A nearly best-possible approximation algorithm for node-weighted Steiner trees

Philip Klein R. Ravi

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-92-54
September 1993

A nearly best-possible approximation algorithm for node-weighted Steiner trees

Philip Klein[*] R. Ravi[†]

Brown University, Providence, RI 02912, USA

Abstract

We give the first approximation algorithm for the node-weighted Steiner tree problem. Its performance guarantee is within a constant factor of the best possible unless $\tilde{P} \supseteq NP$. Our algorithm generalizes to handle other network design problems.

1 Introduction

The Steiner problem in networks is a classic hard problem in combinatorial optimization. Much research has been devoted to heuristics for its solution [4, 7, 14, 16, 17, 22]. Despite a slew of new approximation algorithms for this problem and some of its variants, no approximation algorithm has been given for perhaps the most natural variant: the *node-weighted Steiner tree problem*, in which costs can be assigned to nodes as well as edges. Indeed, Winter's survey [21] on the network Steiner problem closes with the sentence, "Further investigation of the vertex-weighted SPN [Steiner problem in networks] is needed."

One reason for the dearth of results on the node-weighted variant may be that it is harder than the standard problem. Indeed, while constant-factor approximations are known for the standard problem [10, 15, 20, 23] and even some of its generalizations [1, 6], the node-weighted version cannot be approximated to within less than a logarithmic factor unless $\tilde{P} \supseteq NP$ [2, 13]. [1]

In this paper, we give the first approximation algorithm for the node-weighted Steiner tree problem. The performance guarantee is logarithmic. Thus assuming $\tilde{P} \not\supseteq NP$, the accuracy of our approximation is within a constant factor of the best-possible approximation achievable in polynomial time.

The algorithm we propose is only a slight variant of a heuristic proposed by Rayward-Smith and Clare in 1986 [8, 17, 16] for the standard edge-weighted Steiner tree problem. The key to our analysis is a decomposition lemma for trees; this lemma may be useful in other contexts as well.

We also show how to generalize the algorithm and its analysis to handle more general connectivity requirements. Thus we obtain approximation algorithms for node-weighted versions of, e.g, fixed and non-fixed point-to-point connection problems.

Preliminaries

Let G be a graph with nonnegative costs assigned to its nodes and edges. The cost of a subgraph of G is the sum of the costs of its nodes and edges. Let A be a subset of the nodes of G, called *terminals*. A *Steiner tree for A in G* is a connected subgraph of G containing all the nodes of A. (Note that an edge-minimal Steiner tree is indeed a tree.) The Steiner tree problem is to find a minimum-cost Steiner tree.

[*]Research supported by NSF grant CCR-9012357 and NSF PYI award CCR-9157620, together with PYI matching funds from Thinking Machines Corporation and Xerox Corporation. Additional support provided by DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

[†]Research supported by an IBM Graduate Fellowship. Additional support provided by NSF PYI award CCR-9157620 and DARPA contract N00014-91-J-4052 ARPA Order No. 8225.

[1]Here we use $\tilde{P}$ to mean the complexity class Deterministic Quasipolynomial time, or DTIME$[n^{\text{polylog } n}]$.

We introduce a problem that turns out to be closely related. Let B be a ground set, and let $S_1, \ldots, S_s$ be subsets of B with costs $c_1, \ldots, c_s$. A *set cover* is a collection of sets S_i whose union is B. The *set cover problem* is to find a minimum-cost set-cover.

Berman [2] showed that, in the presence of node-weights, approximating the minimum-cost Steiner tree is as hard as approximating set cover. More specifically, he showed that any instance of set-cover can be formulated as an instance of the node-weighted Steiner tree problem. The reduction is illustrated in Figure 1. Thus an approximation algorithm for minimum-cost Steiner tree could be used to achieve the same approximation for set-cover.

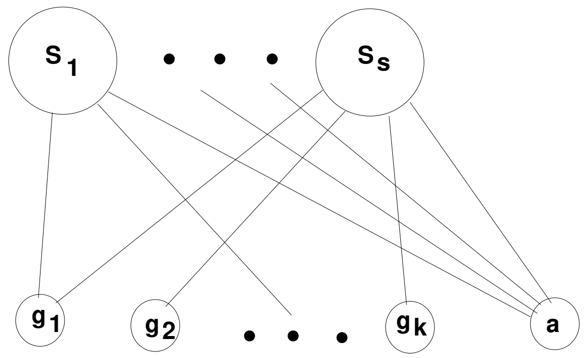

Figure 1: Let G be the bipartite graph with $k+1$ nodes on one side of the bipartition, one for each ground element and one auxiliary node a, and s nodes on the other side, one for each set S_i. There is an edge between a ground element node and a set node if the set contains the element. In addition, the auxiliary node a is adjacent to all the set nodes. The cost of each set node is that of the set it represents. All other nodes and all edges have zero cost. It is easy to check that a Steiner tree for the $k+1$ nodes corresponds to a set cover of the same cost, and vice versa.

It has recently been proved [13] that no polynomial-time approximation algorithm for set-cover achieves an approximation factor smaller than $\frac{1}{4} \ln |B|$ (unless Deterministic time $n^{\text{polylog} n}$ contains NP). By Berman's reduction, the same holds for the node-weighted Steiner tree problem. Thus we cannot expect to obtain an approximation algorithm that achieves a performance ratio better than logarithmic.

Theorem 1.1 *There is a polynomial-time algorithm to approximate the node-weighted Steiner tree problem in networks. The performance ratio is* $2 \ln k$*, where* k *is the number of terminals.*

An example due to Chvátal [3] can be adapted to show that the cost of our algorithm's output can be as much as $\ln k$ times optimal.

Our method can be easily applied to more general node-weighted network-design problems. For example, consider the following generalization of the Steiner problem: given a set of pairs of nodes (s_i, t_i), find a minimum-cost subgraph in which each s_i is connected to t_i. The edge-weighted version of this problem was addressed in [1]; the node-weighted version can be approximately solved using the method of this paper.

We use a framework due to Goemans and Williamson [6] to formulate problems like that described above. Many network-design problems can be formulated as finding a subgraph of minimum-cost that covers a family of cuts in the graph (the particular family depends on the problem). For certain families of cuts, the edge-weighted problem can be approximated to within a factor of two [6]. We show in Section 5 that the node-weighted variants of these network design problems can also be approximated; the performance is as in Theorem 1.1.

2 The algorithm

In this section we describe the algorithm. Note that since any Steiner tree must include all the terminals, we can assume without loss of generality that the terminals have zero cost.

The algorithm maintains a node-disjoint set of trees containing all the terminals. Initially, each terminal is in a tree by itself.

The algorithm uses a greedy strategy to iteratively merge the trees into larger trees until there is only one tree. In each iteration, it selects a node and a subset of the current trees of size at least two so as to minimize the ratio

$$\frac{\text{cost of the node plus sum of distances to the trees}}{\text{number of trees}} \tag{1}$$

Here the distance along a path does not include the costs of its endpoints. Thus the choice minimizes the average node-to-tree distance. The algorithm uses the shortest paths between the vertex and the selected trees to merge the trees into one.

It is easy to implement an iteration. For each node v, define the *quotient cost* of v to be the minimum value of (1), taken over all subsets of the the current trees. To find the quotient cost of v, compute the distances d_i from v to each of the trees T_i; assume without loss of generality that the trees are numbered so that $d_1 \leq d_2 \leq \cdots \leq d_k$. In computing the quotient cost of v, it is sufficient to consider subsets of the form $\{T_1, T_2, \ldots, T_i\}$. Thus the quotient cost for a given vertex can be computed in polynomial time; by computing the quotient cost of all the vertices, we can determine the minimum quotient cost, and thus carry out an iteration in polynomial time.

In Section 4, we show that the cost of all nodes and edges selected by the algorithm is not much more than the minimum cost of a Steiner tree.

3 Spider decomposition

The proof of the performance guarantee of the algorithm involves showing the existence of a node with low quotient cost relative to the minimum cost of a Steiner tree. To show this, we prove that a minimum Steiner tree can be decomposed into subtrees we call *spiders*. It follows that one of these spiders has low quotient cost relative to the cost of the minimum Steiner tree. It then follows that the cost of an iteration of the greedy algorithm is small.

Definition: A *spider* is a tree with at most one node of degree greater than two. A *center* of a spider is a node from which there are edge-disjoint paths to the leaves of the spider. Note that if a spider has at least three leaves, its center is unique. A *foot* of a spider is a leaf, or, if the spider has at least three leaves, the spider's center. Note that every spider contains disjoint paths from its center to all of its feet. A *nontrivial spider* is one with at least two leaves.

Definition: Let G be a graph, and let M be a subset of its nodes. A *spider decomposition* of M in G is a set of node-disjoint nontrivial spiders in G such that the union of the feet of the spiders in the decomposition contains M.

Theorem 3.1 *Let G be a connected graph, and let M be a subset of its nodes such that $|M| \geq 2$. Then G contains a spider decomposition of M.*

For convenience, we call the nodes of M *marked nodes.* Since G is connected, it has a spanning tree T. We prove that T contains a spider decomposition.

Definition: For any two paths P and Q that intersect only at one endpoint, we use QP to denote the path formed by the the concatenation of these paths at their point of intersection.

Claim 3.2 *Suppose T has an even number of marked nodes. Then there is a pairing $(v_1, w_1), \ldots, (v_k, w_k)$ of the marked nodes such that the $v_i - w_i$ paths in T are edge-disjoint.*

Proof: Let $(v_1, w_1), \ldots, (v_k, w_k)$ be the pairing minimizing the quantity

$$\sum_{i=1}^{k} |E(P_i)| \tag{2}$$

where P_i is the path in T from v_i to w_i and $E(P)$ is the set of edges in P. We claim that the paths P_i are edge-disjoint. The proof is depicted in Figure 2.

□

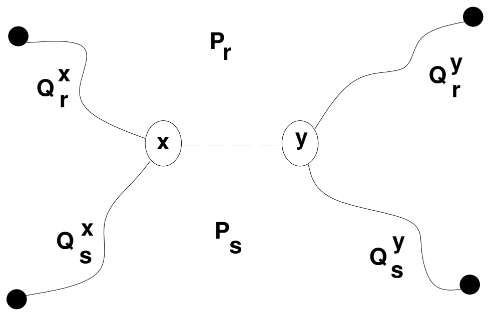

Figure 2: Let $(v_1, w_1), \ldots, (v_k, w_k)$ be the pairing minimizing the number of occurrences of edge-overlap between the $v_i - w_i$ paths in T. Let P_i be the path in T between v_i and w_i. Then the paths P_i are edge-disjoint. Suppose for a contradiction that P_r and P_s share an edge xy. Let Q_r^x be the terminal subpath of P_i ending at x and not passing through y. Similarly define Q_r^y, Q_s^x, and Q_s^y. Then by replacing P_r and P_s with $Q_r^x Q_s^x$ and $Q_r^y Q_s^y$, we obtain a pairing that contradicts the minimality of the initial pairing.

Claim 3.3 *Let S be a nontrivial spider and let P be a v-to-w path intersecting S only at w. Then $S \cup P$ contains one or two node-disjoint nontrivial spiders whose feet contain the feet of S together with v.*

Proof: If w is a center of S, then $S \cup P$ is a spider. Otherwise, let Q be the center-to-leaf path of S going through w (not including the center). Let Q' be the terminal subpath of Q from w to the leaf. Then $S - Q$ and $Q' \cup P$ are node-disjoint spiders (See Figure 3).

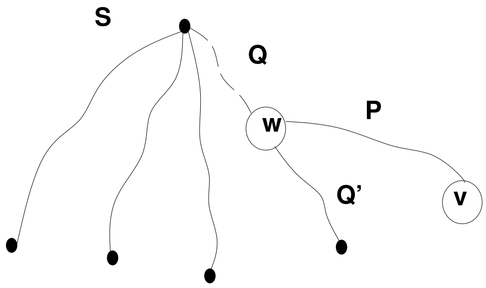

Figure 3: S is a nontrivial spider and P is a v-w path intersecting S only at w. Suppose w is not a center of S. Let Q be the center-to-leaf path of S going through w. Let Q' be the terminal subpath of Q from w to the leaf. Then $S - Q$ and $Q' \cup P$ are node-disjoint spiders. Note that $Q' \cup P$ has two leaves, so it is nontrivial. Moreover, $S - Q$ has one fewer leaves than S. Since S has at least three leaves (else any of its nodes is a center), $S - Q$ has at least two. Thus $S - Q$ is nontrivial.

□

Claim 3.4 *Suppose T contains k edge-disjoint paths whose endpoints are the marked nodes. Then T contains a spider decomposition.*

Proof: By induction on k. The basis $k = 0$ is trivial. Suppose we have $k + 1$ edge-disjoint paths $P_1, \ldots, P_{k+1}$. Consider as marked only the endpoints of the first k of these paths. By the inductive hypothesis, T contains node-disjoint spiders $S_1, \ldots, S_r$ whose feet include these endpoints. If P_{k+1} does not intersect any of these spiders then we are done, for $S_1, \ldots, S_r, P_{k+1}$ is the required set of spiders. Otherwise, let v be an endpoint of P_{k+1}, and let S_t be the spider whose intersection with P_{k+1} is closest to v. Let P be the subpath of P_{k+1} from v to this intersection. By applying Claim 3.3 to S_t and P, we obtain one or two spiders; By replacing S_t with these spiders, we obtain a set of node-disjoint spiders $\{S_i'\}$ whose feet include v along with the endpoints of the first k paths.

Let w be the other endpoint of P_{k+1}. We again apply Claim 3.3 as above to obtain a set of spiders whose feet also include w. □

Now we complete the proof of Theorem 3.1. First assume T has an even number of marked nodes. By Claim 3.2, we have edge-disjoint paths in T whose endpoints are the marked nodes. Hence by Claim 3.4 we obtain the desired spiders.

Now suppose T has an odd number of marked nodes. Let v be any marked node. Considering v as unmarked, proceed as above to obtain node-disjoint spiders. Let S be the spiders closest to v, and let P be the path from v to S. By applying Claim 3.3, we replace S with one or two spiders whose feet include the feet of S together with v. This completes the proof of Theorem 3.1.

4 The performance guarantee

Now we prove the performance guarantee for the greedy algorithm. Let ϕ_i denote the number of subtrees in the solution just after iteration i. Thus, for instance, ϕ_0 is the number of terminals in G. Let the number of trees merged at iteration i be h_i. Then we have

$$\phi_i = \phi_{i-1} - (h_i - 1) \tag{3}$$

Let C_i denote the cost of the subgraph added by the algorithm in iteration i. Let OPT denote the cost of a minimum-cost Steiner tree spanning the terminals. The key ingredient of our proof is the following lemma, which depends on our spider-decomposition theorem.

Lemma 4.1 *At any iteration i of the algorithm,*

$$h_i \geq \frac{C_i \cdot \phi_{i-1}}{OPT} \tag{4}$$

Let T^* be a minimum-cost Steiner tree. Let $T_1, \ldots, T_{\phi_{i-1}}$ be the current trees at the beginning of iteration i of the algorithm. Let T_i^* be the graph obtained from T^* by contracting each T_j to a supernode of zero cost. Note that T_i^* is connected and contains all supernodes. Let M be the set of supernodes. We apply Theorem 3.1 to the graph T_i^* to obtain a spider decomposition of M. Furthermore, the cost of all spiders in the decomposition is at most that of T_i^* which is in turn at most OPT.

Let $c_1, \ldots, c_r$ be the centers of the spiders in the decomposition. For a spider with only two leaves, i.e., a path, pick any node in the path as its center. Let $\ell_1, \ldots, \ell_r$ denote the number of nodes of M in each of these spiders respectively. Let the cost of the spider centered at c_j be $Cost_j$. Since every spider in the decomposition is nontrivial, each ℓ_j is at least two. Moreover, a spider with center c_j induces a subset of the current trees, namely the ℓ_j trees whose supernodes belong to this spider. The cost of c_j plus the sum of distances to these trees is exactly $Cost_j$. Hence the quotient cost of c_j is at most

$$\frac{Cost_j}{\ell_j}$$

Since the algorithm chooses a vertex of minimum quotient cost, for each spider in the decomposition we have

$$\frac{Cost_j}{\ell_j} \geq \frac{C_i}{h_i}$$

Summing over all the spiders in the cover yields

$$\sum_{j=1}^{r} Cost_j \geq \frac{C_i}{h_i} \sum_{j=1}^{r} \ell_j$$

By node-disjointness, the sum $\sum_{j=1}^{r} \ell_j$ of the number of nodes of M in each of the spiders is exactly the number ϕ_{i-1} of current trees. Also $\sum_{j=1}^{r} Cost_j$ is exactly the cost of the spider decomposition, which is at most OPT. Substituting in the above equation and simplifying yields

$$h_i \geq \frac{C_i \phi_{i-1}}{OPT}$$

□

We now use the above lemma in conjunction with an analysis technique due to Leighton and Rao [11] to complete the proof of the performance guarantee.

Substituting Equation (4) into (3) and using the inequality $h_i \leq 2(h_i - 1)$, we get

$$\phi_i \leq \phi_{i-1}(1 - \frac{C_i}{2OPT}) \tag{5}$$

If the total number of iterations of the algorithm is p, then $\phi_p = 0$ while $\phi_{p-1} \geq 2$. Unraveling (5), we obtain

$$\phi_{p-1} \leq \phi_0 \prod_{j=1}^{p-1}(1 - \frac{C_j}{2OPT})$$

Taking natural logarithms on both sides and simplifying using the approximation $\ln(1 + x) \leq x$, we obtain

$$2OPT \ln(\frac{\phi_0}{\phi_{p-1}}) \geq \sum_{j=1}^{p-1} C_j$$

Note that $\phi_0 = k$ and $\phi_{p-1} \geq 2$ and so we have

$$\sum_{j=1}^{p-1} C_j \leq 2(\ln k - \ln 2)OPT < (2 \ln k - 1)OPT \tag{6}$$

Since we assumed that all the terminals have zero cost, note that the cost of the final solution is the exactly the sum $\sum_{j=1}^{p} C_j$.

To complete the proof, we bound the cost of the last iteration. Using Lemma 4.1 and noting that $h_p = \phi_{p-1}$ we have

$$C_p \leq OPT$$

Using the above equation and (6), we have that the cost of the solution output by the algorithm is

$$\sum_{j=1}^{p} C_j < 2 \ln k OPT$$

This proves the performance guarantee in Theorem 1.1. □

5 General Node-weighted Network Design Problems

In this section, we generalize our algorithm to handle more general network design problems. As discussed in the introduction, many such problems can be formulated as cut-covering problems: for a certain family of cuts in a graph, find a minimum-cost subgraph intersecting all the cuts in the family. Fix a graph G with node- and edge-weights. For any node-subset S, there is a corresponding cut $\Gamma(S)$ in G, namely that consisting of edges with exactly one endpoint in S. Thus we can use a 0-1 function f on the set of node-subsets to define a family of cuts: $f(S) = 1$ whenever $\Gamma(S)$ is in the family. Goemans and Williamson [6] proposed a class of cut-families and showed that they are useful in modeling network design problems such as the point-to-point connection problem. They called a function f *proper* if it obeys the following properties: (null property) $f(\emptyset) = 0$; (symmetry property) $f(S) = f(V - S)$ for all $S \subseteq V$; and (disjointness property) if A and B are disjoint, then $f(A) = f(B) = 0$ implies $f(A \cup B) = 0$.

Given a function f, the *terminals* are the nodes v of G such that $f(\{v\}) = 1$. As before, every solution subgraph must contain all the terminals. Hence we can assume without loss of generality that the terminals have zero cost.

Goemans and Williamson gave a 2-approximation algorithm for edge-weighted cut-cover problems where the cut-family corresponds to a proper function f. In this paper we give a complementary result for node-weighted problems.

Theorem 5.1 *Let G be a graph with node- and edge-weights. Let f be a proper function on the node-subsets of G. There is a polynomial-time approximation algorithm for finding a minimum-cost subgraph covering all cuts in the family defined by f. The performance guarantee is $2 \ln k$, where k is the number of terminals.*

Proof of Theorem 5.1: The algorithm in the theorem follows the outline of the algorithm in Section 2 very closely. As before, the algorithm maintains a set of node-disjoint trees. Initially each terminal is in a tree by itself; the algorithm merges them iteratively. An important difference is that only some of the trees are candidates for merging.

We designate a tree to be *active* if $f(\{\text{nodes in the tree}\}) = 1$. At each iteration, the algorithm proceeds as before but considers only active trees in computing the quotient costs of nodes. That is, the algorithm selects a node and a set of at least two active trees so as to minimize

$$\frac{\text{cost of the node plus sum of distances to the trees}}{\text{number of trees}} \tag{7}$$

Then the algorithm uses the paths from the node to the selected trees and merges them into a single tree.

As long as there is at least one active tree in the network, it follows by the symmetry and disjointness properties of f that there are at least two. The algorithm reduces the number of active trees in the network by at least one in each iteration, so it eventually terminates and outputs a set of inactive trees as the final solution. Thus each connected component of the solution is inactive. It follows [6] that the solution is in fact a cut-cover.

The proof of the performance guarantee proceeds as before, except that we define the value of the potential function ϕ_i to be the number of active trees in the current solution, rather than the number of trees. To prove the analogue of Lemma 4.1, we consider an optimal cut-cover, contract the current trees, and use Theorem 3.1 to obtain a spider decomposition of the contracted active trees. We use the result in the inequality

$$\phi_i \leq \phi_{i-1} - (h_i - 1) \tag{8}$$

where h_i is the number of active trees merged in the ith iteration. Note that with our new definition of the potential function ϕ_i we have inequality in (8) instead of equality as in (3) because the tree resulting from the merge may be inactive. The rest of the analysis is identical. This completes the proof of Theorem 5.1. □

6 Conclusions and open problems

We have described approximation techniques for a variety of one-connected network design problems. These are the first approximation algorithms that can handle costs on the nodes.

Two other natural generalizations of the Steiner tree problem are at least as hard to approximate as the set cover problem, the directed Steiner problem and the group Steiner problem. In the directed Steiner problem, we are given a directed arc-weighted graph, a distinguished node, and a set of terminals. The goal is to find a minimum-cost arborescence rooted at the distinguished node and spanning the terminals. A transformation of the node-Steiner problem to the directed version was proposed by A. Segev [19].

The group Steiner problem, proposed by Reich and Widmayer [18], arises in VLSI design. In this problem, we are given an undirected edge-weighted graph and a collection of node-subsets, called *groups*. The goal is to find a minimum-cost connected subgraph containing at least one node from each group.

We now show how to reduce the set cover problem to a group Steiner problem: construct a graph with an auxiliary node and one node for each subset in the set cover problem. Each subset-node has an edge to the auxiliary node of cost equal to the cost of the subset. In formulating the group Steiner problem on this graph, define a group for each ground set element in the set cover problem: namely, the set of nodes corresponding to the subsets that contain the ground element. It is easy to check that a group Steiner tree in this graph corresponds to a set cover of the same cost and vice-versa.

Thus both the group Steiner problem and the directed Steiner problem are candidates for logarithmic-factor approximation algorithms. Indeed, we also observe that an approximation algorithm for the latter would yield one for the former.

An instance of the group Steiner problem can be transformed to an instance of the directed Steiner problem as follows: replace each edge in the input graph with a pair of antiparallel arcs of the same cost. For each group $G = \{u_1, \ldots, u_t\}$ of vertices, introduce a new vertex v_G and add zero-cost arcs from each $u_i \in G$ to v_G. It is easy to verify that a outward-directed Steiner tree originating from a node in any group in this transformed graph can be converted to a group Steiner tree of the same edge-cost in the original graph and vice-versa.

Acknowledgements

Thanks to Esther Jesurum for suggesting the term "spider." Thanks to Bobby Blumofe for pointing out an error in a previous draft.

References

[1] A. Agrawal, P. Klein and R. Ravi, "When trees collide: an approximation algorithm for the generalized Steiner tree problem on networks," *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing* (1991), pp. 134-144.

[2] P. Berman, personal communication, 1991.

[3] V. Chvátal, "A greedy heuristic for the set-covering problem," *Math. of OR* Vol. 4, No. 3, pp. 233-235 (1979).

[4] S. E. Dreyfus, and R. A. Wagner, "The Steiner problem in graphs," *Networks, vol. 1* (1971), pp. 195-207.

[5] J. Edmonds, and E. L. Johnson, "Matching, Euler tours and the Chinese postman", *Math. Prog. 5*, (1973), pp. 88-124.

[6] M. X. Goemans, and D. P. Williamson, "A general approximation technique for constrained forest problems", *Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms* (1992), pp. 307-316.

[7] S. L. Hakimi, "Steiner's problem in graphs and its implications," *Networks, vol. 1* (1971), pp. 113-133.

[8] F. K. Hwang and D. S. Richards, "Steiner tree problems," *Networks*, Vol. 22, No. 1, pp. 55-90 (1992).

[9] D. S. Johnson, "Approximation algorithms for Combinatorial Problems," *J. Comp. System. Sci.*, 9, pp. 256-278 (1974).

[10] L. Kou, G. Markowsky and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica, vol. 15* (1981), pp. 141-145.

[11] F. T. Leighton and S. Rao, "An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms," *Proceedings of the 29th Annual IEEE Conference on Foundations of Computer Science* (1988), pp. 422-431.

[12] L. Lovász, "On the Ratio of Optimal Integral and Fractional Covers," *Discrete Math.*, 13, pp. 383-390 (1975).

[13] C. Lund and M. Yannakakis, "On the Hardness of Approximating Minimization Problems," (manuscript).

[14] K. Mehlhorn, "A faster approximation algorithm for the Steiner problem in graphs," *Information Processing Letters, vol. 27(3)* (1988), pp. 125-128.

[15] J. Plesnik, "A bound for the Steiner tree problem in graphs," *Math. Slovaca, vol. 31* (1981), pp. 155-163.

[16] V. J. Rayward-Smith, "The computation of nearly minimal Steiner trees in graphs," *Internat, J. Math. Ed. Sci. Tech.*, 14 (1), pp. 15-23 (1983).

[17] V. J. Rayward-Smith and A. Clare, "On finding Steiner vertices", *Networks*, 16, pp. 283-294 (1986).

[18] G. Reich and P. Widmayer, "Beyond Steiner's problem: A VLSI oriented generalization," in *Proc., 15th International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 196-210, Castle Rolduc, (1989).

[19] A. Segev, "The Node-Weighted Steiner Tree Problem," *Networks*, Vol. 17, pp. 1-17, (1987).

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

[21] Pawel Winter, "Steiner problem in networks : a survey," *BIT 25* (1985), pp. 485-496.

[22] Y. F. Wu, P. Widmayer and C. K. Wong, "A faster approximation algorithm for the Steiner problem in graphs," *Acta Informatica, vol. 23* (1986), pp. 321-331.

[23] A. Z. Zelikovsky, "The 11/6-approximation algorithm for the Steiner problem on networks," *Institute of Mathematics, Kishinev, USSR.*