{(i,j)\}$;
    end
  end
  $R \leftarrow R \cup \{k\}$;
  $C \leftarrow C \setminus \{k\}$;
  $E' \leftarrow E' \cup \mathcal{X}(k)$;
  $\bar{E}' \leftarrow \bar{E}' \setminus \mathcal{X}(k)$;
end
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 containing all 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 where 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 the first experiment, we compare the original implementation of Chen et al. (2010) of the heuristics `Greedy`, `H1`, and `H2` coupled with the local search (labeled as Orig. `Greedy`, Orig. `H1`, and Orig. `H2`, respectively, in Table 2) with our new implementations of these methods (simply labeled as `Greedy`, `H1`, and `H2`). For each method and each instance size, we consider, in Table 2, the two columns reported in Table 1 of Chen et al. (2010): NR (number of regenerators) and RT (running time). Results in this table clearly indicate that our implementations outperform those in Chen et al. (2010) since they obtain slightly better solutions (lower number of regenerators) in significant lower running times (more than 100 times faster). The methods in Chen et al. (2010) were run on a Pentium D (3.0 GHz processor with 2 GB RAM) while, as described above, our implementations have been run on a similar computer (Pentium 4, 3 GHz with 2 GB of RAM). Note also that our implementations are in Java while those in Chen et al. (2010) are in C++.

In each of the following experiments, 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*.

TABLE 2. Comparison with Chen et al. (2010) implementation.

		Orig. `Greedy`		`Greedy`		Orig. `H1`		`H1`		Orig. `H2`		`H2`	
n	m	NR	RT	NR	RT	NR	RT	NR	RT	NR	RT	NR	RT
40	10	1.40	0.00	1.40	0.00	1.40	0.00	1.40	0.00	1.40	0.10	1.40	0.00
	30	2.00	0.20	2.00	0.01	2.00	0.10	2.00	0.00	2.00	0.00	2.00	0.00
	50	3.00	0.10	3.00	0.01	3.10	0.30	3.00	0.00	3.30	0.10	3.00	0.00
	70	4.60	0.70	4.50	0.01	4.70	0.50	4.60	0.00	4.60	0.20	4.40	0.00
	90	9.50	1.70	10.10	0.02	9.70	1.30	9.90	0.01	9.80	0.50	10.20	0.00
60	10	1.90	0.30	1.90	0.04	1.90	0.30	1.90	0.01	1.90	0.00	1.90	0.00
	30	2.10	0.80	2.00	0.03	2.10	0.30	2.00	0.01	2.20	0.20	2.00	0.00
	50	3.20	1.90	3.10	0.05	3.30	1.10	3.10	0.00	3.40	0.40	3.00	0.00
	70	5.30	4.30	5.10	0.06	5.40	3.90	5.10	0.00	5.30	0.70	5.00	0.01
	90	10.80	15.80	11.00	0.10	11.20	18.00	11.40	0.01	11.20	2.50	11.30	0.01
80	10	1.90	1.30	1.90	0.06	1.90	0.70	1.90	0.03	1.90	0.10	1.90	0.00
	30	2.90	3.90	2.60	0.09	2.80	2.00	2.70	0.02	2.80	0.30	2.70	0.01
	50	3.80	9.10	3.50	0.08	4.00	5.60	3.60	0.01	3.70	0.90	3.50	0.00
	70	6.00	19.40	5.80	0.13	5.90	17.30	5.70	0.00	5.80	3.00	5.70	0.01
	90	11.90	88.90	12.20	0.22	12.50	70.70	12.40	0.03	12.80	13.00	12.10	0.03
100	10	2.00	3.60	2.00	0.14	2.00	1.80	2.00	0.07	2.00	0.50	2.00	0.01
	30	2.90	9.60	2.70	0.17	2.90	7.10	2.80	0.03	2.80	1.40	2.80	0.01
	50	4.00	21.10	4.00	0.20	4.00	13.60	4.00	0.02	4.00	2.60	4.00	0.01
	70	6.00	49.00	6.00	0.25	6.40	56.00	5.80	0.01	6.10	8.20	6.00	0.01
	90	13.40	286.40	12.70	0.38	13.70	295.00	13.50	0.04	13.30	42.90	13.00	0.02
	Avg.	4.93	25.91	4.88	0.10	5.05	24.78	4.94	0.02	5.02	3.88	4.90	0.01

TABLE 3. GRASP Constructive methods

	Value	#Best	Dev(%)	CPU(sec.)
CG				
$\alpha = 0.3$	7.00	7	25.77	12.93
$\alpha = 0.6$	5.65	13	5.92	10.00
$\alpha = 0.9$	5.35	19	0.45	7.96
C1				
$\alpha = 0.3$	6.80	8	20.42	0.77
$\alpha = 0.6$	6.10	10	10.79	0.74
$\alpha = 0.9$	6.25	11	12.10	0.76
C2				
$\alpha = 0.3$	6.30	7	16.59	0.57
$\alpha = 0.6$	6.10	11	10.64	0.56
$\alpha = 0.9$	5.95	12	8.51	0.54

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 3 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 3 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, although the three of them are very fast.

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 of 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. 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 4, Value, #Best, Dev., and CPU time.

Table 4 shows that our three new GRASP heuristics 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 average percent deviations of 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, although they are still very moderate (less than 10 seconds).

TABLE 4. Constructive with local search methods

	Value	#Best	Dev(%)	CPU(sec.)
Greedy+LS	5.65	11	8.96	0.17
H1+LS	5.70	11	9.96	0.05
H2+LS	5.55	14	7.50	0.01
GC(0.9)+LS	5.25	18	1.45	8.19
C1(0.6)+LS	5.15	20	0.00	2.24
C2(0.9)+LS	5.25	18	1.70	0.11

TABLE 5. Comparison of best methods

	Value	#Best	Dev(%)	CPU(sec.)
$n = 40$				
H2+LS	3.96	59	4.46	0.003
C1(0.6)+LS	3.77	70	0.00	0.25
BRKGA	3.83	66	1.71	1.44
$n = 60$				
H2+LS	4.37	62	1.74	0.006
C1(0.6)+LS	4.26	70	0.00	0.65
BRKGA	4.34	64	1.48	4.46
$n = 80$				
H2+LS	4.90	46	8.25	0.01
C1(0.6)+LS	4.50	70	0.00	1.36
BRKGA	4.67	58	4.39	9.74
$n = 100$				
H2+LS	5.27	50	5.69	0.02
C1(0.6)+LS	4.91	70	0.00	2.39
BRKGA	5.00	62	2.13	20.17

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 with $p = 100$, $p_e = 0.15, p_m = 0.10$, and $\rho_e = 0.7$, run for 20 generations. Table 5 reports the Value, #Best, Dev., and CPU time over the entire set of 280 instances divided according to their size.

Table 5 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.

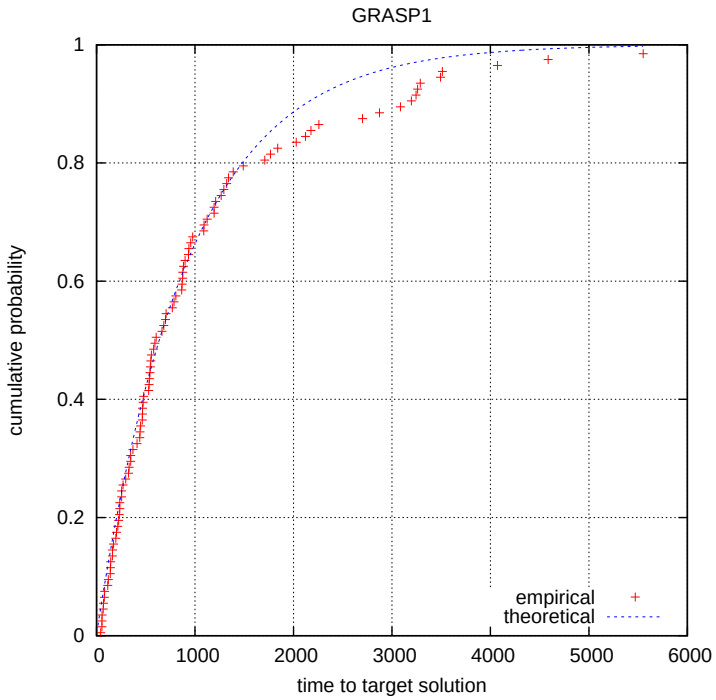


FIGURE 4. Time to target plots (run time distributions) for `C1+LS`.

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 4. 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

This paper deals with the regenerator location problem in optical telecommunication networks. In this problem, we are given a network represented by a weighted graph, where edge weights represent distances, and the maximum distance an optical signal can travel before it starts to degrade. Regenerators strengthen the signal to its original state. They are, however, expensive and one wants to deploy as few of them as possible in the network. We seek the locations of a minimum number of regenerators such that all pairs of nodes in the network are able to communicate using optical signals.

Three constructive procedures and a local improvement procedure were previously proposed by Chen et al. (2010). In this paper, we propose efficient implementations of the three constructive procedure as well as point out an error in the local search procedure for which we propose a correction. Not only are our implementations of the constructive procedures much faster (up to 7375 times faster in

one case) than those of Chen et al. (2010), but, on average, they also find better solutions.

This paper also proposes three GRASP heuristics, each using one of the three constructive procedures of Chen et al. (2010) as well as the corrected local search procedure. The three (multi-start) GRASP heuristics find much better solutions than the corresponding (single-start) constructive procedures with local search.

Finally, this paper proposes a biased random-key genetic algorithm (BRKGA) for this regenerator location problem. The BRKGA uses a decoder based on the greedy constructive procedure of Chen et al. (2010). To enable the genetic algorithm to have running times similar to those of the other algorithms tested, a small population was used as well as a small number of generators were computed. Solutions found by the BRKGA were better than those found by the heuristics of Chen et al. (2010) but worse than those found by the GRASP heuristics. To find solutions of similar quality, we conjecture that the BRKGA would require larger populations, and more importantly, a greater number of generations.

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 versus unbiased random key genetic algorithms: A experimental analysis. *J. of Heuristics*, 17:487–525, 2011.

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 Matematicas, 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) 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@cin.ufpe.br