# Finding Steiner trees with degree 1 terminal nodes

**H. Cancela**[2], **F. Robledo**[1,2] **and G. Rubino**[1]
[1] *ARMOR team, IRISA/INRIA Rennes*
*Campus de Beaulieu 35042, Rennes, France*
[2] *Facultad de Ingeniería, Universidad de la República*
*J. Herrera y Reissig 565, Montevideo, Uruguay*
*cancela@fing.edu.uy, frobledo@irisa.fr, rubino@irisa.fr*

**Abstract:** We consider in this work a variant of the Steiner Problem in Graphs (SPG) with the additional restriction that terminal nodes must have degree 1.

We customize the Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic, testing its performance on a large problem set obtained by modifying 182 SPG instances from the SteinLib repository. The GRASP obtained good results, with low average gaps with respect to known lower bounds in most of the problem classes, and attaining the optimum in 44 cases (24% of the problem set).

**Keywords:** metaheuristic, Steiner problem in graphs, GRASP.
**Classification:** E019000 Science and engineering for electronics

## References

[1] T.A. Feo and M.G.C Resende, "Greedy randomized adaptive search procedures", Journal of Global Optimization 6 (1995), pages 109-133.
[2] M. Jünger, A. Martin, G. Reinelt, and R. Weismantel, "Quadratic 0/1 optimization and a decomposition approach for the placement of electronic circuits", Mathematical Programming 63 (1994), pages 257-279.
[3] T. Koch, A. Martin and S. Voß, "SteinLib: An Updated Library on Steiner Tree Problems in Graphs", Konrad-Zuse-Zentrum für Informationstechnik Berlin, ZIB-Report 00-37 (2000), url: http://elib.zib.de/steinlib.
[4] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos, "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", Journal of Global Optimization 17 (2000), pages 267-283.
[5] M. Minoux, "Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity", INFOR 28 (1990), pages 221-233.
[6] H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs", Math. Jpn. 24 (1980), pages 537-577.
[7] E. Uchoa, M.P. de Aragão and C. Ribeiro, "Preprocessing *Steiner* problems from *VLSI* layout", Catholic University of Rio de Janeiro, Technical Report MCC 32-99 (1999).
[8] J. Y. Yen, "Finding the K shortest loopless paths in a network", Management Science 17 (1971), pages 712-716.
[9] M. Zachariasen and A. Rohe, "Rectilinear group Steiner trees and applications in *VLSI* design", Institute for Discrete Mathematics, Technical Report 00906 (2000).

## 1 Introduction

Given a graph $G = (V, E)$, where $V$ is the set of nodes, $E$ is the set of edges, and given a distinguished subset of nodes $T$ called the terminal vertices, the Steiner Problem in Graphs consists of finding a minimum cost subgraph such that all terminal nodes are connected, eventually using non-terminal nodes (also called Steiner nodes; we denote by $S = V \backslash T$ their set). The cost of the edges of the graph is given by $C = \{c_{ij}\}_{i,j \in V}$ which is the matrix which gives for any pair of sites of $V$, the cost of a direct connection between them. When the direct connection is not possible, we take $c_{ij} = \infty$.

The Steiner Problem in Graphs and its variants is an useful model for a number of circuit design and communication network design problems [2, 7, 9]. We study here an important variant of the problem: suppose we have a fixed node $z$ (called the root), and that we want to connect all nodes in $T$ to $z$ (eventually using some Steiner nodes), in such a way that all nodes in $T$ have degree 1 in the optimal solution (i.e., terminal sites can not be used as intermediate nodes; this implies in particular that connections between pairs of terminal sites are not allowed). We call this problem the Rooted Steiner Problem in Graphs (RSPG); it belongs to the NP-Hard class, as can be proved by reducing in polynomial time the Steiner Problem in Graphs to it. In Section 2 we present a polynomial time heuristic based on the GRASP methodology for approximately solving the RSPG. Section 3 reports computational results, obtained on a set of 182 problem instances, and some conclusions.

## 2 GRASP customization for the RSPG

GRASP [1] is an iterative metaheuristic optimization procedure, in which each iteration consists of two phases: construction and local search [1, 4]. The construction phase builds a feasible solution, whose neighborhood is explored by the local search phase. Figure **1** shows the outline of the GRASP method, adapted to solve the RSPG; in this proposal we employ as building blocks a path-based construction phase and a minimum spanning tree-based local search. The main parameters of the method are $MaxIter$, the number of outer iterations to be performed, and $k$, the candidate list size, which is a parameter of the construction phase controlling the diversity of the solutions built in that phase. The algorithm iterates $Max\_iter$ times, executing first the construction phase, and afterwards two different local search methods; these procedures are described in the following subsections. At the end of all iterations, the best solution found is returned.

### 2.1 Construction Phase

Our construction phase can be seen as a customized version of the Takahashi-Matsuyama algorithm [6]. The algorithm (shown in Figure **1**) takes as inputs the graph $G$, the matrix of costs $C$ and the GRASP parameter $k$ (the candidate list size). The current solution $\mathcal{G}_{sol}$ is initialized with node $z$. Iteratively the construction phase adds new terminal nodes to the current solution. On each iteration, the algo-

rithm chooses randomly a terminal node $t$ not yet included in the current solution $\mathcal{G}_{sol}$ and computes the $k$ shortest paths from $t$ to $\mathcal{G}_{sol}$ using the Yen algorithm [8]. These paths are stored in a restricted candidate list $\mathcal{L}_p$. A path $p$ is randomly selected from $\mathcal{L}_p$ and added to $\mathcal{G}_{sol}$. This process is repeated until all the terminal nodes have been added; then $\mathcal{G}_{sol}$, which is a feasible solution, is returned.

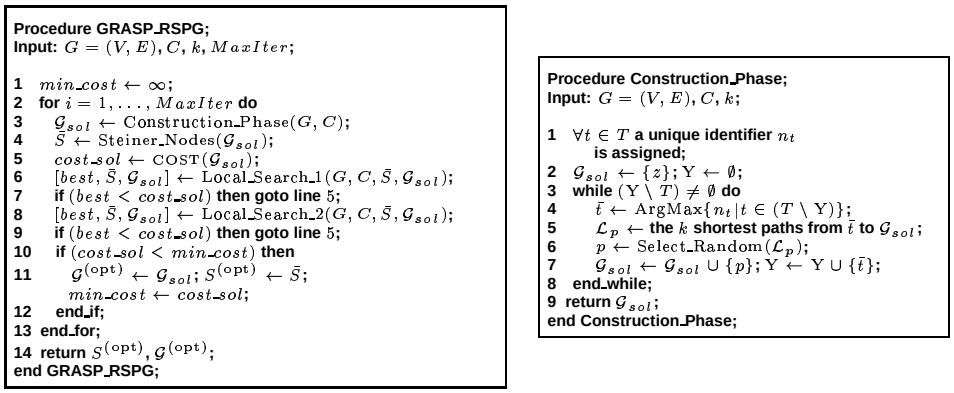


**Fig. 1.** GRASP and Construction methods.

### 2.2 Local Search Phase

Usually the feasible solution built by the construction phase algorithm is not locally optimal. The GRASP metaheuristic applies a local search phase in order to improve this solution. We now introduce two different local search strategies that will be used by our algorithm.

**Definition 2.1 (Neighborhood Structure)** *Let $\bar{S} \subseteq S$ be the subset of Steiner nodes associated with a feasible solution of the $RSPG(G(V, E), C)$. The neighbors of a solution characterized by its set $\bar{S}$ are defined by all the sets of Steiner nodes which can be obtained by adding a new Steiner node to $\bar{S}$, or by eliminating one of the Steiner nodes from $\bar{S}$.*

### 2.3 Local Search Strategy by concentrator site insertion

The algorithm Local_Search_1 (shown in Figure **2**) takes the current solution $\mathcal{G}_{sol}$ and its set of Steiner nodes $\bar{S}$, and tries to improve $\mathcal{G}_{sol}$ by inserting a new Steiner node into $\bar{S}$. Line 1 initializes the best solution with $\mathcal{G}_{sol}$. Lines 2-10 search for a better neighbor solution adding a new Steiner node $s \in S \setminus \bar{S}$ into $\bar{S}$. Line 3 computes the sub-graph $\mathcal{H}$ induced by the set $\{z\} \cup \bar{S} \cup \{s\}$. This graph includes all possible topologies that have as set of Steiner nodes a subset of $\bar{S} \cup \{s\}$. Line 4 computes a minimum spanning tree $\mathcal{G}$ for this graph using an algorithm by Minoux [5]. Line 5 connects the nodes of $T$ to $\mathcal{G}$ by means of edges of minimum cost. The resulting graph is feasible for the RSPG. Iteratively, line 6 removes the pendant nodes. In lines 7-9, we update (if necessary) the best solution. Once all possible ways to insert a new Steiner node have been considered, Local_Search_1 returns the best solution found, its cost, and its set of Steiner nodes.

```
Procedure Local_Search_1;
Input: G = (V, E), C, G_sol, S̄;
1  best ← COST(G_sol); Ŝ ← S̄; G_best ← G_sol;
2  for all s ∈ (S \ S̄) do
3     H ← sub-graph induced by ({z} ∪ S̄ ∪ {s});
4     G ← MST computed by Minoux_Algorithm(H, C);
5     Compute ∀t ∈ T:
            e ← the edge of minimum cost from t to G;
            G ← G ∪ {e};
6     Iteratively remove all Steiner nodes from G with
       degree 1;
7     if (COST(G) < best) then
8        best ← COST(G); Ŝ ← S̄ ∪ {s}; G_best ← G;
9     end_if;
10 end_for;
11 return best, Ŝ, G_best;
end Local_Search_1;
```

```
Procedure Local_Search_2;
Input: G = (V, E), C, G_sol, S̄;
1  best ← COST(G_sol); Ŝ ← S̄; G_best ← G_sol;
2  for all s ∈ S̄ do
3     H ← sub-graph induced by ({z} ∪ (S̄ \ {s}));
4     Ĥ ← connected comp. of H such that z ∈ Ĥ;
5     G ← MST(Ĥ, C);
6     Compute ∀t ∈ T:
            e ← the edge of minimum cost from t to G;
            G ← G ∪ {e};
7     Iteratively remove all Steiner nodes from G with
       degree 1;
8     if (COST(G) < best) then
9        best ← COST(G); Ŝ ← (S̄ \ {s}); G_best ← G;
10    end_if;
11 end_for;
12 return best, Ŝ, G_best;
end Local_Search_2;
```

**Fig. 2.** Local Search by insertion and by elimination moves.

### 2.4 Local Search Strategy by concentrator site elimination

The algorithm Local_Search_2 (shown in Figure **2**) takes the current solution $\mathcal{G}_{sol}$ and its set of Steiner nodes $\bar{S}$, and tries to find a better solution by eliminating Steiner nodes belonging to $\bar{S}$. As in Local_Search_1, the best solution is initialized with $\mathcal{G}_{sol}$. Lines 2-11 search for the best neighbor solution eliminating a Steiner node $s \in \bar{S}$. Line 3 computes the subgraph $\mathcal{H}$ induced by the set $\{z\} \cup (\bar{S} \setminus \{s\})$. Since this graph can be unconnected, line 4 computes the connected component $\hat{\mathcal{H}} \subseteq \mathcal{H}$ containing $z$. Clearly, the Steiner nodes non-belonging to $\hat{\mathcal{H}}$ cannot be considered since they can induce a non-feasible solution. Line 5 computes a minimum spanning tree $\mathcal{T} \subset \hat{\mathcal{H}}$. Line 6 connects the nodes of $T$ to $\mathcal{G}$ by means of edges of minimum cost. The resulting topology is feasible for the RSPG. Iteratively, line 7 removes the pendant nodes. In lines 8-10, we update (if necessary) the best solution. Once all possible ways to eliminate Steiner nodes have been considered, Local_Search_2 returns the best solution found, its cost, and its set of Steiner nodes.

## 3 Performance Tests and Discussion

We present experimental results obtained with an ANSI C implementation of the previous algorithm, running on a Pentium IV, 1.7 GHz computer, with 1 Gigabyte of RAM, under Windows XP. The GRASP parameter settings were chosen from reference literature: candidate list size $k = 10$ and maximum number of iterations $MaxIter = 100$.

We used a large test set, by modifying Steiner Problem instances from the classes C, MC, X, PUC, I080, I160, I320, I640, P6E, P6Z, WRP3 and WRP4 in the SteinLib library [3]. For each problem, we selected the highest degree terminal node as the $z$ node. Also, all the edges between terminal sites were deleted (as they are not allowed in our problem). We run a feasibility check, if unfeasible, the problem instance was discarded. By this process, we obtained 182 problem instances, whose optima was bounded below by the optima of the original SPG instances.

Table 1 shows a summary of computational results. The entries from left to right are the name of the Steinlib classes, the number of instances (NI), the number of nodes and edges of the instances, the number of instances where the lower bound was obtained reaching therefore the optimum (NOPT), the average improvement of the local search phase over the construction phase results (Avg. LSI), the

average running time per iteration (Avg. secs/itr), and the average of the gap of the GRASP solution respect to the lower bound (Avg. LB_GAP). The average improvement is computed as Avg. LSI= $\sum_{p \in Set} \mathrm{LSI}(p)/\mathrm{NI}$, where for problem $p$, $\mathrm{LSI}(p) = \frac{100}{MaxIter} \times \left( \sum_{i=1}^{MaxIter} \frac{(CC_i - LC_i)}{CC_i} \right)$, $CC_i$ and $LC_i$ being the costs of the solutions delivered in iteration $i$ by the Construction Phase and the Local Search Phase respectively. The average gap is Avg. LB_GAP= $\sum_{p \in Set} \mathrm{LB\_GAP}(p)/\mathrm{NI}$, where for problem $p$, $\mathrm{LB\_GAP}(p) = 100 \times \frac{Best\_Cost\_Found - Lower\_Bound}{Lower\_Bound}$, and $Lower\_Bound$ is the optimum value corresponding to the original SPG instance.

| Testset | NI | Nodes | Edges | NOPT | Avg. LSI | Avg. secs/itr | Avg. LB_GAP |
|---|---|---|---|---|---|---|---|
| C | 6 | 500 | 625-2500 | - | 18.17% | 13.83 | 0.43% |
| MC | 2 | 97-120 | 4656-7140 | 1 | 22.43% | 3.18 | 8.16% |
| X | 2 | 52-58 | 1326-1653 | - | 15.13% | 1.43 | 52.33% |
| PUC | 4 | 64-128 | 192-750 | 2 | 19.65% | 2.47 | 0.14% |
| I080 | 67 | 80 | 120-3160 | 15 | 15.52% | 2.17 | 9.83% |
| I160 | 22 | 160 | 240-2544 | 7 | 21.31% | 3.02 | 3.57% |
| I320 | 13 | 320 | 480-10208 | 3 | 22.48% | 10.05 | 2.82% |
| I640 | 10 | 640 | 960-4135 | 2 | 21.33% | 35.74 | 3.73% |
| P6E | 10 | 100-200 | 180-370 | 2 | 22.12% | 1.98 | 17.08% |
| P6Z | 5 | 100-200 | 180-370 | 1 | 20.36% | 1.54 | 27.33% |
| WRP3 | 23 | 84-886 | 149-1800 | 7 | 22.21% | 25.08 | 0.00029% |
| WRP4 | 18 | 110-844 | 188-1691 | 4 | 25.14% | 31.56 | 0.00108% |

**Table I.** Computational results.

We observe that for all problem classes, the local search phase improves significantly (always over 15% average improvement) the solutions built by the greedy construction phase.

The overall results show that GRASP finds in most cases good quality solutions. In 44 instances (out of 182) we reached the lower bound and therefore optimality. As in general only lower bounds and not true optima are known, it is natural that a gap persists in many cases, as shown in the table, with wide variations depending on the problem class. Even then, this gap is usually quite small (less than 5% gap average in 7 over 12 problem classes), showing the interest of the proposed method.