JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 22, 1409-1425 (2006)

FasterDSP: A Faster Approximation Algorithm for Directed Steiner Tree Problem*

MING-I HSIEH, ERIC HSIAO-KUANG WU AND MENG-FENG TSAI
Department of Computer Science and Information Engineering
National Central University
Chungli, 320 Taiwan
E-mail: mihs@wmlab.csie.ncu.edu.tw
E-mail: {hsiao; mftsai}@csie.ncu.edu.tw

Given a weighted directed graph $G = (V, E, c)$, where $c : E \to \mathcal{R}^+$ is an edge cost function, a subset X of vertices (*terminals*), and a root vertex v_r, the directed Steiner tree problem (DSP) asks for a minimum-cost tree which spans the paths from root vertex v_r to each terminal. Charikar *et al*.'s algorithm is well-known for this problem. It achieves an approximation guarantee of $l(l-1)k^{\frac{1}{l}}$ in $\mathcal{O}(n^l k^{2l})$ time for any fixed level $l > 1$, where l is the level of the tree produced by the algorithm, n is the number of vertices, $|V|$, and k is the number of terminals, $|X|$. However, it requires a great amount of computing power, and there are some problems in the proof of the approximation guarantee of the algorithm. This paper provides a faster approximation algorithm improving Charikar *et al*.'s DSP algorithm with a better time complexity, $\mathcal{O}(n^l k^l + n^2 k + nm)$, where m is the number of edges, and an amended $\sqrt{8k} - \delta \ln k$ factor for the 2-level Steiner tree, where $\delta = \sqrt{6} - 2 \doteq 0.4494$.

Keywords: directed Steiner tree, approximation algorithm, multicast distribution tree, content delivery network, multicast routing

1. INTRODUCTION

The Steiner tree problem in graphs asks for a minimum-cost tree, which spans a given set of vertices X in a graph G [9]. This problem has been addressed for different types of graphs: usual networks with edge costs (NSP), node-weighted networks (NWSP) … *etc*. This paper focuses on the Directed Steiner Tree Problem (DSP), which has various practical applications in network design and multicast routing.

The Steiner tree problem is NP-Complete even if the graph is induced by points in the plane [7], and is MAX SNP-hard [10] for general graphs. The directed version of the Steiner tree problem has been well-studied and there are algorithms achieve a $l(l-1)k^{\frac{1}{l}}$ factor in $\mathcal{O}(n^l k^{2l})$ time [1]. The undirected version of the Steiner tree problem is also researched and some algorithms achieve constant factors [11-14].

In fast growing broadband environments, CDNs (Content Delivery Networks) are designed to deliver web-based multimedia contents to a number of distant mirror sites. Consequently, huge amount of data is required to be shared and copied to several sites over Internet. The majority of links between two routers on Internet are full-duplex and

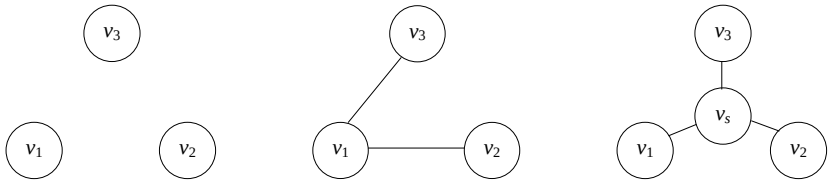
Received September 21, 2004; revised December 10, 2004; accepted January 17, 2005.
Communicated by Gen-Huey Chen.
* This work was supported by Chungshan Institute of Science and Technology under the "Wide Band Mobile Communication System Integration Technology" project, No. 95-EC-17-A-03-R7-02C5 and NSC 95-2524-S-008-001 project.

asymmetric in the upload/download bandwidth [4-6]. A feasible way to select a minimal cost spanning tree for delivering traffic is a critical issue. Charikar *et al.*'s DSP algorithm provides such a solution, but with the cost of a great amount of computing time (power). Furthermore, Charikar *et al.*'s analysis of the algorithm's approximation guarantee is incorrect. The contribution of this paper is to amend Charikar *et al.*'s analysis and provide an improved faster algorithm for the directed Steiner tree problem with better time complexity.

This paper is organized as follows: we will introduce the *l*-restricted Steiner tree, TM algorithm, Charikar *et al.*'s algorithm, and Roos' algorithm in section 2. The fundamental ideas of FasterDSP will be introduced in section 3. The detailed processes of FasterDSP will be illustrated in section 4. The experiment results are discussed in section 5. Finally in section 6, we conclude FasterDSP offers a pragmatic approximation algorithm for DSP in $\mathcal{O}(n^l k^l + n^2 k + nm)$ time with $\sqrt{8k} - \delta \ln k$.

2. RELATED WORK

The Steiner tree problem is a classic combinatorial optimization problem in a graph. The Steiner tree problem originates from *Euclidean Steiner problem* (*ESP*: Fig. 1) that is proposed by Fermat (1601-1665): The problem is to find a point in a plane and the sum of its distances from three given points is minimal.



(a) Three points. (b) Minimum spanning tree. (c) With a Steiner vertex.

Fig. 1. Euclidean Steiner problem.

Definition 1 Steiner vertex: In the Steiner tree, a vertex that is neither a root vertex nor a terminal is called a *Steiner vertex.*

Introducing Steiner vertices can be beneficial to reduce the cost of the extended spanning tree. The Steiner tree problem focuses on discovering Steiner vertices. Generally speaking, a vertex can be assigned with a weight that represents certain properties, such as the number of obtained terminals. An approximation algorithm for DSP selects the minimal weight vertex and related vertices/tree each turn and repeats until enough terminals are obtained. TM algorithm [13], Roos algorithm [3], and Charikar *et al.*'s algorithm [1] all have this general overall procedure.

2.1 TM Algorithm

TM algorithm [13] and KMB algorithm [11] are well-known for the Steiner tree

problem. In an undirected graph, TM algorithm achieves an approximation guarantee of factor 2. However, TM algorithm only achieves an approximation guarantee of $\mathcal{O}(k)$ in a directed graph, where k is number of terminals. As TM algorithm starts, it sets the selected tree as the root vertex. Then, it chooses a terminal that is nearest to the selected tree and the selected tree extends a branch to the terminal.[1] TM algorithm repeats those extension approximation steps until all terminals are covered and outputs the selected tree. TM algorithm generates an acceptable factor tree in most cases. However, it may derive a k factor tree in the worst case.

2.2 *l*-Restricted Steiner Tree

In 1997AD, Zelikovsky provides a proof that there exists a l-restricted Steiner tree which achieves an approximation guarantee of factor $k^{\frac{1}{l}}$, where a l-restricted tree is a tree with every leaf node being at most l edges away from the root vertex. Lemma 1 is what Zelikovsky [2] provides:

Lemma 1 For all $l \geq 1$, $l \in N$, there exists a l-restricted Steiner tree that achieves an approximation guarantee of factor $k^{\frac{1}{l}}$ to the optimal Steiner tree.

This proof has been adopted for most recent DSP algorithms [1, 3, 15]. However, there exists a flaw in this proof. Considering Fig. 2, that shows an optimal Steiner tree with 8 terminals. Besides the tree in Fig. 2, the graph also contains edges from each vertex to its every child with the nearest cost to the path cost on the optimal Steiner tree. In Zelikovsky's proof, such graph must exist a $\lfloor 8^{\frac{1}{2}} \rfloor = 2$ factor 2-level restricted Steiner tree which actually does not exist. This is because the edge set E^l may not be enough to construct paths from the root vertex to each terminal.

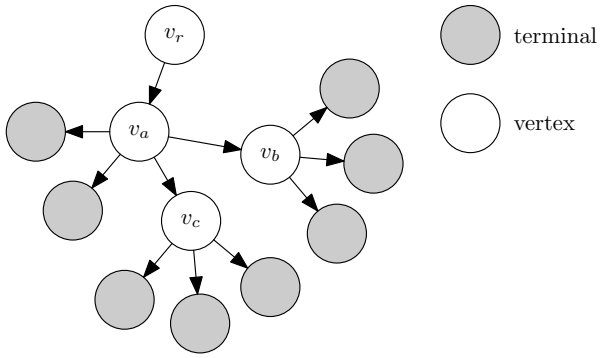
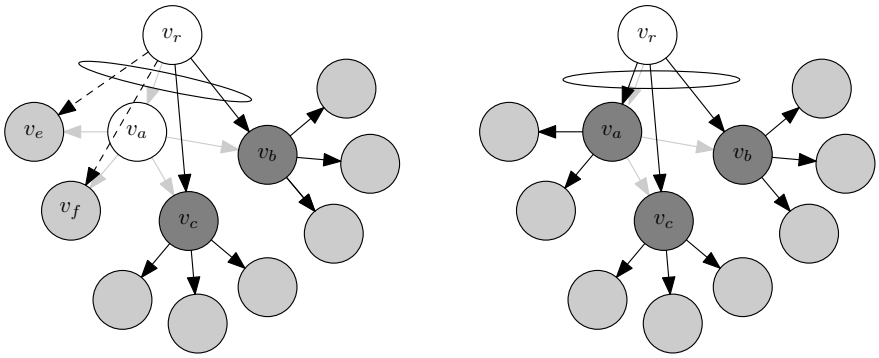


Fig. 2. A counterexample for a l-restricted Steiner tree.

Fig. 3 shows two examples of the 2-restricted Steiner tree based on Fig. 2. The light-gray edges are unselected edges of the optimal Steiner tree in such an example. Since all terminals are leaves in the graph, only vertices v_a, v_b and v_c are *level*-1 *Steiner*

[1] From another point of view, the weight function of TM algorithm returns the distance from the selected tree to this terminal or unavailable if the vertex is not a terminal.



(a) Vertex v_a is level-1 Steiner vertex. (b) Vertices, $v_{a\text{-}c}$ are level-1 Steiner vertex.

Fig. 3. Two examples for 2-restricted Steiner tree.

candidates. In Fig. 3 (a), we selected the vertices v_b and v_c using the rule V_i^l. Since $\kappa(v_a) > k^{\frac{1}{2}}$ and $\kappa(v_b) > k^{\frac{1}{2}}$, vertex v_a will not be selected. Then, we construct the edge set E^l. Since vertices v_e and v_f are terminals and there is no level-1 Steiner vertex between them and the root vertex, the dashed edges of Fig. 3 (a) will not appear in the edge set E^l. Zelikovsky's l-restricted Steiner tree may get a wrong approximation guarantee in such a case. Instead, if we add the dashed edges into the l-restricted Steiner tree, the approximation guarantee will be 4, not 2. If vertex v_b or vertex v_c is not a level-1 Steiner vertex, the approximation guarantee of a 2-restricted level Steiner tree will be 3 at least. In Fig. 3 (b), we show the cases which all Steiner candidates are selected. However, the resulting approximation guarantee of the 2-restricted level Steiner tree is still 3. In suffices to show that in all possible cases of this example, there is no 2-restricted Steiner tree with a 2 factor approximation guarantee.

Thus, the upper bound of Zelikovsky's approximation guarantee is not correct. In the next section, we will derive the proper approximation guarantee for the l-restricted Steiner tree.

2.3 Charikar *et al.*'s Algorithm

Charikar *et al.* brought up a non-trivial approximation algorithm for DSP. This algorithm achieves a $l(l-1)k^{\frac{1}{l}}$ factor based on Zelikovsky's l-restricted Steiner tree.

For a l-level Steiner tree, Charikar *et al.*'s algorithm gives a weight of each vertex based on the density of $(l-1)$-level Steiner tree which is rooted at the vertex with i terminals, where $1 \le i \le k$. Then, the algorithm selects the minimal weight vertex as a level-l Steiner vertex and its subtree.[2] This algorithm repeats the selections until enough terminals are obtained and consequently algorithm will return the desired tree.

In Charikar *et al.*'s paper [1], this algorithm has been proved that it achieves $l(l-1)k^{\frac{1}{l}}$ based on the l-restricted Steiner tree with $\mathcal{O}(n^l k^{2l})$ time. However, the approximation guarantee must be re-evaluated to amend the flaw in Zelikovsky's analysis. On the other hand, Charikar *et al.*'s algorithm could be modified to reduce the time complexity. In the next section, we will address those problems.

[2] When level is 1, the tree will be produced by TM algorithm.

2.4 Roos' Algorithm

Roos launched a faster approximation algorithm for DSP with a $\mathcal{O}(\sqrt{k})$ factor [3]. Roos' algorithm is similar to TM algorithm. Instead of choosing a terminal nearest to the selected tree, Roos algorithm chooses nodes based on assigned weight for each vertex. The weight of a vertex is assigned by the density of the subtree which is rooted at the vertex and extended the shortest path to each available terminals. Roos' algorithm selects the minimal weight vertex and a terminal nearest to this vertex.

However, there is a problem in Roos' analysis. Consider a graph like Fig. 4 with one root vertex, v_r, one vertex, v_s, and $n + 1$ terminals, v_1, …, v_{n+1}. There are cost-1 edges from the root vertex to each terminal and vertex v_s. ε-cost edges are from vertex v_s to terminals, v_1, …, v_n. Additionally, a $n + \varepsilon$-cost edge is from vertex v_s to terminal v_{n+1}.

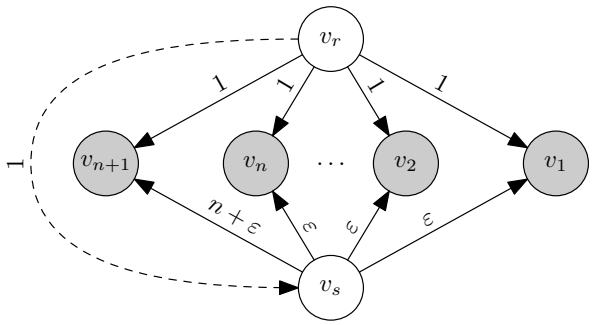


Fig. 4. Tight example for Roos' algorithm.

Let $wt(v_i)$ denote the weight of vertex v_i. When Roos' algorithm starts, $wt(v_i) = 1$, $\forall 1 \le i \le n + 1$ and $wt(v_s) = 1 + \varepsilon$. Therefore, vertex v_1 could be selected and to the next round, where vertex v_2 will be selected in the second round … *etc*. Eventually, Roos' algorithm selects a tree with k one-cost edges. The optimal Steiner tree consists of an edge from vertex v_r to vertex v_s, ε-cost edges, and an edge from vertex v_r to vertex v_{n+1}. In this tight example, Roos' algorithm receives a $\frac{n+1}{2+n\varepsilon} = \mathcal{O}(k)$, not a $\mathcal{O}(\sqrt{k})$ approximation factor.

The miscalculating occurs because the weight function of Roos' algorithm contains a large cost terminal. A modified version will be introduced in section 3.

3. FASTERDSP ALGORITHM

In this section, we will introduce the ideas of FasterDSP and prove the approximation guarantee. Since there are some problems in Zelikovsky's analysis, we will provide an amended proof first. Secondly, we introduce a basic operator of FasterDSP and ideas of FasterDSP. Finally, we proves FasterDSP provides an approximation guarantee of factor $\sqrt{8k} - \delta \ln k$ when $l = 2$.

3.1 Approximation Guarantee of the *l*-Restricted Steiner Tree

We will first define the *uncovered vertex*. Followed by Lemma 2 that defines the approximation guarantee of a *l*-restricted level Steiner tree. In Lemma 3, we will prove the relationship between the number of terminals and the level *l* with a fixed approximation guarantee α.

Definition 2 Uncovered: given a pair of Steiner vertices/terminal, *u* and *v* of a *l*-restricted Steiner tree. Vertex *u* is called uncovered by vertex *v* if no other Steiner vertices are higher level than vertex *v* in the path from the vertex *u* to the vertex *v*.

Let $\beta(v)$ denote the number of uncovered vertices of vertex *e*, ex: $\beta(v_j) = 3$, $\beta(v_i) = 1$. The value of the function $\beta(v)$ equals the approximation guarantee of the subtree from the parent vertex of vertex *v* to the uncovered vertices in *l*-restricted Steiner tree in the worst case. Since the *l*-restricted Steiner tree could be composed of the subtrees from any Steiner vertex $v \in V^l$ to its uncovered vertices, and the approximation guarantee of the subtrees could be bounded by the function $\max \beta(v)$, $v \in$ subtree, the *l*-restricted Steiner tree problem could be formularized by minimizing the maximum $\beta(v)$, $v \in V^l$.

Consider Fig. 2 that shows an optimal Steiner tree with several terminals. The way to construct a *l*-restricted Steiner tree is to select a subset of Steiner vertices from the optimal Steiner tree, assign levels, and connect those to minimizing the maximum $\beta(v)$. For example, in Fig. 3 (b), there is a 3 factor tree because the maximum $\beta(v)$ is 3.

Lemma 2 For all $l > 1$, there exists a *l*-restricted level Steiner tree that achieves $\alpha = \arg\min(k < H_\alpha^{l+1})$ approximation guarantee to the optimal Steiner tree, where $H_m^n = \binom{n+m-1}{m}$ denotes repetitions of permutations and combinations and $l, \alpha, \in N$.

Lemma 2 addresses a capacity problem for terminals of a *l*-restricted Steiner tree. In other words, the capacity of a α factor *l*-restricted level Steiner tree is $H_\alpha^{l+1} - 1$ at least. It means that if an optimal Steiner tree contains $H_\alpha^{l+1} - 1$ or less terminals, there is a *l*-restricted level Steiner tree could achieve an approximation guarantee of α factor.

Before the selection the Steiner vertices, the Steiner tree must be modified so that all terminals must be leaves. When a terminal is an internal node, the modification happens. We can extend a branch with zero cost edge at the terminal and move the terminal into the leaf.

Assuming the approximation guarantee α and the optimal Steiner tree are known, we will choose Steiner vertices by the postfix order of the optimal Steiner tree. For a vertex $v \in T^{opt}$, if $\beta(v)$ is no more than α, vertex *v* will not be a new Steiner vertex. Otherwise, vertex *v* will be a new level-*i* Steiner vertex with greater enough *i* to reduce $\beta(v)$ to less or equal α. If this condition is satisfied for each Steiner vertex of T^{opt}, the maximum $\beta(v)$ will be not more than α. Therefore, the tree composed of those selected Steiner vertices achieves α approximation guarantee.

Let $R = \{r_1, r_2, \ldots, r_l, r_{l+1}\}$ represent the set of the numbers of uncovered vertices for vertex *v*, where r_l denotes the number of uncovered terminals, $r_1, r_2, \ldots$ and r_{l-1} denote the number of uncovered corresponding level-*l* to level-$(l-1)$ Steiner vertices, and $r_{l+1} = \alpha -$

n_l, where $n_i = \sum_{j=1}^{i} r_j$ denotes the number of non-lower level Steiner vertices/terminals. Thus, Lemma 2 can be proved by discussing the capacity of vertex v with its uncovered set R, where v is anyone vertex of the optimal Steiner tree.

Lemma 3 For a vertex v with its uncovered set R, the number of terminals in the subtree of the vertex v will not be smaller than $k(R) = \sum_{i=1}^{l} \sum_{j=1}^{r_i} H_{\alpha-n_i+j}^{l+1-i}$. (On other words, the capacity of vertex v is $k(R)$.)

Definition 3 Let $wt(n_i, r_i, i) = \sum_{j=1}^{r_i} H_{\alpha-n_i+j}^{l+1-i}$ denote the capacity of level-i uncovered vertices. In other hand, $k(R) = \sum_{i=1}^{l} wt(n_i, r_i, i)$.

Proof: By definition, we have $wt(n_i, r_i, i) \le \sum_{m=1}^{b} wt(n_i^m, r_i^m, i)$ if $n_i - r_i \ge n_i^m - r_i^m$, $\forall 1 \le m \le b$ and $\sum_{m=1}^{b} r_i^m \ge r_i$, where $b \in N$.

The proof is by induction. When $k(R) \le \alpha$, no Steiner vertex in the optimal Steiner tree were selected as the new Steiner vertex in l-restricted Steiner tree. Thus, the uncovered set can be written as $R = \{0, 0, \ldots, r_i, \alpha - r_i\}$, where r_i is the number of terminals in the subtree. Additionally, $wt(a, b, l) = b$, $\forall a, b \in N$, the lemma holds for $k(R) = wt(n_i, r_i, l) = r_i \le \alpha$. We now assume that the lemma holds for all $k(R^j)$ less than $k(R^i)$.

Consider a vertex v in the optimal Steiner tree with $b \ge 2$ branches, the uncovered sets for its sons are represented as $R^1, R^2, \ldots, R^b$. If the number of uncovered vertices for the vertex v, $\sum_{j=1}^{b} n_l^j$, is less or equal than α, vertex v will not be selected as Steiner vertex of T^l. Otherwise, vertex v will be selected as a new Steiner vertex to reduce the number of the uncovered vertices to α or smaller.

Case 1. No new Steiner vertex: When no new Steiner vertex is selected, the uncovered set R could be written as $r_i = \sum_{j=1}^{b} r_i^j$, $\forall 1 \le i \le l$ and $r_{l+1} = \alpha - n_l$. The inequality $wt(n_i, r_i, i) \le \sum_{m=1}^{b} wt(n_i^m, r_i^m, i)$, is satisfied since $n_i - r_i = n_{i-1} \ge n_{i-1}^m = n_i^m - r_i^m$, $r_i = \sum_{j=1}^{b} r_i^j$, $\forall 1 \le i \le l$. Thus, the lemma holds for this case.

Case 2. A new Steiner vertex: When a new Steiner vertex is selected and the new Steiner vertex is level-q, the uncovered set R can be written as

$$r_i = \sum_{j=1}^{b} r_i^j, \ \forall 1 \le i < q, \tag{4}$$

$$r_q = 1 + \sum_{j=1}^{b} r_i^j, \tag{5}$$

$r_i = 0$, $\forall q < i \le l$ and $r_{l+1} = \alpha - n_l$. In this case, Eq. (4) can be proved like we did in case 1. Since $r_q = 1 + \sum_{j=1}^{b} r_i^j$, we have $wt(n_q, r_q, q) - wt(n_q, 1, q) \le \sum_{m=1}^{b} wt(n_q^m, r_q^m, q)$. Thus, we can prove that $wt(n_q, 1, q) \le \sum_{i=q+1}^{l} \sum_{m=1}^{b} wt(n_i^m, r_i^m, i)$. Additional, if

$\sum_{m=1}^{b} r_{q+1}^{m} < \alpha - \sum_{m=1}^{b} n_q^m = \alpha - n_q + 1$ is satisfied, a new Steiner vertex with level-q + 1 can reduce uncovered vertices to an acceptable capacity. Therefore, $\sum_{m=1}^{b} r_{q+1}^{m} \geq \alpha - n_q + 1$.

Case 2.1. A new Steiner vertex caused by level-q + 1, $\sum_{m=1}^{b} r_{q+1}^{m} > \alpha - n_q + 1$: If $\sum_{m=1}^{b} r_{q+1}^{m} > \alpha - n_q + 1$, it can be showed

$$wt(n_q, 1, q) = wt(\alpha, \alpha - n_q + 1, q + 1) + 1$$
$$\leq \sum_{m=1}^{b} wt(n_{q+1}^{m}, r_{q+1}^{m}, q + 1)$$

because $n_{q+1}^{m} - r_{q+1}^{m} = n_q^m \leq n_q - 1 = \alpha - (\alpha - n_q + 1)$ and $\sum_{m=1}^{b} r_{q+1}^{m} > \alpha - n_q + 1$. Thus, the lemma holds for this case.

Case 2.2. A new Steiner vertex caused by level > q + 1, $\sum_{m=1}^{b} r_{q+1}^{m} = \alpha - n_q + 1$: Assume the position s, where $\sum_{m=1}^{b} r_s^m > 0$ and $q + 1 < s \leq l$.[3] It can be showed

$$wt(n_q, 1, q) = wt(\alpha, \alpha - n_q + 1, q + 1) + 1$$
$$\leq \sum_{m=1}^{b} wt(n_{q+1}^{m}, r_{q+1}^{m}, q + 1) + \sum_{m=1}^{b} wt(n_s^m, r_s^m, s)$$

because $n_{q+1}^{m} - r_{q+1}^{m} \leq n_q - 1$, $\sum_{m=1}^{b} r_{q+1}^{m} = \alpha - n_q + 1$, and $\sum_{m=1}^{b} wt(n_s^m, r_s^m, s) \geq 1$. Thus, the lemma holds for this case.

Since the lemma holds for any $k(R)$, this completes the proof of the inductive hypothesis. ❑

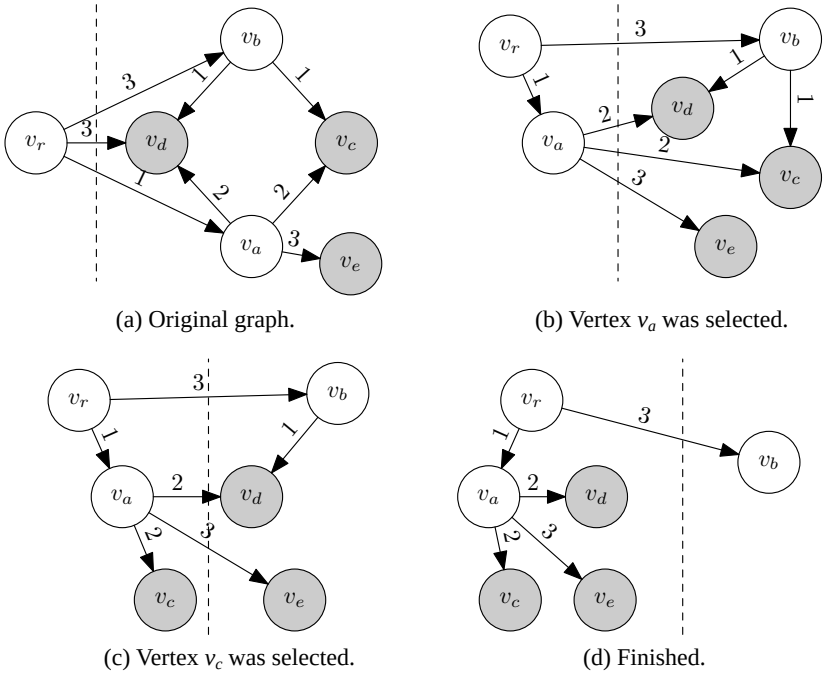
From another point of view, if the uncovered set R of the root vertex is $r_1 = \alpha$ and $r_i = 0$, $\forall 1 < i \leq l + 1$, this l-level Steiner tree has $k(R) = wt(\alpha, \alpha, 1) = H_{\alpha}^{l+1} - 1$ at least, and leads to Lemma 2. ❑

3.2 Select Operator and Special Graph *H*

Fig. 5 shows the basic "*select operator*" and the graph for FasterDSP. Since an approximation algorithm always estimates the weights of vertices and extends a branch from the selected tree, we can design a specific graph called H to reduce the complexity of "operators" in a graph. The graph H can be divided into two parts the selected tree and the unselected vertices. A dashed line separates the H into the part of the selected tree (left) and the part of unselected vertices (right), as showed in Fig. 5. Among the edges according the dashed line, the minimum cost one decides which vertex will be selected to join the left part.

Fig. 5 (a) demonstrates the procedure that root vertex was selected in the initial state. Fig. 5 (b) shows vertex v_a was selected. Since vertex v_a was selected, there are two edges from the selected tree to vertex v_d, the higher cost edge from vertex v_r to vertex v_d will be removed. Fig. 5 (c) shows vertex v_c was selected. Because the edge from vertex v_b to

[3] If there is no position s to make $\sum_{m=1}^{b} r_s^m > 0$, no new Steiner vertex is required.



(a) Original graph. (b) Vertex v_a was selected.

(c) Vertex v_c was selected. (d) Finished.

Fig. 5. Select operator and special graph.

vertex v_c is a edge from an unselected vertex to the selected tree, the edge will be removed. Finally, Fig. 5 (d) shows all terminals are selected. Since there is no unselected terminal, the selected tree is a spanning tree for root vertex and those terminals.

We have the following notations for further discussion, let $\kappa(H)$ denote the number of unselected terminals of graph, H, $X(H)$ denote unselected terminals of graph, H, $V(H)$ denote unselected vertices of graph, H, $H \rightsquigarrow v$ denote select operator that returns a new graph that the vertices in the shortest path from the selected tree to vertex v in H have been selected. Those functions will be used in the next section.

3.3 Algorithm to Construct a FasterDSP Tree

FasterDSP algorithm is based on Charikar *et al.*' algorithm. However, FasterDSP reduces the computing power in duplicate cases. Let $A_c(H, l, k)$ denote Charikar *et al.*'s algorithm. It returns a l-level Steiner tree with k terminals generated by this algorithm. In Fig. 6, we show duplicate cases in this algorithm.

Since Charikar *et al.*'s algorithm will choose a vertex with the assigned number of terminals, vertex v may be chosen twice with the different numbers of terminals, k and k – 1. Then, for building the subtree of v with the number of terminals, Charikar *et al.*'s algorithm will choose another vertex v' and assign the number of terminals with vertex v'. Fig. 6 shows the duplicate cases. To reduce those duplicate cases, FasterDSP requires some functions to build k minimal tree with 1, 2, …, k terminals. We will show that reduces those duplicate cases and improves the time complexity in the next section.

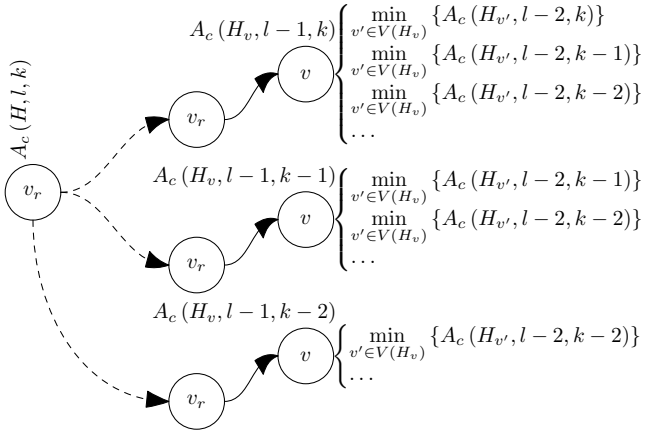


Fig. 6. Duplicate cases in Charikar *et al.*'s algorithm.

In FasterDSP, we apply TM algorithm to build the first level Steiner tree. And building the second level Steiner tree will utilize a modified Roos' algorithm. In fact, we modify the policy for building the subtree used by the weight function of Roos' algorithm. The new policy first extends the branch from the selected tree to Steiner candidate, then calculates its density. A terminal is selected only if (1) it is nearest the selected tree, and (2) the density of the new subtree will decrease. The way of building the third or higher level Steiner tree follows modified Charikar *et al.*'s algorithm.

3.4 Approximation Guarantee for 2-Level Steiner Tree of FasterDSP

In this subsection, we show the approximation guarantee for 2-level Steiner tree of FasterDSP. First, from Lemma 2, we can show that:

Lemma 4 When $l = 2$, there exists a 2-level Steiner tree that achieves a $\left\lceil \frac{\sqrt{8k+9}-3}{2} \right\rceil$ factor to the optimal Steiner tree.

Since there is a 2-level Steiner tree that achieves a $\left\lceil \frac{\sqrt{8k+9}-3}{2} \right\rceil$ factor, this tree also achieves a $\left\lceil \frac{\sqrt{8k+9}-3}{2} \right\rceil \leq \sqrt{2k} - \delta, \ \forall k \in N$ factor, where $\delta = \sqrt{6} - 2 \doteq 0.4494$.

Since the optimal 2-level Steiner tree achieves $\sqrt{2k} - \delta$ factor and the average weight of the selected vertices in each iteration is smaller than the weight of the best vertex in current optimal 2-level Steiner tree, the cost of the 2-level Steiner tree of FasterDSP will be smaller than:

$$\sum_{i=1}^{k} \frac{\sqrt{2i}-\delta}{i} = \sum_{i=1}^{k} \frac{\sqrt{2i}}{i} - \sum_{i=1}^{k} \frac{\delta}{i} < \int_0^k \frac{\sqrt{2y}}{y} dy - \sum_{i=1}^{k} \frac{\delta}{i} < \sqrt{8k} - \delta \ln k. \quad \square$$

Thus, the 2-level Steiner tree of FasterDSP achieves a $\sqrt{8k} - \delta \ln k$ factor to the optimal Steiner tree.

4. PROCESSING IN DETAILS

Since root vertex and a terminal set are given, the focus of the DSP algorithm is how to select Steiner vertices. In this section, we illustrate the detailed process of FasterDSP and prove its time complexity and its space complexity.

Following section 3.2, where we have illustrated how to represent the state of the graph after some vertices were selected.

Definition 4 Density(H_i): let $c(H_i)$ denote the sum of the edge costs in the selected tree of the graph H_i. The average density for each selected terminal in the graph H_i can be defined as $\frac{c(H_i)}{k-\kappa(H_i)}$.

In Definition 4, we defined the average density (cost) for each selected terminal in the graph H_i. FasterDSP must estimate the density (weight) of a Steiner vertex to select the minimal Steiner vertex. If the graph H is an ancestry of the graph H_i, the average density for additional selected terminals in the graph H_i can be defined as:

Definition 5 Density(H_i, H): the density of additional selected terminals in the graph H_i is defined as $\frac{c(H_i)-c(H)}{\kappa(H)-\kappa(H_i)}$.

Since the cost of additional selected terminals can be estimated, the cost for a Steiner vertex can be defined as the (average) density of the terminals in all descendants of the Steiner vertices.

We will illustrate the FasterDSP pseudo code that performs the ideas discussed in section 3.3. First, we illustrate the pseudo code to build level-($l - i$) Steiner vertices when the ($i - 1$)-level Steiner forest are available. Then, we illustrate the pseudo code to build the i-level Steiner forest once the weight of the level-($l - i - 1$) Steiner vertices are available. Those pseudo codes can build the i-level Steiner forest based on the information of the level-($l - i - 1$) Steiner vertices. The 2-level Steiner forest and the 1-level Steiner forest are available from modified Roos' algorithm and TM algorithm. FasterDSP then repeatedly applies those pseudo codes to build Steiner forest with higher enough levels. Since each i-level Steiner forest contains k i-level Steiner trees with 1, …, k terminals, the l-level Steiner tree with k terminals in the l-level Steiner forest is the results of FasterDSP. Next, the procedure, FasterDSPVertices(H, i, k) is used to build the minimal Steiner vertices with the level-($i - 1$) Steiner tree. FasterDSPForest(H, i, k) is used to build the minimal i-level Steiner forest. Since the minimal l-level Steiner tree with k terminals is the result of FasterDSP, FasterDSPForest(H, i, k) could be reduced as the pseudo code, FasterDSPTree(H, l, k), to reduce the time complexity when $i = l$. Additionally, FasterDSP(G, v_r, X, l) is used to convert the graph, root vertex and other parameters into the special graph and perform other preprocesses.

4.1 FasterDSP Vertices Function

The goal of FasterDSPVertices is to select k minimal weight Steiner vertices with

1, …, k terminals as candidates of the next Steiner vertex. Since the weight of a Steiner vertex is the density of the additional selected terminals after the Steiner vertex is selected, this function generates the graphs after each vertex is selected. Then, this function utilizes FasterDSPForest(H_i, $l - 1$, $k - \kappa(H) + \kappa(H_i)$) to generate the subtree of the Steiner vertex. This enables the estimation of the weight for the Steiner vertex.

Table 1. FasterDSP vertices function.

Function FasterDSPVertices(H, l, k)
1. $l \leftarrow \min(k, l)$
2. Let Q_s bet the results returned by FasterDSPForest(H, $l - 1$, k)
3. Let $Q = \{H \rightsquigarrow v \mid v \in V(H)\}$
4. while Q is not empty
Let H_i be the graph in Q with lowest cost and remove it from Q.
Let Q_i be the results returned by FasterDSPForest(H_i, $l - 1$, $k - \kappa(H) + \kappa(H_i)$)
$Q_s \leftarrow \sum \min(Q_{sj}, Q_{ij})$ where Q_{sj} denotes the graph in Q_s with j terminals and so on.
5. return Q_s.

Table 1 shows the procedures to generate the weights for each Steiner candidate. Step 1 normalizes the parameter, l. If $l > k$, there are $l - k$ level useless Steiner vertices. Step 2 generates the weight for a null Steiner candidate or root vertex. The set, Q_s, stores k graphs with 1, …, k extra terminals beyond H. Step 3 generates the graphs after each Steiner candidate is select. Then, step 4 selects a graph from Q, generates the weights for this graph, and merges those weights into Q_s. After the weights for each Steiner candidate are estimated and k minimal Steiner vertices are selected, this function returns the results in the step 5. Notice, in practical situations, step 3 in Table 1 must be modified since the path from the selected tree to the vertex may contain one or more terminals.

4.2 Faster DSP Forest Function

Since FasterDSP vertices function can be used to estimate the weight of a Steiner vertex, FasterDSP forest function could build k minimal l-level Steiner tree with 1, …, k terminals based on such information. As the previous discussions, TM algorithm and modified Roos' algorithm are used to build k 1-level Steiner trees and k 2-level Steiner trees. The focus of FasterDSP forest function is how to select the first level Steiner vertices for graph H. Since FasterDSP vertices function is used to generate the first k minimal Steiner vertices for current graph, a l-level Steiner tree with j terminals will select a minimal weight Steiner vertices with less than j extra terminals. Then those l-level Steiner tree will repeat those steps until enough additional terminals are selected. For example, if $k = 8$ and the densities of k Steiner vertices are 2, 3, 4, 3, 1, 2, 3 and 2, four l-level Steiner trees with 1, …, 4 terminals will select the Steiner vertex with 1 additional terminal, and other l-level Steiner trees will select the Steiner vertex with 4 extra terminals. Since the l-level Steiner trees with 2, 3, 4 terminals will select the same Steiner vertex, the graphs of those are identical after the Steiner vertex is selected. Therefore, the remaining Steiner vertices can be selected using the same function.

Table 2. FasterDSP forest function.

Function FasterDSPForest(H, l, k)
1. $l \leftarrow \min(k, l)$
2. If $k < 1$ then return ϕ
3. If $l < 3$ then return min(RoosForest(H, k), TMForest(H, k))
4. Let Q_s be the results of FasterDSPVertices(H, l, k)
5. $i \leftarrow 0$
6. while $i < k$ do
   Let j be argmin(density(H_j, H)) where density(H_i, H) > density(H_j, H) and $i < j \leq k$
   If $j > i + 1$ then
      $Q_i \leftarrow$ FasterDSPForest(H_i, l, $j - i - 1$)
      $Q_s \leftarrow \sum \min(Q_{sj}, Q_{ij})$ where Q_{sj} denotes the graph in Q_s with j terminals and so on.
   $i \leftarrow j$
7. return Q_s.

In Table 2, the pseudo code shows FasterDSP forest function. Step 1 normalizes the parameter, l. Step 2 checks the parameter, k. This function calls Roos' algorithm and TM algorithm in step 3 if $l < 3$. Step 4 generates the weight for first Steiner vertices. Step 6 partitions k l-level Steiner trees into several groups. The l-level Steiner trees that selects the same Steiner vertex belong to the same group. Then, this function calls itself to generate other Steiner vertices for each group and merges the results.

After applying FasterDSP forest function and FastDSP vertices function to build Steiner vertices at each level, the simplicity version of FasterDSP forest function could be used to build the final Steiner tree with less time complexity. However, due to page limit, we do not show the function here.

4.3 Analysis for Time Complexity and Space Complexity

We now analyze the time complexity and the space complexity for FasterDSP. First, the complexity of select operator is $\mathcal{O}(ni)$ time, where i is the number of new selected vertices in this operation. However, this operator can be done in $\mathcal{O}(n)$ time with additional preprocess data. Since select operator can be done in $\mathcal{O}(n)$ time, TM algorithm can be improved in $\mathcal{O}(nk)$ time.

Lemma 5 The time complexity of FasterDSP is $\mathcal{O}(n^l k^l)$ time without preprocess time, where n is the number of vertices, k is the number of terminals, and l is the level of the tree.

Proof: We will prove this by induction. Since the time complexity of the 1-level Steiner tree of FasterDSP is $\mathcal{O}(nk)$ time, this lemma holds for $l = 1$. We now assume that the lemma holds for all i less than l.

Consider function $f(n, k, l) = \mathcal{O}(n^l k^l)$ that represents the time complexity of a l-restricted Steiner tree with k terminals built by FasterDSP. To generate FasterDSP vertices, it requires no more than n times of FasterDSPForest with $l - 1$ level, $f(n, k, l - 1)$.

After the first Steiner vertices are generated, those l-level Steiner trees will be partitioned into several groups. Let $R = \{r_1, r_2, \ldots, r_c\}$ denote the sizes of those groups. For each group, FasterDSP calls FasterDSPForest to generate others Steiner vertices. Therefore, the time complexity of l-level Steiner tree of FasterDSP could be written as:

$$f(n, k, l) = n(f(n, k, l-1) + kc) + \sum_{r \in R} (f(n, r-1, l) + (r-1)c) = \mathcal{O}(n^l k^l)$$

where c denotes the cost to compare and replace graphs.

This completes the proof of the inductive hypothesis.

Additionally, FasterDSP must build all-pairs shortest path forest and some additional data for a l-level Steiner tree. These tasks can be done in $\mathcal{O}(n^2k + n^2 \log n + nm)$. Therefore, the time complexity of FasterDSP is $\mathcal{O}(n^l k^l + n^2 k + nm)$.

Lemma 6 Space complexity of FasterDSP is $\mathcal{O}(n^2k + nkl)$.

Since $\mathcal{O}(k)$ graphs could only be cached in a level, the number of levels is never more than l, a shortest path forest requires $\mathcal{O}(n^2)$ spaces, and the additional data for 1-level Steiner tree requires $\mathcal{O}(n^2k)$ spaces, the space complexity of FasterDSP is $\mathcal{O}(n^2k + nkl)$.

5. IMPLEMENTATION

To compare the performances of those algorithms, we implemented them using C++ and the graph library of boost. In this section, all results are calculated with FreeBSD 4.8-RELEASE on an IBM eServer x335 with dual Intel Xeon 2.8GHz and 1GB DDR SDRAM.[4]

Because there is no designed testset available in SteinLib, we use testset B of SteinLib[5] to build test cases. Testset B of SteinLib is designed for the undirected graph. We modified testset B by duplicating the edges and assigning different directions with the same cost to fit the requirement of DSP. In addition, the first terminal will be assigned as root vertex. Tables 3 and 4 show the results. Due to page limit, we only show part results here.

In Table 4, the runtime includes preprocess time. The "F l : i" is the i-level Steiner tree of FasterDSP, and the "C l : i" is the i-level Steiner tree of Charikar *et al.*'s algorithm. Since the "C l : 1" should be TM algorithm, we show it as TM. The factor row shows the cost of the tree of the algorithm divides the cost of the optimal Steiner tree.

Since those graphs are not worst cases for DSP, those factors would be better than their guarantees. Besides, their approximation guarantee might not be a tight upper bound; they are possible further improved. In Table 4, TM algorithm is the fastest algorithm in all cases due to no preprocess and simple weight function, thus save 1ms to build the tree in many cases. Although, FasterDSP 1-level Steiner tree is also TM algorithm, it requires some preprocess and spends more time than pure TM algorithm.

[4] The program runs under single thread mode. This means that one-cpu's computing power only can be used in the same time.

[5] http://elib.zib.de/steinlib/showset.php?B.

Table 3. Results of SteinLib testset B.

properties					factor					
name	*n*	*m*	*k*	*opt*	F *l* : 1	F *l* : 2	F *l* : 3	Roos	TM	C *l* : 2
b01	50	126	9	82	1.037	1.000	1.000	1.000	1.037	1.037
b02			13	83	1.048	1.012	1.012	1.012	1.024	1.024
b03			25	138	1.000	1.000	1.000	1.137	1.001	1.000
b04		200	9	59	1.051	1.000	1.000	1.011	1.051	1.051
b05			13	61	1.016	1.015	1.000	1.049	1.016	1.016
b06			25	122	1.041	1.041	1.016	1.139	1.041	1.049
b13	100	250	9	165	1.054	1.054	1.054	1.036	1.054	1.018
b14			13	235	1.013	1.000	1.000	1.123	1.000	1.000
b15			25	318	1.001	1.001	1.000	1.085	1.001	1.001
b16		400	9	127	1.055	1.000	1.000	1.055	1.055	1.031
b17			13	131	1.001	1.000	1.000	1.076	1.001	1.001
b18			25	218	1.018	1.018	1.009	1.266	1.018	1.023

Table 4. Results of SteinLib testset B.

properties					factor					
name	*n*	*m*	*k*	*opt*	F *l* : 1	F *l* : 2	F *l* : 3	Roos	TM	C *l* : 2
b01	50	126	9	82	4ms	4ms	12ms	1ms	< 1ms	211ms
b02			13	83	5ms	6ms	25ms	1ms	< 1ms	461ms
b03			25	138	7ms	9ms	1,869ms	1ms	< 1ms	3,114ms
b04		200	9	59	5ms	5ms	275ms	2ms	< 1ms	452ms
b05			13	61	5ms	6ms	999ms	2ms	< 1ms	1,050ms
b06			25	122	8ms	11ms	9,266ms	3ms	1ms	6,165ms
b13	100	250	9	165	21ms	25ms	111ms	8ms	1ms	4,009ms
b14			13	235	26ms	34ms	6,660ms	9ms	1ms	5,256ms
b15			25	318	43ms	69ms	132,729ms	11ms	3ms	60,882ms
b16		400	9	127	23ms	28ms	149ms	11ms	1ms	6,600ms
b17			13	131	29ms	37ms	21,848ms	12ms	2ms	16,462ms
b18			25	218	48ms	76ms	224,545ms	15ms	5ms	72,869ms

Notice, the sequence to select terminals may be different also complicates the computation. Hence, some results are different in Table 3. In the "F *l* : 2", the algorithm is modified Roos' algorithm. The "F *l* : 2" or Roos are much faster than the "C *l* : 2". However, in some cases b3, b6, b14 and b18, the results of Roos' algorithm is larger than 1.1 factors. Factors of others are never larger than 1.1. This problem has been discussed in related work. The "F *l* : 2" shows that this problem has been solved with a modified weight function. In b18 and b6, an interesting phenomenon shows that the factor of 2-level Steiner tree of Charikar *et al.*'s algorithm is larger than the factor its' 1-level Steiner tree. It is incurred because Charikar *et al.*'s algorithm builds *l*-level Steiner tree by level-1 Steiner vertices. And it means Charikar *et al.*'s algorithm may ignore the better cases found by the lower level tree. Those results do not show that the "C *l* : 3" and the "F *l* : 4" due to limited computing power.

In those results, we show FasterDSP is faster than Charikar *et al.*'s algorithm if memory is large enough to generate and store preprocessed data. It also provides a more stable factor than Roos' algorithm and Charikar *et al.*'s algorithm.

6. CONCLUSION

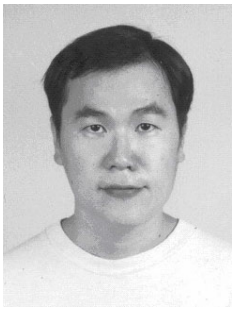
In this paper, we introduce a pragmatic approximation algorithm for the directed Steiner tree problem with a better time complexity and an accurate approximation guarantee analysis. Although the approximation guarantee of FasterDSP has been proved to $\sqrt{8k} - \delta \ln k$, the approximation guarantee of higher level of FasterDSP need to be further analyzed. To improve l-restricted level Steiner tree with better approximation guarantee still demands more efforts, and this continuous pursuing is in our future direction.

REFERENCES

1. M. Charikar, C. Chekuri, T. Cheung, Z. Dai, A. Goel, S. Guha, and M. Li, "Approximation algorithms for directed Steiner problems," *Journal of Algorithms*, Vol. 33, 1999, pp. 73-91.
2. A. Zelikovsky, "A series of approximation algorithms for the acyclic directed Steiner tree problem," *Algorithmica*, Vol. 18, 1997, pp. 99-110.
3. S. Roos, "Scheduling for ReMove and other partially connected architectures," Technical Report, No. 1-68340-44(2001)-05, Laboratory of Computer Engineering, Faculty of Information Technology and Systems, Deflt University of Technology, The Netherlands, 2001.
4. S. Ramanathan, "Multicast tree generation in networks with asymmetric links," *IEEE/ACM Transactions on Networking*, Vol. 4, 1996, pp. 558-568.
5. H. F. Salama, "Evaluation of multicast routing algorithms for real-time communication on high-speed networks," in *Proceedings of the IFIP 6th International Conference on High Performance Networking*, 1995, pp. 27-42.
6. P. Winter, "Steiner problem in networks: a survey," *Networks*, Vol. 17, 1987, pp. 129-167.
7. M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, San Francisco, 1978.
8. T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, 1st ed., The MIT Press, 1990.
9. F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, Amsterdam, 1992.
10. M. Bern and P. Plassman, "The Steiner problems with edge lengths 1 and 2," *Information Processing Letters*, Vol. 32, 1989, pp. 171-176.
11. L. Kou, G. Markowsky, and L. Berman, "A fast algorithm for Steiner trees," *Acta Informatica*, Vol. 15, 1981, pp. 141-145.
12. P. Berman and V. Ramaiyer, "Improved approximation algorithms for the Steiner tree problem," *Journal of Algorithms*, Vol. 17, 1994, pp. 381-408.
13. H. Takahashi and A. Matsuyama, "An approximate solution for the Steiner problem in graphs," *Mathematica Japonica*, Vol. 24, 1980, pp. 573-577.
14. M. Karpinsky and A. Zelikovsky, "New approximation algorithms for the Steiner tree problem," Technical Report No. TR95-030, Electronic Colloquium on Computational Complexity (ECCC), 1995.

15. L. Zosin and S. Khuller, "On directed Steiner trees," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 59-63.

Ming-I Hsieh (謝明益) received his M.S. and B.S degree in Computer Science and Information Engineering from National Central University, Taoyuan, Taiwan, in 2003 and 2001. He is a Ph.D. student of Department of Computer Science and Information Engineering of National Central University, Taoyuan, Taiwan from 2003. His research interests include traffic engineering, quality of service, network security and algorithm.



Eric Hsiao-Kuang Wu (吳曉光) received his B.S. degree in Computer Science and Information engineering from National Taiwan University in 1989. He received his M.S. and Ph.D. in Computer Science from University of California, Los Angeles (UCLA) in 1993 and 1997. He is an Associate Professor of Computer Science and Information Engineering at National Central University, Taiwan. His primary research interests include wireless networks, mobile computing and broadband networks. He is a member of the Institute of Information and Computing Machinery (IICM) and IEEE.

Meng-Feng Tsai (蔡孟峰) received his B.S. and Master degree in Computer Science and Information Engineering from National Taiwan University in 1989 and 1991. He received Ph.D. in Computer Science from University of California, Los Angeles (UCLA) in 2004. He is an Assistant Professor of Computer Science and Information Engineering at National Central University, Taiwan. His primary research interests include data warehouse, distributive aggregation, and data base.