# Light Graphs with Small Routing Cost

**Bang Ye Wu**
*Department of Computer Science and Information Engineering, Shu-Te University, YenChau, KaoShiung, Taiwan 824, Republic of China*

**Kun-Mao Chao**
*Department of Life Science, National Yang-Ming University, Taipei, Taiwan 112, Republic of China*

**Chuan Yi Tang**
*Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, Republic of China*

**Let $G = (\{1, \ldots, n\}, E, w)$ be an undirected graph with nonnegative edge weights $w$ and let $a_{ij}$ be the nonnegative requirement between vertices $i$ and $j$. For any spanning subgraph $H$ of $G$, the weight of $H$ is the total weight of its edges and the routing cost of $H$ is $\sum_{i<j} a_{ij} d_H(i, j)$, where $d_H(i, j)$ is the distance between $i$ and $j$ in $H$. In this paper, we investigated two special cases of the problem of finding a spanning subgraph with small weight and small routing cost. For the case where all the distances in $G$ are 1, we show that the problem is NP-complete, and give a simple approximation algorithm for it. Furthermore, we define some sufficient conditions for the problem to be polynomial-time solvable. For the case where all the requirements are 1, we develop an algorithm for finding a spanning tree with small weight and small routing cost. The algorithm provides trade-offs among tree weights, routing costs, and time complexity. We also extend the results to some other related problems.** © 2002 Wiley Periodicals, Inc.

**Keywords:** approximation algorithms; network design; spanning trees

## 1. INTRODUCTION

Finding spanning subgraphs of a given graph is a classical problem of network design. Typically, we are given a nonnegative edge-weighted graph $G$. The weight on each edge represents the distance and reflects both the cost to install the link (building cost) and the cost to traverse it after the link is installed (routing cost). Let $G = (\{1, \ldots, n\}, E, w)$ be an undirected graph, where $w$ is a nonnegative edge-weight function. The building cost of $G$ is $w(G) = \sum_{e \in E} w(e)$ and the routing cost of $G$ is $c(G) = \sum_{i<j} a_{ij} d_G(i, j)$, where $a_{ij}$ is the requirement between vertices $i$ and $j$ and $d_G(i, j)$ is the length of the shortest path between these two vertices. For the case where all the requirements are 1, the routing cost of a graph is the sum of all distances in the graph, and this problem has been studied under various names in graph theory, for example, *the transmission of a graph* [18].

When considering the building cost only, we are looking for the minimum-weight spanning subgraph. The optimal solution is the *minimum-weight spanning tree* (*MST*) *or forest* of the graph. However, removing any edge from a graph will increase the routing cost unless such an edge can be replaced by a path with the same or smaller length. Therefore, graphs with smaller routing costs might require larger building costs, and we often need to make a trade-off between the two costs.

A number of researchers have studied the problems of finding a spanning tree with minimum-weight or minimum routing cost. Efficient polynomial-time algorithms for the MST have been developed (e.g., see [5,8]). Hu [11] showed how to construct a spanning tree with minimum routing cost for two special cases. When the graph is complete and the distances between every pair of vertices are the same, he gave a polynomial-time algorithm to find the optimal solution. When all the requirements are the same, he gave a sufficient condition for the optimal tree to be a star. The optimal tree for a graph with identical requirements is referred to as a *minimum rout-*

Received April 1999; accepted January 2002

*Correspondence to*: K.-M. Chao; e-mail: kmchao@ym.edu.tw
Contract grant sponsor: National Science Council, Taiwan; contract grant numbers: NSC 90-2213-E-366-005; NSC 89-2213-E-010-011; NSC 89-2213-E-007-068
Contract grant sponsor: Medical Research and Advancement Foundation in Memory of Dr. Chi-Tsou

*ing cost spanning tree* (MRT) or a *shortest total path length spanning tree*. In [9,12], the MRT problem has been shown to be NP-complete, and 2-approximation algorithms were given in [6,19]. Approximation algorithms with better error ratios were developed in [20], and a *polynomial-time approximation scheme* (PTAS) for the MRT problem was presented in [23]. Some extensions of the MRT problem have also been investigated. Given non-negative weights on vertices, the product-requirement (or the sum-requirement) MRT problem assumes that the requirement between two vertices is the product (or the sum, respectively) of their weights. Constant ratio approximation algorithms for these two generalized problems were given in [21], and a PTAS for the product-requirement MRT was shown in [22]. For the more general case that both the distances and the requirements are arbitrary, polylog approximation algorithms have been developed [2,15].

In this paper, we consider the problems of finding spanning subgraphs with small building and routing costs. We shall focus on the following two special cases:

CASE A. For all pairs of vertices $i, j$, the distances $d_G(i, j)$ are all equal to 1, while the requirements $a_{ij}$ are arbitrary. We shall call the optimum graph in this case the *optimum requirement graph* (ORG).

CASE B. The distances are arbitrary, while the requirements between every pair of vertices are all equal to 1. In this case, we shall focus on how to find a spanning *tree* minimizing the two costs.

Let $n$ be the number of vertices, and $k$, the building cost constraint in Case A, that is, the ORG is restricted to graphs with $k$ edges. If $k = n - 1$, the ORG problem is equivalent to the *optimum requirement spanning tree* problem [11], and it can be optimally solved in polynomial time. Here, we consider a more general case where $n - 1 \le k \le n(n-1)/2$ and obtain the following results:

- The NP-completeness of the ORG problem for general $k$.
- Some sufficient conditions for the ORG problem to be polynomial-time solvable. Interestingly, they include the product-requirement and sum-requirement cases.
- An approximation algorithm with error ratio $1 + (n - 1)/k$. It should be noted that this error ratio is no more than 2.

For the case where all the requirements are the same (i.e., Case B), we present an algorithm for finding a spanning tree $T$ that simultaneously approximates both costs with constant ratios, that is, the weight of $T$ is a constant times that of the MST and the routing cost of $T$ is also a constant times that of the MRT. Such a tree is called a *light approximate routing cost spanning tree* (LART). We extend the results to some related problems:

Although both the MST and the MRT tend to use light edges, a tree with small weight may have a large routing cost and vice versa. For instance, we can easily construct a graph such that the routing cost of its MST is $\Theta(n)$ times the routing cost of its MRT (Fig. 1). Similarly, a spanning tree with a constant times the minimum routing cost may have a tree weight as large as $\Theta(n)$ times the weight of the MST.

Several results for trees realizing trade-offs between weight and some distance requirements have been studied before. For a vertex $r \in V(G)$, an $(\alpha, \beta)$-LAST (*light approximate shortest-path tree*) rooted at $r$ is a spanning tree $T$ of $G$ with $d_T(r, v) \le \alpha d_G(r, v)$ for all $v \in V(G)$ and $w(T) \le \beta w(MST(G))$. In [13], Khuller et al. showed that it is possible to construct an $(\alpha, 1 + 2/(\alpha - 1))$-LAST for any $\alpha > 1$. The NP-completeness of finding the minimum diameter subgraph with a budget constraint was established in [17], while a polynomial-time algorithm for finding the minimum diameter spanning tree of a graph with arbitrary edge weights was given in [10]. Considerable work has been done on the *spanner* of a graph. In general, a $t$-spanner of $G$ is a low-weight subgraph of $G$ such that, for any two vertices, the distance within the spanner is at most $t$ times the distance in $G$. Some results of finding spanners of a weighted graph can be found in [1]. Obviously, the spanners can be used to approximate the minimum routing cost subgraph with arbitrary requirements. But since the criteria for a spanner

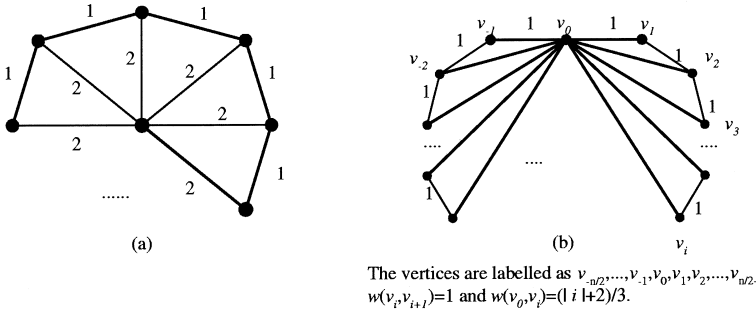


FIG. 1. (a) A simple example of an MST with large routing cost: The routing cost of the MST is $\Theta(n)$ times that of the star. (b) A small routing cost tree with large weight: The routing cost of the star is about one-half that of the MST (a path in this example), while the weight of the star is $\Theta(n)$ times that of the MST.

are much stricter, we may often obtain better results if we wish to minimize the routing cost only.

The remaining sections are organized as follows: Some basic assumptions are given in Section 2. The results for the ORG problem and the LART problem are presented in Sections 3 and 4, respectively.

## 2. PRELIMINARIES

In this paper, a graph is a simple, connected, and undirected graph with a nonnegative weight on each edge. The requirements are also assumed to be nonnegative. Let $G = (V, E)$ be a graph and $w(e)$ denote the weight on edge $e \in E$. Let $n$ be the number of vertices of $G$. The distance $d_G(u, v)$ between vertices $u$ and $v$ in $G$ is the minimum weight of any path in $G$ between them. The weight of graph $G$ is the total weight of all edges in $G$. The distance $d_G(v, U)$ between a vertex $v$ and a vertex set $U$ is the minimum distance between $v$ and any vertex in $U$. We use $V(G)$ to denote the vertex set of a graph $G$. For a subgraph $S$ of $G$, the distance $d_G(v, S)$ means $d_G(v, V(S))$.

**Definition 1.** Let $G = (\{1, \ldots, n\}, E, w)$ be a graph and $a_{ij}$ be the requirement between vertices $i$ and $j$. The routing cost of a graph $G$ is defined as $c(G) = \sum_{i<j} a_{ij} d_G(i, j)$.

For convenience, the input requirements are assumed to be symmetric, that is, $a_{ij} = a_{ji}$ for all $i$ and $j$. Asymmetric requirements $a_{ij}$ can be easily transformed to symmetric requirements $b_{ij}$ by setting $b_{ij} = b_{ji} = a_{ij} + a_{ji}$. Since we consider only undirected graphs, the routing cost $\sum_{ij} a_{ij} d_G(i, j)$ is the same as $\sum_{i<j} b_{ij} d_G(i, j)$. The MST and the MRT of $G$ are denoted by $MST(G)$ and $MRT(G)$, respectively. A spanning tree $T$ of $G$ is a shortest-path tree rooted at $r$ if $d_T(r, v) = d_G(r, v)$ for all $v \in V(G)$.

## 3. THE ORG PROBLEM

In this section, we shall present some results for the ORG problem. If the vertex set can be partitioned into two subsets such that there is no positive requirement crossing the two subsets, we may solve the problem for each subset individually. Here, we assume that there is at least one positive requirement for any cut.

**Definition 2.** Given are a set of nonnegative requirements $A = \{a_{ij} | 1 \le i < j \le n\}$ and an integer $k$, where $n - 1 \le k \le n(n-1)/2$. The *ORG* is a graph $G = (\{1, \ldots, n\}, E)$ such that the routing cost of $G$ is minimum among all graphs with $k$ edges.

**Definition 3.** Let $G = (V, E)$ be a connected graph. A subgraph $H = (V, F), F \subset E$, of $G$ is a *2-spanner* of $G$ if $d_H(i, j) \le 2 d_G(i, j)$ for all vertices $i$ and $j$.

**Definition 4.** Given a graph $G = (V, E)$, the minimum 2-spanner problem is to find the 2-spanner with a minimum number of edges. Given an integer $k \ge n - 1$, the decision version of the minimum 2-spanner problem asks if there exists a 2-spanner with no more than $k$ edges. The decision problem is referred to as the *2-spanner problem* in this paper.

**Theorem 1.** The ORG problem is NP-complete even for the case that all requirements are either 1 or 0.

**Proof.** We transform the 2-spanner problem to the ORG problem. Given an instance of the 2-spanner problem of the graph $G = (V, E)$ and an integer $k$, we construct an instance of the ORG problem as follows: For any edge $(i, j) \in E$, we set the requirement $a_{ij} = 1$ and $a_{ij} = 0$ otherwise. The edge constraint of the ORG problem is set to $k$. We claim that the ORG problem has a solution with routing cost at most $2a^* - k$ if and only if there is a 2-spanner of $G$ with $k$ edges, where $a^* = \sum_{i<j} a_{ij}$ is the total requirement.

Suppose that $H = (V, F)$ is a 2-spanner of $G$ and $|F| = k$. Consider $H$ as a solution of the ORG problem. Every edge in $H$ corresponds to one requirement. There are $k$ requirements directly routed by one edge. Since $H$ is a 2-spanner, the other requirements are routed by two edges. Consequently, the routing cost of $H$ is $k + 2(a^* - k) = 2a^* - k$. Conversely, suppose that the ORG problem has a solution $H$ with routing cost $2a^* - k$. For any $k$-edge graph, there are at most $k$ requirements routed by only one edge and the others need at least two edges. The lower bound of the routing cost of a $k$-edge subgraph is $2a^* - k$, and such a routing cost is achieved only when $k$ requirements are routed by one edge and all others by two edges. Therefore, every edge in $H$ is also an edge in $G$ and $d_H(i, j) = 2$ for any edge $(i, j)$ in $G$ but not in $H$. It also implies that $H$ is a 2-spanner of $G$. Since the ORG problem is obviously in NP and the 2-spanner problem is NP-complete [4,16], by the above reduction, the ORG problem is NP-complete. ∎

In the following, we shall define some sufficient conditions for the ORG problem to be polynomial-time solvable:

**Definition 5.** For a set of requirements $\{a_{ij}\}$, a vertex $m$ is a heavy vertex if $a_{im} \ge a_{ij}$ for all vertices $i$ and $j$.

A heavy vertex does not necessarily exist and may not be unique. For an extreme case where all requirements are identical, each of the vertices is a heavy vertex. The next theorem shows that the ORG can be easily constructed if the input contains a heavy vertex.

**Theorem 2.** The optimal requirement graph problem can be solved optimally in $O(n^2)$ time if the input contains a heavy vertex.

**Proof.** Without loss of generality, we assume that vertex $n$ is the heavy vertex. Let $Y = \{a_{in} | 1 \le i < n\}$

and $X$ be the subset of the largest $k-(n-1)$ requirements of the set $\{a_{ij}|1 \le i < j \le n-1\}$. Let $a^*$ be the total requirement and $x$ and $y$ be the total requirement in $X$ and $Y$, respectively. For a graph with $k$ edges, only $k$ requirements may be routed directly and the others will be routed by at least two edges. Since there is at least one edge incident to each vertex and $n$ is the heavy vertex, for any graph $G$ with $k$ edges, the routing cost $c(G) \ge (x+y) + 2(a^* - (x+y))$, and the equality holds when the requirements in $X \cup Y$ are routed directly and others are routed by exactly two edges. It is easy to see that the graph with edge set $\{(i,j)|a_{ij} \in X \cup Y\}$ achieves the lower bound of the routing cost. Therefore, such a graph is optimal and can be constructed in $O(n^2)$ time by a linear time algorithm [3] for finding the largest $k$ elements in a set of $O(n^2)$ numbers. ∎

The next two corollaries can be proved similarly:

**Corollary 3.** Any graph with diameter 2 is an optimal solution of the ORG problem with identical requirements.

**Corollary 4.** Let $A$ be a set of requirements over a vertex set $\{1,\ldots,n\}$ and $A_1$ be the subset of the largest $k$ requirements in $A$. Assume that $H = (\{1,\ldots,n\}, E)$, where $E = \{(i,j)|a_{ij} \in A_1\}$. If the diameter of $H$ is 2, then $H$ is the ORG.

Let $q_i$ be a nonnegative weight on vertex $i$. As defined in [21], requirements are called *product-requirements* (or *sum-requirements*) if $a_{ij} = q_i q_j$ (or $a_{ij} = q_i + q_j$, respectively) for all vertices $i$ and $j$.

**Corollary 5.** The ORG problem with product-requirements or sum-requirements can be solved in $O(n \log n + k)$ time.

**Proof.** Sort the vertices by their weights in nondecreasing order. We have $q_i \le q_{i+1}$ for all $i < n$. The requirement matrix is a sorted matrix, that is, all rows and columns are sorted. Obviously, the vertex $n$ is a heavy vertex. By Theorem 2, all we need to do is to find the largest $k-(n-1)$ numbers in the requirement matrix. Since it is a sorted matrix, the selection can be done in $O(n)$ time [7]. Then, it takes $O(k)$ time to report the selected requirements, and the total time complexity is $O(n \log n + k)$. ∎

We now propose a simple approximation algorithm for the ORG problem with arbitrary requirements. The idea is to treat an arbitrary vertex $m$ as if it was a heavy vertex. We use $(n-1)$ edges to connect all other vertices to $m$. The remaining edges are used to connect the vertex pairs, on which the requirement is one of the largest $(k-n+1)$ requirements. Since the diameter of the constructed graph is two, the cost is within double the optimal routing cost. The next theorem gives us a slightly better result for large $k$.

**Theorem 6.** The ORG problem with arbitrary requirements can be approximated with error ratio $1+(n-1)/k$ in $O(n^2)$ time.

**Proof.** Let $A_1$ be the summation of the largest $k$ requirements and the remaining total requirement be $A_2$. Obviously, the routing cost of the optimal graph is no less than $A_1 + 2A_2$. Let $A_3$ be the summation of the largest $k-(n-1)$ requirements. Since the diameter of the constructed graph is two and the largest $k-(n-1)$ requirements are routed directly, its routing cost is no more than $A_3 + 2(A_1 - A_3 + A_2)$. Furthermore, we know that $A_3 \ge ((k-n+1)/k)A_1$. Therefore, the relative error ratio is $1+(n-1)/k$. The time complexity is dominated by the time to select the largest $k-(n-1)$ requirements. By a linear time algorithm for the selection problem, the time complexity is $O(n^2)$ since there are $O(n^2)$ requirements. ∎

Practically speaking, a carefully chosen vertex $m$ might deliver a better solution. However, we did not find a proof that the error ratio is asymptotically better.

## 4. THE LART PROBLEM

### 4.1. Overview

In this section, the routing requirements are all equal to one, while the distances are arbitrary. We define a *light approximate routing cost spanning tree* (LART) of $G$ as follows:

**Definition 6.** For $\alpha \ge 1$ and $\beta \ge 1$, an $(\alpha,\beta)$-LART is a spanning tree $T$ of $G$ with $c(T) \le \alpha \times c(MRT(G))$ and $w(T) \le \beta \times w(MST(G))$.

Due to Wong's work [19], there exists a vertex such that the routing cost of any shortest-path tree rooted at that vertex is no more than twice that of the whole graph. If the input graph $G$ is a general graph and each edge has weight 1, we can easily find a shortest-path tree such that its weight is $n-1$ and its routing cost is no more than twice $c(G)$. We shall focus on the case where the input distances are arbitrary:

**Definition 7.** A *metric graph* $G$ is a complete graph in which the edge weights satisfy the triangle inequality.

We briefly describe our main idea of finding an $(\alpha,\beta)$-LART of a graph as follows:

1. We shall first focus on the case where the input is a metric graph. Then, by the previous work in [23], we show that the algorithm can also be applied to a general graph with arbitrary nonnegative distances.
2. A $k$-star is a spanning tree with at most $k$ internal nodes. It was shown [23] that, for a metric graph $G$,

there exists a $k$-star whose routing cost is at most $(k+3)/(k+1)$ times the minimum routing cost. The PTAS in the paper is to find the minimum routing cost $k$-star in $O(n^{2k})$ time. However, the weight of the minimum routing cost $k$-star may be large. For example, the weight of the minimum routing cost 1-star may be $\Theta(n)$ times $w(MST(G))$.

3. Consider the algorithm in [13] for constructing an $(\alpha, 1 + 2/(\alpha - 1))$-LAST rooted at a vertex. If we check the LAST rooted at each vertex and choose the one with the minimum routing cost, we can show that it is a $(2\alpha, 1 + 2/(\alpha - 1))$-LART. To find a LART achieving more general trade-off, we use a general $k$-star. Let $R$ be a vertex set containing the $k$ internal nodes of the $k$-star with the approximate routing cost of the $MRT(G)$. We construct a *light approximate shortest-path forest* with multiple roots $R$ by the algorithm in [13]. Then, we connect the forest into a tree $T$ by adding the edges of the minimum weight tree spanning $R$. Although the routing cost of $T$ cannot be arbitrarily close to the optimum as the minimum $k$-star does, we show that it also ensures a good error ratio.

### 4.2. *The Algorithm*

In this subsection, we present the algorithm for finding a LART and analyze its time complexity. The following definition is a variant of the LART in [13]. Recall that the distance $d_G(v, R)$ between a vertex and a vertex set is the minimum distance between $v$ and any vertex in $R$.

**Definition 8.** Let $G = (V, E, w)$ and $R \subset V$. For $\alpha \geq 1$ and $\beta \geq 1$, a light approximate shortest-path forest $(\alpha, \beta)$-LASF with roots $R$ is a spanning forest $T$ of $G$ with $d_T(v, R) \leq \alpha \times d_G(v, R)$ for all $v \in V$ and $w(T) \leq \beta \times w(MST(G))$.

An example of a LASF of a graph is given in Figure 2. In [13], Khuller et al. gave an algorithm for finding a LAST for graphs with nonnegative edge weights. They also observed that their algorithm can be easily extended to the multiple roots variant. For the case where the input is a metric graph, the following corollary can be directly obtained by the results in [13], and the proof is omitted here:

**Corollary 7.** Let $G$ be a metric graph. For any $k$-vertex subset $R$ of $V(G)$ and $\alpha > 1$, there exists an algorithm which constructs an $(\alpha, 1 + 2/(\alpha - 1))$-LASF with roots $R$ in $O(kn)$ time if $MST(G)$ is given.

Our algorithm for finding a LART is listed below, and the constructed tree is illustrated in Figure 3:

**Algorithm FIND-LART**
**Input:** A metric graph $G$, a real number $\alpha > 1$, and an integer $k$.
**Output:** A LART of $G$,
STEP 1. Find $MST(G)$.
STEP 2. For each $R \subset V(G)$ and $|R| \leq k$, use the following method to construct a spanning tree, and keep $T$ with minimum $c(T)$:
STEP 2.1. Find an $(\alpha, 1 + 2/(\alpha - 1))$-LASF $T_1$ with roots $R$.
STEP 2.2. Find the MST $T_0$ of the induced subgraph $G|_R$.
STEP 2.3. Set $T = T_0 \cup T_1$ and compute $c(T)$.

Before showing the time complexity, we show how to compute the routing cost of a tree in $O(n)$ time:

**Definition 9.** Let $T$ be a tree and $e \in E(T)$. Assume that $X$ and $Y$ are the two subtrees resulting from deleting $e$ from $T$. The *routing load* on edge $e$ is defined by $l(T, e) = |V(X)| \times |V(Y)|$.

By definition, it is easy to see that $l(T, e) \leq |V(T)|^2/4$ for any edge $e$. The routing load of an edge is the number of vertex pairs between the two subtrees connected by the edge. The next lemma gives us an alternative formula to compute the routing cost of a tree:

**Lemma 8.** For a tree $T$ with edge-weight function $w$, $c(T) = \sum_{e \in E(T)} l(T, e)w(e)$. In addition, $c(T)$ can be computed in $O(n)$ time, where $n$ is the number of vertices in $T$.

**Proof.** Since $l(T, e)$ is the number of vertex pairs between which the path in $T$ contains $e$, the weight of $e$

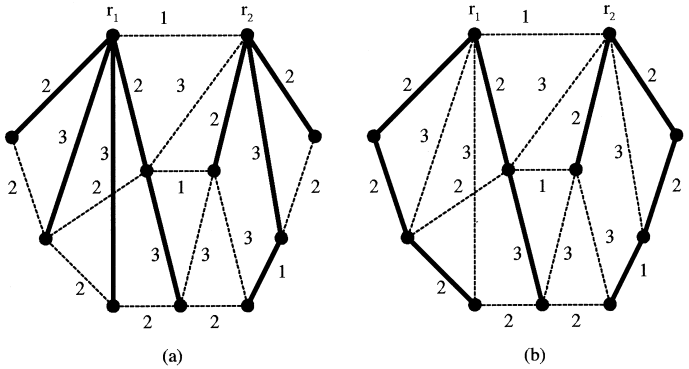


FIG. 2. (a) A shortest-path forest with roots $R = \{r_1, r_2\}$ of a graph, in which $d_T(v, R) = d_G(v, R)$ for each vertex $v$. (b) A LASF $d_T(v, R) \leq 2d_G(v, R)$ for each vertex $v$.

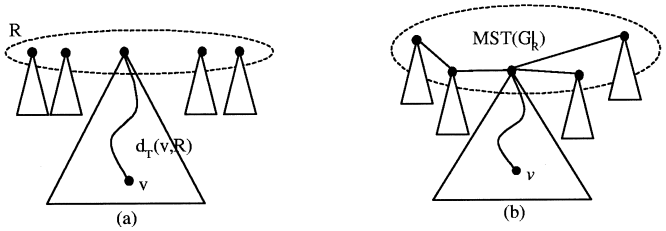


FIG. 3. The tree constructed in each iteration of **Algorithm FIND-LART:** (a) The LASF: For every vertex $v, d_T(v,R) \le \alpha d_G(v,R)$. (b) Connecting the forest by an MST of $R$.

will be counted $l(T,e)$ times while computing $c(T)$. By summing up over all edges of $T$, we get the formula. To compute $c(T)$ by this formula, we only need to find the routing load on each edge. This can be done in $O(n)$ time by rooting $T$ at any node and traversing $T$ in a postorder sequence. ∎

The time complexity of the algorithm is shown in the next lemma.

**Lemma 9.** The time complexity of **Algorithm FIND-LART** is $O(n^{k+1})$.

**Proof.** Step 1 takes $O(n^2)$ time. By Lemma 8 and Corollary 7, each step within the loop can be done in $O(kn)$ time. Since the loop is executed $\binom{n}{1}+\binom{n}{2}+\cdots+\binom{n}{k} = O(n^k)$ times, the time complexity is $O(n^k(kn) + n^2)$, or $O(n^{k+1})$ for any constant $k \ge 1$. ∎

### *4.3. The Performance Analysis*

In this subsection, we analyze the weight and the routing cost of the tree $T$ constructed by algorithm **FIND-LART.** The bound on $w(T)$ is shown in the following lemma:

**Lemma 10.** $w(T) \le (f(k) + (2/\alpha - 1))w(MST(G))$, where $f(1) = 1, f(2) = 2$, and $f(k) = 3$ for $k > 2$.

**Proof.** By Corollary 7, when $k = 1$,

$$w(T) \le \left(1 + \frac{2}{\alpha - 1}\right) w(MST(G)).$$

When $k = 2$, the tree $T_0$ contains only one edge. Since $G$ is a metric graph, $w(T_0) \le w(MST(G))$. We now consider the case $k > 2$. The minimum-weight tree spanning a subset of a graph is known as the *Steiner minimum tree*. The weight of the Steiner minimum tree spanning $R$ in $G$ is no more than $w(MST(G))$. The ratio of the weight of the minimum tree without Steiner vertices to that of the Steiner minimum tree is known as the *Steiner ratio* in the literature. For a metric graph, the ratio is 2 [14]. Therefore, $w(MST(G|_R)) \le 2w(MST(G))$ for $k > 2$. ∎

To show the routing cost ratio, we need some notation and results introduced in [20,23]. We shall give some necessary descriptions, and readers can refer to [20,23] for more details.

Let $\delta \le 1/2$ be a positive number. For any tree $T$ spanning $n$ vertices, there is a connected subtree $S$ of $T$ such that if we remove $S$ from $T$ each remaining connected component contains no more than $\delta n$ vertices. We call $S$ a *$\delta$-separator* of $T$. A $\delta$-separator is minimal if any proper subgraph of $S$ is not a $\delta$-separator of $T$.

An algorithmic description shall help us understand the separator and its importance to the routing cost. For any tree $T$, there is a vertex called the *centroid* such that if we root the tree at the centroid every subtree contains no more than one-half of the vertices. Root $T$ at its centroid and let $des(v)$ be the number of descendants of vertex $v$ (including itself). Then, remove those vertices with $des(v) \le \delta n$ and all incident edges. That way we can obtain a connected subgraph $S$ and it is a $\delta$-separator. Obviously, a centroid is a $1/2$-separator which contains only one vertex. Intuitively, a separator is like a routing center of the tree. Starting from any vertex, there are sufficiently many vertices which can be reached only after reaching the separator. For two vertices $i$ and $j$ in different components separated by $S$, the path between them can be divided into three subpaths: from $i$ to $S$, a path in $S$, and from $S$ to $j$. Since each component contains no more than $\delta n$ vertices, the distance $d_T(i,S)$ will be counted at least $(1 - \delta)n$ times as we compute the routing cost of $T$. For each edge $e$ in $S$, since there are at least $\delta n$ vertices on either side of the edge, the routing load of $e$ is at least $\delta(1 - \delta)n^2$. We now state a lemma providing a lower bound on the routing cost of the MRT. The lemma is a simplification of a result in [23], and the formal proof is omitted.

**Lemma 11.** Let $S$ be a minimal $\delta$-separator of a spanning tree $T$ of $G$. Then,

$$c(T) \ge (1 - \delta)n \sum_{v \in V(G)} d_T(v,S) + \delta(1 - \delta)n^2 w(S).$$

Let $v$ be a vertex in $S$ and $P$ be a path in $S$. We say that a vertex $u$ is hung at $v$ if its lowest ancestor in $S$ is $v$ and that $u$ is hung at $P$ if it is hung at some vertex in $P$ but not at the endpoints. A path in $S$ is a $\delta$-path if there are no more than $\delta n/2$ vertices hung at it. By cutting a tree at a vertex, we mean to divide the tree into several subtrees and all subtrees contain the vertex, as in Figure 4. We shall cut $S$ into $\delta$-paths. First, cut $S$ into paths at

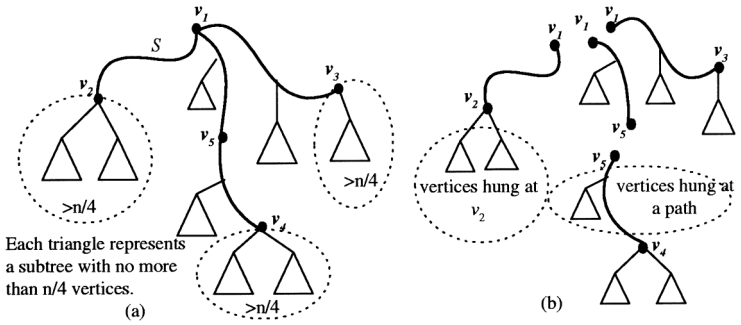


FIG. 4. (a): $S$ (bold line) is a minimal 1/4 separator of the tree. The vertex $v_1$ is the centroid, and vertices $v_2, v_3$, and $v_4$ are leaves in $S$. (b): The separator is cut into a 1/4-spine of the tree. The CAL of the spine is $\{v_1, v_2, v_3, v_4, v_5\}$. The path between $v_1$ and $v_4$ is cut at vertex $v_5$ to ensure that the number of vertices hung at each path is no more than $n/8$.

the vertices with at least three neighbors in $S$. For each path that is not a $\delta$-path, we can cut it into $\delta$-paths at some vertices of the path, and we shall give a bound on the number of such cutting vertices. The set of resulting $\delta$-paths is called a $\delta$-spine, and the set of the endpoints of the $\delta$-paths is called the *cut and leaf set* (CAL) of the spine. Let $v$ be a leaf in $S$, that is, $v$ has only one neighbor in $S$. There are at least $\delta n$ vertices hung at $v$; otherwise, $S$ is not minimal. Therefore, there are at most $\lceil 1/\delta \rceil - 1$ leaves in $S$. Assume that $S$ has $i$ leaves. The number of vertices needed to cut the paths into $\delta$-paths is less than

$$(1 - i\delta)n/(\delta n/2) = \lceil 2/\delta \rceil - 2i.$$

Since $S$ is a tree, the number of vertices with at least three neighbors in $S$ is no more than $i-2$. Consequently, we can find a spine such that the number of vertices in its CAL is less than $i + (i-2) + (\lceil 2/\delta \rceil - 2i)$ or no more than $\lceil 2/\delta \rceil - 3$. The next lemma is given in [23] and the formal proof is omitted here:

**Lemma 12.** For any constant $0 < \delta \le 0.5$ and spanning tree $T$ of $G$, there exists a $\delta$-spine $Y$ of $T$ such that $|CAL(Y)| \le \lceil 2/\delta \rceil - 3$.

The following lemma is used to bound the routing cost ratio:

**Lemma 13.** Let $T$ be a spanning tree of a metric graph $G$. If $S$ and $Y$ are the minimal $\delta$-separator and $\delta$-spine of $T$, respectively, then $\sum_{v \in V} d_G(v, CAL(Y)) \le \sum_{v \in V} d_T(v, S) + (\delta n/4) w(S)$.

**Proof.** Let $T$ be a spanning tree rooted at its centroid. For each vertex $v$ hung at a vertex in $CAL(Y)$, we have

$$d_G(v, CAL(Y)) \le d_T(v, CAL(Y)) = d_T(v, S).$$

Furthermore, for each vertex $v$ hung at a $\delta$-path $P$, by the triangle inequality, we have

$$d_G(v, CAL(Y)) \le d_T(v, S) + w(P)/2.$$

Since $P$ is a $\delta$-spine, there are at most $\delta n/2$ vertices hung at $P$. Summing up the distances over all vertices, we have

$$\begin{aligned} \sum_{v \in V} d_G(v, CAL(Y)) &\le \sum_{v \in V} d_T(v, S) + (\delta n/4) \sum_{P \in Y} w(P) \\ &= \sum_{v \in V} d_T(v, S) + (\delta n/4) w(S). \end{aligned}$$

∎

Let $\hat{T}$ be the tree output by algorithm **FIND-LART.** Its routing cost ratio is given in the next lemma:

**Lemma 14.** Let $T^* = MRT(G)$. $c(\hat{T}) \le [(k+3)/(k+1)]\alpha c(T^*)$, for any integer $k \le 6\alpha - 3$.

**Proof.** Let $\delta = 2/(k+3)$. By Lemma 12, there exists a $\delta$-spine $Y$ of $T^*$ such that $|CAL(Y)| \le k$. Let $T_1$ be a LASF with roots $R$, where $R = \{r_i | 1 \le i \le q\} = CAL(Y)$, and $T_0$ be the MST of the induced subgraph $G|_R$. Furthermore, let $T = T_1 \cup T_0$. Since $c(\hat{T}) \le c(T)$, we only need to prove the ratio of the routing cost of $T$. Let $V = V(G) = V(T)$. By Lemma 8 and the fact that the routing load on any edge is no more than $n^2/4$, we have

$$\begin{aligned} c(T) &\le n \sum_{v \in V} d_T(v, R) + \sum_{e \in E(T_0)} l(T, e) w(e) \\ &\le n \sum_{v \in V} d_T(v, R) + (n^2/4) w(T_0). \end{aligned} \tag{1}$$

Since $T_1$ is a LASF, we have

$$\sum_{v \in V} d_T(v, R) \le \alpha \sum_{v \in V} d_G(v, R). \tag{2}$$

By the inequalities (1) and (2), we have

$$c(T) \le \alpha n \sum_{v \in V} d_G(v, R) + (n^2/4) w(T_0). \tag{3}$$

Let $S_1 = (R, E_S)$, in which the edge set $E_S$ contains all the edges $(u, v)$ if $u$ and $v$ are the two endpoints of some path in $Y$. By the triangle inequality, $w(S_1) \le w(S)$. Since $S_1$ is a spanning tree of the induced graph $G|_R$ and $T_0$ is the minimal one, $w(T_0) \le w(S_1) \le w(S)$. Then, by Lemma 13 and the inequality (3),

$$c(T) \le \alpha n \sum_{v \in V} d_{T^*}(v, S) + (\alpha\delta + 1)(n^2/4) w(S).$$

By Lemma 11,

$$c(T^*) \geq (1-\delta)n\sum_{v\in V} d_{T^*}(v,S) + \delta(1-\delta)n^2 w(S).$$

Comparing the two inequalities, we have $c(T) \leq \max\{\alpha/(1-\delta), (\alpha\delta+1)/(4\delta(1-\delta))\}c(T^*)$. Note that $\alpha > 1$ and $0 < \delta \leq 1/2$. Let $g(\delta) = \max\{\alpha/(1-\delta), (\alpha\delta+1)/(4\delta(1-\delta))\}$. When $\delta \geq 1/(3\alpha), \alpha/(1-\delta) \geq (\alpha\delta+1)/(4\delta(1-\delta))$, and $g(\delta)$ decreases as $\delta$ decreases from $1/2$ to $1/(3\alpha)$. When $\delta \leq 1/(3\alpha), (\alpha/(1-\delta)) \leq ((\alpha\delta+1)/(4\delta(1-\delta))$, and $g(\delta)$ increases as $\delta$ decreases from $1/(3\alpha)$. Therefore, $g(\delta)$ reaches its minimum when $\delta = 1/(3\alpha)$. Since $\delta = 2/(k+3), c(T) \leq ((k+3)/(k+1))\alpha c(T^*)$, for any $k \leq 6\alpha - 3$. ∎

In summary, the performance of the algorithm is given in the following theorem:

**Theorem 15.** Given a metric graph $G$, a $[((k+3)/(k+1))\alpha, (f(k)+2/(\alpha-1))]$-LART can be constructed in $O(n^{k+1})$ time for any real number $\alpha > 1$ and an integer $1 \leq k \leq 6\alpha - 3$, where $f(1) = 1, f(2) = 2$, and $f(k) = 3$ for $k > 2$.

### *4.4. Extensions of the Algorithm*

We now extend the algorithm **FIND-LART** to the case where the input is a general graph:

**Definition 10.** The *metric closure* of a graph $G$ is a complete graph with vertex set $V(G)$, in which the weight on an edge $(u, v)$ is equal to $d_G(u, v)$.

Given a spanning tree $T$ of the metric closure of a graph $G$, it is shown in [23] that, in $O(n^3)$ time, we can transform $T$ into a spanning tree $Y$ of $G$ such that $c(Y) \leq c(T)$. By observing that $w(Y) \leq w(T)$ in the process of the construction, we have the following corollary:

**Corollary 16.** Given an $(\alpha, \beta)$-LART of the metric closure of a graph $G$, an $(\alpha, \beta)$-LART of $G$ can be constructed in $O(n^3)$ time.

Therefore, to find a LART of a general graph, we can first compute its metric closure in $O(n^3)$ [5] and then find a LART of the metric graph. Finally, we transform the tree into the desired LART of the original graph.

**Corollary 17.** Given a graph $G$, a $[((k+3)/(k+1))\alpha, (f(k)+2/(\alpha-1))]$-LART can be constructed in $O(n^{k+1}+n^3)$ time for any real number $\alpha > 1$ and an integer $1 \leq k \leq 6\alpha - 3$, where $f(1) = 1, f(2) = 2$, and $f(k) = 3$ for $k > 2$.

The algorithm for constructing a LART can be applied to the following related problems. Since the MRT problem is NP-hard, it is easy to see that these problems are also NP-hard. All the results can be obtained directly from Theorem 15:

1. The weight-constrained MRT problem: Given a graph $G$ and a real number $\alpha \geq 1$, the goal is to find a spanning tree $T$ with minimum $c(T)$ subject to $w(T) \leq \alpha \times w(MST(G))$. For any fixed $\alpha > 1$, the optimal solution can be approximated with a constant ratio in polynomial time.
2. The routing-cost-constrained MST problem: Given a graph $G$ and a real number $\alpha \geq 1$, find a spanning tree $T$ with minimum $w(T)$ subject to $c(T) \leq \alpha \times c(MRT(G))$. For any graph $G$ and a fixed $\alpha > 3/2$, the optimal solution can be approximated with a constant ratio in polynomial time.
3. Let $\alpha_T = c(T)/c(MRT(G)), \beta_T = w(T)/w(MST(G))$, and $h(\alpha_T, \beta_T)$ be a specified function for evaluating the total cost of $T$. Assume that $h$ is monotonically increasing, that is, $h(x+\Delta x, y+\Delta y) \geq h(x, y)$ for all $\Delta x, \Delta y \geq 0$. The goal is to find a spanning tree $T$ with minimum total cost $h(\alpha_T, \beta_T)$. For example, $h(\alpha_T, \beta_T) = \alpha_T \times \beta_T$, or $h(\alpha_T, \beta_T) = x \times \alpha_T + y \times \beta_T$, in which $x$ and $y$ are constants. By Theorem 15, for any real number $\alpha > 1$, a $[((k+3)/(k+1))\alpha, (f(k)+2/(\alpha-1))]$-LART can be constructed in polynomial time, and the ratio $h[((k+3)/(k+1))\alpha, (f(k)+2/(\alpha-1))]/h(1,1)$ is a constant. Since $h(1,1)$ is a trivial lower bound, such a tree is an approximate solution with a constant ratio for any monotonically increasing function $h$. To obtain a better ratio and reasonable time complexity, suitable $\alpha$ and $k$ should be chosen.


## Acknowledgments

The authors thank the anonymous referees for their careful reading and many useful comments. The authors also thank Dr. Douglas R. Shier for improving the presentation of this paper.


## REFERENCES


[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, On sparse spanners of weighted graphs, Discr Comput Geom 9 (1993), 81–100.

[2] Y. Bartal, On approximating arbitrary metrics by tree metrics, Proc 30th Ann ACM Symp Theory Comput ACM, Dallas, Texas, 1998, pp. 161–168.

[3] M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, Time bounds for selection, J Comput Syst Sci 7 (1972), 448–461.

[4] L. Cai, NP-completeness of minimum spanner problems, Discr Appl Math 48 (1994), 187–194.

[5] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, Introduction to algorithms, MIT Press, Cambridge, MA, 1994.

[6] R.C. Entringer, D.J. Kleitman, and L.A. Szekely, A note on spanning trees with minimum average distance, Bull Inst Combin Appl 17 (1996), 71–78.

[7] G.N. Frederickson and D.B. Johnson, Generalized selection and ranking: Sorted matrices, SIAM J Comput 13 (1984), 14–30.

[8] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J ACM 34 (1987), 596–615.

[9] M.R. Garey and D.S. Johnson, Computers and intractability: A guide to the theory of NP-completeness, W.H. Freeman, San Francisco, 1979.
[10] R. Hassin and A. Tamir, On the minimum diameter spanning tree problem, Info Process Lett 53 (1995), 109–111.
[11] T.C. Hu, Optimum communication spanning trees, SIAM J Comput 3 (1974), 188–195.
[12] D.S. Johnson, J.K. Lenstra, and A.H.G. Rinnooy Kan, The complexity of the network design problem, Networks 8 (1978), 279–285.
[13] S. Khuller, B. Raghavachari, and N. Young, Balancing minimum spanning trees and shortest-path trees, Algorithmica 14 (1995), 305–321.
[14] L. Kou, G. Markowsky, and L. Berman, A fast algorithm for Steiner trees, Acta Info 15 (1981), 141–145.
[15] D. Peleg and E. Reshef, Deterministic polylog approximation for minimum communication spanning trees, Proc 25th Int Colloq on Automata, Languages and Programming (ICALP'98), Aalborg, Denmark, 1998, K.G. Larsen, S. Skyum, and G. Winskel (Editors), LNCS 1443, Springer-Verlag, 1998, pp. 670–681.
[16] D. Peleg and A.A. Schäffer, Graph spanners, J Graph Theory 13 (1989), 99–116.
[17] J. Plesnik, The complexity of designing a network with minimum diameter, Networks 11 (1981), 77–85.
[18] J. Plesnik, On the sum of all distances in a graph or digraph, J Graph Theory 8 (1984), 1–21.
[19] R. Wong, Worst-case analysis of network design problem heuristics, SIAM J Alg Discr Methods 1 (1980), 51–63.
[20] B.Y. Wu, K.-M. Chao, and C.Y. Tang, Approximation algorithms for the shortest total path length spanning tree problem, Discr Appl Math 105 (2000), 273–289.
[21] B.Y. Wu, K.-M. Chao, and C.Y. Tang, Approximation algorithms for some optimum communication spanning tree problems, Discr Appl Math 102 (2000), 245–266.
[22] B.Y. Wu, K.-M. Chao, and C.Y. Tang, A polynomial time approximation scheme for optimal product-requirement communication spanning trees, J Alg 36 (2000), 182–204.
[23] B.Y. Wu, G. Lancia, V. Bafna, K.-M. Chao, R. Ravi, and C.Y. Tang, A polynomial time approximation scheme for minimum routing cost spanning trees, SIAM J Comput 29 (2000), 761–778.