Hybrid Local Search for the Steiner Problem in Graphs

Marcus Poggi de Aragão * Celso C. Ribeiro * Eduardo Uchoa *
Renato F. Werneck *

* Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro
Rua Marquês de São Vicente, 225, Rio de Janeiro, 22453-900, Brasil.
Email: {poggi, celso, uchoa, rwerneck}@inf.puc-rio.br

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 $w : E \rightarrow \mathbb{R}_+$ associated with its edges and a subset $X \subseteq V$ of terminal nodes, the Steiner problem in graphs (SPG) consists in finding a minimum weighted connected subtree of G spanning all terminal nodes in X. The solution of SPG is a Steiner minimum tree. The non-terminal nodes that end up in the Steiner minimum tree are called Steiner nodes.

The Steiner problem in graphs is a classic combinatorial optimization problem, see e.g. Hwang, Richards, and Winter [5], Maculan [8], and Winter [20] for surveys. Karp [6] showed earlier that its decision version is NP-complete in the general case. Applications can be found in many areas, such as telecommunication network design, VLSI design, and computational biology, among others. Several heuristics are available for the approximate solution of the Steiner problem in graphs, see e.g. Duin and Voss [3], Hwang et al. [5], and Voss [18] for recent surveys. The shortest-path heuristic (SPH) of Takahashi and Matsuyama [14] described in Section 3 is one of the most effective algorithms for computing greedy approximate solutions to SPG.

Local search heuristics based on the insertion of Steiner nodes were proposed by Minoux [10] and Voss [17]. A *node-based* neighborhood for the SPG may be defined by all solutions which can be attained from the current one by inserting or eliminating a Steiner node. Since solutions of the Steiner problem may be characterized by their associated sets of Steiner nodes and one of the corresponding minimum spanning trees, the search for a optimal Steiner tree can be reduced to the search for the optimal set of Steiner nodes.

An efficient local search procedure based on the exchange of key-paths was described by Verhoeven et al. [16]. A *key-node* is a Steiner node with degree at least three. A *key-path* is a path in a Steiner tree of which all (if any) intermediate nodes are Steiner nodes with degree two and whose ends are either terminals or key-nodes. A *path-based* neighborhood for the SPG may be defined by replacing key-paths of the current solution by shortest paths connecting the two components of the tree obtained by the removal of the former.

Duin and Voss discussed in [4] the MultiplePass improvement procedure based on the modification of the set of key-nodes. This procedure is embedded within their Pilot metaheuristic. In this work we study the behavior of a local search procedure using a similar neighborhood structure. The main components of this procedure are: (i) the *key-node* neighborhood structure, (ii) a fast implementation of the constructive algorithm of Takahashi and Matsuyama [14] used for computing solutions in the neighborhood, and (iii) the use of simpler neighborhoods (node-based and path-based) to speed up the search. The key-node neighborhood and the local search procedure are described in Section 2. The construction procedure and its fast implementation are discussed in Section 3. Section 4 reports some computational results on benchmark instances. Concluding remarks and extensions are discussed in the last section.

2 Local search

Given a current solution, each of its neighbors in the *key-node neighborhood* is obtained by trying to either insert or eliminate a single key-node. In the first case, a non-terminal node v of the original graph which is not a key-node in the current tree is temporarily considered as a terminal node. Then, the shortest-path heuristic SPH described in the next section is applied to an extended set of terminal nodes formed by v itself, all original terminals, and the key-nodes in the current tree, using v as the root. In the second case, a key-node v of the current tree is tentatively removed from the set of key-nodes. Again, SPH is applied using the remaining key-nodes and the original terminals as the set of temporary terminals, one of which randomly selected as the root. In both cases, non-terminal nodes with degree one are pruned. The new solution becomes the incumbent if it is not worse than the current one. The procedure stops after a full pass through all non-terminal nodes without improvement in the value of the incumbent solution.

Although this neighborhood consistently leads to better local optima than node-based and path-based neighborhoods, it is also more costly, since each pass requires $|V|-|X|$ calls to the SPH procedure. Therefore, we ciclically combine the three neighborhoods to speed up the local search procedure. We start from the initial solution constructed by SPH exploring the path-based neighborhood until a locally optimal solution is found. We then proceed using the node-based neighborhood. Only then the key-node-based neighborhood is applied. We iterate until all three neighborhoods fail to improve the incumbent solution. This hybrid local search strategy using three neighborhood structures (HLS3) is a straightforward extension of the local search procedures successfully applied in [9, 13] using only the two first neighborhoods. Moreover, since the path-based and the key-node-based neighborhoods are not necessarily contained in one another, their combination is at least as effective in terms of solution quality as each of them applied individually.

3 Construction algorithm

A fast implementation of the shortest-path heuristic (SPH) of Takahashi and Matsuyama [14] is essential to our heuristic, since repeated calls to this procedure are the bottleneck of HLS3.

The algorithm starts from a given terminal as the root of the initial tree. At the beginning of any iteration i, let H_i be the set of terminal nodes already spanned by the current partial tree P_i and $s_i \in X \setminus H_i$ be its closest non-spanned terminal node. The algorithm obtains a new tree P_{i+1} by appending to P_i all nodes in the shortest path from s_i to P_i, and sets $H_{i+1} \leftarrow H_i \cup \{s\}$. It stops after $|X|-1$ iterations, when all terminal nodes are spanned.

A straightforward implementation of this algorithm performs one independent shortest-path routine to find the closest non-spanned terminal node in each iteration. Dijkstra's algorithm [2] typically uses a heap to store distances from each node to the current tree. In this straightforward implementation, the heap is reinitialized (with all vertices of P_i) in each iteration i, and Dijkstra's algorithm is applied from scratch. The key idea to obtain a faster implementation of SPH is to use the information gathered by early executions of Dijkstra's algorithm to give a "warm start" to the ones that follow. At the end of any given iteration i, the distances on the heap are consistent, in the sense that they are upper bounds to the actual distances to the new tree (P_i). There is no need to reset the heap. All we have to do is add to it (with distance 0) the nodes in $P_{i+1} \setminus P_i$ and proceed with Dijkstra's algorithm. The upper bounds will be automatically updated as the execution proceeds.

With this implementation, the heap will not be empty until the end of SPH. Moreover, each terminal node (except the first and the last ones) will be inserted into the heap only twice: one with a strictly positive distance, another with distance zero (right after it is incorporated into the tree). Non-terminal nodes, on the other hand, may be inserted up to $|X| - 1$ times, one in each iteration of the algorithm. Our implementation tries to avoid these reinsertions whenever possible. For each node v, there is an associated variable d_v that represents how far v was from the tree the last time it was removed from the heap (at the beginning, $d_v = \infty$ for all nodes). Whenever Dijkstra's algorithm tries to insert a node v into the heap, we first check if the value with which it would be inserted is strictly smaller than d_v. If it is not, the insertion is not performed. This does not affect the correctness of the algorithm, but leads to great speedups.

Using a binary heap, both implementations of SPH run in $O(|X||E| \log |V|)$ worst case time. However, our implementation is much faster in practice, and the $|X|$ factor is barely noticed. To further improve the solution obtained by SPH, we compute a minimum spanning tree of the graph induced in G by the nodes of the tree and then prune all degree one non-terminal nodes.

4 Computational results

Computational experiments were performed on three sets of test problems with quite different characteristics. The first set of test problems are series C, D, and E from the OR-Library [1]. Each series contains randomly generated instances, with edge weights taken from a uniform distribution in the interval $[1, 10]$. The second set is made up by instances defined over grid graphs with holes, extracted from real VLSI layout problems by Koch and Martin [7]. Finally, the third set is formed by incidence instances created by Duin and Voss [3] so as to make the reduction tests ineffective. Different preprocessing strategies were applied to the OR-Library and to the VLSI instances, as described in [13, 15]. All instances are available from the SteinLib repository [19].

The heuristic described in the previous sections was coded in `C++` and compiled with `GCC` using the full optimization `-O3` flag. Computational results obtained on a 400 MHz Pentium II machine with 64 MB of memory are summarized in Table 1, which excludes instances solved to optimality by preprocessing. We also omit the results for the incidence instances with 640 nodes, since the optimal solutions are not known for many of them. We report the number of instances in each series, followed by the results obtained by the combination of the shortest path heuristic with the hybrid local search procedure (SPH+HLS3): the number of optimal solutions found, the average percentual gap with respect to the optimal value, and the computation time in seconds (not including preprocessing times, which may be found in [13]). The last three columns report the same information for a more elaborate heuristic (GHLS3). This is essentially the hybrid GRASP with perturbations and path-relinking described in [13] using HLS3 as the local search procedure.

It is also interesting to mention that the tighter primal bounds obtained by GHLS3 were essential to accelerate the branch-and-ascent algorithm [11] used to solve to optimality all formerly open instances in series `i320`.

5 Concluding remarks

In this work, we presented a hybrid local search procedure for the Steiner problem in graphs, exploring a neighborhood defined by insertions and eliminations of key-nodes and making use of a fast implementation of the SPH heuristic. This local search strategy may also be viewed as an implementation the well known node-based local search acting on the distance network, with two advantages. First, there is no need to store the full $|V| \times |V|$ distance matrix. Second, SPH is able to find improving shortcuts due to the fact that it works on the original graph, and not in the distance network (in which the actual

		SPH+HLS3			GHLS3		
Series	Instances	Optima	Avg. gap (%)	Time (s)	Optima	Avg. gap (%)	Time (s)
C	19	17	0.46	0.27	19	0	44.1
D	19	19	0	1.19	19	0	190.3
E	19	13	0.07	9.53	18	0.0093	1502.0
alue	14	4	0.20	139.00	12	0.0042	24185.0
alut	8	3	0.11	206.83	7	0.0017	35412.0
diw	12	9	0.06	2.57	12	0	407.5
dmxa	8	5	0.30	0.06	8	0	9.7
gap	6	6	0	2.18	6	0	337.8
msm	19	17	0.09	0.38	19	0	64.3
taq	10	4	0.31	3.50	10	0	534.8
i080	100	25	2.22	0.06	97	0.0066	9.9
i160	100	16	2.14	0.47	84	0.0834	76.1
i320	100	14	2.38	4.20	74	0.1567	697.9

Table 1: Computational results for SPH+LS

intermediary nodes in each shortest path are not known to the minimum spanning tree algorithm). In the worst case, SPH obtains the exact same solution that would be obtained by Prim's algorithm [12] when applied to the distance network. In many situations, the SPH solution is actually better.

Computational results obtained with this hybrid local search procedure are competitive with other simple heuristics for the Steiner problem in graphs. This local search procedure leads to significant improvements with respect to previously used strategies when embedded in more sophisticated approaches such as the hybrid GRASP with perturbations and path-relinking described in [13], although at the cost of additional computation time. Detailed computational results and comparisons with other approaches will be reported elsewhere.

References

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

[2] E.W. Dijkstra, "A note on two problems in connexion with graphs", *Numerische Mathematik* 1 (1959), 269–271.

[3] C.W. Duin and S. Voss, "Efficient path and vertex exchange in Steiner tree algorithms", *Networks* 29 (1997), 89–105.

[4] C.W. Duin and S. Voss, "The Pilot method: A strategy for heuristic repetition with application to the Steiner problem in graphs", *Networks* 34 (1999), 181–191.

[5] F.K. Hwang, D.S. Richards, and P. Winter, *The Steiner tree problem*, North-Holland, Amsterdam, 1992.

[6] R.M. Karp, "Reducibility among combinatorial problems", in *Complexity of Computer Computations* (E. Miller and J.W. Thatcher, eds.), 85–103, Plenum Press, New York, 1972.

[7] T. Koch and A. Martin, "Solving Steiner tree problems in graphs to optimality", *Networks* 32 (1998), 207–232.

[8] N. Maculan, "The Steiner problem in graphs", in *Surveys in Combinatorial Optimization* (S. Martello, G. Laporte, M. Minoux, and C.C. Ribeiro, eds.), *Annals of Discrete Mathematics* 31 (1987), 185–212.

[9] S.L. Martins, P. Pardalos, M.G. Resende, and C.C. Ribeiro, "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy", *Journal of Global Optimization* 17 (2000), 267–283.

[10] M. Minoux, "Efficient greedy heuristics for Steiner tree problems using reoptimization and supermodularity", *INFOR* 28 (1990), 221–233.

[11] M.V. Poggi de Aragão, E. Uchoa, and R.F. Werneck, "Dual heuristics on the exact solution of large Steiner problems", *Electronic Notes in Discrete Mathematics* 7 (2001), to appear.

[12] R.C. Prim, "Shortest connection networks and some generalizations", *Bell Systems Technical Journal* 36 (1957), 1389–1401.

[13] C.C. Ribeiro, E. Uchoa, and R.F. Werneck, "A hybrid GRASP with perturbations for the Steiner problem in graphs", submitted for publication, presented at the *International Conference on Adaptive Memory and Evolution: Tabu Search and Scatter Search*, 2001.

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

[15] E. Uchoa, M.V. Poggi de Aragão, and C.C. Ribeiro, "Preprocessing Steiner problems from VLSI layout", submitted for publication, 1999.

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

[17] S. Voss, *Steiner-Probleme in Graphen*, Anton Hain, 1990.

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

[19] S. Voss, A. Martin, and T. Koch, "SteinLib Testdata Library", online document at http://elib.zib.de/steinlib/steinlib.html, last visited on March 28, 2001.

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