A PARALLEL GRASP FOR THE STEINER TREE PROBLEM IN GRAPHS USING A HYBRID LOCAL SEARCH STRATEGY

S.L. MARTINS, M.G.C. RESENDE, C.C. RIBEIRO, AND P.M. PARDALOS

Abstract. In this paper, we present a parallel greedy randomized adaptive search procedure (GRASP) for the Steiner problem in graphs. GRASP is a two phase metaheuristic. In the first phase, solutions are constructed using a greedy randomized procedure. Local search is applied in the second phase, leading to a local minimum with respect to a specified neighborhood. In the Steiner problem in graphs, feasible solutions can be characterized by their non-terminal nodes (Steiner nodes) or by their key-paths. According to this characterization, two GRASP procedures are described using different local search strategies. Both use an identical construction procedure. The first uses a node-based neighborhood for local search, while the second uses a path-based neighborhood. Computational results comparing the two procedures show that while the node-based variant produces better quality solutions, the path-based variant is about twice as fast. A hybrid GRASP procedure combining the two neighborhood search strategies is then proposed. Computational experiments with a parallel implementation of the hybrid procedure are reported, showing that the algorithm found optimal solutions for 45 out of 60 benchmark instances and was never off by more than four percent of the optimal solution value. The average speedup results observed for the test problems show that increasing the number of processors reduces elapsed times with increasing speedups. Moreover, the main contribution of the parallel algorithm concerns the fact that larger speedups of the same order of the number of processors are obtained exactly for the most difficult problems.

1. Introduction

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 $w : E \to \mathbb{R}_+$ associated with its edges and a subset $X \subseteq V$ of terminal nodes, in the Steiner problem (SPG) one seeks a minimum weighted subtree of G spanning all terminal nodes in X. The solution of SPG is a Steiner minimum tree and the non-terminal nodes that end up in the Steiner minimum tree are called Steiner nodes. Karp [22] showed earlier that the decision version of SPG is NP-complete. Applications can be found in many areas, such as telecommunication network design, VLSI design, and computational biology, among others.

The Steiner problem in graphs can be formulated as an integer linear program or a global concave minimization problem. Many exact algorithms for small size problems are based on these formulations [1, 3, 4, 6, 8, 18, 23, 24, 26, 27, 37].

Date: November 1999.

Key words and phrases. Combinatorial optimization, global optimization, Steiner problem in graphs, heuristics, local search, GRASP, network design .

AT&T Labs Research Technical Report: 99.13.1.

In memory of Professor P. D. Panagiotopoulos.

Several heuristics are available for the approximate solution of SPG, see e.g. Duin and Voss [11], Hwang, Richards, and Winter [19], and Voss [36] for recent surveys. Constructive methods have been proposed, e.g. by Choukmane [7], Kou et al. [25], Minoux [31], Plesník [20], Rayward-Smith and Clare [32], and Takahashi and Matsuyama [34]. We also find implementations of metaheuristics such as genetic algorithms [12, 21], tabu search [2, 33], GRASP [28, 29], and simulated annealing [9].

A greedy randomized adaptive search procedure (GRASP) is a metaheuristic for combinatorial optimization. A GRASP [14] is an iterative process, where each 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 iterations is returned as the result. In this paper, we present a parallel GRASP for the Steiner problem in graphs. Feasible solutions can be characterized by their non-terminal nodes called Steiner nodes or by its key-paths. According to this characterization, two GRASP procedures are described in Section 2, using different search strategies. Both use an identical construction procedure. The first uses a vertex-based neighborhood for local search, while the second uses a key-path based neighborhood. In Section 3, we present computational results comparing the two procedures. A hybrid GRASP procedure combining the two neighborhood search strategies is presented in Section 4. Computational experiments with a parallel implementation of the hybrid procedure are reported in Section 5. Concluding remarks are made in Section 6.

2. Sequential GRASP for the Steiner problem in graphs

Approximate solutions for the Steiner problem in graphs can be obtained by many techniques, including node-based, spanning tree-based, 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. Two local search strategies are described, the first one using a node-based neighborhood while the second uses a path-based neighborhood. Combining the two local search strategies with the single construction phase yields two versions of GRASP.

A greedy randomized adaptive search procedure (GRASP) [13, 14] 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 to explore the solution space). 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 best 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 in an 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.

2.1. **Construction phase.** 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 best one.

The construction phase of our GRASP is based on the distance network heuristic, suggested by Kou, Markowsky, and Berman [25] and later improved by Mehlhorn [30]. This heuristic consists of computing the modified distance network graph $G' = (V, E')$ proposed by Mehlhorn and using Kruskal's algorithm to solve the minimum spanning tree problem for this graph [28]. In this graph, edge weights w' correspond to weights of shortest paths in the original graph G.

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 smallest 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 parameter α is either fixed or randomly selected. For a detailed description of this construction procedure, see [28].

2.2. **Local search using a node-based neighborhood.** We can associate a feasible solution of SPG with each subset $S \subseteq V \setminus X$ of Steiner nodes such that the graph induced in $G = (V, E)$ by $S \cup X$ is connected, corresponding to any of its minimum spanning trees. Let S^* be the set of Steiner nodes in the optimal solution of SPG. Then, the optimal solution T^* is a minimum spanning tree of the graph induced in G by the node set $S^* \cup X$. Solutions of the Steiner problem SPG may be characterized by their associated sets of Steiner nodes and one of the corresponding minimum spanning trees. Accordingly, the search for the Steiner minimum tree T^* can be reduced to the search for the optimal set S^* of Steiner nodes.

Let S be the set of Steiner nodes, to which we associate a solution of SPG given by one of the minimum spanning trees T of the graph induced in G by $S \cup X$. In the *node-based neighborhood*, the neighbors of this solution 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 any non-terminal node $s \in (V \setminus X) \setminus S$, the computation of the neighbor 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 [31]. For each non-terminal node $t \in S$, the neighbor 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 in the graph induced in G by $(S \setminus \{t\}) \cup X$.

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. For more details about local search using a node-based neighborhood, see [28, 29].

2.3. **Local search using a path-based neighborhood.** 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 at 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 [35].

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. The *path-based neighborhood* of the current tree T is defined as the set of trees $N(T) = \{C_i \cup C_i' \cup \mathrm{sp}(C_i, C_i') \mid i = 1, \ldots, K\}$, 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.

Local search is performed by replacing a key-path by the shortest path between the two subsets of nodes that remain after its removal, updating the key-paths and key-vertices, and checking if the cost is improved. This procedure is done for all key-paths. If the cost of the current solution is improved, the incumbent solution is updated by the new tree and the above procedure is repeated for the new solution until there is no more improvement [28].

We notice 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 and that the neighborhood is not connected.

3. Comparative results for the two local search approaches

In this section, we report on preliminary computational results obtained with the two implementations of GRASP described in Section 2 using the two different local search strategies. These experiments have been performed on a set of 40 benchmark problems from series C and D of the OR-Library [5]. The graphs have been previously reduced using the reduction tests proposed by Duin and Vogenant [10]. First, the special distance test and the nearest special vertex test are applied until no further reduction is possible. The special distance test eliminates edges which may not belong to an optimal solution, while the nearest special vertex test contracts edges which necessarily belong to every optimal solution. Next, all non-terminal nodes with degree one in the partially reduced graph are also eliminated. Finally, non-terminal nodes with degree two and their adjacent edges are contracted into a single edge. All of the above steps are repeated, until no further reduction can be identified. Such tests are quite effective and lead to significant reductions in the input graphs.

TABLE 1. Results for series C and D for GRASP with node-based and path-based neighborhoods

		Node-based		Path-based	
Problem	Optimal	Value	Time	Value	Time
C.01	85	85	1.02	85	0.75
C.02	144	144	1.09	144	0.92
C.03	754	754	5.24	754	2.84
C.04	1079	1079	2.48	1079	1.95
C.05	1579	1579	0.61	1579	0.57
C.06	55	55	3.34	55	0.95
C.07	102	102	19.49	103	2.99
C.08	509	509	112.74	509	50.59
C.09	707	707	144.32	707	69.84
C.10	1093	1093	0.85	1093	0.69
C.11	32	32	3.64	33	1.00
C.12	46	46	5.21	46	1.76
C.13	258	258	161.92	258	74.91
C.14	323	323	99.27	323	48.73
C.15	556	556	8.93	556	5.33
C.16	11	11	5.97	11	1.25
C.17	18	18	7.36	18	1.98
C.18	113	116	262.98	116	146.98
C.19	146	147	184.42	147	124.16
C.20	267	267	13.88	268	2.21
D.01	106	106	1.42	106	0.93
D.02	220	220	5.07	220	2.28
D.03	1565	1565	33.66	1565	15.83
D.04	1935	1935	1.93	1935	1.36
D.05	3250	3250	1.10	3254	0.93
D.06	67	68	6.34	70	1.25
D.07	103	103	6.55	103	2.02
D.08	1072	1072	560.37	1077	257.92
D.09	1448	1448	287.95	1449	148.58
D.10	2110	2110	22.35	2111	11.68
D.11	29	29	11.58	29	1.39
D.12	42	42	17.38	42	2.49
D.13	500	501	859.21	502	395.80
D.14	667	669	384.77	667	242.31
D.15	1116	1117	16.27	1120	2.96
D.16	13	13	26.77	13	2.58
D.17	23	23	50.75	23	5.67
D.18	223	228	1098.73	228	604.97
D.19	310	313	986.76	317	528.38
D.20	537	537	131.33	539	62.75

The two variants of GRASP have been implemented in C and were compiled on the IBM xlC compiler version 3.1.3 with compiler options `-O3 -qstrict`. The experiments were done on an IBM RS6000 model 390 computer with 256 Mbytes of memory.

Each variant of GRASP was run for 500 iterations on each test problem using a fixed value for $\alpha = 0.1$ in the construction phase. Table 1 summarizes the results. For each instance, the table lists the instance name, its optimal value, and the value of the best solution found along with the computation time in seconds for each variant.

Table 2. Summary of results for series C and D

	Node-based				Path-based			
Series	# opt	Avg err.	Max err.	Time	# opt	Avg err.	Max err.	Time
C	18	0.17%	2.65%	52.23	15	0.39%	3.13%	27.02
D	14	0.26%	2.24%	225.51	10	0.54%	4.47%	114.60

We summarize in Table 2 the results from Table 1. For each series of test problems and for each GRASP variant, we report the number of optimal solutions found, the average and the maximum percentage of deviation from the optimal value, and the average computational time in seconds. As it can be seen, both variants find very good approximate solutions. Optimal solutions were found on a large number of problems, with average error of less than one percent. The worst quality solution was less than five percent away from the optimal. In general, the node-based neighborhood produced the best-quality solutions, finding a larger number of optima and having smaller errors. However, the computation times observed for the node-based neighborhood are approximately twice those of the path-based neighborhood. Based on these observations, we present in the next section a hybrid procedure combining the two local search strategies.

4. Hybrid local search strategy

In this section, we propose a hybrid local search strategy where we limit the application of the expensive local search strategy (node-based neighborhood) only to sufficiently good starting solutions. To accomplish this, each GRASP local search phase is divided into two stages. In the first stage, the path-based local search procedure is applied. Let z_p be the value of the objective function of the locally optimal solution T_p with respect to the path-based neighborhood. Given a cutoff parameter $\lambda > 0$, the node-based local search is applied in a second stage whenever $z_p < (1+\lambda)z^*$, i.e. whenever the locally optimal solution with respect to the path-based neighborhood is sufficiently close to the objective function value z^* of the best solution T^* found so far.

The pseudo-code with the complete description of procedure `GRASP_HYBR_SPG` for the Steiner problem in graphs is given in Figure 1.

The value of the best solution found is initialized in line 1. The preprocessing computations associated with Mehlhorn's version of the distance network heuristic are performed in line 2, as described in [29]. The loop from lines 3 to 15 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 described in Section 2.1. Next, the two-stage local search strategy attempts to improve this solution. In line 5, we check if this solution was already constructed. If this solution was not visited in previous iterations, the path-based local search will be applied to it. In line 6, the path-based local search routine `PATH_LS` attempts to replace key-paths by shortest paths, as described in Section 2.3. In line 7 we check if the solution T_p returned by `PATH_LS` was not found in previous iterations and if its cost z_p is less than $(1+\lambda)z^*$ to decide if a node-based local search will be performed. If these conditions are met, procedure `NODE_LS` performs a node-based local search in line 9, as described in Section 2.2. If the solution T_n found at the end of the node-based local search is better than the best solution found so far, we update in line 11 the best solution found and its value.

```
procedure GRASP_HYBR_SPG(V, E, X, w);
    z* ← ∞;
    Compute the distance graph G' = (X, E') and weights w'_ij, ∀(i, j) ∈ E';
    for k = 1, ..., max_iterations do
        Apply a randomized version of Kruskal's algorithm to obtain a
        spanning tree T of G' = (X, E');
        if T was not visited in previous iterations then do
            T_p ← PATH_LS(V, E, w, X, T);
            if T_p was not visited in previous iterations and z_p < (1 + λ)z*
            then do
                T_n ← NODE_LS(V, E, w, X, T_p);
                if z_n < z* then do
                    T* ← T_n; z* ← z_n;
                end then;
            end then;
        end then;
    end for;
    return z*, T*;
end GRASP_HYBR_SPG;
```

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

5. Parallelization and computational results

The most straightforward GRASP parallelization scheme is characterized by the the distribution of the iterations among the available processors. In a homogeneous parallel environment, each processor performs a fixed number of GRASP iterations, equal to the total number of iterations divided by the number of processors. Once all processors have finished their computations, the best solution among those found by each processor is obtained using a reduction operation.

We use this strategy in the parallelization of the `GRASP_HYBR_SPG` sequential algorithm, whose pseudo-code was presented in Figure 1. The `max_iterations` GRASP iterations to be performed are uniformly distributed to the available processors. The problem data is input by a single processor and distributed to all others. Each processor has a copy of the hybrid GRASP procedure and keeps its own local list of already-visited solutions. Different initial seeds for random number generation are used by each processor, so as to avoid the repetition of random sequences.

The GRASP using the hybrid local search strategy was also implemented in C and compiled with the IBM xlC compiler version 3.1.3 using compiler options `-O3 -qstrict`. The computational experiments have been performed on a set of 60 benchmark instances from series C, D, and E of the OR-Library [5]. The graphs have been previously reduced using some of the reduction tests proposed by Duin and Volgenant [10] and implemented by Ribeiro and Souza [33]. The above parallelization scheme was implemented using the Message Passing Interface (MPI) library [15] on an IBM SP-2 computer with 32 RS6000 model 390 processors, each having 256 Mbytes of RAM. The parallel procedure was executed using 2, 4,

TABLE 3. Solutions obtained by the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series C

		Number of processors				
Problem	Optimal	1	2	4	8	16
C.01	85	85	85	85	85	85
C.02	144	144	144	144	144	144
C.03	754	754	754	754	754	754
C.04	1079	1079	1079	1079	1079	1079
C.05	1579	1579	1579	1579	1579	1579
C.06	55	55	55	55	55	55
C.07	102	102	102	102	102	102
C.08	509	509	509	509	509	509
C.09	707	707	707	707	707	707
C.10	1093	1093	1093	1093	1093	1093
C.11	32	33	33	33	33	33
C.12	46	46	46	46	46	46
C.13	258	258	258	258	258	258
C.14	323	323	323	323	323	323
C.15	556	556	556	556	556	556
C.16	11	11	11	11	11	11
C.17	18	18	18	18	18	18
C.18	113	115	115	115	115	115
C.19	146	148	148	148	148	148
C.20	267	267	267	267	267	267

8, and 16 processors of the SP-2 machine, performing a total of 512 iterations over all processors (the number of iterations performed by each processor was 512 divided by the number of processors). The parallel GRASP was implemented with parameters $\lambda = 1\%$ and α randomly generated uniformly in the interval $[0, 0.3]$ at each iteration.

In Tables 3 to 5 we present the problem name, its optimal solution value, and the values of the solutions found by the parallel GRASP using 1, 2, 4, 8, and 16 processors. These results are summarized in Table 6 in which we indicate, for each series, the number of optimal solutions found using up to 16 processors. We also give the average and the maximum deviation from the optimal value for each series. As expected, solution quality (in terms of the number of optimal solutions found) is not affected by the number of processors used, as long as the total number of iterations is always the same. Furthermore, the three sub-optimal solutions in series C are off of the optimal value by at most two units. For series D, two of the five sub-optimal solutions are off by at most two units, while one is off by five units. For series E, five of the seven sub-optimal solutions are within one percent of optimality. For each series, the average deviation from the optimal is at most 0.32%, while the maximum deviation is less than four percent.

Tables 7 to 9 report elapsed times in seconds observed for a total of 512 iterations using 1, 2, 4, 8, and 16 processors. For each instance in a series, the table shows the elapsed time for a single processor (sequential algorithm), and the elapsed time and the speedup (ratio of elapsed time of parallel algorithm to elapsed time of sequential algorithm) for 2, 4, 8, and 16 processors. These results are summarized in Table 10, where for each series, we give the average elapsed time for the sequential algorithm, together with the average elapsed times and the average speedups using 2, 4, 8,

TABLE 4. Solutions obtained by the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series D

		Number of processors				
Problem	Optimal	1	2	4	8	16
D.01	106	106	106	106	106	106
D.02	220	220	220	220	220	220
D.03	1565	1565	1565	1565	1565	1565
D.04	1935	1935	1935	1935	1935	1935
D.05	3250	3250	3250	3250	3250	3250
D.06	67	67	67	67	67	67
D.07	103	103	103	103	103	103
D.08	1072	1073	1074	1075	1073	1073
D.09	1448	1448	1448	1448	1448	1448
D.10	2110	2110	2110	2110	2110	2110
D.11	29	29	29	29	29	29
D.12	42	42	42	42	42	42
D.13	500	502	502	502	502	502
D.14	667	667	667	667	667	667
D.15	1116	1116	1116	1116	1116	1116
D.16	13	13	13	13	13	13
D.17	23	23	23	23	23	23
D.18	223	229	229	229	228	228
D.19	310	316	316	316	315	315
D.20	537	538	538	538	538	538

TABLE 5. Solutions obtained by the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series E

		Number of processors				
Problem	Optimal	1	2	4	8	16
E.01	111	111	111	111	111	111
E.02	214	214	214	214	214	214
E.03	4013	4016	4015	4015	4015	4015
E.04	5101	5101	5101	5101	5101	5101
E.05	8128	8128	8128	8128	8128	8128
E.06	73	73	73	73	73	73
E.07	145	145	145	145	145	145
E.08	2640	2648	2648	2648	2648	2648
E.09	3604	3608	3608	3608	3607	3607
E.10	5600	5600	5600	5600	5600	5600
E.11	34	34	34	34	34	34
E.12	67	67	67	67	67	67
E.13	1280	1292	1292	1292	1291	1291
E.14	1732	1735	1735	1735	1735	1735
E.15	2784	2784	2784	2784	2784	2784
E.16	15	15	15	15	15	15
E.17	25	25	25	25	25	25
E.18	564	584	584	583	584	584
E.19	758	770	768	769	769	769
E.20	1342	1343	1343	1342	1342	1342

Table 6. Number of optimal solutions found using up to 16 processors (512 iterations over all processors) and solution quality for 16 processors for series C, D, and E

	Number of processors					% error (16 procs)	
Series	1	2	4	8	16	avg	max
C	17	17	17	17	17	0.31	3.13
D	15	15	15	15	15	0.23	2.24
E	12	12	13	13	13	0.32	3.54

Table 7. Computation times and speedups σ observed with the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series C

	Processors								
	1	2		4		8		16	
Problem	secs	secs	σ	secs	σ	secs	σ	secs	σ
C.01	0.81	0.44	1.82	0.30	2.74	0.34	2.36	0.31	2.61
C.02	1.42	0.82	1.75	0.51	2.80	0.42	3.36	0.40	3.60
C.03	4.83	2.98	1.62	1.90	2.55	1.46	3.31	0.82	5.92
C.04	3.39	2.13	1.60	1.63	2.09	1.18	2.89	0.94	3.61
C.05	0.73	0.47	1.55	0.39	1.84	0.19	3.79	0.17	4.20
C.06	1.37	0.95	1.45	0.86	1.59	1.01	1.36	1.29	1.06
C.07	6.36	3.93	1.62	2.54	2.51	1.91	3.33	1.07	5.96
C.08	63.67	34.52	1.84	18.82	3.38	11.78	5.41	6.78	9.40
C.09	101.00	53.05	1.90	29.52	3.42	16.53	6.11	9.54	10.58
C.10	0.87	0.62	1.40	0.31	2.83	0.25	3.44	0.20	4.44
C.11	1.58	1.25	1.26	1.24	1.28	1.75	0.90	1.99	0.79
C.10	0.87	0.62	1.40	0.31	2.83	0.25	3.44	0.20	4.44
C.12	4.67	3.37	1.39	2.58	1.81	2.47	1.89	2.16	2.16
C.13	98.14	54.25	1.81	30.69	3.20	17.16	5.72	11.01	8.91
C.14	65.58	36.58	1.79	20.37	3.22	10.83	6.06	6.90	9.51
C.15	6.91	4.15	1.66	2.57	2.69	1.72	4.01	1.59	4.34
C.16	2.72	2.46	1.11	2.29	1.19	3.17	0.86	3.14	0.87
C.17	4.14	4.17	0.99	3.84	1.08	3.91	1.06	4.25	0.97
C.18	155.64	80.97	1.92	44.46	3.50	24.22	6.43	15.34	10.15
C.19	128.14	65.65	1.95	35.63	3.60	19.81	6.47	12.58	10.19
C.20	2.53	2.06	1.23	1.67	1.51	1.75	1.45	0.93	2.73

and 16 processors. These results are further illustrated in Figures 2 and 3. The average speedup results show that increasing the number of processors improves elapsed times for all three series with increasing speedups. The figures show that the leveling-off of the speedup curves is beyond 16 processors.

We note that for some instances (for example, E.08 and E.15), the speedup is almost linear (i.e. of the same order of the number of processors), while for others (for example, E.05 and E.16) parallelization does not contribute much and the speedup is sometimes even smaller than one. This behavior is mainly due to the use of the memory structures which keep track of the solutions visited during the search. The local search phase, which is the most time consuming part of the algorithm, is performed only at iterations in which the constructed solution was not found in previous iterations and, consequently, is not stored in the memory structures. In general, larger speedups are observed for more difficult instances in

TABLE 8. Computation times and speedups σ observed with the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series D

	Processors								
	1	2		4		8		16	
Problem	secs	secs	σ	secs	σ	secs	σ	secs	σ
D.01	1.18	0.78	1.52	0.62	1.91	0.76	1.55	0.37	3.15
D.02	3.88	2.52	1.54	1.92	2.02	1.44	2.70	1.01	3.83
D.03	21.77	12.33	1.77	7.22	3.01	4.68	4.65	3.46	6.29
D.04	2.40	1.47	1.63	1.02	2.35	0.87	2.75	0.46	5.18
D.05	1.15	0.89	1.29	0.61	1.88	0.65	1.78	0.28	4.10
D.06	2.97	2.26	1.32	2.08	1.43	2.68	1.11	2.88	1.03
D.07	3.46	2.43	1.42	1.96	1.76	1.79	1.93	1.33	2.60
D.08	365.94	200.37	1.83	110.53	3.31	60.54	6.04	41.28	8.87
D.09	394.46	210.41	1.87	109.18	3.61	59.40	6.64	32.30	12.21
D.10	15.12	8.54	1.77	4.91	3.08	2.87	5.28	1.61	9.37
D.11	6.83	6.41	1.07	6.15	1.11	6.54	1.04	6.03	1.13
D.12	28.07	25.80	1.09	24.31	1.15	22.19	1.26	20.29	1.38
D.13	553.25	300.29	1.84	179.05	3.09	107.15	5.16	68.91	8.03
D.14	324.72	176.09	1.84	121.14	2.68	72.52	4.48	41.23	7.88
D.15	10.24	6.23	1.64	3.27	3.13	2.53	4.05	2.17	4.72
D.16	27.03	26.70	1.01	29.83	0.91	30.10	0.90	30.52	0.89
D.17	15.55	14.75	1.05	14.18	1.10	14.64	1.06	16.63	0.93
D.18	655.37	369.38	1.77	208.85	3.14	136.20	4.81	95.18	6.89
D.19	582.04	301.79	1.93	160.95	3.62	93.77	6.21	67.95	8.57
D.20	66.97	34.51	1.94	19.59	3.42	12.35	5.42	7.82	8.57

TABLE 9. Computation times and speedups σ observed with the parallel hybrid GRASP algorithm using up to 16 processors (512 iterations over all processors) for series E

	Processors								
	1	2		4		8		16	
Problem	secs	secs	σ	secs	σ	secs	σ	secs	σ
E.01	1.42	1.02	1.39	0.94	1.50	1.10	1.29	1.02	1.39
E.02	5.99	4.53	1.32	3.78	1.58	3.17	1.89	2.94	2.04
E.03	107.16	56.37	1.90	30.47	3.52	17.07	6.28	12.13	8.83
E.04	21.02	12.61	1.67	6.78	3.10	4.57	4.60	3.28	6.41
E.05	0.67	0.51	1.32	0.37	1.79	0.54	1.25	1.32	0.51
E.06	8.48	7.97	1.06	7.75	1.09	8.12	1.05	8.76	0.97
E.07	37.94	25.90	1.46	19.35	1.96	15.41	2.46	13.23	2.87
E.08	6091.43	3131.68	1.95	1595.11	3.82	810.73	7.51	466.12	13.07
E.09	2948.36	1832.04	1.61	1071.02	2.75	599.24	4.92	370.42	7.96
E.10	188.58	102.12	1.85	54.78	3.44	29.91	6.31	16.88	11.17
E.11	26.62	26.06	1.02	25.77	1.03	25.85	1.03	27.44	0.97
E.12	75.85	61.91	1.23	56.23	1.35	52.93	1.43	51.15	1.48
E.13	5644.48	3856.96	1.46	2278.27	2.48	1279.05	4.41	839.72	6.72
E.14	3233.90	2223.69	1.45	1252.86	2.58	754.90	4.28	457.39	7.07
E.15	219.97	110.32	1.99	56.49	3.89	30.19	7.29	16.43	13.39
E.16	134.64	134.99	1.00	185.64	0.73	189.35	0.71	186.54	0.72
E.17	288.90	255.16	1.13	242.58	1.19	325.25	0.89	321.10	0.90
E.18	4712.09	2527.18	1.86	1599.76	2.95	1164.09	4.05	914.32	5.15
E.19	3248.99	1883.86	1.72	1057.85	3.07	728.38	4.46	553.22	5.87
E.20	601.41	308.91	1.95	173.52	3.47	108.98	5.52	73.76	8.15

Table 10. Average computation times and average speedups using up to 16 processors (512 iterations over all processors) for series C, D, and E

	Number of processors					
Series	1	2	4	8	16	
C	32.77	17.77	10.12	6.10	4.08	seconds
	–	1.84	3.24	5.37	8.03	speedup
D	154.12	85.19	50.36	31.68	22.08	seconds
	–	1.80	3.06	4.86	6.98	speedup
E	1379.90	828.19	485.97	307.44	216.86	seconds
	–	1.67	2.84	4.49	6.36	speedup

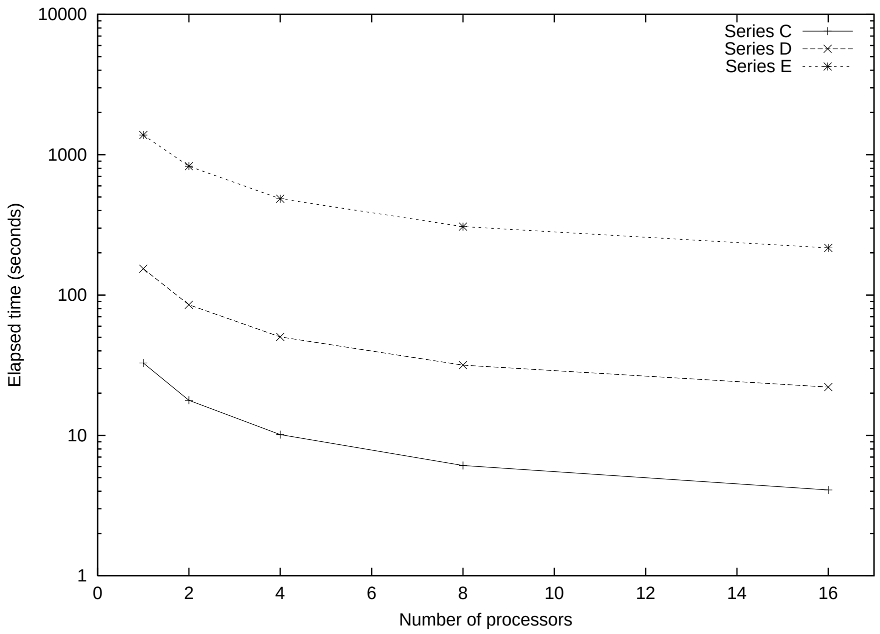

Figure 2. Elapsed times using up to 16 processors for Series C, D, and E.

which fewer repetitions of constructed solutions occur. For problems in which many repetitions are observed, the memory structures contribute to strongly reduce the computation times of the sequential algorithm, avoiding local search and leading to smaller speedups of the parallel version (since less room is left to the reduction of computation times through parallelization). We note that the main contribution of the parallel algorithm concerns exactly the most difficult problems, for which larger speedups can be obtained.

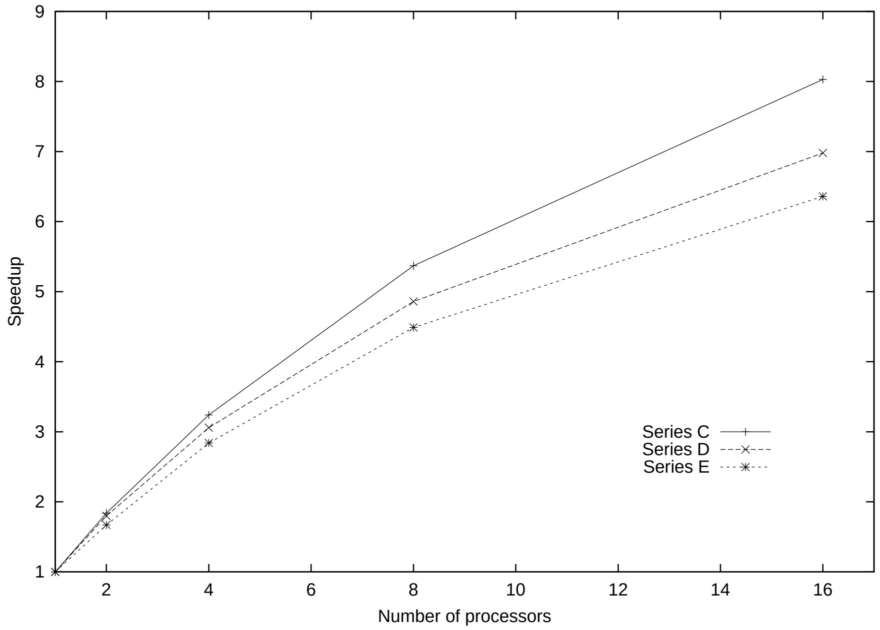

FIGURE 3. Speedups using up to 16 processors for Series C, D, and E.

6. Concluding remarks

We described a parallel greedy randomized adaptive search procedure for the Steiner problem in graphs, using a hybrid local search strategy combining path-based and node-based neighborhoods. The algorithm found optimal solutions for 45 out of 60 benchmark instances and was never off by more than four percent of the optimal solution. To illustrate the effectiveness of this parallel approach in terms of solution quality, we point out that a recent state-of-the-art tabu search algorithm [33] found 42 optimal solutions for the same set of test problems. Also, an improved reactive tabu search implementation [2] found 44 optima, with seven additional ones through the application of path-relinking [16, 17] as a post-optimization strategy.

The average speedup results observed for each series of test problems show that increasing the number of processors reduces elapsed times with increasing speedups. Moreover, the main contribution of the parallel algorithm concerns the fact that larger speedups of the same order of the number of processors are obtained exactly for the most difficult problems.

References

[1] Y. P. Aneja. An integer programming approach to the Steiner problem in graphs. *Networks*, 10:167–178, 1980.

[2] M.P. Bastos and C.C. Ribeiro. Reactive tabu search with path-relinking for the Steiner problem in graphs. In *Proceedings of the Third Metaheuristics International Conference*, pages 31–36, 1999.

[3] J. E. Beasley. An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16, 1989.

[4] J.E. Beasley. An algorithm for the Steiner problem in graphs. *Networks*, 14:147–159, 1984.

[5] J.E. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[6] S. Chopra, E. R. Gorres, and M. R. Rao. Solving the Steiner tree problem using branch and cut. *ORSA Journal on Computing*, 4:320–335, 1992.
[7] E.-A. Choukmane. Une heuristique pour le problème de l'arbre de Steiner. *RAIRO Recherche Opérationnelle*, 12:207–212, 1978.
[8] A. Claus and N. Maculan. Une nouvelle formulation du probleme de Steiner sur un graphe. Technical Report 280, Centre de Recherche sur les Transports, University of Montreal, 1983.
[9] K.A. Dowsland. Hill-climbing simulated annealing and the Steiner problem in graphs. *Engineering Optimization*, 17:91–107, 1991.
[10] C.W. Duin and A. Volgenant. Reduction tests for the Steiner problem in graphs. *Networks*, 19:549–567, 1989.
[11] C.W. Duin and S. Voss. Efficient path and vertex exchange in Steiner tree algorithms. *Networks*, 29:89–105, 1997.
[12] H. Esbensen. Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm. *Networks*, 26:173–185, 1995.
[13] 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.
[14] T.A. Feo and M.G.C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.
[15] Message Passing Interface Forum. MPI: A new message-passing interface standard (version 1.1). Technical report, University of Tennessee, 1995.
[16] F. Glover. Tabu search and adaptive memory programming – Advances, applications and challenges. Technical report, University of Colorado, 1996.
[17] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.
[18] M. X. Goemans. The Steiner tree polytope and related polyhedra. *Mathematical Programming*, 63:157–182, 1994.
[19] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner tree problem*. North-Holland, 1992.
[20] J. Plesník. A bound for the Steiner problem in graphs. *Math. Slovaca*, 31:155–163, 1981.
[21] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Solving the graphical Steiner tree problem using genetic algorithms. *Journal of the Operational Research Society*, 44:397–406, 1993.
[22] R.M. Karp. Reducibility among combinatorial problems. In E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.
[23] B.N. Khoury, P.M. Pardalos, and D.W. Hearn. Equivalent formulations for the Steiner problem in graphs. In D.-Z. Du and P.M. Pardalos, editors, *Network Optimization Problems*, pages 111–123. World Scientific, 1993.
[24] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998.
[25] L.T. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Informatica*, 15:141–145, 1981.
[26] A. Lucena. Tight bounds for the Steiner problem in graphs. Technical report, IRC for Process Systems Engineering, Imperial College, 1993.
[27] F. Margot, A. Prodon, and Th. M. Liebling. Tree polyhedron on 2-tree. *Mathematical Programming*, 63:183–192, 1994.
[28] S.L. Martins, P.M. Pardalos, M.G. Resende, and C.C. Ribeiro. GRASP procedures for the Steiner problem in graphs. In P. Pardalos, S. Rajasekaran, and J. Rolim, editors, *Randomization Methods in Algorithm Design*, volume 43 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 133–145. American Mathematical Society, 1998.
[29] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A parallel GRASP for the Steiner problem in graphs. In *Proceedings of IRREGULAR'98– 5th International Symposium on Solving Irregularly Structured Problems in Parallel*, volume 1457 of *Lecture Notes in Computer Science*, pages 310–331. Springer-Verlag, 1998.
[30] K. Mehlhorn. A faster approximation for the Steiner problem in graphs. *Information Processing Letters*, 27:125–128, 1988.
[31] M. Minoux. Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity. *INFOR*, 28:221–233, 1990.
[32] V.J. Rayward-Smith and A. Clare. On finding Steiner vertices. *Networks*, 16:283–294, 1986.
[33] C.C. Ribeiro and M.C. Souza. Tabu search for the Steiner problem in graphs. *Networks*, 2000. To appear.

[34] H. Takahashi and A. Matsuyama. An approximate solution for the Steiner problem in graphs. *Math. Japonica*, 24:573–577, 1980.

[35] 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 and Sons, 1996.

[36] S. Voss. Steiner's problem in graphs: Heuristic methods. *Discrete Applied Mathematics*, 40:45–72, 1992.

[37] R. T. Wong. A dual ascent approach for Steiner tree problems on directed graphs. *Mathematical Programming*, 28:271–287, 1984.

(S.L. Martins) Department of Computer Science, Catholic University of Rio de Janeiro, r. Marquês de São Vicente, 225, Rio de Janeiro, RJ 22453-900 Brazil
E-mail address, S.L. Martins: simone@inf.puc-rio.br

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

(C.C. Ribeiro) Department of Computer Science, Catholic University of Rio de Janeiro, r. Marquês de São Vicente, 225, Rio de Janeiro, RJ 22453-900 Brazil
E-mail address, C.C. Ribeiro: celso@inf.puc-rio.br

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