A GRASP algorithm with tree based local search for designing a survivable Wide Area Network backbone

Héctor Cancela*
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
and
Franco Robledo
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay
and
Gerardo Rubino
IRISA, INRIA
Rennes, France

Abstract

System survivability is the ability to give service in spite of failures of some of the components. To assure survivability is an important goal when designing a communications network backbone, to ensure that it can resist to failures in the switch sites as well as in the connection lines. Previous work has employed a Greedy Randomized Adaptive Search Procedure (GRASP), based on path algorithms, to build low cost network topologies which comply with heterogeneous node-connectivity requirements, which can model the survivability goals. In this work, we present another variant of the GRASP procedure, based on a tree search, which obtains good results in topologies with a large number of switch nodes.
Keywords: meta-heuristics; GRASP; topological design; survivability; node connectivity.

1 Introduction

A wide area network (WAN) can be seen as a set of sites and a set of communication lines that interconnect these sites. A typical WAN is organized as a hierarchical structure integrating two levels: one *backbone network* and a number of *local access networks* [9]. Each local access network usually has a tree-like structure, rooted at a single switch site of the backbone network, and connects users (terminal sites) either directly to this switch site or to a hierarchy of intermediate concentrator sites (local servers) which are connected to the switch site. The backbone network has usually a meshed topology, to allow efficient and reliable communication between the switch sites of the network that act as connection points for the local access networks (eventually incorporating other switch sites for efficiency purposes).

One important aspect about the topology of a backbone network is its survivability, which is the ability to continue in operation after failures of some of its switch sites and links. To take this aspect into account when designing the network, it is possible to specify a connectivity level, and search for low cost topologies which have at least this number of disjoint paths (either edge disjoint or node disjoint) between pairs of switch sites. In the most general case, the connectivity level can be fixed independently for each pair of switch sites (heterogeneous connectivity requirements). This problem can be modeled as a *Generalized Steiner Problem with Node-Connectivity* [12, 16]; finding a minimum cost topology with these constraints is a NP-hard problem. Some references in this area are [7, 9, 13, 15]; these works are either focused on the edge-disjoint flavor of the problem, or explore particular cases (for example, where the connection requirements are the existence of two disjoint paths for all pairs of switch sites [1, 8]).

Topologies verifying edge-disjoint path connectivity constraints assure that the network can survive to failures in the connection lines; while node-disjoint path constraints assure that the network can survive to failures both in in switch sites as well as in the connection lines. In a previous work [3], we have used the GRASP methodology and proposed a path-based algorithm for finding a Backbone network topology that satisfies connection requirements on the number of node-disjoint paths, working upon the *Generalized Steiner Problem with Node-Connectivity* model.

In this paper, we present a different, tree-based algorithm, for the search phase of the GRASP method. Section 2 formalizes our backbone network design problem (BNDP). Section 3 presents the GRASP meta-heuristic, and Section 4 introduces the tree-based local search for solving the BNDP. Experimental results are presented in Section 5, showing that the algorithm works very well

*cancela@fing.edu.uy, robledo@fing.edu.uy, rubino@irisa.fr

for a number of test cases corresponding to quite different topologies and connection requirements. Finally, conclusions and future work are presented in Section 6.

2 Notation and Problem Formalization

We introduce the notation used to formalize the problem:

- S_D is the set of sites where the switch equipments can be installed; these sites also will be called potential switch sites or backbone sites. The number of the switch sites is given by $n_D = |S_D|$.
- $S_D^{(I)} \subseteq S_D$ is a distinguished subset of switch sites, which will always be included in the backbone network topology (usually because they are the access points for some local access subnetworks). We usually call these the *fixed sites* of the backbone. The sites in $S_D - S_D^{(I)}$ are called non-fixed sites or Steiner nodes.
- $R = \{r_{ij}\}_{i,j \in S_D^{(I)}}$ is a matrix of requirements of connection between pairs of sites of $S_D^{(I)}$. We will require r_{ij} node disjoint paths between fixed sites i and j, where r_{ij} usually is strictly greater than 1.
- $C = \{c_{ij}\}_{i,j \in S_D}$ is the matrix which gives for any pair of sites of S_D, the cost of laying a line between them. When the direct connection among both places is not possible, we take $c_{ij} = \infty$.
- $E = \{(i,j); \forall i \in S_D, \forall j \in S_D / c_{ij} < \infty\}$, this is the set of feasible connections between two switch sites.
- $G_B = (S_D, E)$ is the graph of feasible connections on the Backbone Network.

We define our Backbone Network Design Problem $BNDP(S_D, E, C, R)$ as the problem of finding a subgraph $\mathcal{H}_B$ of G_B of minimum cost such that $\mathcal{H}_B$ satisfies the connection requirements specified in R. We will denote by Γ_{BNDP} the space of feasible solutions associated with the problem.

3 A Greedy Randomized Adaptive Search Procedure (GRASP) for the BNDP

GRASP is a well known meta-heuristic, which has been applied for solving many hard combinatorial optimization problems with very good results [5, 6]. A GRASP is an iterative process, where each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

Figure 1 presents the GRASP pseudo-code. The GRASP takes as input parameters the candidate list size, the maximum number of GRASP iterations and the seed for the random number generator. After reading the instance data (line 1), the GRASP iterations are carried out in lines 2-6. Each GRASP iteration consists of the construction phase (line 3), the local search phase (line 4) and, if necessary, the incumbent solution update (lines 5 and 6).

```
Procedure GRASP(ListSize, MaxIter, RandomSeed);

1  InputInstance();
2  for k = 1 to MaxIter do
3     InitialSol = BNDP_ConstructSol(ListSize);
4     LocalSol = BNDP_Local_Search(InitialSol);
5     if cost(LocalSol) < cost(BestSolFound) then
6        UpdateSolution(BestSolFound, LocalSol);
7  end_for;
8  return BestSolutionFound;
```

Figure 1: GRASP pseudo-code.

In the construction phase, a feasible solution is built, one element at a time. At each step of the construction phase, a candidate list is determined by ordering all non already selected elements with respect to a greedy function that measures the (myopic) benefit of including them in the solution. The heuristic is adaptive because the benefits associated with every element are updated at each step to reflect the changes brought on by the selection of the previous elements. Then one element is randomly chosen from the best candidates list and added into the solution. This is the probabilistic component of GRASP, which allows for different solutions to be obtained at each GRASP iteration, but does not necessarily jeopardize the power of the adaptive greedy component.

The pseudo-code of BNDP_ConstructSol is given in Figure 2. The algorithm works over the network G_B of feasible connections on the backbone network, the matrix R of connection requirements between fixed switch sites of $S_D^{(I)}$, and the matrix of connection costs C. The current solution is initialized with the fixed sites without any connection among them. An auxiliary matrix M is initialized with the values of R. This is used with the purpose of maintaining on each step the connection requirements not yet satisfied between sites of $S_D^{(I)}$. The paths found on each iteration are stored in a data structure $\mathcal{P}$ where the elements in $\mathcal{P}_{ij}$ are paths found from s_w^i to s_w^j, with $s_w^i, s_w^j \in S_D^{(I)}$.

Iteratively the construction phase searches for node-disjoint paths between fixed sites of $S_D^{(I)}$ that have not yet satisfied their connection requirements. The algorithm chooses on each iteration such a pair of fixed switch sites $s_w^i, s_w^j \in S_D^{(I)}$. The current solution is updated by adding a new node-disjoint path between the

chosen sites. For this, we work upon an auxiliary network and find the $ListSize$ shortest paths from s^i_w to s^j_w using a classical algorithm [17]. These paths are stored in a restricted candidate list $\mathcal{L}_p$. By construction, each of these paths will be node disjoint with those already in $\mathcal{P}_{ij}$. A path is randomly selected from $\mathcal{L}_p$ and the current solution is updated by adding this new path. Moreover the set $\mathcal{P}$ of node-disjoint paths and the matrix M can be updated efficiently.

The $ListSize$ parameter is used to provide diversity while limiting the computational effort. If $ListSize = 1$, the algorithm would be a pure greedy method, building the same solution at every invocation. Larger values of $ListSize$ allow to consider more path options at each step, at the cost of more lengthy computations.

This process is repeated until all the connection requirements have been satisfied; then the feasible solution $\mathcal{G}_{sol}$ and the set of node-disjoint paths $\mathcal{P}$ are returned.

Procedure BNDP_ConstructSol($ListSize$);

1 $\mathcal{G}_{sol} \leftarrow (S_D^{(I)}, \emptyset)$; $M \leftarrow R$; $\mathcal{P}_{ij} \leftarrow \emptyset \ \forall s^i_w, s^j_w \in S_D^{(I)}$;
2 **while** $\exists m_{ij} > 0$ **do**
3 Let $s^i_w, s^j_w \in S_D^{(I)}$ be a randomly chosen pair of fixed switch sites such that $m_{ij} > 0$;
4 $\bar{\mathcal{G}} \leftarrow G_B \setminus (\text{NODES}(\mathcal{P}_{ij}) \setminus \{s^i_w, s^j_w\})$;
5 Compute $\bar{C}$:

$$\bar{c}_{ij} \leftarrow \begin{cases} 0 & \text{if } (s^i_w, s^j_w) \in \mathcal{G}_{sol}, \\ c_{ij} & \text{if } (s^i_w, s^j_w) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases} ;$$

6 $\mathcal{L}_p \leftarrow$ the $ListSize$ shortest paths from s^i_w to s^j_w on $\bar{\mathcal{G}}$ considering the matrix $\bar{C}$;
7 $p \leftarrow \text{Select\_Random}(\mathcal{L}_p)$;
8 $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p\}$; $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup \{p\}$;
9 $M \leftarrow \text{Update\_Matrix}(\mathcal{G}_{sol}, M, R, p)$;
10 **end_while;**
11 **return** $\mathcal{G}_{sol}$, $\mathcal{P}$;

Figure 2: Construction Phase pseudo-code.

The solutions generated by the construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood. The local search algorithm depends on the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution.

In the following section, we present a Local Search procedure which is based on a decomposition of the solution in small building blocks (called key-paths), and their substitution by trees which are equivalent from the point of view of the network connectivity.

4 A Tree-Based Local Search

The local search strategy proposed is a generalization of the key-path based local search, proposed by Verhoeven, Severens and Aarts [14] in context of the Steiner tree problem. A key-path is a path such that its end-points are either non-fixed switch sites of degree at least 3, or fixed switch sites; and all its intermediate nodes are non-fixed switch sites of degree 2. We employ the following property:

Proposition 4.1 *Let $\mathcal{G} \in \Gamma_{BNDP}$ be a feasible solution. If each edge of $\mathcal{G}$ belongs to some path between two terminals, then it is possible to decompose $\mathcal{G}$ in key-paths (i.e., there is a set of key-paths such that every edge of $\mathcal{G}$ belongs to one and only one key-path). We will denote by $K(\mathcal{G}) = (p_1, \ldots, p_h)$ the decomposition of $\mathcal{G}$ in key-paths, ordered by decreasing cost.*

Definition 4.2 (Neighborhood Structure)
Let $\mathcal{G}_s \in \Gamma_{BNDP}$ be a feasible solution. Given a key-path $p \subset \mathcal{G}_s$, we define a neighbor solution of $\mathcal{G}_s$ as: $\hat{\mathcal{G}}_s = (\mathcal{G}_s \setminus \text{EDGES}(p)) \cup \mathcal{T}$, where $\mathcal{T}$ is a tree rooted in one end-point of p and built maintaining the feasibility in the new network $\hat{\mathcal{G}}_s$. The Tree Neighborhood of $\mathcal{G}_s$ is composed of the neighbor solutions obtained by applying the previous operation to each of the different key-paths in $K(\mathcal{G}_s) = (p_1, \ldots, p_h)$.

The pseudo-code of BNDP_Local_Search is given in Figure 3. The local search algorithm works upon the original problem data (the network G_B, the matrix R, and the matrix C) and takes as additional inputs the set $\mathcal{P}$ of already found paths, and the decomposition in key-paths $K(\mathcal{G}_{sol})$ associated to $\mathcal{G}_{sol}$. The algorithm searches the best neighbor in the *Tree Neighborhood*. Iteratively the local search analyzes each key-path of the decomposition $K(\mathcal{G}_{sol})$, replacing (if it is possible) a key-path by a tree of smaller cost, maintaining the feasibility of the current solution. This tree is computed by an algorithm (represented in the pseudo-code by the Find_Tree function) which works upon an auxiliary network searching a tree rooted at one of the end-points of the current key-path; we describe below the details.

When there are no more improvements to be obtained by substituting key-paths by trees, the best feasible solution and the set of found paths are returned.
In order to describe the algorithm Find_Tree we introduce the following notations and definitions.

Notation 4.3 *Let $\mathcal{G}$ be the the current network built iteratively by* BNDP_ConstructSol. *Given a key-path $q \in K(\mathcal{G})$, we denote:*

$$\mathcal{V}_q(\mathcal{G}) = \left\{(s^i_w, s^j_w) \in S_D^{(I)} \times S_D^{(I)} \,|\exists p \in \mathcal{P}_{ij}, \text{ so that } q \subseteq p\right\}.$$

This is the set of pairs of switch sites of $S_D^{(I)}$ with at least one path $p \in \mathcal{P}_{ij}$ such that $q \subseteq p$ (i.e., the pairs of switch sites whose connection depends on the key-path q).

Procedure BNDP_Local_Search($\mathcal{P}$,$K(\mathcal{G}_{sol})$);

repeat
  $improve \leftarrow$ FALSE;
  for $k = 1, \ldots, |K(\mathcal{G}_{sol})|$ **do**
    Let p_k **be the** k**-th key-path;**
    $\mathcal{T} \leftarrow$ Find_Tree();
    if $(\text{COST}(\mathcal{T}) < \text{COST}(p_k))$ **then**
      $\mathcal{G}_{sol} \leftarrow (\mathcal{G}_{sol} \setminus p_k) \cup \mathcal{T}$;
      $\mathcal{P} \leftarrow$ Update_Paths$(\mathcal{P}, p_k, \mathcal{T})$;
      $K(\mathcal{G}_{sol}) \leftarrow$ Update_KeyPaths$(K(\mathcal{G}_{sol}), p_k, \mathcal{T})$;
      $improve \leftarrow$ TRUE;
    end_if;
  end_for;
until not$(improve)$;
return $\mathcal{G}_{sol}$,$\mathcal{P}$;

Figure 3: Local Search Phase pseudo-code.

Procedure Find_Tree($\mathcal{P}$,$K(\mathcal{G}_{sol})$,p_k);

$min\_cost \leftarrow \infty$; $\mathcal{T} \leftarrow p_k$;
Let $\{s_1, s_2\}$ **be the end-points of** p_k;
for each $\hat{s} \in \{s_1, s_2\}$ **do**
  $\bar{\mathcal{G}} \leftarrow G_B \setminus \mathcal{X}_{(p_k,\bar{s})}(\mathcal{G}_{sol})$; $\bar{s} \leftarrow \{s_1, s_2\} \setminus \hat{s}$;
  Compute $\bar{C}$**:**

$$\bar{c}_{ij} \leftarrow \begin{cases} 0 & \text{if } (s_w^i, s_w^j) \in \mathcal{G}_{sol}, \\ c_{ij} & \text{if } (s_w^i, s_w^j) \in (\bar{\mathcal{G}} \setminus \mathcal{G}_{sol}). \end{cases};$$

  $\mathcal{H} \leftarrow \text{ArgMin}\{\text{COST}(p_{min}), \text{COST}(\mathcal{T}_{min})\}$ **where:**
    p_{min} **is the shortest path from** s_1 **to** s_2 **on** $\bar{\mathcal{G}}$**,**
    $\mathcal{T}_{min}$ **is a tree of minimum cost connecting** $\hat{s}$ **with a node of** $p \setminus \{\bar{s}\}, \forall p \in \text{T}_{(p_k,\bar{s})}$;
  if $(\text{COST}(\mathcal{H}) < min\_cost)$ **then**
    $min\_cost \leftarrow \text{COST}(\mathcal{H})$; $\mathcal{T} \leftarrow \mathcal{H}$;
  end_if;
end_for_each;
return $\mathcal{T}$;

Figure 4: Find_Tree pseudo-code.

Notation 4.4 *Given a path $p \in \mathcal{P}_{ij}$ and two switch sites $s_1, s_2 \in p$, we denote by $p_{(s_1,s_2)}$ the sub-path from s_1 to s_2 on p.*

Definition 4.5 *Let $\mathcal{G} \in \Gamma_{BNDP}$ be the feasible solution computed by* BNDP_ConstructSol*. Given a key-path $q \in K(\mathcal{G})$ and given $\bar{s} \in S_D$ one of its end-points, we define the following set:*

$$\mathcal{X}_{(q,\bar{s})}(\mathcal{G}) = \bigcup_{(s_w^i, s_w^j) \in \mathcal{V}_q(\mathcal{G})} \left(\bigcup_{p \in \mathcal{P}_{ij}, q \not\subseteq p, \bar{s} \neq s_w^j} \text{NODES}(p) \right).$$

This set is the union over every pair of sites s_w^i, s_w^j which depend on key-path q, of all the nodes belonging to paths in $\mathcal{P}_{ij}$ which do not contain q.

Definition 4.6 *Let $\mathcal{G} \in \Gamma_{BNDP}$ be the feasible solution built by* BNDP_ConstructSol*. Given a key-path $q \in K(\mathcal{G})$ and given $\bar{s} \in S_D$ one of its end-points, we define the key-graph associated to $(q, \bar{s})$ as:*

$$\text{T}_{(q,\bar{s})} = \{p_{(\bar{s},s_w^j)} | s_w^j \in S_D^{(I)}, \exists s_w^i \in S_D^{(I)} / (s_w^i, s_w^j) \in \mathcal{V}_q(\mathcal{G}) \text{ and } p_{(\bar{s},s_w^j)} \subset p_{(s_w^i,s_w^j)} \in \mathcal{P}_{ij}\}.$$

This is a graph formed by the union over every pair of sites s_w^i, s_w^j which depend on key-path q, of all the sub-paths $\{p_{(\bar{s},s_w^j)} | p_{(\bar{s},s_w^j)} \subset p_{(s_w^i,s_w^j)} \in \mathcal{P}_{ij}\}$.

Figure 4 shows the pseudo-code of Find_Tree. This algorithm tries to compute a low-cost tree $\mathcal{T}$ on the auxiliary network $\bar{\mathcal{G}}$ (defined in Figure 4), considering the auxiliary matrix $\bar{C}$, so that replacing in the current solution the key-path p_k by the computed tree $\mathcal{T}$ the feasibility is preserved, creating thus a new neighbor solution, that is: $\mathcal{T} \cup (\mathcal{G}_{sol} \setminus p_k) \in \Gamma_{BNDP}$. For this, it considers each endpoint $\hat{s}$ of p_k building a tree $\mathcal{T}_{min}$ with root $\hat{s}$ having as endpoints nodes of $\text{T}_{(p_k,\bar{s})}$ (where $\bar{s}$ is the other endpoint of p_k). In particular, this tree is integrated by the shortest paths from $\hat{s}$ to each path $p_{(\bar{s},s_w^j)} \in \text{T}_{(p_k,\bar{s})}$ on network $\bar{\mathcal{G}}$, guaranteeing thus the non-loss of one level of connectivity (once the key-path p_k is substituted) with respect to the set of fixed switch sites $\{s_w^j | p_{(\bar{s},s_w^j)} \in \text{T}_{(p_k,\bar{s})}\}$. The problem is a variant of the single-source shortest paths problem, and can be solved using Dijkstra's algorithm.

Moreover the shortest path p_{min} between both ends of p_k is computed on network $\bar{\mathcal{G}}$. By construction, $\mathcal{T}_{min}$ and p_{min} will satisfy:

- $\mathcal{T}_{min} \cup (\mathcal{G}_{sol} \setminus p_k) \in \Gamma_{BNDP}$,
- $p_{min} \cup (\mathcal{G}_{sol} \setminus p_k) \in \Gamma_{BNDP}$.

Both of them are taken into account in order to find the best neighbor alternative (i.e. with smaller cost), which will be returned by Find_Tree.

5 Implementation and Performance Tests

We present here some experimental results obtained with the algorithm presented in the previous section.

The algorithm was implemented in ANSI C language. Adjacency matrices were used as data structures for the representation of graphs. The experiments were obtained on a Pentium III computer with 800 MHz, and 256 MB of RAM memory, running under Windows NT 4 operating system.

All instances were solved with identical parameter settings. The candidate list size was $ListSize = 5$, and the maximum number of iterations $MaxIter = 50$. These values were chosen considering the GRASP reference literature [5, 6].

We selected four test problems as experimental suite, to investigate the effectiveness of the proposed method. A requirement was that an optimal solution (or at least

the optimal value) was known, so that GRASP results could be properly evaluated. Two instances of large size with known solutions were obtained from previous literature. Besides, we introduce other two instances which were designed constructively applying certain topological properties that preserve the optimality of a known optimal solution. In [4] we give a detailed description for these and other instances, with information about known optimal solutions, and their construction.

Figure 5 shows the topologies associated to the test cases 1, 2, 3 and 4. The black nodes represent the fixed switch sites, while the white nodes represent other potential switch sites, which may or may not be included in the solution (Steiner nodes). We describe below the main characteristics of the four problem instances.

- Network 1 is a simplified version of a HSODTN (High Speed Optical Data Transmission Network) connecting different parts of a war ship. The reduced topology has 9 fixed switch sites (modeling strategic point in an aircraft carrier), 55 Steiner nodes (non-fixed switch sites) and 124 edges. The objective is to find a 3-node-survivable subnetwork with minimal cost (all connection requirement between fixed sites are equal to 3). This model and other variants can be found in [7, 13]. An optimal solution having cost 74 has been found by an exact parallel-distributed backtracking algorithm in [10].

- Network 2 has 41 fixed switch sites, 68 Steiner nodes and 383 edges. This network was designed taking as basis four Brinkman sub-graphs (four Brinkman graphs with two edges less on each one) interconnected according to Figure 5. The Brinkman graph is 4-regular, 4-connected and of girth at least 5 (see for instance [2]). In Figure 5 we show the built topology; the edges of the Brinkman sub-graphs are represented by straight lines and the other connections are represented by dashed lines. This instance is formulated as a $\mathrm{NCON}(\cdot)$ problem [7, 13], which is a particular case of the *Generalized Steiner Problem*, where each fixed switch site $s_w^i \in S_D^{(I)}$ is labeled with a positive integer number r_i, and the aim is to find a minimum cost subnetwork such that for every pair of fixed switch sites $s_w^i, s_w^j \in S_D^{(I)}$ there exists at least $r_{ij} = \min\{r_i, r_j\}$ node-disjoint paths. We have 10 fixed switch sites with $r_i = 4$, 13 fixed switch sites with $r_i = 3$ and 18 fixed switch sites with $r_i = 2$. The edge costs were chosen within the $[1, 200]$ range. The topology of a global optimal solution (of cost 3980) can be seen in [4].

- Network 3 has 22 fixed sites of $S_D^{(I)}$, 61 Steiner sites and 262 feasible connections. The edges have costs randomly selected in the interval $[1, 200]$, satisfying the triangular inequality. The objective is to find a 4-node-survivable subnetwork spanning $S_D^{(I)}$ with minimal cost (all connection requirement between fixed sites are equal to 4). Like in the previous instance, we known an optimal solution (of cost 680) obtained from the constructive procedure used to build this test case. Details of its design as well as an optimal topology can be seen in [4].

- Networks 4 is a test case based on real network from Bell Communications Research (Bellcore) [13]. Particularly, we have selected the problem called LATADL in order to test the performance of our algorithm. Link costs are linked to geographical distances. The LATADL-problem has 116 fixed sites and 173 feasible connections. In this problem, there are two classes of nodes: nodes of type 1, shown as circles in Figure 5, and nodes of type 2, shown as small squares. The connectivity requirements are that between two nodes of type 2 there must be two node-disjoint paths; and between a node of type 2 and a node of type 1, or between two nodes of type 1, there must be at least one path. For Network 4 an optimal solution has been published in [13], with cost 7400.

Table 1 shows some data about the performance of our GRASP algorithm for each instance. The column entries are from left to right:

T - the total running time for the $MaxIter = 50$ GRASP iterations,

IT - the GRASP iteration number where the best feasible solution was found,

COPT - the optimum cost,

BCF - the cost of the best feasible solution found by our algorithm,

GAP - $100 \times \frac{\mathrm{BCF}-\mathrm{COPT}}{\mathrm{COPT}}$ (=percent relative error),

GAPC - $100 \times \frac{\sum_{i=1}^{MaxIter} \mathrm{CCP_i}-\mathrm{COPT}}{MaxIter \cdot \mathrm{COPT}}$ is the average of the percentage relative errors computed over the feasible solutions built in the Construction Phase; being $\mathrm{CCP_i}$ the cost of the built feasible solution in the GRASP iteration i,

GAPL - $100 \times \frac{\sum_{i=1}^{MaxIter} \mathrm{CLS_i}-\mathrm{COPT}}{MaxIter \cdot \mathrm{COPT}}$ is the average of the percentage relative errors computed over the feasible solutions built in the Local Search Phase; being $\mathrm{CLS_i}$ the cost of the best found neighbor feasible solution in the GRASP iteration i.

We observe that the GRASP algorithm has very good results, as an optimal solution is obtained for the first three cases, and a sub-optimal solution with percentage relative error of 0.60 for the last case. To understand the contributions of the initial construction procedure and of the local search algorithm, we show in the last two columns the average gaps of the solutions found in these two steps. We can observe that the construction procedure already obtains good results, as the average gaps

Topology	T	IT	COPT	BCF	GAP	GAPC	GAPL
Network 1	93	3	74	74	0%	2.29%	0.47%
Network 2	145	11	3980	3980	0%	2.40%	0.34%
Network 3	117	7	680	680	0%	2.05%	0.39%
Network 4	139	14	7400	7445	0.60%	3.11%	0.73%

Table 1: GRASP results.

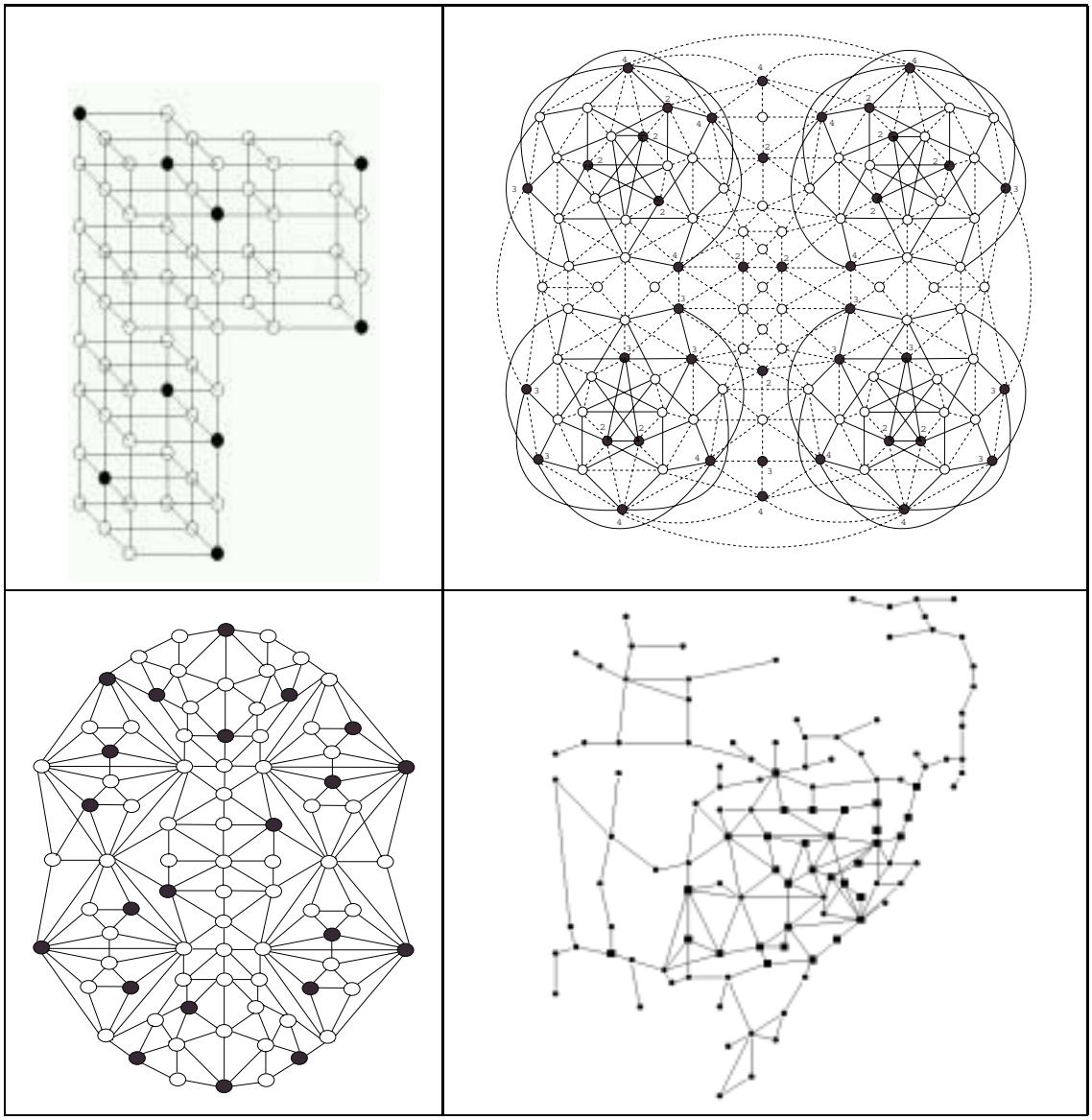

Figure 5: Topologies associated to the instances 1, 2, 3 and 4.

are about $2-3\%$ in all the cases. All the same, the local search algorithm obtains a very significant improvement, as average gaps after its application are less than 0.5% in three cases, and about 0.75% in the remaining one. This shows that even when the construction procedure furnishes good quality solutions, the tree neighborhood proposed is powerful enough and can be used with a simple local search to find improved solutions.

6 Conclusions

We described a greedy randomized search adaptive procedure for the Backbone network design problem with heterogeneous connectivity requirements. This problem can be modeled as a *Generalized Steiner Problem* with node-connectivity requirements.

The local search procedure that we propose is based on decomposing the current solution into its key-path components, and trying to substitute each of these key-paths by an equivalent (from the point of view of terminal connectivity) tree, of smaller cost.

The implementation of our algorithm was tested on different problems with known optimal solutions, either taken from literature related to the problem or generated in order to preserve known optimal, and was shown to find good quality solutions within few iterations (in most cases an optimal solution was found; the only exception was one case where the cost of the best solution found was within 0.61% of the optimum). The construction phase of the GRASP obtained solutions which were usually between 2% and 3% more costly than the optimal ones; on the other hand, each local search iteration obtained results which were on average much nearer the optimum (average gaps in the order of 0.5%). These are interesting results considering that to compute the best solution is a NP-Hard problem.

7 Acknowledgments

This work is a result of the PAIR project, funded by the INRIA, France. The participation of Franco Robledo has been funded by the PDT program (MEC, Uruguay).

References

[1] M.Baïou and A.R. Mahjoub, "Steiner 2-edge-connected subgraph polytopes on series-parallel graphs", SIAM Journal on Discrete Mathematics 10 (3), pages 505-514 (1997).

[2] B. Bollobas, Modern Graph Theory, Springer (1998).

[3] H. Cancela, F. Robledo and G. Rubino, "Network design with node connectivity constraints", LANC'03 (IFIP/ACM Latin America Networking Conference 2003), October 3-5, La Paz, Bolivia (2003).

[4] H. Cancela, F. Robledo and G. Rubino, "A GRASP algorithm for the Generalized Steiner Problem with Node Connectivity Constraints", Internal Report IRISA 1586 (2003).

[5] 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).

[6] T.A. Feo and M.G.C Resende, "Greedy randomized adaptive search procedures", Journal of Global Optimization, 6:109-133 (1995).

[7] M. Grötschel, C.L. Monma, and M. Stoer, "Polyhedral and computational investigations for designing communication networks with high survivability requirements", Operations Research 43 (1995), pages 1012-1024.

[8] A.R. Mahjoub, "Two-edge-connected spanning subgraphs and polyhedra", Mathematical Programming 64 (1994), pages 199-208.

[9] M. Priem and F. Priem, "Ingénierie des WAN", ISBN 2-10-004510-5, Dunod InterEditions (1999).

[10] F. Robledo, "Diseño topológico de redes : casos de estudio *The generalized Steiner problem* y *The Steiner 2-Edge-Connected subgraph problem*". Master Thesis (2000). F. de Ingeniería, UDELAR, Montevideo, Uruguay.

[11] F. Robledo and O. Viera, "A parallel algorithm for the Steiner 2-edge-survivable network problem", Internal Report 1504, IRISA (2002).

[12] K. Stiglitz, P. Weiner, D. J. Kleitman, "The design of minimum-cost survivable networks", IEEE Trans. on Circuit Theory, CT-16, 4 (1969) pages 455-460.

[13] M. Stoer, "Design of Survivable Networks", Lecture Notes in Mathematics, ISBN 3-540-56271-0, ISBN 0-387-56271-0, Springer-Verlag (1996).

[14] M.G.A. Verhoeven, M.E.M. Severens, and E.H.L. Aarts, "Local search for Steiner trees in graphs", In *Modern Heuristics Search Methods*, V.J. Rayward-Smith et al. (Eds.), John Wiley, 117-129 (1996).

[15] P. Winter, "Generalized Steiner problem in series-parallel networks", Journal of Algorithms 7 (1986), pages 549-566.

[16] P. Winter, "Steiner problem in networks: A survey", Networks 17 (1987), pages 129-167.

[17] J. Y. Yen, "Finding the K shortest loopless paths in a network", Management Science 17 (1971), pages 712-716.