IFIP/ACM LANC 2003, oct. 2003

Network design with node connectivity constraints

Héctor Cancela [1], Franco Robledo [1,2], Gerardo Rubino[2]
[1] Universidad de la República
Address : Facultad de Ingeniería, J.Herrera y Reissig 565, Montevideo, Uruguay
Phone:+598-2-7114244 ext. 112, fax: +598-2-7110469
Email: [cancela,frobledo]@fing.edu.uy
[2] IRISA
Address : Campus de Beaulieu, Rennes 35042 CEDEX, France
Phone: +33-2-99847296, fax: +33-2-99847171
Email:Gerardo.Rubino@irisa.fr

Abstract

The Generalized Steiner Problem with Node-Connectivity constraints (GSPNC) consists of determining of a minimum cost subnetwork of a given network where some pairs of nodes are required to satisfy node-connectivity requirements. The GSPNC has applications to the design of low-cost communications networks which can survive failures in the servers as well as in the connection lines. The GSPNC is known to be NP-Complete. In this paper, we introduce an algorithm based on GRASP (Greedy Randomized Adaptive Search Procedure), an effective combinatorial optimization metaheuristic. Experimental results are obtained over a set of problem instances with different characteristics and connectivity requirements, obtaining in all these cases optimal or near-optimal results.
Keywords: network design; node connectivity; Steiner problems; metaheuristics; GRASP.

1 Introduction

When designing a communication network, the aim consists of finding a minimum cost topology which satisfies some additional requirements, generally chosen to improve the survability of the network, that is, its capacity to resist to the failures of some of its components. One way to do this is to specify a connectivity level, and to search for topologies which have at least this number of disjoint paths (either edge disjoint or node disjoint) between pairs of nodes. In the most general case, the connectivity level can be fixed independently for each pair of nodes (heterogeneous connectivity requirements). This problem is known as Generalized Steiner Problem (GSP) [17, 22], and it is a NP-Complete problem [22]. Some references on the GSP and related problems are [1, 8, 9, 13, 15, 18, 21]; these works are either focused on the edge-disjoint flavour of the problem, or explore particular cases (for example, when it is required to have two disjoint paths between all pairs of nodes [2, 3, 10, 11, 12, 14]).

Topologies verifying edge-disjoint path connectivity constraints ensure that the network can survive to failures in the connection lines; while node-disjoint path constraints ensure that the network can survive to failures both in server sites as well as in connection lines. Finding the least cost network topology that satisfies these node-disjoint path connection requirements is modeled as a Generalized

Steiner Problem with Node-Connectivity constraints (GSPNC), which is formalized in Section 2. As mentioned before, this class of problem belongs to the NP-complete class. This means that all (known) algorithms for solving it take computing time exponential in the size of the problem instances.

The purpose of this work is to develop a heuristic method which can give small cost topologies satisfying all constraints in reasonable computing times. The method developed here is based on GRASP (Greedy Randomized Adaptive Search Procedure). This metaheuristics, which has been used to solve different combinatorial optimization problems with good results, is discussed in Section 3. Its adaptation for solving the GSPNC is presented in Section 4.

Exploring the performance of the proposed method involves applying it to a number of different test problems. Section 5 presents experimental results obtained on a set of problems which include both unstructured and highly structured topologies of more than a hundred nodes. Some conclusions and lines for future work are discussed in Section 6.

2 Problem formalization and example

We formalize the GSPNC as follows. Given a non-directed simple graph $G = (V, E)$, a matrix $C = \{c_{ij}\}_{i,j \in V}$ of nonnegative edge-costs, a subset $T \subseteq V$ called "set of terminal nodes", a matrix $R = \{r_{ij}\}_{i,j \in T}$ of required local node-connectivities between any pair of different nodes in T (for any i,j in T, r_{ij} is a non-negative integer number), the goal is to find a subgraph G_T of G with minimal cost so that for every pair of nodes $i, j \in T, i \neq j$, there are at least r_{ij} node-disjoint paths connecting i and j in G_T. The nodes in $V \backslash T$ are usually called Steiner nodes. We will denote by Γ_{GSPNC} the space of feasible solutions associated with the problem.

We present a small problem instance example, based on the network shown in Figure 1. In this network there are four terminal nodes, colored white and labeled T_1, T_2, T_3 and T_4, and six Steiner nodes, colored black. The edges that can be used to build a solution are shown, annotated with their costs. The connection requirements are the following (supposing the R elements ordered by the terminal indexes):

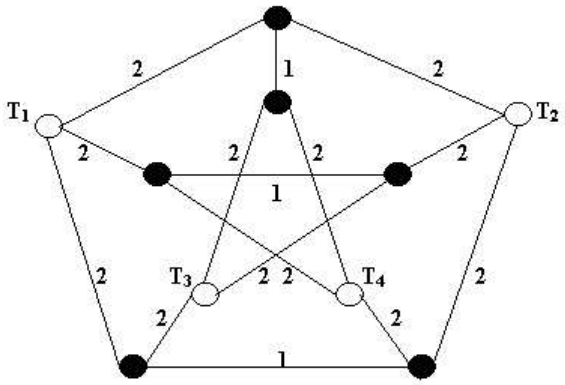

Figure 1: Example based on Petersen graph

$$R = \begin{pmatrix} - & 0 & 2 & 3 \\ 0 & - & 1 & 0 \\ 2 & 1 & - & 0 \\ 3 & 0 & 0 & - \end{pmatrix}.$$

This corresponds to asking for two node disjoint paths between T_1 and T_3, three node disjoint paths between T_1 and T_4, and one path between T_2 and T_3.

Figure 2 presents a solution to this problem instance. As it can be seen, not all Steiner nodes are needed in order to satisfy the connection requirements.

3 Greedy Randomized Adaptive Search Procedure (GRASP)

GRASP is a well known metaheuristic, which has been applied for solving many hard combinatorial optimization problems with very good results [6, 7].

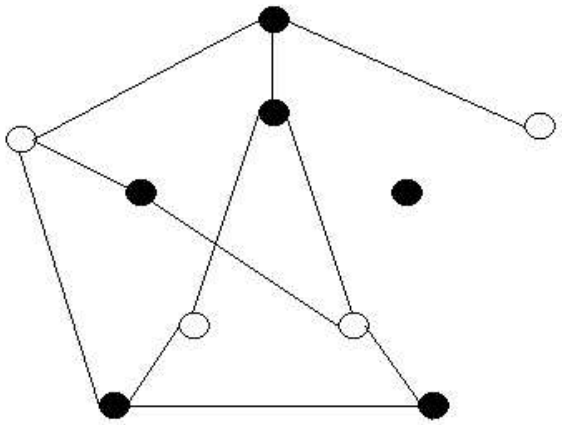

Figure 2: A cost 20 solution for Petersen graph example

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 in the local search phase, looking for an improvement. The best solution over all GRASP iterations is returned as the result.

Figure 3 illustrates a generic GRASP implementation pseudo-code. The GRASP has two main parameters: the candidate list size and the maximum number of GRASP iterations. The GRASP iterations are carried out in lines 1-7. Each GRASP iteration consists of the construction phase (line 2), the local search phase (line 3) and, if necessary, the solution update (lines 4 and 5).

In the construction phase, a feasible solution is built. Usually the solution is represented as a set of elements in the problem domain; the construction phase starts from an empty set and iteratively adds an element until the set corresponds to a feasible solution. 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 to the solution. This probabilistic component of GRASP allows for different solutions to be obtained at each GRASP iteration, while taking into account the adaptive greedy benefit function.

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. The construction phase plays an important role with respect to this last point, since it must produce good starting solutions for local search. Normally, a local optimization procedure, such as "two-exchange", is employed. While such procedures can require exponential time from an arbitrary starting point, empirically their efficiency significantly improves as the initial solutions improve. Through the use of customized data structures and careful implementation, an efficient construction phase that produces good initial solutions for efficient local search can be created. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the solution obtained from a random starting point.

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

1  for k = 1 to MaxIter do
2        InitialSolution = ConstructGreedyRandomizedSolution(ListSize);
3        LocalSearchSolution = LocalSearch(InitialSolution);
4        if cost(LocalSearchSolution) < cost(BestSolutionFound) then
5              UpdateSolution(BestSolutionFound, LocalSearchSolution);
6        end_if;
7  end_for;
8  return BestSolutionFound;
```

Figure 3: GRASP pseudo-code

4 A customized GRASP for network design

In order to apply GRASP to our particular optimization problem, it is necessary to customize the different components outlined above. In this section we define a path-based construction phase and a path-based local search suitable for the GSPNC, called respectively GSPNC_Construction_Phase and GSPNC_Local_Search.

4.1 Solution Construction Phase

This phase corresponds to the procedure GSPNC_Construction_Phase; a pseudo-code is given in Figure 4.

The method proposed can be seen as an extension of the Takahashi-Matsuyama algorithm [19], which is a heuristic for computing a (hopefully low cost) Steiner tree, and works by searching for shortest paths between pairs of nodes not already connected. Our extension has a quite different objective, as we need to efficiently compute k low cost node-disjoint paths from i to j, with $i, j \in T$.

The algorithm takes as input the original graph G, the matrix R of connection requirements between terminal nodes, and the matrix of connection costs C.

Line 1 of the pseudo-code corresponds to the initialization procedure. The current solution $\mathcal{G}_{sol}$ is initially defined as the network containing only the terminal nodes without any edge connecting them. An auxiliary matrix M is defined and initialized with the values of R. This matrix will be used for maintaining updated the requirements of connection between nodes of T not yet satisfied by the current solution $\mathcal{G}_{sol}$. Also, the paths found in each iteration are stored in a data structure $\mathcal{P}$ which initially is empty. Moreover, all terminal nodes are randomly assigned unique identifiers; so that each time the procedure is called, it will consider these nodes in a different order.

The main loop from line 2 to line 9 starts from the current solution $\mathcal{G}_{sol}$, incrementing it with additional paths between terminal nodes of T, until all connection requirements are satisfied. In line 3 we chose two terminal nodes $i, j \in T$ whose connection requirement are not yet satisfied and which maximize the sum of their identifiers. In line 4 we consider the graph $\bar{\mathcal{G}}$ obtained by eliminating from G all the nodes belonging to the paths already stored in the set $\mathcal{P}_{ij}$. This auxiliary graph is implicitly computed when we search for a new node-disjoint path between i and j (it is not necessary to store it in a data structure apart). In line 5 we consider the auxiliary matrix $\bar{C}$ defined from C so that the edges that belong to $\mathcal{G}_{sol}$ have cost 0. In line 6 we work upon network $\bar{\mathcal{G}}$ with costs $\bar{C}$, and find the k shortest paths from i to j (using a classical algorithm [16]), which will be stored in the restricted candidate list $\mathcal{L}_p$. By their construction, each of

Procedure GSPNC_Construction_Phase(G,R,C);

1 $\mathcal{G}_{sol} \leftarrow (T, \emptyset)$; $M \leftarrow R$; $\mathcal{P}_{ij} \leftarrow \emptyset$ $\forall i, j \in T$;
$\forall k \in T$, k **is assigned a random identifier** n_k;
2 **while** $\exists m_{ij} > 0$ **do**
3 **Let** $i, j \in T$ **such that** $m_{ij} > 0$ **and** $n_i + n_j = \max\{n_u + n_v \ : m_{uv} > 0\}$;
4 $\bar{\mathcal{G}} \leftarrow G \setminus (\text{NODES}(\mathcal{P}_{ij}) \setminus \{i, j\})$;
5 **Let** $\bar{C}$ **be the matrix given by:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_{sol}, \\ c_{uv} & \textbf{if } (u,v) \in \bar{\mathcal{G}} \setminus \mathcal{G}_{sol}; \end{cases}$
6 $\mathcal{L}_p \leftarrow$ **the** k **shortest paths from** i **to** j **on** $\bar{\mathcal{G}}$**, considering cost matrix** $\bar{C}$;
7 $p \leftarrow \text{Select\_Random}(\mathcal{L}_p)$;
8 $\mathcal{G}_{sol} \leftarrow \mathcal{G}_{sol} \cup \{p\}$; $\mathcal{P}_{ij} \leftarrow \mathcal{P}_{ij} \cup \{p\}$; $M \leftarrow \text{Update\_Matrix}(\mathcal{G}_{sol}, M, R, p)$;
9 **end_while;**
10 **return** $\mathcal{G}_{sol}$, $\mathcal{P}$;
end GSPNC_Construction_Phase;

Figure 4: Construction Phase pseudo-code

these paths will be node disjoint with those already in $\mathcal{P}_{ij}$. One path p is selected at random from the candidate list $\mathcal{L}_p$ (line 7). Then, the current solution is modified by adding this new node-disjoint path p between the chosen nodes; this corresponds to line 8, where the current graph $\mathcal{G}_{sol}$, the sub-set of node-disjoint paths $\mathcal{P}_{ij}$, and the auxiliary matrix M are updated.

In line 10, once all the connection requirements have been satisfied, the solution $\mathcal{G}_{sol}$ and the set of node-disjoint paths $\mathcal{P}$ are returned.

4.2 Local Search

Since the feasible solution built by the construction phase algorithm usually is not even a local optimum, the GRASP metaheuristics applies a local search phase in order to improve this solution. The local search strategy this work proposes for the GSPNC is an extension of the key-path based local search, proposed by Verhoeven, Severens and Aarts [20] in the context of the Steiner tree problem. Before defining the local search strategy, we introduce some notation and auxiliary definitions, and in particular we define a suitable structure for the neighborhood.

Definition 4.1 *Given a feasible solution $\mathcal{G} \in \Gamma_{GSPNC}$, we define a joint-node as a Steiner node (that is, a node in $V \setminus T$) with degree at least three in $\mathcal{G}$.*

Definition 4.2 *Given a feasible solution $\mathcal{G} \in \Gamma_{GSPNC}$, we define a key-path as a path in $\mathcal{G}$ such that all its intermediate nodes are Steiner nodes with degree two in $\mathcal{G}$, and whose end nodes are either terminal nodes or joint-nodes.*

Notation 4.3 *Let $\mathcal{G} \in \Gamma_{GSPNC}$ be a feasible solution. We will denote by $K(\mathcal{G}) = (p_1, \ldots, p_h)$ its decomposition in key-paths ordered by decreasing cost.*

Definition 4.4 *Let $\mathcal{G}$ be the current graph being built iteratively by* GSPNC_Construction_Phase. *Given a key-path $q \in K(\mathcal{G})$, we denote:*

$$\mathcal{V}_q(\mathcal{G}) = \{(i,j) \in T \times T | \exists p \in \mathcal{P}_{ij},\ q \subseteq p\}\,.$$

This is the set of pairs of terminal nodes of T which "depend on" key-path q, that is to say, q is part of at least one path p in the set $\mathcal{P}_{ij}$ of the node-disjoint paths connecting i and j.

Definition 4.5 *Let $\mathcal{G} \in \Gamma_{GSPNC}$ be the feasible solution computed by the algorithm*

Procedure GSPNC_Local_Search(G,R,C,$\mathcal{P}$,$\mathcal{G}_s$,$K(\mathcal{G}_s)$);

1 $improve \leftarrow$ TRUE;
2 **while** $improve$ **do**
3 $improve \leftarrow$ FALSE;
4 **for** $k = 1, \ldots, |K(\mathcal{G}_s)|$ **do**
5 **Let** p_k **be the** k**-th key-path;** s_1, s_2 **be the ends of** p_k**;**
Let $\bar{p}_k = p_k - s_1 - s_2$ **be** p_k **minus its first and last edges;**
6 $\bar{\mathcal{G}}_s \leftarrow G \setminus \mathcal{X}_{(p_k)}(\mathcal{G}_s)$;
7 **Matrix** $\bar{C}$ **is computed as:** $\bar{c}_{uv} \leftarrow \begin{cases} 0 & \textbf{if } (u,v) \in \mathcal{G}_s, \\ c_{uv} & \textbf{if } (u,v) \in \bar{\mathcal{G}}_s \setminus \mathcal{G}_s; \end{cases}$
8 $\hat{p}_k \leftarrow$ **the shortest path from** s_1 **to** s_2 **on** $\bar{\mathcal{G}}_s$ **considering** $\bar{C}$;
9 **if** $\mathrm{COST}(\hat{p}_k) < \mathrm{COST}(p_k)$ **then**
10 $\mathcal{G}_s \leftarrow (\mathcal{G}_s \setminus \bar{p}_k) \cup \hat{p}_k$; $\mathcal{P} \leftarrow \mathrm{Update\_Paths}(\mathcal{P}, \bar{p}_k, \hat{p}_k)$;
11 **if** *necessary* **then** $K(\mathcal{G}) \leftarrow \mathrm{Update\_KeyPaths}(K(\mathcal{G}), \bar{p}_k, \hat{p}_k)$;
12 $improve \leftarrow$ TRUE;
13 **end_if;**
14 **end_for;**
15 **end_while;**
16 **return** $\mathcal{G}$,$\mathcal{P}$;
end GSPNC_Local_Search;

Figure 5: Local Search Phase pseudo-code

GSPNC_Construction_Phase. *Given a key-path $q \in K(\mathcal{G})$, we define the following set:*

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

This set is the union over every pair of nodes i, 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 (Neighborhood) *Let $\mathcal{G}_s \in \Gamma_{GSPNC}$ be a feasible solution. Given a key-path $p \subset \mathcal{G}_s$, we define a neighbor solution of $\mathcal{G}_s$ as: $\hat{\mathcal{G}}_s = (\mathcal{G}_s \setminus \mathrm{EDGES}(p)) \cup \hat{p}$, where $\hat{p}$ is a minimum cost path connecting the endpoints of p and maintaining the feasibility in the new graph $\hat{\mathcal{G}}_s$.*

The Path 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 GSPNC_Local_Search pseudo-code is given in Figure 5. The algorithm takes as inputs the original graph G, matrix R, matrix C, the set $\mathcal{P}$ of found paths, the current solution $\mathcal{G}_s$ (computed in the construction phase), and its decomposition in key-paths $K(\mathcal{G}_s)$. The method loops while it finds smaller cost neighbor solutions, according to the previously defined *Path Neighborhood*. There is a inner iteration, lines 4 to 12, where each key-path of the decomposition $K(\mathcal{G}_s)$ is analyzed, and if possible replaced by a new path of smaller cost connecting its ends, so that the feasibility of the current solution is preserved; this loop finalizes when there are no more improvements to be obtained by substituting key-paths by new paths. In line 5 we consider the k-th key-path of $K(\mathcal{G}_s)$. In line 6 we consider an auxiliary graph $\bar{\mathcal{G}}_s$ derived from G by eliminating the nodes of $\mathcal{X}_{(p_k)}(\mathcal{G}_s)$. In line 7, we consider also an auxiliary matrix $\bar{C}$, where the costs of the edges present in the current solution are 0. Using $\bar{\mathcal{G}}_s$ and $\bar{C}$, in line 8 we compute a path which can replace the current key-path creating a new neighbor so-

lution, thus maintaining its feasibility. In line 9 the cost of this path is compared to the cost of the current key-path. If the cost of the path is strictly smaller, a new feasible solution is computed in line 10, replacing the current key-path by the found path, and moreover the set of paths $\mathcal{P}$ is updated. The decomposition in key-paths of the new solution is built in line 11, and the variable indicating there was an improvement found is set in line 12.

Once the search has finished, the best feasible solution and the set of paths are returned in line 16.

5 Experimental results

In order to select a set of test problems for the experiments, we scanned previous literature on GSP and selected four problem instances with relatively important size (more than 100 nodes in three cases). In the four cases an optimal solution has been published [14, 15, 18]; this allows us to study the effectiveness of our GRASP technique. In addition, we generated another problem instance which was designed constructively in order to have a known optimal solution.

We describe below the main characteristics of the five problem instances. Figure 6 shows the topologies associated with the test cases 1, 2, 3 and 5 (test case 4, not shown for space reasons, is similar to test case 5). The white nodes represent the terminal nodes, while the black nodes represent the Steiner nodes, which may or may not be included in the solution.

- Network 1 has a double grid structure, with 33 terminal nodes, 87 Steiner nodes and 268 edges. A link between Steiner nodes has cost 1, a link between a Steiner node and a terminal node has cost 2 and a link between two terminal nodes has cost 4. The objective is to find a 2-node-survivable subnetwork with minimal cost (all connection requirement between terminal nodes are equal to 2).

- Network 2 represents a simplified version of a HSODTN (High Speed Optical Data Transmission Network) connecting different parts of a war ship, allowing for communication between weapon systems. The reduced topology has 9 terminal nodes (modeling strategic points in an aircraft carrier), 55 Steiner nodes, and 134 edges. A link between Steiner nodes has cost 1, a link between a Steiner node and a terminal node has cost 2 and a link between two terminal nodes has cost 4. The objective is to find a 3-node-survivable network with minimal cost. This model and other variants can be found in [9, 8, 18].

- Network 3 has been constructed applying certain topological properties related to graphs satisfying the triangular inequality, cited in [5], leading to the existence of a known optimal solution. In particular, we employ a theorem proved by Mader in 1978, which specifies sufficient conditions for the existence of a k node connected subgraph in a given network, and also holds for the Steiner case. The network has 22 terminal nodes, 61 Steiner nodes and 262 edges. The links have costs randomly selected in the interval $[1, 200]$, satisfying the triangular inequality. The objective is to find a 4-node-survivable subnetwork spanning the terminal nodes set T (all connection requirement between terminal nodes are equal to 4).

- Network 4 and Network 5 are the test cases respectively called LATAD5S and LATADL [4, 8, 18], based on real networks from Bell Communications Research (later Bellcore, now Telcordia Technologies). Link costs are linked to geographical distances. The objective is to find a 2-node-survivable subnetwork with minimal cost. The LATA5S-problem has 38 nodes and 71 edges, and the LATADL-problem has 116 nodes

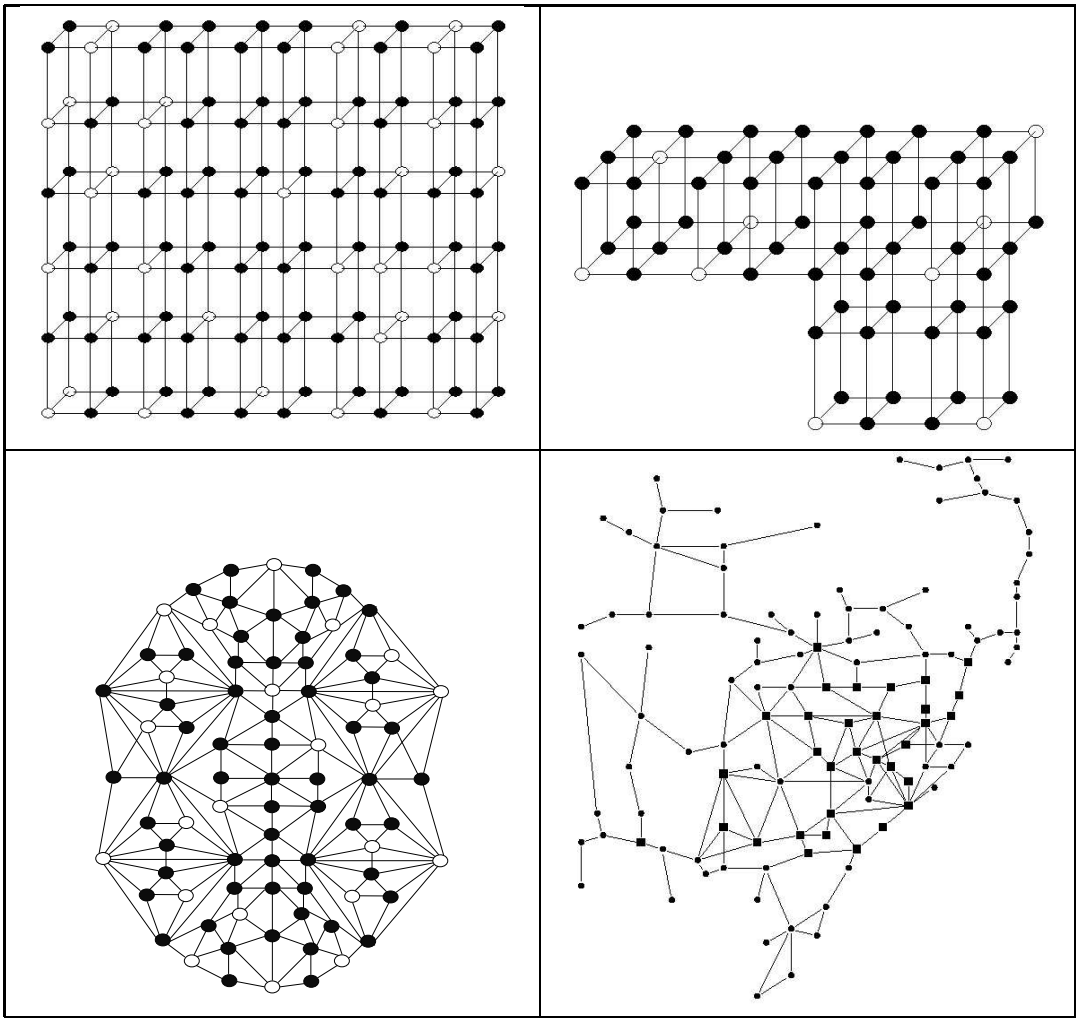

Figure 6: Topologies associated with the instances 1, 2, 3 and 5.

and 173 edges. In these problems, there are two classes of nodes: nodes of type 1, shown as circles in Figure 6, and nodes of type 2. 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.

Optimal solutions for Network 1 and Network 2 have been published in [15]; the respective minimum costs are 140 and 74 respectively. These optimal topologies were found by an exact parallel-distributed backtracking algorithm. For Network 4 and Network 5, optimal solutions have been published in [18], with costs 4739 and 7400 respectively. For Network 3, we know an optimal solution of cost 680, obtained from the constructive procedure used to build this test case.

The method presented in Section 4 was implemented in ANSI C, using adjacency matrices for the representation of graphs. The experiments were obtained on a Pentium III computer with clock speed of 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 $MaxIter = 30$. These values were chosen based on the GRASP reference literature [6, 7].

The runs performed on these instances

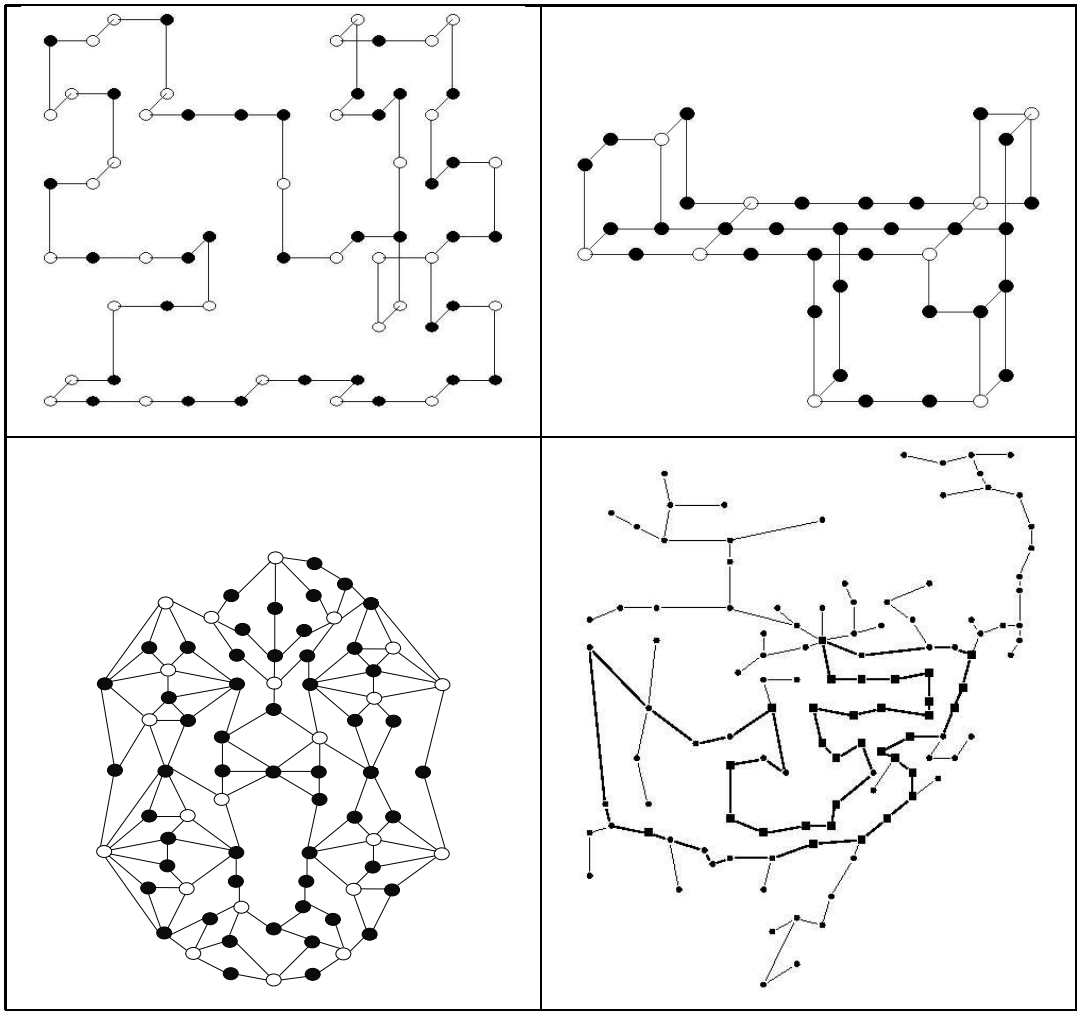

Figure 7: Optimal solutions associated with the instances 1, 2, 3 and 5.

proved to be successful, the optimal solution being obtained by GRASP for the first four cases, and a sub-optimal solution with percent relative error of 0.71 for the last case. Table 1 shows some data about the performance of our GRASP algorithm for each instance. The entries correspond to the test case names, from left to right are:

Nodes- total number of nodes,

Edges- total number of edges,

T- the total running time for the $MaxIter = 30$ 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).

These results show the potential of our algorithm, computing good feasible solutions with polynomial time (on only one instance the percentage relative error was slightly greater than 0.7%). This must be confirmed with more extensive testing, including other test topologies with different characteristics.

Figure 7 shows optimal solutions associated with the test cases 1, 2, 3 and 5.

6 Conclusions

By modelling the network design problem with heterogenous connectivity require-

Topology	Nodes	Edges	T (sec)	IT	COPT	BCF	GAP
Network 1	120	286	82	7	145	145	0
Network 2	64	124	54	6	74	74	0
Network 3	83	262	73	8	680	680	0
Network 4	38	71	41	10	4739	4739	0
Network 5	116	173	101	18	7400	7453	0.71

Table 1: Computational results

ments as a Generalized Steiner Problem with node-connectivity requirements, we were able to develop a greedy randomized search adaptive procedure which can give an approximate solution.

The implementation of our algorithm was tested on a number of different problems taken from literature related to the problem, and was shown to find good quality solutions within few iterations for these problem instances (in most cases an optimal solution was found; in the only exception the cost of the best solution found was within 0.71% of the optimum). These are very good results considering that to compute the best solution is a NP-Hard problem [18, 22]; in particular, the known exact algorithms have worst case computing time which grows exponentially in the number of terminal nodes and edges.

Execution time of the proposed GRASP method is also dependent on the number of nodes, edges, and the connection requirements, but increases much slower.

These are (up to our knowledge) the first results on the use of metheuristics for solving the Generalized Steiner Problem with node connectivity constraints applicable to a general graph class.

As future work, it is possible to search for new methods which improve either the initial construction or the local search phases of the GRASP. In particular, we are working on the design of an alternative local search strategy, which is based on the idea of replacing key-paths by trees. This new approach has a more complex implementation but is more flexible than the one presented here; experimental comparison between the two alternatives is needed to decide which is the more effective strategy.

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] S. Arraga, M. Aroztegui, and S. Nesmachnow. Resolución del problema de Steiner generalizado utilizando un algoritmo genético paralelo (text in Spanish). In *3er Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pages 38–45, 1995.

[2] M. Baïou. *Le problème du sous-graphe Steiner 2-aréte-connexe : approche polyédrale (text in French)*. PhD thesis, University of Rennes 1, 1996.

[3] M. Baïou and A.R. Mahjoub. Steiner 2-edge-connected subgraph polytopes on series-parallel graphs. *SIAM Journal on Discrete Mathematics*, 1993.

[4] Bellcore. Fiber options. accessed at http//www.bellcore.com, 1988.

[5] R. Diestel. *Graph Theory*. Springer, 1998.

[6] T.A. Feo and M.G.C Resende. A probabilistic heuristic for a computionally difficult set covering problem. *Operations Research Letters*, 8:67–71, 1989.

[7] T.A. Feo and M.G.C Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[8] M. Grötschel, C.L. Monma, and M. Stoer. Design of survivable networks. In Ball M. O., T. L. Magnanti, C. L. Monma, and G.L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*. Informs, North Holland, 1995.

[9] M. Grötschel, C.L. Monma, and M. Stoer. Polyhedral and computational investigations for designing communication networks with high survivability requirements. *Operations Research*, 43, 1995.

[10] I. Ljubic and J. Kratica. A genetic algorithm for biconnectivity augmentation problem. In *Proc. of the 2000 IEEE Congress on Evolutionary Computation*, pages 89–96. IEEE Press, 2000.

[11] I. Ljubic, G. Raidl, and J. Kratica. A hybrid GA for the edge-biconnectivity augmentation problem. In *Proceedings of the 2000 Parallel Problem Solving from Nature VI Conference*, pages 641–650. Springer, 2000.

[12] A.R. Mahjoub. Two-edge-connected spanning subgraphs and polyhedra. *Mathematical Programming*, 64:199–208, 1994.

[13] M. Priem and F. Priem. *Ingénierie des WAN (text in French)*. Dunod InterEditions, 1999.

[14] F. Robledo and O. Viera. An Ant-System algorithm for the Generalized Steiner Problem with edge-connectivity. Research report PI 1503 (`http://www.irisa.fr/bibli/publi/pi/2002/1503/1503.html`), IRISA, Rennes, France, 2002.

[15] F. Robledo and O. Viera. A parallel algorithm for the Steiner 2-edge-survivable network problem. Research Report PI 1504 (`http://www.irisa.fr/bibli/publi/pi/2002/1504/1504.html`), IRISA, Rennes, France, 2002.

[16] D. R Shier. Iterative methods for determining the k shortest paths in a network. *Networks*, 6:205–229, 1976.

[17] K. Stiglitz, P. Weiner, and D. J. Kleitman. The design of minimum-cost survivable networks. *IEEE Trans. on Circuit Theory*, 16(4):455–460, 1969.

[18] M. Stoer. *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, 1996.

[19] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Jpn.*, 24:537–577, 1980.

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

[21] P. Winter. Generalized Steiner problem in series-parallel networks. *Journal of Algorithms*, 7:549–566, 1986.

[22] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.