GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES FOR THE STEINER PROBLEM IN GRAPHS

SIMONE L. MARTINS, PANOS M. PARDALOS, MAURICIO G.C. RESENDE, AND CELSO C. RIBEIRO

Abstract. We describe four versions of a Greedy Randomized Adaptive Search Procedure (GRASP) for finding approximate solutions of general instances of the Steiner Problem in Graphs. Different construction and local search algorithms are presented. Preliminary computational results with one of the versions on a variety of test problems are reported. On the majority of instances from the OR-Library, a set of standard test problems, the GRASP produced optimal solutions. On those that optimal solutions were not found, the GRASP found good quality approximate solutions.

1. Introduction

Posed independently by Hakimi [21] and Levin [30], the Steiner problem in graphs (SPG) consists in connecting a subset of given nodes on a graph with the minimum cost tree. The SPG has also many equivalent formulations as an integer program [22] or as a continuous nonconvex global optimization problem [26]. Karp [25] showed earlier that the SPG decision problem is NP-complete in general. Even for some restrictions like grid graphs [19], bipartite graphs [20], chordal and split graphs [46], the problem remains NP-complete. Moreover, on directed graphs, Provan and Ball [38] showed that the variant of the rooted Steiner arborescence of a directed graph is NP-complete. NP-completeness still sticks to the problem with planar acyclic digraphs with indegree and outdegree at most two [37]. However, there are some classes of instances for which polynomial time algorithms exist. In what follows, we give a short survey of existing algorithms and heuristics for the Steiner problem in graphs.

All known exact SPG algorithms for general graphs are in some way enumerative algorithms. However, they differ in how the enumeration is done and how clever their strategies are to avoid total enumeration. In order to avoid total enumeration, topological and other properties of general graphs can be used. Another alternative is to develop good lower and upper bounds in order to reduce the possible search space. Two simple approaches are proposed by Hakimi [21]. While one of them is based on spanning tree enumeration, the other approach is a topology enumeration recursive algorithm. Another algorithm, which generates spanning trees for a derived problem in order of increasing total length until a solution to the original problem can be inferred, is proposed by Balakrishnan and Patel [2]. Also, implicit enumeration approaches are given by Shore, Foulds, and Gibbons [41], and Yang

Date: July 6, 1998.

Key words and phrases. Combinatorial optimization, Steiner Problem in Graphs, local search, GRASP, network design.

AT&T Labs Research Technical Report: TR 98.14.1.

and Wing [50], where the set of all trees spanning the set of given points is, in a systematic way, separated into smaller subsets to be analyzed, using upper and lower bounds, in order to determine whether or not they contain the optimal feasible solution. Along another line, straightforward dynamic programming approaches are proposed by Dreyfus and Wagner [10], and Levin [30]. A branch-and-bound approach that uses heuristics to provide good lower bounds and to choose the next edge for consideration in the backtracking process is proposed by Foulds and Gibbons [17], and Shore, Foulds, and Gibbons [41]. Other approaches, including branch-and-bound, are proposed by Beasley [3, 4], Aneja [1], Maculan [32], Lucena [31], Wong [48], and Chopra, Gorres, and Rao [7]. Exact algorithms for the SPG can benefit from preprocessing the graph so that the size of the problem to solve is reduced. Reduction techniques are discussed in Beasley [3], Balakrishnan and Patel [2], Duin and Volgenant [13, 14], Voss [44, 45], Iwainsky et al. [24], Chopra, Gorres, and Rao [7], Khoury and Pardalos [27], and Duin [11, 12].

Many heuristics have been proposed to find approximate solutions for the SPG. For example, in Aneja [1], a greedy variation of a set covering algorithm is presented, and an MST-based procedure is used as a shortcut in the dual ascent algorithm given by Wong [48]. Another MST-based heuristic was proposed by Choukmane [8] and independently by Plesník [36], Kou, Markowsky, and Berman [28], Bharath-Kumar and Jaffe [6], and Iwainsky et al. [24]. A heuristic based on Kruskal's MST algorithm is given by Wu, Widmayer, and Wong [49]. Another heuristic approach based on Prim's MST algorithm is given by Takahashi and Matsuyama [42]. While all of the above MTS-based approaches implicitly choose the Steiner nodes (optional nodes in the Steiner tree), an average distance heuristic that selects those nodes is given by Rayward-Smith [39]; refer also to Rayward-Smith and Clare [40], and Foulds and Rayward-Smith [18]. A different heuristic based on neighborhood contraction techniques is provided by Plesník [36]. The reader is referred to Winter [47], Hwang and Richards [22], Hwang, Richards, and Winter [23], and Cieslik [9] for surveys.

2. GRASP for the Steiner problem in graphs

Approximate solutions for the Steiner minimal tree problem can be obtained by many techniques, including node, spanning tree, and path-based approaches. In this section, we apply the concepts of GRASP to the approximate solution of the Steiner problem in graphs, using a spanning tree based construction phase and a node-based local search phase. We also propose a path-based construction phase and a path-based local search. Combining these two construction phases with the two local search phases, yields four versions of GRASP.

2.1. **GRASP.**

A greedy randomized adaptive search procedure (GRASP) [15, 16] can be seen as a metaheuristic which captures good features of pure greedy algorithms (e.g. fast local search convergence and good quality solutions) and also of random construction procedures (e.g. diversification). Each iteration consists of the construction phase, the local search phase and, if necessary, the incumbent solution update. In the construction phase, a feasible solution is built, one element at a time. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that estimates the benefit of selecting each element. The probabilistic component of a

GRASP is characterized by randomly choosing one of the best candidates in the list, but usually not the top one.

The solutions generated by a GRASP construction are not guaranteed to be locally optimal. Hence, it is almost always beneficial to apply 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 one from its neighborhood. It terminates when there are no better solutions in the neighborhood. Success for a local search algorithm depends on the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution. The GRASP construction phase plays an important role with respect to this last point, since it produces good starting solutions for local search. The customization of these generic principles into an approximate algorithm for the Steiner problem in graphs is described in the following.

Before we describe the algorithms, we define some notation. Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes and E denotes the set of edges. Given a non-negative weight function $w : E \rightarrow \mathbb{R}$ associated with its edges and a subset $X \subseteq V$ of terminal nodes, the Steiner problem SPG(V,E,w,X) consists in finding a minimum weighted connected subgraph of G spanning all terminal nodes in X. The solution of SPG(V,E,w,X) is a Steiner minimum tree (SMT). The non-terminal nodes that end up in the SMT are called *Steiner nodes*.

A graph $H = (V', E')$ is said to be a subgraph of $G = (V, E)$ if $V' \subseteq V$ and E' is a subset of the edges in E having both extremities in V'. This graph H is said to span a subset $U \subseteq V$ of the nodes in G if $U \subseteq V'$. For any subset of nodes $W \subseteq V$, the edge subset $E(W) = \{(i, j) \in E \mid i \in W, j \in W\}$ defines the induced subgraph $G(W) = (W, E(W))$ in G by the nodes in W.

We denote by $T_{V,E,w}(X)$ the Steiner minimum tree solving the Steiner problem SPG(V, E, w, X) formulated above. Given the graph $G = (V, E)$ with non-negative weights w associated with its edges, the minimum spanning tree problem MSTP(V, E, w) consists in finding a minimum weighted subtree of G spanning all nodes in V. This problem can be seen as particular case of the Steiner problem SPG(V, E, w, X), in which $X = V$. Accordingly, we denote by $T_{V,E,w}(V)$ the minimum spanning tree solving MSTP(V, E, w). We can associate a feasible solution of the Steiner problem SPG(V, E, w, X) with each subset $S \subseteq V \setminus X$ of Steiner nodes such that the graph $G(S \cup X) = (S \cup X, E(S \cup X))$ is connected, given by a minimum spanning tree solving problem MSTP$(S \cup X, E(S \cup X), w)$. Let S^* be the set of Steiner nodes in the optimal solution of SPG(V, E, w, X). The optimal solution $T_{V,E,w}(X)$ is a minimum spanning tree of the graph induced in G by the node set $S^* \cup X$, i.e., the solution to the minimum spanning tree problem MSTP$(S^* \cup X, E(S^* \cup X), w)$.

In the following, solutions of the Steiner problem SPG(V, E, w, X) will be characterized by the associated set of Steiner nodes and one of the corresponding minimum spanning trees. Accordingly, the search for the Steiner minimum tree $T_{V,E,w}(X)$ will be reduced to the search for the optimal set S^* of Steiner nodes.

2.1.1. *Spanning tree based construction phase.* The spanning tree based construction phase of GRASP makes use of the distance network heuristic, suggested by Choukmane [8], Iwainsky, Canuto, Taraszow and Villa [24], Kou, Markowsky and Berman [28], and Plesník [36], with time complexity $O(|X||V|^2)$. Mehlhorn [34] proposed a modification of the original version, leading to a procedure using simple

data structures and presenting an improved time complexity. For every terminal node $i \in X$, let $N(i)$ be the subset of non-terminal nodes of V that are closer to i than to any other terminal node. A non-terminal node may belong to the neighborhood of more than one terminal node. The first step of this procedure consists in the computation of a graph $G' = (X, E')$, where $E' = \{(i,j), i,j \in X \mid \exists (k,\ell) \in E, k \in N(i), \ell \in N(j)\}$. Then, a weight $w'_{ij} = \min\{d(i,k) + w_{k\ell} + d(\ell,j) \mid (k,\ell) \in E, k \in N(i), \ell \in N(j)\}$ is associated with each edge $(i,j) \in E'$, where $d(a,b)$ denotes the shortest path from a to b in the original graph $G = (V,E)$ in terms of the weights w. Next, Kruskal's greedy algorithm [29] is used to solve the minimum spanning tree problem $\text{MSTP}(X, E', w')$. This algorithm works as follows. A minimum spanning tree is constructed, one edge at a time, until all nodes are connected. At each iteration, the edge with the least weight is selected for insertion in the tree, from among those whose insertion does not create a cycle in the current solution. The edges in the minimum spanning tree $T_{X,E',w'}(X)$ so obtained are replaced by the edges in the corresponding shortest paths in the original graph G. Neighborhoods $N(i), \forall i = 1, \ldots, |X|$ can be computed in time $O(|E| + |V| \log |V|)$ [34], which is the same complexity of the minimum spanning tree computation. Then, the overall complexity of the distance heuristic network with Mehlhorn's improvements is only $O(|E| + |V| \log |V|)$, which is much better than the original bound.

As discussed in the previous section, the construction phase of GRASP relies on randomization to build different solutions at different iterations. Graph $G' = (X, E')$ is created only once and does not change throughout all computations. In order to add randomization to Mehlhorn's version of the distance network heuristic, we make the following modification in Kruskal's algorithm. Instead of selecting the feasible edge with the least weight, we build a restricted candidate list with all edges $(i,j) \in E'$ such that $w'_{ij} \leq w'_{\min} + \alpha(w'_{\max} - w'_{\min})$, where $0 \leq \alpha \leq 1$ and $w'_{\min}$ and $w'_{\max}$ denote, respectively, the least and the largest weights among all edges still unselected to form the minimum spanning tree. Then, an edge is selected at random from the restricted candidate list. The operations associated with the construction phase of our GRASP are implemented in lines 2 and 4 of the pseudo-code of algorithm `GRASP_for_SPG` outlined in Figure 1.

2.1.2. *Node-based local search.* Since the solution produced by the construction phase is not necessarily a local optimum, local search can be applied as an attempt to improve it. The first step towards the implementation of a local search procedure consists in identifying an appropriate neighborhood definition.

Let S be the set of Steiner nodes in the current Steiner tree. We have noticed in Section 2.1 that each subset S of Steiner nodes can be associated with a feasible solution of the Steiner problem $\text{SPG}(V, E, w, X)$, given by a minimum spanning tree $T_{S\cup X, E(S\cup X), w}(S \cup X)$ solving problem $\text{MSTP}(S \cup X, E(S \cup X), w)$. Moreover, let $W(T)$ denote the weight of a tree T. The neighbors of a solution characterized by its set S of Steiner nodes are defined by all sets of Steiner nodes which can be obtained either by adding to S a new non-terminal node, or by eliminating from S one of its Steiner nodes.

Given the current tree $T_{S\cup X, E(S\cup X), w}(S \cup X)$ and a non-terminal node $s \in \{V \setminus X\} \setminus S$, the computation of neighbor $T_{S\cup\{s\}\cup X, E(S\cup\{s\}\cup X), w}(S \cup \{s\} \cup X)$ obtained by the insertion of s into the current set S of Steiner nodes can be done in $O(|V|)$ average time, using the algorithm proposed by Minoux [35]. For each non-terminal node $t \in S$, neighbor $T_{S\setminus\{t\}\cup X, E(S\setminus\{t\}\cup X), w}(S \setminus \{t\} \cup X)$ obtained by the

elimination of t from the current set S of Steiner nodes is computed by Kruskal's algorithm as the solution of the minimum spanning tree problem MSTP$(S \setminus \{t\} \cup X, E(S \setminus \{t\} \cup X), w)$.

In order to speedup the local search, since the computational time associated with the evaluation of all insertion moves is likely to be much smaller than that of the elimination moves, only the insertion moves are evaluated in a first pass. The evaluation of elimination moves is performed only if there are no improving insertion moves.

2.1.3. *Algorithm description.* We now describe the GRASP consisting of a a spanning tree based construction phase and a node-based local search phase. The pseudo-code with the complete description of procedure `GRASP_for_SPG` for the Steiner problem in graphs is given in Figure 1. The procedure takes as input the original graph $G = (V, E)$, the set X of terminal nodes, the edge weights w, the restricted candidate list parameter α ($0 \leq \alpha \leq 1$), a seed for the pseudo random number generator, and the number of iterations (`max_iterations`). Graph G' is computed in line 2. The GRASP procedure is repeated `max_iterations` times. In each iteration, a greedy randomized solution T is constructed in line 4 using the randomized version of Kruskal's algorithm. Let S be the set of Steiner nodes in T.

Next, the local search attempts to produce a better solution. In line 5 we initialize the best set of Steiner nodes as those in the current solution, and the weight of the best neighbor as that of the current solution. The loop from line 6 to 11 searches for the best insertion move. In line 7 we compute the minimum spanning tree T^{+s} associated with problem MSTP$(S \cup \{s\} \cup X, E(S \cup \{s\} \cup X), w)$ defined by the insertion of node s into the current set of Steiner nodes. Let $W(T^{+s})$ be its weight. In line 8 we check whether the new solution T^{+s} improves the weight of the current best neighbor. The best set of Steiner nodes and the weight of the best neighbor are updated in line 9. Once all insertion moves have been evaluated, we check in line 12 whether an improving neighbor has been found. If this is the case, the set of Steiner nodes, the current Steiner tree and its weight are updated in line 13, and the local search resumes from this new current solution.

If no improving insertion move is found, then the elimination moves are evaluated. In line 15 we reinitialize the best set of Steiner nodes as those in the current solution, and the weight of the best neighbor as that of the current solution. We check in line 17 whether the graph $G^{-t} = ((S \setminus \{t\}) \cup X, E((S \setminus \{t\}) \cup X))$ obtained by the elimination of node t is connected or not. If it is connected, we compute in line 18 the minimum spanning tree T^{-t} associated with problem MSTP$((S \setminus \{t\}) \cup X, E((S \setminus \{t\}) \cup X), w)$ defined by the elimination of node t from the current set of Steiner nodes. Again, let $W(T^{-t})$ be its weight. In line 19 we check whether the new solution T^{-t} improves the weight of the current best neighbor. Once again, the best set of Steiner nodes and the weight of the best neighbor are updated in line 20. Once all elimination moves have been evaluated, we check in line 24 whether an improving neighbor has been found. If this is the case, the set of Steiner nodes, the current Steiner tree and its weight are updated in line 25, and the local search resumes from this new current solution.

If the solution found at the end of the local search phase is better than the best solution found so far, we update in line 28 the best set of Steiner nodes, the current Steiner tree and its weight. The best set S^* of Steiner nodes and the best Steiner tree T^* are returned in line 31.

```
procedure GRASP_for_SPG(V, E, w, X, α,seed,max_iterations)
1    best_value ← ∞;
2    Compute graph G' = (X, E') and weights w'_ij, ∀(i, j) ∈ E';
3    for k = 1, ..., max_iterations do
                                                    /* Construction phase */
4        Apply a randomized version of Kruskal's algorithm to obtain a
         spanning tree T of G' = (X, E') with S as its set of Steiner nodes;
                                                    /* Insertion moves */
5        best_set ← S; best_weight ← W(T);
6        for all s ∈ (V \ X) \ S do
7            Compute the minimum spanning tree T^{+s};
8            if W(T^{+s}) < best_weight then do
9                best_set ← S ∪ {s}; best_weight ← W(T^{+s});
10           end then;
11       end for;
12       if best_weight < W(T) then do
13           S ← S ∪ {s}; T ← T^{+s}; W(T) ← W(T^{+s}); go to line 5;
14       end then;
                                                    /* Elimination moves */
15       best_set ← S; best_weight ← W(T);
16       for all t ∈ S do
17           if G^{-t} = ((S \ {t}) ∪ X, E((S \ {t}) ∪ X)) is connected then do
18               Compute the minimum spanning tree T^{-t};
19               if W(T^{-t}) < best_weight then do
20                   best_set ← S \ {t}; best_weight ← W(T^{-t});
21               end then;
22           end then;
23       end for;
24       if best_weight < W(T) then do
25           S ← S \ {t}; T ← T^{-t}; W(T) ← W(T^{-t}); go to line 5;
26       end then;
                                                    /* Best solution update */
27       if W(T) < best_value then do
28           S* ← S; T* ← T; best_value ← W(T);
29       end then;
30   end for;
31   return S*, T*;
end GRASP_for_SPG;
```

FIGURE 1. Pseudo-code of the sequential GRASP procedure for the Steiner problem in graphs

2.1.4. *Acceleration scheme.* In order to accelerate the local search phase, we implemented a faster evaluation scheme for the insertion moves (lines 5-14 in procedure `GRASP_for_SPG` in Figure 1). The basic idea consists in keeping a candidate list with promising insertion moves, which is periodically updated.

In the first GRASP iteration, we build a list containing the `k_best` improving insertion moves, which is kept in nondecreasing order of the associated move values. At each following iteration, let S be the set of Steiner nodes in the current solution. Instead of reevaluating all insertion moves, we just take the node s corresponding to the first move in this candidate list and reevaluate the weight of a minimum spanning tree associated with the insertion of this node into the current set of Steiner nodes. If this move reveals itself to be an improving one, then the current solution is updated, the move is eliminated from the candidate list, and the local search resumes from the new set $S \cup \{s\}$ of Steiner nodes. Otherwise, if the move is not an improving one, it is eliminated from the candidate list and the next candidate move is evaluated. Once the candidate list becomes empty, a new full iteration is performed, all insertion moves are evaluated, and the candidate list is rebuilt.

TABLE 1. Effect of the acceleration scheme on OR-Library problems of series C

	`GRASP_for_SPG`		With acceleration	
Problem	$W(T^*)$	secs	$W(T^*)$	secs
C.01	85	4.79	85	4.51
C.02	144	5.05	144	4.58
C.03	754	82.16	754	32.77
C.04	1079	116.03	1080	46.07
C.05	1579	148.73	1579	57.93
C.06	55	4.55	55	4.39
C.07	102	5.87	102	5.47
C.08	509	91.80	509	38.29
C.09	707	164.61	707	61.41
C.10	1093	270.07	1093	114.21
C.11	32	3.95	32	3.99
C.12	46	6.47	46	6.14
C.13	261	104.20	260	45.89
C.14	324	171.57	324	81.48
C.15	556	335.22	556	158.02
C.16	11	5.50	11	5.49
C.17	18	7.65	18	6.34
C.18	115	104.66	116	56.97
C.19	147	188.99	147	131.20
C.20	267	485.02	267	337.78

We present in Table 1 some computational results illustrating the efficiency of the above acceleration scheme on an SGI Challenge (20 196 MHZ MIPS R10000 processors) with 6144 Mbytes of RAM. For each of the 20 series C test problems from the OR-Library [5], we present the weight $W(T^*)$ of the best solution found and the computation time in seconds (secs) obtained by (i) a straightforward version of algorithm `GRASP_for_SPG` with a random values for α, and (ii) the same algorithm using the above acceleration scheme with `k_best` = 100. Each algorithm was run for 10 iterations. These results show that this technique significantly reduced the computational times, even attaining a speedup factor of 2.68 in the case of problem C.09, with almost no loss in terms of solution quality.

2.2. **Other versions of GRASP.** We now propose alternative construction and local search phases for a GRASP for the Steiner problem in graphs. Both algorithms are path-based.

```
procedure PATH_TM(V,E,w,X)
1     Compute shortest paths from terminal nodes to nodes in V;
2     Select a terminal node v, at random, and set T ← {v};
3     while there are terminal nodes not in T do
4           Γ ← {x | x ∈ X and x ∉ T};
5           u ← argmin{dist(x,T) | x ∈ Γ};
6           T ← T ∪ {shortest path from u to T};
7     end while;
end PATH_TM;
```

FIGURE 2. A path-based construction procedure

2.2.1. *Path-based construction phase.* This construction procedure uses a randomized version of the algorithm described in the pseudo-code in Figure 2. This algorithm was proposed by Takahashi and Matsuyama [42].

The randomization is applied at line 5, where instead of selecting the closest terminal node, a restricted candidate list with close nodes is constructed and a node is selected at random from this list.

2.2.2. *Path-based local search.* We begin with some definitions. A *key-node* is a Steiner node with degree at least three. A *key-path* is a path in a Steiner tree T of which all intermediate nodes are Steiner nodes with degree two in T, and whose end nodes are either terminal or key-nodes. A Steiner tree has a most $|S|-2$ key-nodes and $2|S|-3$ key-paths. A minimum Steiner tree consists of key-paths that are shortest paths between key-nodes or terminals. We use the key-path based local search, proposed by Verhoeven, Severens, and Aarts [43].

Let $T = \{l_1, l_2, \ldots, l_K\}$ be a Steiner tree, where each l_i, $i = 1, \ldots, K$, denotes a key-path. Also, let C_i and C_i' denote the two components that result from the removal of the key-path l_i from T. Any local search requires the definition of a neighborhood. In this case, we define the neighborhood of the current tree T as

$$N(T) = \{C_i \cup C_i' \cup \mathrm{sp}(C_i, C_i') \mid i = 1, \ldots, K\},$$

where $\mathrm{sp}(C_i, C_i')$ is the shortest path between C_i and C_i'. Observe that $C_i \cup C_i' \cup \{l_i\} = T$ and $N(T)$ contains at most $2|S|-3$ neighbors.

The pseudo-code of the path-based local search is given in Figure 3. Given a set of K key-paths, the procedure considers the removal of each key-path in the loop in lines 4–12. The removal of key-path l_i results in the decomposition of the tree into components C_i and C_i'. In line 5 the length of the shortest path from C_i to C_i' is compared to the length of key-path l_i. If the shortest path is smaller than the length of the key-path, a new tree is set up by removing key-path l_i and replacing it with the shortest path from C_i to C_i' in line 6. If the new tree is not a set of key-paths, a new set of key-paths that make up the tree is computed in lines 7–9. In case there was an improvement, an indicator is set in line 10, to restart the local search.

Note that solutions only have neighbors with lower or equal cost. A replacement of a key-path in T can lead to the same Steiner tree if no shorter path exists. This implies that local minima have no neighbors.

```
procedure PATH_LS(V,E,w,X,T = {l_1, l_2, ... , l_K})
1     improve ← .true.;
2     while improve do
3           improve = .false.;
4           for i = 1, 2, ... , K do
5                 if sp(C_i, C'_i) < length(l_i) then do
6                       T ← T \ {l_i} ∪ sp(l_i);
7                       if necessary then do
8                             update T to be a set of key-paths;
9                       end then;
10                      improve ← .true.;
11                end then;
12          end for;
13    end while;
end PATH_LS;
```

FIGURE 3. A path-based local search

3. Preliminary experimental results

In this section, we report on preliminary computational results with an implementation of the GRASP described in Subsection 2.1, i.e. using the spanning tree based construction procedure and the vertex-based local search.

We considered the 60 instances from classes C, D, and E of Beasley's OR-Library [5]. The experiments were done on a Silicon Graphics Challenge computer with 20 196 MHz MIPS R10000 processors and 6.144 Gbytes of memory. The code was executed each time on a single processor.

The programs were written in the C programming language and were compiled with the SGI C compiler `cc` using compiler flags `-O3 -64`. User running times were measured with the system routine `getrusage`.

All instances were solved with identical parameter settings. The restricted candidate list parameter α was selected at random using the uniform distribution in the interval $[0, 1]$. Each GRASP iteration used the same parameter during the entire iteration. The acceleration scheme described in Subsubsection 2.1.4 was activated with the parameter `k_best` $= 100$.

Table 2 summarizes the computational results. For each instance, the table lists the number of nodes, arcs, and terminals in the graph, the optimal or best known solution for that instance, the weight $W(T^*)$ of the best solution T^* found, the iteration in which the best solution was found, and the average running time per iteration in CPU seconds. We make the following observations regarding the computational results:

- In 47 out of 60 test problems, the GRASP produced an optimal solution. This broke down to 20 out of 20 for series C, 16 out of 20 for series D, and 11 out of 20 for series E.
- Of the 13 instances for which sub-optimal solutions were produced, in four instances GRASP was off by 1 unit, in six instances GRASP was off by 2 to 5 units, and in three it was off by 6 to 11 units. The largest percentage relative error (difference between solution found and optimal solution divided

TABLE 2. Statistics for OR-Library problems of series C, D, and E

Problem	Nodes	Arcs	Terminals	Opt	$W(T^*)$	itr. T^* found	avg. secs/itr
C.01	500	625	5	85	85	1	0.35
C.02	500	625	10	144	144	1	0.35
C.03	500	625	83	574	574	19	4.19
C.04	500	625	125	1079	1079	2	4.37
C.05	500	625	250	1579	1579	1	6.33
C.06	500	1000	5	55	55	1	0.29
C.07	500	1000	10	102	102	1	0.38
C.08	500	1000	83	509	509	1	3.58
C.09	500	1000	125	707	707	6	6.46
C.10	500	1000	250	1093	1093	1	12.04
C.11	500	2500	5	32	32	1	0.29
C.12	500	2500	10	46	46	1	0.46
C.13	500	2500	83	258	258	60	4.92
C.14	500	2500	125	323	323	60	8.82
C.15	500	2500	250	556	556	1	18.47
C.16	500	12500	5	11	11	1	0.27
C.17	500	12500	10	18	18	1	0.34
C.18	500	12500	83	113	113	772	6.43
C.19	500	12500	125	146	146	386	14.76
C.20	500	12500	250	267	267	1	17.50
D.01	1000	1250	5	106	106	1	0.99
D.02	1000	1250	10	220	220	1	1.47
D.03	1000	1250	167	1565	1565	70	22.10
D.04	1000	1250	250	1935	1935	1	26.37
D.05	1000	1250	500	3250	3250	2	49.12
D.06	1000	2000	5	67	68	1	1.04
D.07	1000	2000	10	103	103	1	1.27
D.08	1000	2000	167	1072	1072	263	30.28
D.09	1000	2000	250	1448	1448	33	51.69
D.10	1000	2000	500	2110	2110	2	129.45
D.11	1000	5000	5	29	29	1	0.87
D.12	1000	5000	10	42	42	1	1.00
D.13	1000	5000	167	500	500	1	35.77
D.14	1000	5000	250	667	669	18	57.31
D.15	1000	5000	500	1116	1116	1	23.85
D.16	1000	25000	5	13	13	1	1.37
D.17	1000	25000	10	23	23	1	1.56
D.18	1000	25000	167	223	228	20	58.79
D.19	1000	25000	250	310	315	1	122.48
D.20	1000	25000	500	537	537	1	209.53
E.01	2500	3125	5	111	111	1	5.03
E.02	2500	3125	10	214	214	3	6.66
E.03	2500	3125	417	4013	4014	46	353.35
E.04	2500	3125	625	5101	5102	34	689.68
E.05	2500	3125	1250	8128	8128	1	2113.88
E.06	2500	5000	5	73	73	1	4.85
E.07	2500	5000	10	145	145	1	7.65
E.08	2500	5000	417	2640	2645	2	525.25
E.09	2500	5000	625	3604	3605	20	1476.68
E.10	2500	5000	1250	5600	5600	1	5582.31
E.11	2500	12500	5	34	34	1	5.00
E.12	2500	12500	10	67	67	1	6.99
E.13	2500	12500	417	1280	1291	1	813.23
E.14	2500	12500	625	1732	1735	7	1526.08
E.15	2500	12500	1250	2784	2788	14	8915.61
E.16	2500	62500	5	15	15	1	9.16
E.17	2500	62500	10	25	25	1	9.54
E.18	2500	62500	417	564	573	68	1326.42
E.19	2500	62500	625	758	768	18	3228.76
E.20	2500	62500	1250	1342	1342	1	12565.35

by the optimal) was 2.24% for D.18. On only five instances was the percentage relative error greater than 1%.

- In 37 out of the 60 instances, the GRASP produced the best solution in the first iteration. In only three of those 37 instances did GRASP not find an optimal solution.
- We compare the results obtained by GRASP in classes D and E with those found with the key-path based local search approach in Verhoeven, Severens, and Aarts [43]. In all cases, the GRASP solutions were as good or better than the key-path based local search solutions. In 26 out of the 40 instances, the GRASP solution is better than the key-path based solution. In the few cases in which the local search solution matches the solution produced by GRASP, the number of terminal nodes is very small (five or ten). However, we should point out that the average running time per iteration of the key-path based local search approach is substantially smaller than that of the average GRASP iteration.

4. Concluding remarks

We described greedy randomized adaptive search procedures for the Steiner problem in graphs. One implementation of our procedures was tested on 60 standard test problems and was shown to find good quality solutions. It found optimal solutions for 47 instances and was never off by more than 2.24% of the optimal. Running times increase with the number of nodes, arcs, and terminals.

We compare the results obtained by GRASP in classes D and E with those found with the key-path based local search approach in Verhoeven, Severens, and Aarts [43]. In all cases, the GRASP solutions were as good as or better than the key-path based local search solutions. We noted that the average running time per iteration of the key-path based local search approach is substantially smaller than that of the average GRASP iteration. This is an indication that the GRASP based on the key-path local search approach could be more efficient than the current version tested using a node-based local search. We have already implemented the key-path based local search procedure and are in the process of integrating it to the GRASP code. Computational results on the new version will be reported soon.

References

[1] Y.P. Aneja. An integer linear programming approach to the Steiner problem in graphs. *Networks*, 10:167–178, 1980.
[2] A. Balakrishnan and N.R. Patel. Problem reduction methods and a tree generation algorithm for the Steiner network problem. *Networks*, 17:65–85, 1987.
[3] J.E. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984.
[4] J.E. Beasley. An SST-based algorithm for the Steiner problem on graphs. *Networks*, 19:1–16, 1989.
[5] J.E. Beasley. OR-Library: Distributing test problems by electronic mail. *J. of Oper. Res. Soc.*, 41:1069–1072, 1990.
[6] K. Bharath-Kumar and J.M. Jaffe. Routing to multiple destinations in computer networks. *IEEE Trans. Commun.*, COM-31:343-351, 1983.
[7] S. Chopra, E.R. Gorres, and M.R. Rao. Solving the Steiner tree problem on graphs using branch and cut. *ORSA Journal on Computing*, 4:320–335, 1992.
[8] E.-A. Choukhmane. Une heuristique pour le problème de l'arbre de Steiner. *RAIRO Recherche Opérationnelle*, 12:207–212, 1978.
[9] D. Cieslik. *Steiner minimal trees*, Kluwer Academic Publishers, Dordrecht, 1998.
[10] S.E. Dreyfus and R.A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.
[11] C.W. Duin. Steiner's problem in graphs. *Ph.D. Thesis*, University of Amsterdam, 1993.

[12] C.W. Duin. Reducing the graphical Steiner problem with a sensitivity test. In *Network Design: Connectivity and Facilities Location*, P.M. Pardalos and D.-Z. Du (Eds.), *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 40:79–107, 1998.
[13] C.W. Duin and A. Volgenant. Some generalizations of the Steiner problem in graphs. *Networks*, 17:353–364, 1987.
[14] C.W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989.
[15] 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.
[16] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
[17] L.R. Foulds and P.B. Gibbons. A branch and bound approach to the Steiner problem in graphs. *Proceedings of the 14th Annual Conference of the Operational Research Society of New Zealand*, 1:61–70, 1978.
[18] L.R. Foulds and J.V. Rayward-Smith. Steiner problems in graphs: Algorithms and applications. *Eng. Optimization*, 7:7–16, 1983.
[19] M.R. Garey and D.S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32:826–834, 1977.
[20] M.R. Garey and D.S. Johnson. *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.
[21] S.L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.
[22] F.K. Hwang and D.S. Richards. Steiner tree Problems. *Networks*, 2:55–89, 1992.
[23] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem,* Elsevier, Amsterdam, 1992.
[24] A. Iwainsky, E. Canuto, O. Taraszow, and A. Villa. Network decomposition for the optimization of connection structures. *Networks*, 16:205–235, 1986.
[25] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Communications*, R.E. Miller and J.W. Thatcher (Eds.), Plenum Press, New York, 85–103, 1972.
[26] B.N. Khoury, P.M. Pardalos, and D.W. Hearn. Equivalent formulations for the Steiner problem in graphs. In *Network Optimization Problems,* P.M. Pardalos and D.-Z. Du (Eds.), World Scientific, 111–124, 1993.
[27] B.N. Khoury and P.M. Pardalos. A heuristic for the Steiner problem on graphs. *Comp. Opt. & Appl.*, 6:5–14, 1996.
[28] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Info.*, 15:141–145, 1981.
[29] J.B. Kruskal. On the shortest apanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
[30] A. Levin. Algorithms for the shortest connection of a group of graph vertices. *Soviet Math. Doklady*, 12:1477–1481, 1971.
[31] A. Lucena. Steiner problem in graphs: Lagrangean relaxation and cutting planes. *COAL Bulletin*, 21:2–7, 1992.
[32] N. Maculan. The Steiner problem in graphs. *Ann. Discrete Math.*, 31:185–222, 1987.
[33] T.L. Magnanti and R.T. Wong. Network design and transportation planning: Models and algorithms. *Trans. Science*, 18:1–55, 1984.
[34] K. Mehlhorn. A faster approximation for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.
[35] M. Minoux. Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity. *INFOR*, 28:221–233, 1990.
[36] J. Plesník. A bound for the Steiner tree problem in graphs. *Math. Slovoca*, 31:155-163, 1981.
[37] J.S. Provan. A polynomial algorithm for the Steiner tree problem on terminal-planar graphs. Technical Report UNC/ORSA/Tech. Rep.-83/10, Dep. Operations Research, University of North Carolina, Chapel Hill, 1983.
[38] J.S. Provan and M.O. Ball. Computing network reliability. *Operations Res.*, 32:516–526, 1984.
[39] V.J. Rayward-Smith. The computation of nearly minimal Steiner trees in graphs. *Int. J. Math. Ed. Sci. Technol.*, 14:15–23, 1983.
[40] V.J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.
[41] M.L. Shore, L.R. Foulds, and P.B. Gibbons. An algorithm for the Steiner problem in graphs. *Networks*, 12:323–333, 1982.
[42] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Jpn.*, 24:573–577, 1980.
[43] 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.
[44] S. Voss. Steiner's problem in directed graphs: Logical tests in a branch and bound algorithm. *NATO Advanced Research Workshop on Topological Network Design*, Copenhagen, 1989.
[45] S. Voss. A reduction-based algorithm for the Steiner problem in graphs. *Methods of Operations Res.* 58:239–252, 1989.

[46] K. White, M. Farber, and W. Pulleyblank. Steiner trees, connected domination and strongly chordal graphs. *Networks*, 15:109–124, 1985.
[47] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.
[48] R.T. Wong. A dual ascent approach to Steiner tree problems on a directed graph. *Mathematical Programming*, 28:271–287, 1984.
[49] Y.F. Wu, P. Widmayer, and C. K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Acta Info.*, 23:223–229, 1986.
[50] Y.Y. Yang and O. Wing. An algorithm for the wiring problem. *Digest IEEE Int. Symp. Electrical Networks*, 14–15, 1971.

Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
E-mail address: simone@inf.puc-rio.br

Center for Applied Optimization, Department of Industrial and Systems Engineering, University of Florida, Gainesville, FL 32611 USA.
E-mail address: pardalos@ufl.edu

Information Sciences Research, AT&T Labs Research, Florham Park, NJ 07932 USA.
E-mail address: mgcr@research.att.com

Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil
E-mail address: celso@inf.puc-rio.br