SIAM J. DISC. MATH.
Vol. 8, No. 3, pp. 359–387, August 1995

TREE SPANNERS *

LEIZHEN CAI[†] AND DEREK G. CORNEIL[‡]

Abstract. A tree t-spanner T of a graph G is a spanning tree in which the distance between every pair of vertices is at most t times their distance in G. This notion is motivated by applications in communication networks, distributed systems, and network design.

This paper studies graph-theoretic, algorithmic, and complexity issues about tree spanners. It is shown that a tree 1-spanner, if it exists, in a weighted graph with m edges and n vertices is a minimum spanning tree and can be found in $O(m \log \beta(m, n))$ time, where $\beta(m, n) = \min\{i | \log^{(i)} n \leq m/n\}$. On the other hand, for any fixed $t > 1$, the problem of determining the existence of a tree t-spanner in a weighted graph is proven to be NP-complete. For unweighted graphs, it is shown that constructing a tree 2-spanner takes linear time, whereas determining the existence of a tree t-spanner is NP-complete for any fixed $t \geq 4$. A theorem that captures the structure of tree 2-spanners is presented for unweighted graphs. For digraphs, an $O((m + n)\alpha(m, n))$ algorithm is provided for finding a tree t-spanner with t as small as possible, where $\alpha(m, n)$ is a functional inverse of Ackerman's function. The results for tree spanners on undirected graphs are extended to "quasi-tree spanners" on digraphs. Furthermore, linear-time algorithms are derived for verifying tree spanners and quasi-tree spanners.

Key words. graph algorithm, NP-complete, tree spanner, spanning tree, distance

AMS subject classifications. 05C05, 05C12, 05C85, 68Q25, 68R10

1. Introduction.

1.1. Motivation.

A *t-spanner* of a graph G is a spanning subgraph H in which the distance between every pair of vertices is at most t times their distance in G. This notion was introduced in 1987 by Peleg and Ullman [27], who showed that spanners can be used to construct synchronizers for transforming synchronous algorithms into asynchronous algorithms. A similar notion appeared in 1986 when Chew [16] studied approximations of complete Euclidean graphs by their planar subgraphs.

The key idea behind the notion of spanners is the approximation of pairwise vertex-to-vertex distances in the original graph by spanning subgraphs. The quality of the distance approximation by a t-spanner is measured by the parameter $t \geq 1$, which is referred to as the *stretch factor* of the t-spanner. This distance approximation property makes spanners quite useful in areas such as communication networks, distributed systems, motion planning, network design, and parallel machine architectures [5], [3], [6], [16], [27]–[29], [25]. For example, a sparse spanner (a spanner with few edges) of small stretch factor can be used to plan efficient routing schemes in a communication network while maintaining succinct routing tables [28]. Such a spanner can also be used as a substitute for its original network to reduce the construction cost of the network while keeping similar communication costs [29], [23]–[25]. In motion planning, when the input of a simple polygon is inaccurate, a special spanner of the visibility

* Received by the editors September 24, 1992; accepted for publication (in revised form) August 30, 1994. This research was supported in part by a grant from the Natural Sciences and Engineering Research Council of Canada.

[†] Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada. Current address: Department of Computer Science, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong (lcai@cs.cuhk.hk).

[‡] Department of Computer Science, University of Toronto, Toronto, Ontario M5S 1A4, Canada. (dgc@cs.toronto.edu).

graph of the input polygon, called the visibility skeleton, can be used to plan collision-free paths inside the real polygon [14].

In most applications, the sparseness of a spanner is the main concern; the sparsest t-spanner in a connected graph is a tree t-spanner, that is, a t-spanner that is a tree. Therefore, as far as sparseness is concerned, tree t-spanners are the best possible t-spanners. Furthermore, tree spanners have other interesting applications besides those mentioned for general graph spanners. Tree spanners of small stretch factors can be used to perform multisource broadcasts in a network [5], which can greatly simplify the message routing at the cost of only small delays in message delivery. The existence of a tree 2-spanner in a 2-connected network guarantees that the communication between operative sites will not be affected by any isolated failure of communication sites and lines [10]. There are also some surprising connections between tree 2-spanners and cycles in graphs. Certain cycle-extremal weighted graphs can be represented as a weighted union of tree 2-spanners ([8], where they were called tritrees), and graphs that contain tree 2-spanners ([7], where they were called trigraphs) appear to be the only graphs that require a large number of cycles to cover the edges of the graph exactly twice.

In this paper we consider graph-theoretic, algorithmic, and complexity issues about tree spanners. We study tree spanners in weighted, unweighted, and directed graphs, as well as "quasi-tree" spanners in directed graphs. By exploring graph-theoretic characterizations, we obtain several efficient algorithms for finding tree and quasi-tree t-spanners for some values of t. On the other hand, we show the intractability of determining the existence of tree and quasi-tree t-spanners for almost all other values of t. Furthermore, we present linear-time algorithms for verifying tree and quasi-tree t-spanners for all values of t.

1.2. Notation and definitions. We use the terminology of Bondy and Murty [9]. Graphs in this paper can be either weighted or unweighted, directed or undirected; they are connected graphs without loops, multiedges, and multiarcs. For any graph $G, V(G)$ denotes the vertex set of G; if G is undirected then $E(G)$ denotes the edge set of G, and if G is directed then $A(G)$ denotes the arc set of G. For a subset V' of vertices of $G, G[V']$ denotes the induced subgraph of G on V'; for a subset E' of edges of $G, G[E']$ denotes the edge-induced subgraph of G on E'. The induced subgraph $G[V(G) \setminus V']$ is denoted by $G - V'$, and the edge-induced subgraph $G[E(G) \setminus E']$ is denoted by $G - E'$. For any subgraph H of $G, G - H$ denotes the subgraph obtained from G by deleting edges (or arcs) of H from G. Throughout this paper, unless specified otherwise, m denotes the number of edges (or arcs) of G and n denotes the number of vertices of G. For any real number x, $\lfloor x \rfloor$ denotes the largest integer $\leq x$ and $\lceil x \rceil$ denotes the least integer $\geq x$.

We shall assume that the weight $w(e)$ of an edge (or arc) e is a positive real number and regard an unweighted graph as a weighted graph where each edge (or arc) has unit weight. Given a subgraph H of $G, w(H)$ denotes the *weight* of H, i.e., the sum of the weights of all edges in H; when H is a (directed) path, $w(H)$ is the *length* of H. For any two vertices x and y of G, a path from x to y is an *(x, y)-path* and an (x, y)-path of minimum length is a *shortest (x, y)-path.* We use $d_G(x, y)$ to denote the *weighted distance* in G from x to y, i.e., the length of a shortest (x, y)-path in G. Note that $d_G(x, y) = \infty$ if there is no (x, y)-path in G and $d_G(x, y) = d_G(y, x)$ if G is undirected.

For any real number $t \geq 1$, a spanning subgraph H of G is a *t-spanner* if $d_H(x, y) \leq t \cdot d_G(x, y)$ for every pair of vertices x and y of G. The parameter t is called the *stretch* *factor* of H. The *stretch index* of a spanner H is the minimum number t for which H is a t-spanner. A t-spanner H is a *minimal t-spanner* if no subgraph of H is a t-spanner of G, a *minimum t-spanner* if it has the least number of edges among all t-spanners of G, and an *optimal t-spanner* if H has the least weight among all t-spanners of G.

For an undirected graph G, a spanning subgraph T of G is a *tree t-spanner* if T is both a t-spanner and a tree; in this case G is *tree t-spanner admissible.* A spanning subgraph T of G is a *tree spanner* if it is a tree t-spanner for some $t \geq 1$ and a *minimum tree spanner* if it has the smallest stretch factor among all tree spanners of G. Thus a spanning tree of G is always a tree spanner.

For a directed graph (digraph) $G = (V, A; w)$, we use $\tilde{G} = (V, E; \tilde{w})$ to denote its underlying undirected graph, i.e., $xy \in E$ iff either $(x, y) \in A$ or $(y, x) \in A$ or both, and

$$\tilde{w}(x,y) = \begin{cases} w((x,y)) & \text{if } (x,y) \in A, (y,x) \notin A, \\ w((y,x)) & \text{if } (y,x) \in A, (x,y) \notin A, \\ \min\{w((x,y)), w((y,x))\} & \text{if } (x,y), (y,x) \in A. \end{cases}$$

A vertex x *reaches* vertex $y(y$ is *reachable* from $x)$ in G if there is a directed (x, y)-path in G. It follows that $d_G(x, y) = \infty$ if y is not reachable from x in G.) A *spanning tree* of G is a spanning subgraph T that contains no directed cycle and such that $\tilde{T}$ is a tree. Then, as with undirected graphs, a *tree t-spanner* of a digraph is a spanning tree that is a t-spanner. A *quasi tree* of G is a spanning subgraph T such that $\tilde{T}$ is a tree; T is a *quasi-tree t-spanner* if it is a t-spanner of G. Note that a quasi tree may contain a cycle consisting of two arcs (x, y) and (y, x). Other terms on tree spanners of undirected graphs are naturally extended to tree spanners and quasi-tree spanners of digraphs. However, a spanning tree (quasi tree) of a digraph is not necessarily a tree (quasi tree) spanner.

A few more definitions are in order for undirected graphs. (For simplicity, we will use these definitions for digraphs as well; it is understood that whenever we do so, we either refer to the underlying graphs or mean that the underlying graphs have the property.) For a connected graph, a *k-cut* is a set of k vertices whose deletion disconnects the graph. A graph G is *nonseparable* if it has no 1-cut and *triconnected* if it has no k-cut for $k \leq 2$. A *block* of a graph is a maximal nonseparable subgraph, and a *triconnected component* of a graph is a maximal triconnected subgraph. A vertex is *universal* if it is adjacent to all other vertices of the graph. An edge e is a *binding edge* if its two ends form a minimal cut set. Two disjoint subgraphs S and S' of G are *fully joined* if every vertex in S is adjacent to every vertex in S'. A *star* is any complete bipartite graph $K_{1,n}$ with $n \geq 1$.

Finally, by the *tree t-spanner problem,* we usually mean the problem of finding a tree t-spanner in a graph, but it may refer to the problem of determining whether a graph contains a tree t-spanner when we talk about NP-completeness. Its meaning should be clear from the context. The meanings of other spanner problems are similarly defined.

1.3. Observations. We gather here some fundamental results on spanners in a graph. For simplicity, we will state our results only in terms of undirected graphs. These results also hold for digraphs and will be used in our discussions throughout the paper.

First, because edge weights are assumed to be positive, each of the following statements gives an equivalent definition of a t-spanner in a weighted graph.

THEOREM 1.1. *Let H be a spanning subgraph of a weighted graph $G = (V, E; w)$. The following statements are then equivalent:*

(1) *H is a t-spanner of G (i.e., $d_H(x,y) \leq t \cdot d_G(x,y)$ for every pair $x, y \in V$).*

(2) *For every edge $xy \in E, d_H(x,y) \leq t \cdot d_G(x,y)$.*

(3) *For every edge $xy \in E \setminus E(H), d_H(x,y) \leq t \cdot d_G(x,y)$.*

(4) *For every edge $xy \in E, d_H(x,y) \leq t \cdot w(xy)$.*

(5) *For every edge $xy \in E \setminus E(H), d_H(x,y) \leq t \cdot w(xy)$.*

Proof. The implications (1) $\Rightarrow$ (2), (2) $\Rightarrow$ (3), and (4) $\Rightarrow$ (5) are trivial. To see that (3) $\Rightarrow$ (4), we need only note that, for any edge $xy \in E$, we have $d_G(x,y) \leq w(xy)$ and that $d_H(x,y) \leq w(xy) \leq t \cdot w(xy)$ if $xy \in E(H)$, since $t \geq 1$ and $w(xy) > 0$.

We now show that (5) $\Rightarrow$ (1). It suffices to show that $d_H(x,y) \leq t \cdot d_G(x,y)$ for two arbitrary vertices x, y of G. Let P be a shortest (x,y)-path in G. Then for each edge uv on P, if $uv \in E(H)$, then $d_H(u,v) \leq w(uv) \leq t \cdot w(uv)$, since $t \geq 1$ and $w(uv) > 0$; otherwise, $d_H(u,v) \leq t \cdot w(uv)$ by statement (5). Therefore,

$$d_H(x,y) \leq \sum_{uv \in P} d_H(u,v) \leq t \sum_{e \in P} w(e).$$

Since $d_G(x,y) = \sum_{e \in P} w(e)$ by the choice of P, we obtain

$$d_H(x,y) \leq t \cdot d_G(x,y)$$

This completes the proof. □

Quite often we will use statement (5) in the above theorem as the definition of a t-spanner, since it is easy to handle in most cases. Based on the above theorem, we can easily observe the following facts.

Observation 1.2. Let F be a t-spanner of G and H be a k-spanner of F. Then H is a kt-spanner of G.

Observation 1.3. For any $k > 1, H = (V, E'; w)$ is a t-spanner of $G = (V, E; w)$ iff $H' = (V, E'; w')$ is a t-spanner of $G' = (V, E; w')$, where $w'(e) = k \cdot w(e)$ for every $e \in E$.

It is easy to see that we can consider each block separately in dealing with most spanners, such as minimal, minimum, and optimal t-spanners. In particular, we can restrict our attention to nonseparable graphs when we deal with tree spanners.

Observation 1.4. Let T be a spanning tree of a graph G. Then T is a tree t-spanner of G iff for every block H of $G, T \cap H$ is a tree t-spanner of H.

Finally, for an unweighted graph G, the distance between any two vertices in G is always an integer. Therefore, in light of statement (4) of Theorem 1.1, we need only consider t-spanners for integral t.

Observation 1.5. Let H be a spanning subgraph of an unweighted graph G. Then H is a t-spanner iff H is a $\lfloor t \rfloor$-spanner.

1.4. Outline of the paper.

We begin by discussing the verification of tree spanners and quasi-tree spanners in §2. We present $O(m)$ time algorithms for verifying tree t-spanners in graphs and in digraphs as well as quasi-tree t-spanners in digraphs.

In §3, we consider tree spanners in weighted graphs. We show that a tree 1-spanner, if it exists, is a minimum spanning tree and can be found in $O(m \log \beta(m,n))$ time, where $\beta(m,n) = \min\{i \mid \log^{(i)} n \leq m/n\}$. On the other hand, we prove that, for any fixed $t > 1$, the problem of finding a tree t-spanner in a weighted graph is intractable.

In §4, we investigate tree spanners in unweighted graphs. We show that a tree 2-spanner can be constructed in linear time and that the tree t-spanner problem is

NP-complete for any fixed integer $t \geq 4$. We also present a skeleton tree theorem, which captures the structure of tree 2-spanners.

We deal with tree spanners of digraphs in §5. We present an $O((m+n)\alpha(m+n,n))$ algorithm for finding a minimum tree spanner in a digraph, where $\alpha(m,n)$ is a functional inverse of Ackerman's function. For general digraphs, we extend the results of §§3 and 4 to quasi-tree spanners.

We conclude the paper with a short summary and some open problems in §6.

2. Verifying a tree t-spanner.

Given a graph G, a spanning tree T, and a positive number t, we wish to verify whether T is a tree t-spanner of G. We may also wish to know if T is a tree spanner and, if it is, determine its stretch index, i.e., the smallest t for which T is a t-spanner. Similar problems can also be explored for quasi-tree spanners. These problems will come forth naturally in later sections, and, for convenience, we will refer to these problems as *tree spanner verification problems.*

In this section, we will provide linear-time algorithms for the above verification problems. The main results of this section are summarized in the following theorem, which will be used in later sections.

THEOREM 2.1. *Let D and G be directed and undirected weighted graphs, respectively. Let S and T be spanning trees of D and G, respectively. Let Q be a quasi tree of D. Then the following problems can be solved in $O(m)$ time:*

(a) *Determine the stretch index of T.*

(b) *Is S a tree spanner? If it is, determine its stretch index.*

(c) *Is Q a quasi-tree spanner? If it is, determine its stretch index.*

2.1. A verification algorithm paradigm.

We first describe an algorithm paradigm for tree spanner verification problems. Clearly, statement (5) of Theorem 1.1 provides us with a simple method for solving these problems. By taking this approach, we need to compute the distances in T of all $m-n+1$ vertex pairs defined by nontree edges. Thus the cost of distance computation dominates the running time of verification algorithms based on this approach. If we compute the distance of each vertex pair directly and independently, it may take $O(mn)$ time to compute these distances, since each distance may take $O(n)$ time to compute. We can reduce the cost to $O(n^2)$ by computing all pairwise distances in T together. Unfortunately, this is not satisfactory for sparse graphs. To speed up the verification, we need a better way to compute the distances of these $m-n+1$ vertex pairs.

For simplicity, we will describe an algorithm for verifying a tree t-spanner in an undirected graph. The algorithm is easily extended to other verification problems. Several definitions are in order. A *rooted tree* T is a tree with a distinguished vertex r, called the *root.* For any two vertices x and y in T, if x is on the path from r to y, then x is an *ancestor* of y. The *least common ancestor* of x and y, denoted by $\mathrm{LCA}(x,y)$, is the common ancestor z of x and y such that for any common ancestor z' of x and y, z' is an ancestor of z. We will take advantage of the structure of a tree to compute distances more efficiently. To achieve this, we arbitrarily choose a vertex r to be the root of T and then label vertices of T in such a way that distance $d_T(x,y)$ of any vertex pair (x,y) can be quickly computed from the labels of x,y, and $\mathrm{LCA}(x,y)$. Notice that $d_T(x,y) = d_T(x,\ \mathrm{LCA}(x,y)) + d_T(\mathrm{LCA}(x,y),y)$.

```
ALGORITHM VERIFICATION($G, T, t$){Verify if $T$ is a tree $t$-spanner of $G$.}
Input: A graph $G$, a spanning tree $T$ and a positive number $t$;
Output: "Yes" if $T$ is a tree $t$-spanner; "No" otherwise.

begin
1.   Arbitrarily choose a vertex $r$ as the root of $T$;
2.   Compute a label label($x$) for each vertex $x$ of $T$;
3.   Compute LCA($x, y$) for every nontree edge $xy$ of $G$;
4.   for each nontree edge $xy$ do
     begin
4.1.      Compute $d_T(x, y)$ by using the labels of $x, y$ and LCA($x, y$);
4.2.      if $d_T(x, y) > t \cdot w(xy)$ then output "No" EXIT;
     end;
5.   output "Yes";
end.
```

By statement (5) of Theorem 1.1, we note that the stretch index of T equals

$$\max\{1, d_T(x,y)/w(xy) | xy \in E(G) \setminus E(T)\}.$$

The above algorithm can thus be modified (line (4.2)) to compute the stretch index of T as well. To apply this algorithm to a digraph D, we take the underlying tree $\tilde{T}$ of D's spanning tree (quasi tree) T to define a rooted tree and carry out the computation of the algorithm with respect to this rooted tree. In this case, when we notice that x reaches y in T iff $d_T(x, y)$ is finite, we can use the algorithm to check the reachability from x to y in T as well.

Regarding the complexity of the algorithm, we see that line (1) is trivial. To carry out the computation of line (3), we use a linear-time least common ancestor algorithm of Harel and Tarjan [21]. Clearly, line (4.2) takes $O(1)$ time. In the next two subsections, we will discuss efficient implementations of line (2) and line (4.1) for undirected graphs and digraphs so as to obtain the results in Theorem 2.1.

2.2. Undirected case. Let G be an undirected weighted graph and T be a spanning tree of G. Arbitrarily choose a vertex r in T as the root of T. For each vertex x in T, label x by the root-to-vertex distance of x, i.e., label(x) $= d_T(r, x)$. See Fig. 1 for an example.

We show that for any two vertices x and y, their distance $d_T(x, y)$ in T can be computed in constant time from label(x), label(y), and label(LCA(x, y)). Notice that

$$\text{label}(x) = d_T(r, x) = d_T(r,\ \text{LCA}(x, y)) + d_T(\text{LCA}(x, y), x)$$

and

$$\text{label}(y) = d_T(r, y) = d_T(r,\ \text{LCA}(x, y)) + d_T(\text{LCA}(x, y), y).$$

We obtain

$$d_T(x, y) = \text{label}(x) + \text{label}(y) - 2 \cdot \text{label}(\text{LCA}(x, y)).$$

Therefore $d_T(x, y)$ can be determined in $O(1)$ time.

It is easy to see that by either a depth-first or a breadth-first search from the root r, we can obtain the labels for all vertices of T. Thus line (2) of algorithm VERIFICATION can be carried out in $O(n)$ time. Furthermore, line (4.1) can be done in $O(1)$ time; thus step (4) takes $O(m - n)$ time. Therefore, the overall time

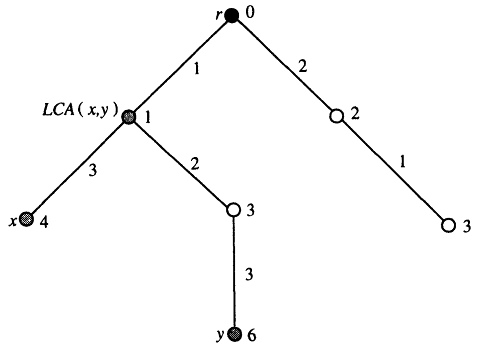

FIG. 1. *Labeling the vertices of T by their root-to-vertex distances.*

of the algorithm is linear. Thus verifying a tree t-spanner takes linear time. Once $d_T(x,y)$ is obtained for every nontree edge xy, we can easily determine the stretch index of T in linear time, thereby establishing Theorem 2.1(a).

2.3. Directed case. Let D be a weighted digraph and S be a spanning tree of D. Then by statement (5) of Theorem 1.1, it is easy to see that S is a tree spanner of D iff x reaches y in S for any nontree arc (x,y) of D. Therefore, in order to verify that S is a tree spanner of D, we need to verify that S preserves reachability for each nontree arc of D. We apply VERIFICATION to D and S together with the underlying tree $\tilde{S}$ of S.

Arbitrarily choose a vertex r in $\tilde{S}$ as the root of $\tilde{S}$. An edge xy of $\tilde{S}$, where x is an ancestor of y, is a *forward edge* if (x,y) is an arc of S and a *backward edge* if (y,x) is an arc of S. For an arbitrary vertex x in $\tilde{S}$, let $P(x)$ be the unique (r,x)-path in $\tilde{S}$. Label x by a triple $(b(x), f(x), l(x))$, where

$b(x)$ is the number of backward edges on $P(x)$,

$f(x)$ is the number of forward edges on $P(x)$, and

$l(x)$ is the total weight of forward edges on $P(x)$ minus the total weight of backward edges on $P(x)$.

The first two components in the triple are used for verifying reachability; the third one is used for computing distances. It is easy to see that all vertex labels can be computed in $O(n)$ time by either a depth-first or breadth-first search of $\tilde{S}$ from the root r. See Fig. 2 for an example. Backward and forward edges are indicated by upward and downward arrows, respectively.

For any two vertices x and y of S, it is easy to see that x reaches y in S iff

$$f(x) = f(\mathrm{LCA}(x,y)) \text{ and } b(y) = b(\mathrm{LCA}(x,y)).$$

Since these two conditions can be easily checked in $O(1)$ time, the overall cost of verifying a tree spanner is linear. Furthermore, it is not difficult to see that if x reaches y in S, then

$$d_S(x,y) = l(y) - l(x).$$

Thus $d_S(x,y)$ can be computed in $O(1)$ time. Therefore, it takes linear time to compute the stretch index of a tree spanner of D. This also implies that verifying a tree t-spanner of D takes linear time. Hence, we have Theorem 2.1(b).

We now turn our attention to quasi-tree spanners. Let Q be a quasi tree of D. Like the situation for tree spanners in digraphs, in order to verify that Q is a quasi-tree

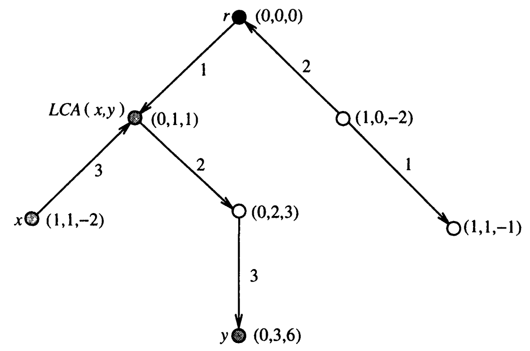

FIG. 2. *Labeling the vertices of $\tilde{S}$ by triples.*

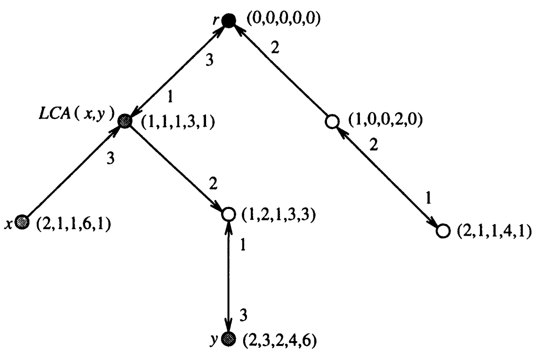

FIG. 3. *Labeling the vertices of $\tilde{Q}$ by quintuples.*

spanner of D, we need to verify that Q preserves reachability for each arc of D that is not in Q. We apply VERIFICATION to D and Q together with the underlying tree $\tilde{Q}$ of Q.

Arbitrarily choose a vertex r in $\tilde{Q}$ as the root of $\tilde{Q}$. An edge xy of $\tilde{Q}$ is a *double edge* if both (x,y) and (y,x) are arcs in Q. Note that a double edge is also a forward edge and a backward edge. For an arbitrary vertex x in $\tilde{Q}$, let $P(x)$ be the unique (r,x)-path in $\tilde{Q}$. Label x by a quintuple $(b(x), f(x), d(x), lb(x), lf(x))$, where

$b(x)$ is the number of backward edges on $P(x)$,

$f(x)$ is the number of forward edges on $P(x)$,

$d(x)$ is the number of double edges on $P(x)$,

$lb(x)$ is the total weight of arcs of Q corresponding to backward edges on $P(x)$, and

$lf(x)$ is the total weight of arcs of Q corresponding to forward edges on $P(x)$. The first three components in the quintuple are used for verifying reachability; the last two are used for computing distances. Again all vertex labels can be computed in $O(n)$ time by either a depth-first or breadth-first search of $\tilde{Q}$ from the root r. See Fig. 3 for an example. Each double edge is shown with both an upward and downward arrow; the numbers beside the arrows indicate the weights of the corresponding arcs of Q.

For any two vertices x and y, it is easy to see that x reaches y iff

$$f(x) - f(\mathrm{LCA}(x,y)) = d(x) - d(\mathrm{LCA}(x,y))$$

and

$$b(y) - b(\mathrm{LCA}(x, y)) = d(y) - d(\mathrm{LCA}(x, y)).$$

We can check these two conditions in $O(1)$ time and thus verify a quasi-tree spanner in linear time. Furthermore, if x reaches y, then we have

$$d_Q(x, y) = lb(x) - lb(\mathrm{LCA}(x, y)) + lf(y) - lf(\mathrm{LCA}(x, y)).$$

Thus $d_Q(x, y)$ can be computed in $O(1)$ time. Therefore, it takes linear time to compute the stretch index of a quasi-tree spanner of D, which implies that verifying a quasi-tree t-spanner takes linear time. This establishes Theorem 2.1(c), and thus completes the proof of Theorem 2.1.

3. Tree spanners in weighted graphs.

In this section, we consider the complexity of tree spanner problems on weighted graphs. By statement (5) of Theorem 1.1, a spanning subgraph H of a weighted graph $G = (V, E; w)$ is a t-spanner iff for every edge $xy \in E \setminus E(H)$, we have $d_H(x, y) \leq t \cdot w(xy)$. We present an $O(m \log \beta(m, n))$ algorithm for finding a tree 1-spanner in a weighted graph; on the other hand, we show that for any fixed $t > 1$, the tree t-spanner problem is NP-complete on weighted graphs. This completely settles the issue of complexity of tree spanner problems for weighted graphs. Henceforth in this section, by a graph we mean a weighted graph.

3.1. Finding a tree 1-spanner.

Let $G = (V, E; w)$ be a weighted graph, and let H be a 1-spanner of G. Since H is a subgraph of G, it is clear that $d_H(x, y) \geq d_G(x, y)$ for any two vertices $x, y \in V$. Therefore, $d_H(x, y) = d_G(x, y)$ for any $x, y \in V$, i.e., H preserves pairwise distances in G.

The distance-preserving property of a 1-spanner is useful in many applications. For example, a 1-spanner of a communication network can be used as a substitute for the original network without introducing any extra delay in communication. It is also closely related to the metric realization problem [2], [31], [20] (to construct a graph with a minimum total weight that realizes an n-by-n symmetric distance matrix $M = (m_{i,j})$). To see this, we construct a complete graph $G(M)$ on n vertices such that $w(ij) = m_{i,j}$ for each edge ij of $G(M)$; then the optimal 1-spanner of $G(M)$ gives an optimal realization of G if we allow only n vertices. Regarding tree 1-spanners, we see that a tree 1-spanner is a distance-preserving spanning tree. Therefore, using a tree 1-spanner of a network to perform broadcast in the network guarantees the minimum delay. Furthermore, a tree 1-spanner can also be used as a compact encoding of the distance information of G.

Remark. Because of the connection between 1-spanners and metric realizations, some results in this subsection regarding minimal 1-spanners have appeared in the literature on metric realizations. In particular, Corollary 3.3 has been previously obtained by Hakimi and Yau [20].

We shall first explore the properties of 1-spanners of G. These properties lead us to polynomial algorithms for constructing a minimum or an optimal 1-spanner in G, and these algorithms can be used to find a tree 1-spanner in G. We then establish a relationship between a tree 1-spanner and a minimum spanning tree and use this relationship to derive a more efficient algorithm for finding a tree 1-spanner.

LEMMA 3.1. *Let H be a 1-spanner of a weighted graph G. Then H is minimal iff $d_{H-xy}(x, y) > w(xy)$ for every edge xy of H.*

Proof. If there is an edge xy of H such that $d_{H-xy}(x, y) \leq w(xy)$, then $H - xy$ is a 1-spanner of H by statement (5) of Theorem 1.1, and thus $H - xy$ is a 1-spanner of G by Observation 1.2. Hence, H is not minimal. Conversely, if H is not minimal, then there is an edge uv of H such that $H - uv$ is a 1-spanner of G. This implies that $d_{H-uv}(u, v) \le w(uv)$. □

We are now ready to present necessary and sufficient conditions for an edge of G to be in a minimal 1-spanner. Bear in mind that edge weights of G are positive.

THEOREM 3.2. *Let H be a minimal 1-spanner of a weighted graph G, and let xy be an edge of G. Then the following statements are equivalent:*

(1) *Edge xy belongs to H.*

(2) *For every vertex $z \in V \setminus \{x, y\}, d_G(x, z) + d_G(z, y) > w(xy)$.*

(3) *Distance $d_{G-xy}(x, y) > w(xy)$.*

Proof. (1) $\Rightarrow$ (2). Let z be an arbitrary vertex in $V \setminus \{x, y\}$. If $d_H(x, z) + d_H(z, y) \le w(xy)$, then $d_H(x, z) < w(xy)$, since edge weights are positive, and thus any shortest (x, z)-path P in H avoids edge xy. Let $H' = H - xy$. Then $d_{H'}(x, z) = d_H(x, z)$, since P is in H'. Similarly, $d_{H'}(z, y) = d_H(z, y)$.

By the definition of distance, we have

$$d_{H'}(x, y) \le d_{H'}(x, z) + d_{H'}(z, y).$$

Therefore,

$$d_{H'}(x, y) \le d_H(x, z) + d_H(z, y) \le w(xy).$$

Then by Lemma 3.1, H is not a minimal 1-spanner, which is a contradiction. Hence,

$$d_H(x, z) + d_H(z, y) > w(xy).$$

Since H is a 1-spanner of G, we now have $d_H(x, z) = d_G(x, z)$ and $d_H(z, y) = d_G(z, y)$. Therefore, $d_G(x, z) + d_G(z, y) > w(xy)$.

(2) $\Rightarrow$ (3). Let $G' = G - xy$. By the definition of distance, we have

$$d_{G'}(x, y) = \min_{z \in V \setminus \{x, y\}} \{d_{G'}(x, z) + d_{G'}(z, y)\}$$

It follows from statement (2) that $d_{G'}(x, y) > w(xy)$.

(3) $\Rightarrow$ (1). Because edge weights are positive, statement (3) implies that xy is the only (x, y)-path in G with length $\le w(xy)$. Since H is a 1-spanner of G, we have $d_H(x, y) \le w(xy)$. Therefore, xy must appear in H. □

COROLLARY 3.3 (Hakimi and Yau [20]). *Every weighted graph G has a unique minimal 1-spanner.*

Proof. By Theorem 3.2, each edge of a minimal 1-spanner of G is uniquely determined. □

Since both a minimum and an optimal 1-spanner of G are minimal 1-spanners, Corollary 3.3 implies the following result.

COROLLARY 3.4. *For any weighted graph G, the following statements are equivalent:*

(1) *H is a minimal 1-spanner of G.*

(2) *H is a minimum 1-spanner of G.*

(3) *H is an optimal 1-spanner of G.*

If G contains a tree 1-spanner T, then T is also a minimal 1-spanner. Thus Corollary 3.3 also implies the uniqueness of a tree 1-spanner.

COROLLARY 3.5. *A weighted graph can contain at most one tree 1-spanner.*

In light of Theorem 3.2 and Corollary 3.4, we see that the minimum (or optimal) 1-spanner of a weighted graph can be constructed in polynomial time, since for each edge of G, we can use either statement (2) or statement (3) of Theorem 3.2 to decide if the edge belongs to the minimal 1-spanner H of G. For a single edge, it is more efficient to use statement (3) to determine whether the edge is in H if pairwise distances are not given. However, it seems that the algorithm using statement (2) is more efficient for constructing H, especially when pairwise distances are given; using the fastest known algorithm to compute pairwise distances [17], we can implement the algorithm in $O(mn + n^2 \log n)$ time.

THEOREM 3.6. *The minimum (or optimal) 1-spanner of a weighted graph can be found in $O(mn + n^2 \log n)$ time.*

Clearly, we can find the tree 1-spanner (if it exists) of G in $O(mn + n^2 \log n)$ time by first computing the minimal 1-spanner of G and then checking if it is a tree. However, this approach is not efficient. To obtain a more efficient algorithm for computing the tree 1-spanner, we will establish a relationship between the tree 1-spanner of G and a minimum spanning tree of G.

THEOREM 3.7. *The tree 1-spanner of a weighted graph G is a minimum spanning tree. Moreover, every tree 1-spanner admissible weighted graph contains a unique minimum spanning tree.*

Proof. Let T be a minimum spanning tree of G. We first claim that T is contained in any 1-spanner H of G. To see this, let xy be an arbitrary edge of T and P be a shortest (x, y)-path in $G - xy$. Then there is an edge e on P that is not in T. If $w(P) \leq w(xy)$, then $w(e) < w(xy)$, since edge weights are positive and P contains at least two edges. This implies that $T + e - xy$ is a spanning tree whose weight is less than $w(T)$, contrary to T being a minimum spanning tree. Therefore, $d_{G-xy}(x, y) = w(P) > w(xy)$, and, by Theorem 3.2, xy is an edge of H.

Now let T' be the tree 1-spanner of G. By the above claim, T is a subgraph of T'. Since both T and T' are spanning trees of G, we have $T = T'$. The theorem follows immediately. □

In light of the above theorem, we have the following algorithm for constructing the tree 1-spanner of G: we first find a minimum spanning tree T of G and then verify whether T is a 1-spanner of G. A minimum spanning tree can be found in $O(m \log \beta(m, n))$ time [18], where $\beta(m, n) = \min\{i | \log^{(i)} n \leq m/n\}$ and $\log^{(i)} n$ is defined by $\log^{(0)} n = n, \log^{(i)} n = \log \log^{(i-1)} n$ for $i \geq 1$. Since verification takes linear time by Theorem 2.1(a), we have the following result.

THEOREM 3.8. *The tree 1-spanner of a weighted graph can be found in $O(m \log \beta(m, n))$ time.*

Remark. The above algorithm can be applied to find tree 1-spanners in a weighted graph G where zero weight is allowed. Let G_0 be the subgraph of G induced by zero-weighted edges of G and $Z_1, \ldots, Z_k$ be the connected components of G_0. We construct a new weighted graph G' as follows: contract such Z_i to a single vertex z_i, remove all loops, and, for all parallel edges (formed from the contraction) between two vertices, delete all but one with the lightest weight. Then G' is a weighted graph with no zero weight on edges, and its tree 1-spanner can be found by the algorithm in this subsection. It is not hard to see that G admits a tree 1-spanner iff G' admits one. Actually, a tree 1-spanner of G can be obtained from the tree 1-spanner of G' by "replacing" each z_i with a spanning tree of Z_i. However, G may contain many tree 1-spanners when it has zero-weighted edges. In fact, the number of tree 1-spanners in G equals the product of the number of spanning trees of $Z_i, 1 \leq i \leq k$. Figure 4 illustrates the above construction.

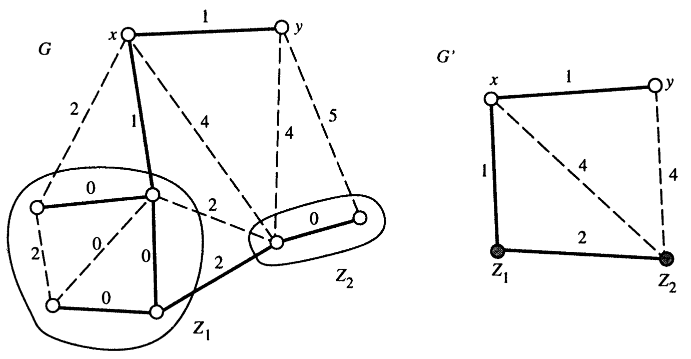

FIG. 4. *Graphs G, G' and their tree 1-spanners (solid edges).*

TABLE 1
Bridge length (number of edges) and edge weights of G (note $t = 1 + \varepsilon$).

		Edge weights $w(e)$		
t	Bridge length	Bridge edge	Literal edge	Clause edge
$1 < t < 2$	1	1	$\lfloor 1/\varepsilon \rfloor$	$\lceil (1 + 2\lceil 1/\varepsilon \rceil)/\varepsilon \rceil$
$2 \le t < 4$	$2\lfloor \varepsilon \rfloor$	1	2	$\lceil (4 + 2\lfloor \varepsilon \rfloor)/\varepsilon \rceil$

3.2. NP-completeness for $t > 1$. We now consider the complexity of finding tree t-spanners ($t > 1$) in a weighted graph. It turns out that the tree t-spanner problem on weighted graphs is intractable for any fixed rational number $t > 1$. As a consequence, the minimum t-spanner problem on weighted graphs is intractable for any fixed rational number $t > 1$. Furthermore, we deduce that the optimal t-spanner problem on weighted graphs is also intractable for any fixed rational number $t > 1$. In this subsection, we assume that all edge weights are positive rational numbers and that $t > 1$ is a fixed rational number.

Recall that an instance (U, C) of 3SAT (cf. [LO2] in [19]) consists of a set U of n distinct Boolean variables and a collection C of m 3-element clauses over U. For any variable $u \in U$, both u and $\bar{u}$ are *literals*; for a truth assignment ξ, a literal l is *true* if $\xi(l) = 1$ and *false* otherwise.

THEOREM 3.9. *For any fixed rational number $t > 1$, it is NP-complete to determine whether a weighted graph contains a tree t-spanner, even if all edge weights are positive integers.*

Proof. It is clear that the problem is in NP. To establish the NP-completeness of the problem, we present a polynomial transformation from 3SAT. Here we will only consider the case $1 < t < 4$; the proof for $t \ge 4$ is the same as that for unweighted graphs (see Theorem 4.10 of §4.3), where a more complicated construction is employed.

Let $t \in (1, 4)$ be a fixed rational number, and, for convenience, let $\varepsilon = t - 1$; then $0 < \varepsilon < 3$. For an arbitrary instance (U, C) of 3SAT, we construct a weighted graph G such that C is satisfiable iff G admits a tree t-spanner. Graph G is constructed as follows:

1. Take a vertex x and vertices $U' = \{u_1, \bar{u}_1, \ldots, u_n, \bar{u}_n\}$, where $n = |U|$, and construct a star H centered at x by joining x to each vertex in U'. Each vertex in U' is a *literal vertex* and each edge in H is a *literal edge.*

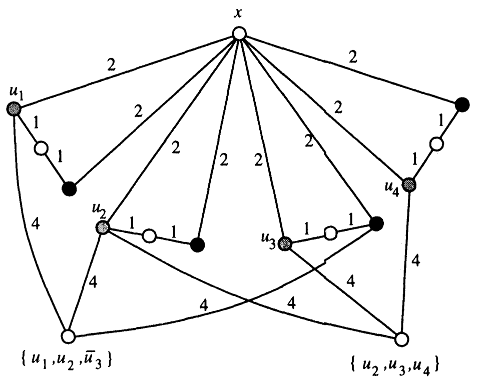

FIG. 5. *The graph g for $t = 2.5$ and $C = \{\{u_1, u_2, \bar{u}_3\}, \{u_2, u_3, u_4\}\}$.*

2. Create a new vertex c for each clause in C and add an edge between c and each of its three distinct literal vertices in H. These new vertices and edges are called *clause vertices* and *clause edges,* respectively.

3. Connect each pair $u_i, \bar{u}_i$ of literal vertices by a distinct path (whose length is specified in Table 1), called a *bridge,* to complete the construction of G. Each edge on the bridge is called a *bridge edge.*

4. For each edge e of G, assign it weight $w(\varepsilon)$ according to Table 1.

Figure 5 shows an example of G. It is easy to see that G can be constructed in polynomial time.

Now, for any tree t-spanner T of G (Remember that $1 < t < 4$ and $\varepsilon = t - 1$), we note the following two important properties:

P1. Every bridge is contained in T.

P2. For every pair $xu_i, x\bar{u}_i$ of literal edges, exactly one of them belongs to T.

To see property P1, let ab be an arbitrary bridge edge and P_{ab} be an (a, b)-path in T. Suppose that ab is not in T. If $0 < \varepsilon < 1$, then P_{ab} contains either two literal edges or at least two clause edges; otherwise $1 \leq \varepsilon \leq 3$ and P_{ab} contains all other $2\lfloor\varepsilon\rfloor - 1$ bridge edges on the bridge containing ab and either two literal edges or at least two clause edges. It is readily checked that we would then have $d_T(a, b) > t \cdot w(ab)$ in both cases, which contradicts T being a t-spanner. Hence ab belongs to T.

To see property P2, without loss of generality, we consider edge xu_i. If the (x, u_i)-path in T contains neither edge xu_i nor $x\bar{u}_i$, then it must contain a literal edge and at least two clause edges. It is easy to see that $d_T(x, u_i) > t \cdot w(xu_i)$ if $0 < \varepsilon < 1$; otherwise $1 \leq \varepsilon < 3$ and

$$d_T(x, u_i) \geq 2 + 2\left\lceil \frac{4 + 2\lfloor\varepsilon\rfloor}{\varepsilon} \right\rceil.$$

It can be shown that

$$\left\lceil \frac{4 + 2\lfloor\varepsilon\rfloor}{\varepsilon} \right\rceil > \varepsilon \quad \text{for} \quad 1 \leq \varepsilon < 3$$

(consider $\varepsilon \in [1, 2)$ and $[2, 3)$ separately). Thus it follows that $d_T(x, u_i) > t \cdot w(xu_i)$ for $1 < t < 4$, which contradicts T being a tree t-spanner. So at least one of $xu_i, x\bar{u}_i$ is in T. Since T is a tree, it can be deduced from property P1 that exactly one of them is in T.

We now prove that C is satisfiable iff G contains a tree t-spanner. Suppose that C is satisfiable and let ξ be a satisfying truth assignment for C. Call a literal vertex of G a *true vertex* if its corresponding literal is a true literal under ξ. Construct a spanning tree T of G by taking all bridge edges, all literal edges incident with true vertices, and, for each clause, an arbitrary clause edge that is incident with a true vertex. Since C is satisfied by ξ, it is clear that T is a spanning tree and each clause vertex is a leaf.

To see that T is a t-spanner, we note that, for any literal edge xl not in T, the (x, l)-path in T consists of a literal edge and a bridge and that, for any clause edge cl' not in T, the (c, l')-path in T consists of a clause edge, two literal edges, and at most one bridge. It is a routine matter to check that $d_T(x, l) \le t \cdot w(xl)$ and $d_T(c, l') \le t \cdot w(cl')$. Therefore, it follows from statement (5) of Theorem 1.1 that T is a tree t-spanner of G.

Conversely, suppose that T is a tree t-spanner of G. Then by property P2, exactly one of the two literal edges $xu_i, x\bar{u}_i$ is contained in T. Therefore we can define a truth assignment ξ_T be setting $\xi_T(u_i) = 1$ if $xu_i \in E(T)$ and $\xi_T(u_i) = 0$ if $x\bar{u}_i \in E(T)$. It remains to be shown that ξ_T satisfies C.

It is easy to see by property P1 that any two literal vertices are connected by a path in T that avoids clause vertices. Thus each clause vertex is a leaf of T. Suppose that there is a clause vertex c which contains only false literals under ξ_T. Then for a clause edge cl not in T, the (c, l)-path in T consists of a clause edge, two literal edges, and two bridges. If $0 < \varepsilon < 1$, then it is easy to check that $d_T(c, l) \ge t \cdot w(cl)$ (notice $1/\varepsilon > 1$). Otherwise $1 \le \varepsilon < 3$, and again it is a routine matter to check that $d_T(c, l) \ge t \cdot w(cl)$ (notice $2\lfloor \varepsilon \rfloor > \varepsilon$); this contradicts T being a t-spanner. Therefore, each clause in C contains at least one true literal under ξ_T, and thus C is satisfiable. The proof is complete. □

Since a tree t-spanner has the least number of edges among all t-spanners, Theorem 3.9 implies that finding a minimum t-spanner in a weighted graph is intractable for any fixed rational number $t > 1$.

COROLLARY 3.10. *For any fixed rational number $t > 1$, it is NP-complete to determine, given a weighted graph G and a positive integer K, whether G contains a t-spanner with at most K edges, even if all edge weights are positive integers.*

Furthermore, the tree t-spanner T in the proof of Theorem 3.9 also achieves the minimum total weight (sum of weights of all edges in the spanning subgraph) over all t-spanners of G. Therefore, Theorem 3.9 also implies the following result.

COROLLARY 3.11. *For any fixed rational number $t > 1$, it is NP-complete to determine, given a weighted graph G and a positive rational number W, whether G contains a t-spanner of total weight at most W, even if all edge weights are positive integers.*

4. Tree spanners in unweighted graphs.

We now consider tree spanners in unweighted graphs, which can be considered as a special case of tree spanners in weighted graphs, i.e., tree spanners in unit-weighted graphs. Henceforth in this section, by a graph we always mean an unweighted graph. By statement (5) of Theorem 1.1, a spanning subgraph H is a t-spanner of a graph $G = (V, E)$ iff for every edge $xy \in E \setminus E(H)$ we have $d_H(x, y) \le t$.

In light of Observation 1.5, we need to consider only tree t-spanners for integral t. Clearly, a graph contains a tree 1-spanner iff it itself is a tree. We show that a tree 2-spanner in a graph can be found in linear time. We also study the structure of tree 2-spanners and give a characterization of tree 2-spanner-admissible graphs.

In particular, we present a structural theorem for tree 2-spanners in terms of the "skeleton tree" of a graph. This structural theorem is useful in dealing with various tree 2-spanner problems. On the other hand, we show that the tree t-spanner problem is NP-complete for any fixed $t \geq 4$. The complexity of the tree 3-spanner problem remains an open issue.

4.1. Finding a tree 2-spanner. Our main concern is to find a tree 2-spanner in a graph. In order to design an efficient tree 2-spanner-finding algorithm, we first investigate the structure of tree 2-spanners in a graph. We then describe a linear-time algorithm. Furthermore, we give a characterization of tree 2-spanner-admissible graphs in terms of decomposition. Because of Observation 1.4, we can restrict our attention to nonseparable graphs.

Remark. It is interesting to note that tree 2-spanner-admissible graphs coincide with trigraphs introduced by Bondy [7] in his work on cycle double covers; our characterization results are quite similar to his. In particular, our decomposition theorem (Theorem 4.4) for tree 2-spanner-admissible graphs and Lemma 4.1 have been previously obtained by Bondy. They are reformulated here in our terminology for the sake of completeness.

LEMMA 4.1 (Bondy [7]). *Let G be a nonseparable graph and T be an arbitrary tree 2-spanner of G. Then for every 2-cut $\{u, v\}$ of $G, uv \in E(T)$.*

Proof. Let $\{u, v\}$ be a 2-cut of G. Let H_1 be a connected component of $G-\{u, v\}$. Let $G_1 = G[V(H_1) \cup \{u, v\}]$ and $G_2 = G - V(H_1)$. Then each $G_i, i = 1, 2$, contains a (u, v)-path P_i of length at least two. If $uv \notin E(T)$, then for each edge in $E(P_i) \setminus E(T)$, there is a path of length two in $T \cap G_i$ between its two ends. Thus there is a (u, v)-path Q_i in $T \cap G_i$. Q_1 and Q_2 would then be a cycle in T, which is a contradiction. Hence $uv \in E(T)$. □

THEOREM 4.2. *Let G be a nonseparable graph. Then a spanning tree T of G is a tree 2-spanner iff for each triconnected component H of $G, T \cap H$ is a spanning star of H.*

Proof. If $T \cap H$ is a spanning star of H for each triconnected component H of G, then $T \cap H$ is a tree 2-spanner of H. Since each edge of G belongs to some triconnected component, it follows that T is a 2-spanner of G.

Conversely, suppose that T is a tree 2-spanner of G. We first show that for each triconnected component H of $G, T' = T \cap H$ is a tree 2-spanner of H. It is trivial if H consists of a single edge. Thus we may assume that $|V(H)| \geq 3$. Then H contains at least one edge $uv \notin E(T)$, since H contains a cycle. Because T is a tree 2-spanner, there exists a vertex w such that $uw, vw \in E(T)$. If w is not in H, then u and v are the only vertices in H adjacent to w, since H is a triconnected component and $|V(H)| \geq 3$. This implies that $\{u, v\}$ is a 2-cut. By Lemma 4.1, uv would be an edge in T, which is a contradiction. Therefore, w is in H and thus uw, vw are in T'. This implies that T' is a tree 2-spanner of H.

It remains to be shown that T' is a star. Suppose that T' is not a star; then there is an edge $xy \in E(T')$ that is not incident to any leaf. Let T'_x and T'_y be the two connected components of $T' - xy$ containing vertices x and y, respectively. Note that both T'_x and T'_y contain at least two vertices. For two arbitrary vertices $u \in V(T'_x) \setminus \{x\}$ and $v \in V(T'_y) \setminus \{y\}$, it is easy to see that $d_{T'}(u, v) \geq 3$. This implies $uv \notin E(H)$. Then $\{x, y\}$ would be a 2-cut of H, contrary to H being a triconnected component. Therefore, T' is a spanning star of H. □

COROLLARY 4.3. *A triconnected graph G admits a tree 2-spanner iff it contains a universal vertex.*

Lemma 4.1 and Theorem 4.2 can be used to obtain a characterization of tree 2-spanner-admissible graphs in terms of decomposition. A graph G is an *edge bonding* of two graphs G_1 and G_2 if $G = G_1 \cup G_2$ and $G_1 \cap G_2$ is an edge.

THEOREM 4.4 (Bondy [7]). *A graph G is tree 2-spanner admissible iff each block H of G is either*

(1) *a triconnected graph with a universal vertex or*

(2) *an edge bonding (on edge e) of two tree 2-spanner-admissible graphs where e is a tree edge in both graphs.*

Proof. Because of Observation 1.4, we only need to consider a block H of G. If H contains a universal vertex u, then the set of edges incident with u induces a tree 2-spanner of H. If H is an edge bonding of two tree 2-spanner-admissible graphs H_1 and H_2 on a tree edge e, then the edge bonding of a tree 2-spanner T_1 of H_1 and a tree 2-spanner T_2 of H_2 on edge e yields a tree 2-spanner of H.

Conversely, suppose that H is tree 2-spanner admissible. If H has no 2-cut, then it is triconnected and by Corollary 4.3 contains a universal vertex. Otherwise, H has a 2-cut $\{x, y\}$; by Lemma 4.1, then, xy is an edge of H and belongs to every tree 2-spanner of H. Let H' be a connected component of $H - \{x, y\}, H_1 = H[V(H') \cup \{x, y\}]$, and $H_2 = H - V(H_1)$. It can then be deduced from Theorem 4.2 that $T \cap H_1$ and $T \cap H_2$ are tree 2-spanners of H_1 and H_2, respectively. Hence, H is an edge bonding of two tree 2-spanner-admissible graphs H_1 and H_2 on edge xy. This completes the proof. □

We now use the above results to derive an algorithm for finding a tree 2-spanner T (if it exists) in a graph G. The algorithm can be outlined as follows (details are left to the reader). First, find all blocks of G. Then, for each block, find all 2-cuts (if there is a 2-cut that does not induce a binding edge, then G contains no tree 2-spanner, by Lemma 4.1) and triconnected components. Put all binding edges in T. Now, for each triconnected component H, find a spanning star containing all edges of H that have been put into T so far and put it in T (if such a spanning star does not exist, then G contains no tree 2-spanner by Lemma 4.1 and Theorem 4.2). Finally, if T is a spanning tree, then it is a tree 2-spanner; otherwise, G contains no tree 2-spanner. This algorithm can be implemented in linear time by using the triconnected component-finding algorithm of Hopcroft and Tarjan [22] and standard techniques.

THEOREM 4.5. *A tree 2-spanner (if it exists) of a graph can be found in $O(m + n)$ time.*

4.2. The skeleton tree.

A tree spanner may be required to have some additional properties, such as a degree constraint, a bound on the diameter, or a limit on the number of leaves. Here we conduct a further investigation of the structure of tree 2-spanners to provide a useful tool in dealing with the construction of tree 2-spanners with additional properties. Henceforth, we assume that all graphs in this subsection are tree 2-spanner-admissible.

By Theorem 4.2, every tree 2-spanner of a triconnected graph is a spanning star. Thus there is nothing more to be said about the structure of tree 2-spanners in a triconnected graph. In light of Lemma 4.1, we can restrict our attention to nonseparable graphs with binding edges. We show that any tree 2-spanner of such a graph can be obtained from a "skeleton tree" of the graph by properly adding "compound leaves" to the skeleton tree. This result also gives another clear picture of the structure of tree 2-spanner-admissible graphs. In the rest of this subsection, we assume that G is a tree 2-spanner-admissible graph that is nonseparable and contains at least one binding edge. We start with the subgraph of G induced by the set of its binding edges.

LEMMA 4.6. *The set of binding edges of G induces a tree.*

Proof. Let B be the set of binding edges and $T_B = G[B]$. Clearly, T_B is a forest, since G admits a tree 2-spanner T and every edge in B belongs to T by Lemma 4.1. We need to show only that T_B is connected.

Let $\mathcal{H}$ be the set of triconnected components of G. Construct a bipartite graph F with vertex set $B \cup \mathcal{H}$ in which $e \in B$ and $H \in \mathcal{H}$ are adjacent iff edge e is in the triconnected component H. Since G is connected, it is clear that F is connected.

Let v and v' be two arbitrary vertices of T_B and e and e' be two binding edges incident with v and v', respectively. Consider the bipartite graph F. Since F is connected, there is an (e, e')-path $P = e_1H_1e_2 \ldots H_{k-1}e_k$ in F, where $e_i \in B, H_i \in \mathcal{H}, e_1 = e$, and $e_k = e'$. Thus for any two elements $e_i, e_{i+1} \in B, H_i$ is a triconnected component of G that contains edges e_i and e_{i+1}. By Theorem 4.2, e_i and e_{i+1} share a vertex. It is now easy to deduce that there is a (v, v')-path in T_B, since each e_i is an edge in T_B. Therefore, T_B is connected and the proof is complete. $\square$

Let $\mathcal{T}(G)$ denote the set of tree 2-spanners of G and $\mathcal{E}(G)$ denote the set of edges of G contained in every tree 2-spanner of G, i.e., $\mathcal{E}(G) = \cap_{T\in\mathcal{T}(G)}E(T)$. Note that a *nontrivial tree* is a tree with at least one edge.

LEMMA 4.7. *$G[\mathcal{E}(G)]$ is a nontrivial tree.*

Proof. Obviously, $G[\mathcal{E}(G)]$ is a forest. Since G is tree 2-spanner admissible, by Lemma 4.1, every binding edge of G is contained in $G[\mathcal{E}(G)]$. Furthermore, for any edge $e \in \mathcal{E}(G)$, if e is not a binding edge of G, then it belongs to a unique triconnected component H of G. Since H contains at least one binding edge of G, it follows from Theorem 4.2 that edge e shares a vertex with at least one binding edge of G. By Lemma 4.6 and the assumption that G contains a binding edge, we see that $G[\mathcal{E}(G)]$ is connected and hence a nontrivial tree. $\square$

Because $G[\mathcal{E}(G)]$ induces a nontrivial tree that belongs to every tree 2-spanner of G, we call it the *skeleton tree* of G and denote it by $\mathcal{S}(G)$. Let us now examine the structure of $G - V(\mathcal{S}(G))$. Recall that two disjoint subgraphs are fully joined iff every vertex in one subgraph is adjacent to every vertex in the other.

LEMMA 4.8. *Each connected component C of $G - V(\mathcal{S}(G))$ is fully joined with a unique edge of $\mathcal{S}(G)$. Moreover, for any tree 2-spanner T of G, each vertex in C is a leaf of T.*

Proof. By Lemma 4.1, the skeleton tree $\mathcal{S}(G)$ contains the subgraph $\mathcal{S}$ induced by the set of binding edges of G. Therefore, each connected component C of $G - V(\mathcal{S}(G))$ is a subgraph of a connected component C' of $G - V(\mathcal{S})$. Since C' belongs to a unique triconnected component H of G, so does C. If H contains more than one edge of $\mathcal{S}(G)$, say e_1 and e_2, then it follows from Theorem 4.2 that e_1 and e_2 must share a vertex u and all the edges between u and C belong to T. This implies that every vertex of C is in $\mathcal{S}(G)$, which contradicts the choice of C. Therefore, H contains a unique edge e of $\mathcal{S}(G)$. If one end of e is not a universal vertex of H, then by Theorem 4.2, all the edges between the other end of e and C belong to T. Again, this contradicts the choice of C. Therefore, C is fully joined with a unique edge e of $\mathcal{S}(G)$. By Theorem 4.2, each vertex of C is a leaf of T. $\square$

Because of the above lemma, we call each connected component of $G - V(\mathcal{S}(G))$ a *compound leaf*. Then every edge e of the skeleton tree $\mathcal{S}(G)$ has a set (possibly empty) of compound leaves fully joined with it. Let the two ends of e be x and y. Then for any compound leaf C fully joined with e, $V(C) \cup \{x, y\}$ induces a triconnected component H of G. Note that both x and y are adjacent to every vertex in C. The set of edges

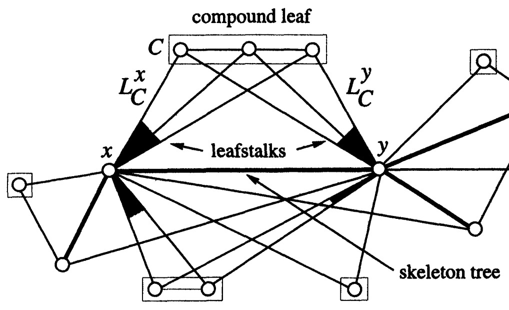

FIG. 6. *The skeleton tree, compound leaves, and leafstalks.*

between x and C(y and C) forms a star $L_C^x(L_C^y)$ and will be referred to as a *leafstalk* of C. These concepts are illustrated in Fig. 6, where thick lines depict the skeleton tree, each box contains a compound leaf, and each shaded triangle indicates a nontrivial leafstalk (a leafstalk with more than one edge).

THEOREM 4.9 (skeleton tree theorem). *Let G be a tree 2-spanner-admissible, nonseparable graph that contains binding edges. A spanning tree T is a tree 2-spanner of G iff it is obtained from the skeleton tree $\mathcal{S}(G)$ of G by adding to $\mathcal{S}(G)$ exactly one leafstalk for each compound leaf of G.*

Proof. This follows from Theorem 4.2, Lemma 4.7, and Lemma 4.8. □

We now turn to the construction of the skeleton tree $\mathcal{S}(G)$ of G. First, we find all binding edges and triconnected components of G. As discussed in Lemma 4.6 and the proof of Lemma 4.7, these binding edges form a tree S that is a subtree of $\mathcal{S}(G)$. Then we extend S to $\mathcal{S}(G)$ by considering triconnected components one by one. For each triconnected component H, if H contains two distinct edges e_1 and e_2 of S, then e_1 and e_2 share a vertex u and we put into $\mathcal{S}(G)$ all the edges of H that are incident with u (by Theorem 4.2); if H contains only one edge $e = uv$ of S and one end of e, say u, is not a universal vertex of H, then we put into $\mathcal{S}(G)$ all the edges of H that are incident with v (by Theorem 4.2). The correctness of the above algorithm follows from our previous discussions.

By using the linear-time, triconnected component-finding algorithm of Hopcroft and Tarjan [22] and standard techniques, we can construct the skeleton tree and find all compound leaves in linear time. Therefore, the skeleton tree provides a handy and useful tool in constructing tree 2-spanners with certain properties. For instance, with the aid of the skeleton tree, the problem of finding a tree 2-spanner of bounded degree is easily solved in linear time. It is interesting to note that the corresponding degree-bounded spanning tree problem is NP-complete ([ND1] in [19]). Skeleton trees have also been used in the design of a polynomial-time algorithm for determining whether a 2-connected graph contains a tree 2-spanner isomorphic to a given tree [11], [13]. We note again that the corresponding isomorphic spanning tree problem is NP-complete ([ND8] in [19]). Further applications of skeleton trees can be found in the construction of quasi-tree 2-spanners (§5.2), as well as in the construction of nearly distance-preserving spanning trees [11].

4.3. NP-completeness for $t \geq 4$.

Although a tree 2-spanner in a graph can be constructed in linear time, the problem of finding a tree t-spanner seems to be very hard for $t \geq 3$; in fact, as we show here, the problem is intractable for any fixed $t \geq 4$. As a consequence, the minimum t-spanner problem on unweighted graphs is

NP-complete for any fixed $t \geq 4$. The complexity of finding a tree 3-spanner in a graph is still unknown.

Remark. Stronger NP-completeness results hold for the minimum t-spanner problem on unweighted graphs. In fact, the problem is NP-complete for any fixed $t \geq 2$ [12], [26], even when restricted to graphs of bounded degree [15].

THEOREM 4.10. *For any fixed $t \geq 4$, the tree t-spanner problem is NP-complete.*

Proof. It is clear that the problem is in NP. To establish the NP-completeness, we present a polynomial transformation from 3SAT. By Observation 1.5, we need to consider only integral values of t. Let $t \geq 4$ be a fixed integer and (U, C) be an arbitrary instance of 3SAT. We construct a graph G such that C is satisfiable iff G has a tree t-spanner. Call a path with t edges a *t-path*. The following result is useful in our construction.

LEMMA 4.11. *Let G be a graph and e an edge of G. Let G' be a graph formed from G by adding two distinct t-paths P_1, P_2 (all internal vertices of P_1 and P_2 are new vertices) between the two ends of e and T be a tree t-spanner of G'. Then $e \in E(T)$.*

Proof. We first notice that for any edge e' of either P_1 or P_2, there is only one path in $G' - e'$ of length $\leq t$ between the two ends of e'. Furthermore, this unique path contains edge e. It follows that if e is not in T, then all edges of P_1 and P_2 would have to be in T. However, P_1 and P_2 form a cycle, which contradicts T being a tree. □

From now on, by forcing an edge, we mean adding two distinct t-paths between the two ends of the edge. Such an edge will be called a *forced edge,* and the two t-paths will be called *forcing paths.* Denote $|U|$ by n and $|C|$ by m. The graph G is constructed as follows.

For each variable $u_i \in U, 1 \leq i \leq n$, construct a graph H_i by

1. taking five vertices $x_i, u_i, \bar{u}_i, y_i$ and z_i,
2. adding edges $x_iy_i, x_iu_i, x_i\bar{u}_i, z_iu_i$, and $z_i\bar{u}_i$,
3. joining y_i with z_i by a $(t-2)$-path (all internal vertices on the path are new vertices) and forcing every edge on the path, and
4. joining u_i with $\bar{u}_i$ by a $(t-3)$-path (all internal vertices on the path are also new vertices) and forcing every edge on the path as well.

Figure 7 shows the graph H_i for $t = 4$. Next, put $H_1, \ldots, H_n$ together by identifying vertices $x_1, \ldots, x_n$ into a single vertex x to form the variable setting component H. Vertices u_i and $\bar{u}_i$ of H_i will be used to represent the literals u_i and $\bar{u}_i$, respectively, and they are called *literal vertices.*

For each clause $c_j \in C, 1 \leq j \leq m$, create a new vertex c_j, called a *clause vertex,* and add an edge between c_j and each of its three distinct literal vertices in H. Note that each literal vertex is either a vertex u_i or a vertex $\bar{u}_i$.

It is easy to see that G can be constructed in polynomial time. It remains to be shown that C is satisfiable iff G has a tree t-spanner. Before describing the proof, we note the following important property of the graph G, which enables us to define a proper truth assignment for C in terms of a tree t-spanner of G.

LEMMA 4.12. *Any tree t-spanner T of G contains exactly one of the two edges xu_i and $x\bar{u}_i$ for each $1 \leq i \leq n$.*

Proof. Clearly, because T contains the forced path between u_i and $\bar{u}_i$, at most one of xu_i and $x\bar{u}_i$ can be in T (otherwise we would have a cycle in T). We need to show that T contains at least one of xu_i and $x\bar{u}_i$. Suppose that neither xu_i nor $x\bar{u}_i$ is in T. Then the shortest path in $F = G - E(H_i)$ between x and u_i consists of at least three edges (edge xl for some literal l and edges lc and cu_i for some clause c that

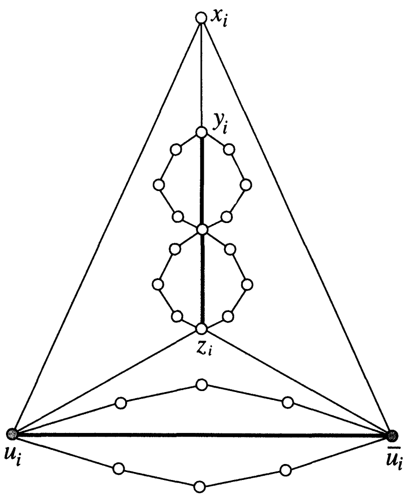

FIG. 7. *Component H_i for $t = 4$.*

contains both l and u_i). Thus $d_F(x, u_i) \geq 3$, and, similarly, $d_F(x, \bar{u}_i) \geq 3$. Consider two cases that depend on whether xy_i is in T.

Case 1. $xy_i \in E(T)$. If neither z_iu_i nor $z_i\bar{u}_i$ is in T, then no tree (x, u_i)-path is present inside H_i and thus

$$d_T(x, u_i) \geq d_F(x, u_i) \geq 3.$$

But then

$$d_T(z_i, u_i) = d_T(z_i, x) + d_T(x, u_i) \geq (t-1) + d_F(x, u_i) \geq t + 2,$$

contrary to T being a t-spanner. Therefore, at least one of z_iu_i and $z_i\bar{u}_i$ is in T. Without loss of generality, we may assume that $z_iu_i \in E(T)$. Note that $z_i\bar{u}_i \notin E(T)$, as there is a forced $(u_i, \bar{u}_i)$-path in T. Since a tree path is unique between any two vertices, we have

$$d_T(x_i, \bar{u}_i) = d_T(x_i, z_i) + d_T(z_i, u_i) + d_T(u_i, \bar{u}_i) = 2t - 3 > t$$

for $t \geq 4$, again a contradiction to T being a t-spanner.

Case 2. $xy_i \notin E(T)$. Then no tree path is present inside H_i between x and u_i or x and $\bar{u}_i$, and thus

$$d_T(x, u_i) \geq d_F(x, u_i) \geq 3$$

and

$$d_T(x, \bar{u}_i) \geq d_F(x, \bar{u}_i) \geq 3.$$

Then

$$d_T(x, z_i) = \min\{d_T(x, u_i) + d_T(u_i, z_i), d_T(x, \bar{u}_i) + d_T(\bar{u}_i, z_i)\} \geq 4.$$

It follows that

$$d_T(x, y_i) = d_T(x, z_i) + d_T(z_i, y_i) \geq t + 2,$$

contrary to T being a t-spanner.

Since both cases lead to contradictions, we conclude that T contains exactly one of two edges xu_i and $x\bar{u}_i$ for each $1 \le i \le n$. □

We now prove that C is satisfiable iff G has a tree t-spanner. Suppose that C is satisfiable and let ξ be a satisfying truth assignment for C. We construct a spanning tree T of G as follows:

1. for each forced edge e, put edge e in T;

2. for each forcing path, arbitrarily delete one edge and then put the remaining edges in T;

3. for each variable $u_i, 1 \le i \le n$, if $\xi(u_i) = 1$, then put edges xu_i and z_iu_i in T; otherwise ($\xi(u_i) = 0$), put edges $x\bar{u}_i$ and $z_i\bar{u}_i$ in T;

4. for each clause $c_j, 1 \le j \le m$, arbitrarily pick a true literal l_j in c_j and put edge c_jl_j in T.

Note that each clause vertex is a leaf in T. It is a routine matter to verify that T is a tree t-spanner.

Conversely, suppose that T is a tree t-spanner of G. We need to present a truth assignment ξ_T that satisfies C. By Lemma 4.12, T contains exactly one of two edges xu_i and $x\bar{u}_i$ for each $1 \le i \le n$. Therefore, we can define a truth assignment ξ_T by setting, for $1 \le i \le n, \xi_T(u_i) = 1$ whenever $xu_i \in E(T)$ and $\xi_T(u_i) = 0$ otherwise. It remains to be shown that ξ_T satisfies C.

Suppose that some clause c_j only contains false literals under ξ_T. Notice that any two literals are joined by a path in T that avoids clause vertices. Therefore, c_j is a leaf in T. Then for a clause edge c_jl_j not in $T, d_T(c_j, l_j) = 2t - 3 \ge t + 1$ for $t \ge 4$, since the distance between any two false literals in T is $2t - 4$, contrary to T being a t-spanner. Therefore, each clause in C contains at least one true literal under ξ_T and thus C is satisfiable. This completes the proof. □

5. Tree spanners in digraphs.

In this section, we consider tree spanners and quasi-tree spanners in digraphs. At first glance, it seems that tree spanner problems on digraphs are at least as hard as tree spanner problems on undirected graphs. Surprisingly, a tree t-spanner of a digraph can be found in almost-linear time. In fact, even a minimum tree spanner (i.e., a tree t-spanner with t as small as possible) of a digraph can be found in almost-linear time. On the other hand, the situation for quasi-tree spanners in digraphs is closer to that for tree spanners in undirected graphs. We will use the results developed in §§3 and 4 for undirected graphs to obtain similar results for quasi-tree spanners.

Throughout this section, $G = (V, A; w)$ is a weighted digraph and $\tilde{G} = (V, E; \tilde{w})$ denotes the underlying undirected graph of G. Recall that for an arc $(x, y) \in A, \tilde{w}(xy) = w((x, y))$ if $(y, x) \notin A$ and $\tilde{w}(xy) = \min\{w((x, y)), w((y, x))\}$ if $(y, x) \in A$. Also, recall that an *in-neighbor* of a vertex x in G is a vertex y such that $(y, x) \in A$ and an *out-neighbor* of x is a vertex z such that $(x, z) \in A$. A vertex v is a *source* if it has no in-neighbors and an *intermediate vertex* if it has both in- and out-neighbors. A spanning subgraph T of G is a spanning tree if T contains no directed cycle and $\tilde{T}$ is a tree. For convenience, we say that G is connected (triconnected) whenever its underlying graph $\tilde{G}$ is connected (triconnected). The meanings of blocks and connected components of G should be understood in the same manner.

5.1. Finding a minimum tree spanner.

Recall that in a digraph G, a vertex x reaches vertex y (i.e., y is reachable from x) if there is a directed (x, y)-path in G. By the definition of a tree spanner, it is easy to see that a spanning tree T of G is a tree spanner iff it preserves reachability of G, i.e., x reaches y in G iff x reaches y in T. Unlike an undirected graph, then, a spanning tree of G is not necessarily a tree spanner. In fact, we have the following result:

LEMMA 5.1. *A digraph G contains at most one tree spanner.*

Proof. Let S and T be two arbitrary tree spanners of G. If $S \neq T$, then there is an arc (x, y) in S that is not in T. Thus there is a directed (x, y)-path P in T, since T is a tree spanner. Let z be an internal vertex of P. Then there is a directed (x, z)-path Q and a directed (z, y)-path Q' in S, since S is a tree spanner. This implies that there are two distinct, directed (x, y)-paths QQ' and xy in S, which contradicts S being a tree. Therefore, $S = T$ and G contains at most one tree spanner. $\square$

Because of the above lemma, we need to consider only the problem of finding the tree spanner T in a digraph G, since T is automatically a minimum tree spanner and we can use it to solve the tree t-spanner problem by comparing t with the stretch index of T. Recall that an *acyclic digraph* is a digraph that contains no directed cycle.

LEMMA 5.2. *If a digraph G admits a tree spanner, then G is acyclic.*

Proof. Let T be a tree spanner of G. Suppose that G contains a directed cycle C. Any two vertices of C are then mutually reachable in T since T is a spanner of G, which contradicts T being a tree. Hence G is acyclic. $\square$

In light of the above lemma, we will hereafter assume that G is acyclic. G then contains a source s. We now present a necessary and sufficient condition for G to admit a tree spanner in terms of $G - s$. Note that $N_G^+(s)$ is the set of out-neighbors of s in G and that a *trivial digraph* (a digraph with a single vertex) is itself a tree spanner.

THEOREM 5.3. *Let G be an acyclic digraph and s be a source of G. G then admits a tree spanner iff each connected component H_i of $G - s$ contains a tree spanner T_i such that there is a vertex $v_i \in V(H_i) \cap N_G^+(s)$ that reaches every vertex of $V(H_i) \cap N_G^+(s)$ through arcs of T_i.*

Proof. If the condition of the theorem is satisfied, then it is readily checked that $T_1 \cup \cdots \cup T_k + \{sv_1, \ldots, sv_k\}$, where k is the number of connected components of $G - s$, is the tree spanner of G, since for $i \neq j$ there are no arcs between H_i and H_j.

Conversely, suppose that G contains a tree spanner T. Then s is a source in T. We first show that each connected component H_i contains a unique vertex v_i adjacent to s in T. Since G is connected, H_i by definition contains at least one such vertex. Suppose that H_i contains two such vertices u and u'. Then there is a (u, u')-path P in $\tilde{H}_i$. Each edge in P corresponds to a unique arc in H_i, as G is acyclic; for each such arc (x, y), there is a directed (x, y)-path P_{xy} in T, since T is a tree spanner. Let $T_i = T \cap H_i$. It is easy to see that P_{xy} lies entirely in T_i. It follows that there is a (u, u')-path P' in $\tilde{T}_i$. However, P' together with su and su' forms a cycle in $\tilde{T}$, a contradiction. Therefore, s is adjacent to a unique vertex v_i of H_i in tree T. Clearly, $v_i \in V(H_i) \cap N_G^+(s)$.

Now, by the definition of H_i, we easily see that T_i is connected; T_i is thus a tree spanner of H_i. Furthermore, for each out-neighbor v of s in H_i, there is a directed (s, v)-path Q in T, since T is a tree spanner. Then the (v_i, v)-section of Q is a directed (v_i, v)-path in T_i, and hence v is reachable from v_i in T_i. $\square$

It is trivial to transform the above theorem into a recursive procedure for finding the tree spanner of G. In order to implement the procedure efficiently, we present an iterative version. Let $\{1, \ldots, n\}$ be the vertex set of G. Since G is acyclic, we can assume that the vertices of G have been topologically ordered, i.e., if (i, j) is an arc of G, then $i < j$. Let G_i denote the subgraph of G induced by vertices $\{i, \ldots, n\}$. Vertex i is then a source of G_i by the definition of the topological ordering of G. Therefore, vertex v_i in Theorem 5.3 is the vertex with smallest number in $V(H_i) \cap N_G^+(s)$. Note that the out-neighbors of vertex i in G_i are the same as those of i in G and hence will be denoted by $N^+(i)$. Figure 8 depicts an acyclic digraph, its tree spanner, and a

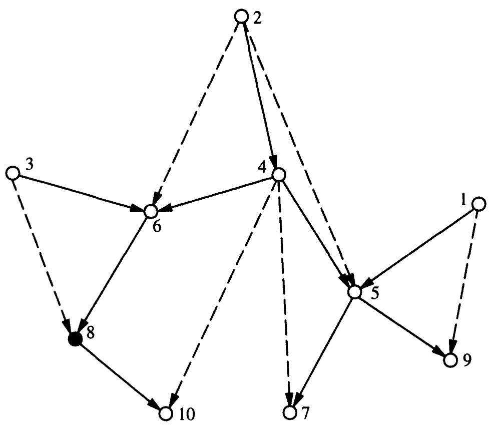

FIG. 8. *An acyclic digraph G, its tree spanner, and a topological ordering of V.*

topological ordering of its vertices. The following procedure finds the tree spanner of G when it contains one.

```
PROCEDURE TREESPANNER($G, T$); {Find the tree spanner $T$ of an acyclic digraph $G$.}
begin
1.  $T \leftarrow$ the trivial tree consisting of vertex $n$;
2.  for $i \leftarrow n-1$ downto 1 do
3.      $\mathcal{H}_i \leftarrow \{H \mid (H$ is a connected component of $G_{i+1}) \wedge (V(H) \cap N^+(i) \neq \emptyset)\}$;
4.      for each $H \in \mathcal{H}_i$ do
5.          $v_{i,H} \leftarrow \min\{j \mid j \in v(H) \cap N^+(i)\}$;
6.          if $v_{i,H}$ reaches every vertex in $V(H) \cap N^+(i)$ through arcs of $T$
              then $T \leftarrow T + (i, v_{i,H})$ else output "No" EXIT;
        end for;
    end for;
7.  return $T$;
end TREESPANNER.
```

Notice that T is a forest during the computation of the above procedure. Let T_i denote the forest T after the normal completion of the $(n-i)$th iteration of the "for" loop at line (2). By Theorem 5.3, it is clear that T_i consists of tree spanners of the connected components of G_i; thus T_1 is the tree spanner of G. However, a straightforward implementation of the procedure may take $O(mn)$ time.

We now refine the procedure to obtain a more efficient algorithm. We notice the following: first, the check at line (6) can be postponed after the completion of the "for" loop at line (2), since $v_{i,H}$ reaches all vertices in $V(H) \cap N^+(i)$ through arcs of T_i iff it reaches these vertices through arcs of T_1; second, the computation at lines (3) and (5) requires only the vertex sets of connected components of G_{i+1}; and third, the connected components of G_i can be obtained from those of G_{i+1} by merging vertex i and all connected components in $\mathcal{H}_i$ into a single component.

Based on the above observations, we use sets to maintain the connected components of G_i. Initially, we have n sets consisting of n single vertices. These sets will be merged to represent the connected components of G_i during the process. Thus line (3) can be carried out by finding all sets containing the out-neighbors of vertex i. We are now ready to present an algorithm that finds the tree spanner of G. The algorithm first decides if G is acyclic, then finds a spanning tree T of G, and finally verifies if T is the tree spanner of G. It also computes the stretch index t of T when T is a tree spanner.

```
ALGORITHM TREE-SPANNER($G, T, t$){Find the tree spanner $T$ of $G$.}
Input: A weighted digraph $G = (V, A; w)$;
Output: The tree spanner $T$ and its stretch index $t$ if $G$ admits a tree spanner;
        otherwise output "No".

begin
1.  if $G$ is not acyclic then output "No" EXIT
       else compute a topological ordering of $G$;
    {Vertices $1, \ldots, n$ is a topological ordering of $G$.}
2.  $T \leftarrow$ the trivial tree consisting of vertex $n$;
3.  Create set $\{i\}$ for each vertex $i$ of $G$;
4.  for $i \leftarrow n-1$ downto 1 do
5.      $\mathcal{H}_i \leftarrow \emptyset$;
6.      for each out-neighbor $k$ of $i$ do
        {Compute $\{H | (H$ is a connected component of $G_{i+1}) \wedge (V(H) \cap N^+(i) \neq \emptyset)\}$.}
7.          Find the set $H_k$ containing vertex $k$;
8.          $\mathcal{H}_i \leftarrow \mathcal{H}_i \cup \{H_k\}$;
        end for;
9.      for each set $H \in \mathcal{H}_i$ do
10.         $v_{i,H} \leftarrow \min\{j | j$ is an out-neighbor of $i$ in $H\}$;
11.         $T \leftarrow T + (i, v_{i,H})$;
        end for;
12.     Merge $\{i\}$ and all $H \in \mathcal{H}_i$ into a single set;
    end for;
13. if $T$ is a tree spanner
       then compute the stretch index $t$ of $T$
       else output "No";
end TREE-SPANNER.
```

We now consider the complexity of the above algorithm. Line (1) takes $O(m+n)$ (cf. [1], [30]). Vertex $v_{i,H}$ at line (10) can be found in $O(|N^+(i)|)$ time by keeping track of the set H_k for each out-neighbor k of i at line (7). By Theorem 2.1(b), line (13) can be carried out in linear time. The merge operation at line (12) and the find operation at line (7) constitute a sequence of union-and-find operations on disjoint sets; there are at most $m+n$ operations in total. By using the well-known "path compression on balanced trees" technique for disjoint set manipulation, these $\leq m+n$ union-and-find operations can be implemented in $O((m+n)\alpha(m+n,n))$ time (cf. [1], [30]), where α is a functional inverse of Ackermann's function and, for all feasible large m and $n, \alpha(m,n) \leq 4$ [30].

The remaining computation takes linear time. The overall running time of the algorithm is thus $O((m+n)\alpha(m+n,n))$. Therefore, we can state the following theorem.

THEOREM 5.4. *The minimum tree spanner of a weighted digraph and its stretch index can be computed in $O((m+n)\alpha(m+n,n))$ time.*

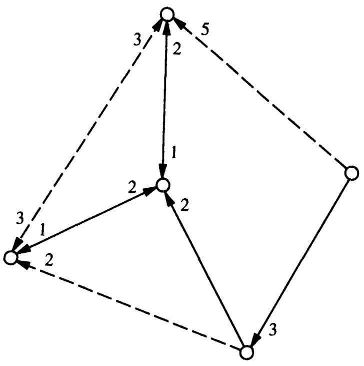

FIG. 9. *A quasi-tree* 1.5-*spanner.*

5.2. Quasi-tree spanners in digraphs. Recall that a quasi-tree of G is a spanning subgraph T such that $\tilde{T}$ is a tree; recall also that T is a quasi-tree t-spanner if it is a t-spanner of G. See Fig. 9 for an example of a quasi-tree spanner. The notion of quasi-tree spanners is intended to capture the underlying tree structure of the spanner. As we will see, results on quasi-tree spanners in digraphs are quite similar to those of tree spanners in undirected graphs. We begin by considering relationships between quasi-tree spanners in G and tree spanners in $\tilde{G}$.

LEMMA 5.5. *Let T be a quasi-tree t-spanner of G. If both arcs (x, y) and (y, x) belong to G, then $(x, y) \in A(T)$ iff $(y, x) \in A(T)$.*

Proof. It suffices to show that $(x, y) \in A(T)$ implies $(y, x) \in A(T)$. Suppose $(y, x) \notin A(T)$. Then there is a directed (y, x)-path P in T. It follows that the corresponding edges of P in $\tilde{T}$ together with edge xy form a cycle in $\tilde{T}$, which contradicts $\tilde{T}$ being a tree. □

LEMMA 5.6. *If T is a quasi-tree t-spanner of G, then $\tilde{T}$ is a tree t-spanner of $\tilde{G}$.*

Proof. Let xy be an arbitrary edge in $\tilde{G}-\tilde{T}$. We need to show $d_{\tilde{T}}(x, y) \leq t \cdot \tilde{w}(xy)$. By the definition of $\tilde{T}$, it is easy to see that $d_{\tilde{T}}(u, v) \leq d_T(u, v)$ for any two vertices $u, v \in V$. If exactly one of arcs (x, y) and (y, x), say (x, y), is in G, then (x, y) is in $G - T$, and thus

$$d_{\tilde{T}}(x, y) \leq d_T(x, y) \leq t \cdot w((x, y)) = t \cdot \tilde{w}(xy),$$

since $\tilde{w}(xy) = w((x, y))$. Otherwise, both (x, y) and (y, x) are arcs in G and thus are in $G - T$ by Lemma 5.5. Then $d_T(x, y) \leq t \cdot w((x, y))$ and $d_T(y, x) \leq t \cdot w((y, x))$. Therefore,

$$d_{\tilde{T}}(x, y) \leq \min\{d_T(x, y), d_T(y, x)\} \leq t \cdot \min\{w((x, y)), w((y, x))\} = t \cdot \tilde{w}(xy).$$

This proves the lemma. □

In light of the above two results, we can use the results on tree spanners in §§3 and 4 to obtain similar results for quasi-tree spanners. Given a weighted undirected graph F, we construct a weighted digraph D by replacing each edge xy of F with two arcs (x, y) and (y, x) and setting $w'((x, y)) = w'((y, x)) = w(xy)$, where w and w' are the weighting functions of F and D, respectively. The following two NP-completeness results can be readily obtained from Theorem 3.9 in §3.2 and Theorem 4.10 in §4.3, respectively, by using Lemmas 5.5 and 5.6.

THEOREM 5.7. *For any fixed $t > 1$, it is NP-complete to determine whether a weighted digraph contains a quasi-tree t-spanner, even if all arcs have integral weights.*

THEOREM 5.8. *For any fixed $t \geq 4$, it is NP-complete to determine whether an unweighted digraph contains a quasi-tree t-spanner.*

On the other hand, the results in §§3.1 and 4.1 can be extended to quasi-tree 1-spanners in weighted digraphs and quasi-tree 2-spanners in unweighted digraphs, respectively.

We first discuss the weighted case. Suppose that G admits a quasi-tree 1-spanner T. Then by Lemma 5.6, $\tilde{T}$ is a tree 1-spanner of $\tilde{G}$. Therefore, $\tilde{T}$ is the unique minimum spanning tree of $\tilde{G}$ by Theorem 3.7 of §3.1. By Lemma 5.5, T is uniquely determined by $\tilde{T}$. Therefore, it is easy to see that the following algorithm finds a quasi-tree 1-spanner in a weighted digraph. First find a minimum spanning tree $\tilde{T}$ of $\tilde{G}$; then construct the maximum quasi-tree T corresponding to $\tilde{T}$ by putting into T, for every edge xy of $\tilde{T}$, all arcs between vertices x and y in G; and finally verify if T is a 1-spanner. Since $\tilde{G}$ can be obtained from G in linear time, $\tilde{T}$ can be found in $O(m \log \beta(m,n))$ time, T can be constructed from $\tilde{T}$ in linear time, and verification takes linear time by Theorem 2.1(c), we have the following result.

THEOREM 5.9. *The quasi-tree 1-spanner of a weighted digraph can be found in $O(m \log \beta(m,n))$ time.*

We now turn our attention to finding a quasi-tree 2-spanner in an unweighted digraph G. We first consider triconnected digraphs. Remember that by a triconnected digraph G, we mean that $\tilde{G}$ is triconnected. Also, bear in mind that a vertex u will be referred to as a universal vertex of G whenever it is a universal vertex of $\tilde{G}$. Finally, recall that an intermediate vertex is any vertex with both in- and out-neighbors.

THEOREM 5.10. *A triconnected digraph G admits a quasi-tree 2-spanner iff it contains a universal vertex u such that, for any intermediate vertex v of $G - u$, both (u,v) and (v,u) are arcs of G.*

Proof. If G contains such a universal vertex u, then it is readily checked that the set of arcs between u and the remaining vertices of G induces a quasi-tree 2-spanner of G. Conversely, suppose that G admits a quasi-tree 2-spanner T. It follows from Lemma 5.6 and Theorem 4.2 that $\tilde{T}$ is a spanning star of $\tilde{G}$ centred at a vertex, say u. Therefore, u is a universal vertex of $\tilde{G}$ and hence of G. Let v be an arbitrary intermediate vertex of G. If (u,v) is an arc of G, then there must be a vertex x such that (v,x) is an arc of G, since v is an intermediate vertex. If $x \neq u$, then (v,x) is an arc of $G - T$. Then there is a directed (v,x)-path P of length 2 in T, since T is a quasi-tree 2-spanner of G. Notice that $\tilde{T}$ is a spanning star. Thus P passes through vertex u, which implies that (v,u) is an arc of G. By a similar argument, we can deduce that (u,v) is an arc of G if (v,u) is an arc of G. □

To illustrate the above theorem, a triconnected digraph and its quasi-tree 2-spanner are depicted in Fig. 10. It is clear that we can use the above theorem to find a quasi-tree 2-spanner in a triconnected digraph. We need only to find all intermediate vertices I and all universal vertices U of G and then check if there is a vertex $u \in U$ such that all possible arcs between u and I appear in G. If such a vertex u exists, then the set of arcs between u and the remaining vertices of G forms a quasi-tree 2-spanner; otherwise, G has no quasi-tree 2-spanner. It is easy to see that this method takes linear time.

It is possible to extend the structural results in §4.1 to quasi-tree 2-spanners and then obtain an algorithm for constructing quasi-tree 2-spanners. However, an easy way to construct a quasi-tree 2-spanner is to use the skeleton tree. Because of Theorem 5.10, we need to consider only a nonseparable digraph G that is not triconnected. As we have mentioned, if G contains a quasi-tree, then $\tilde{G}$ is tree 2-spanner admissible. Then $\tilde{G}$ contains a skeleton tree $\tilde{S} = \mathcal{S}(\tilde{G})$, which corresponds to a subgraph S of G.

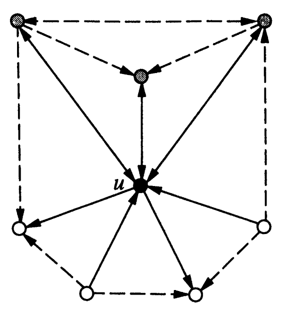

FIG. 10. *A quasi-tree 2-spanner in a triconnected digraph.*

TABLE 2
The complexity status of tree spanner problems.

t	Weighted graphs	Unweighted graphs	Directed graphs
1	$O(m \log \beta(m,n))$	$O(m+n)$	$O((m+n)\alpha(m+n,n))$
(1, 3)	NPc	$O(m+n)$	$O((m+n)\alpha(m+n,n))$
[3, 4)	NPc	?	$O((m+n)\alpha(m+n,n))$
$[4,\infty)$	NPc	NPc	$O((m+n)\alpha(m+n,n))$
∞	$O(m \log \beta(m,n))$	$O(m+n)$	$O((m+n)\alpha(m+n,n))$

Since $\tilde{S}$ belongs to every tree 2-spanner of $\tilde{G}$, by Lemma 5.6, any quasi-tree 2-spanner of G must contain S. Therefore, S must be a 2-spanner of the subgraph of G induced by vertices in S. For a compound leaf C of $\tilde{G}$, let e_C be the unique edge in $\tilde{S}$ with which C is fully joined and $\tilde{H}_C$ be the triconnected component of $\tilde{G}$ containing C. Let $\tilde{L}_C^1$ and $\tilde{L}_C^2$ denote the two leafstalks in $\tilde{H}_C$ with the addition of edge e_C. Denote the subgraphs in G corresponding to $\tilde{H}_C, \tilde{L}_C^1$, and $\tilde{L}_C^2$ by H_C, L_C^1, and L_C^2, respectively. In light of the skeleton tree theorem (Theorem 4.9 in §4.2), for each compound leaf C we need to check only if either L_C^1 or L_C^2 is a 2-spanner of H_C.

To summarize, we outline the quasi-tree 2-spanner algorithm as follows. First, decide if $\tilde{G}$ is tree 2-spanner admissible. If it is, then find all blocks of G. For each block B of G, if it is triconnected, then use Theorem 5.10 to find its quasi-tree 2-spanner; otherwise, construct the skeleton tree $\tilde{S}_B$ of $\tilde{B}$ and the corresponding subgraph S_B in B. Check if S_B is a 2-spanner of $B[V(S_B)]$. Finally, for each compound leaf C of B, check if either L_C^1 or L_C^2 is a 2-spanner of H_C. If G passes all of the above checks, then it contains a quasi-tree 2-spanner; otherwise, it does not. The actual quasi-tree 2-spanner of G can be obtained by keeping track of S_B, L_C^1, and L_C^2.

The correctness of this algorithm follows from our discussions. We now estimate the complexity of the algorithm. It has been shown in §§4.1 and 4.2 that whether $\tilde{G}$ is tree 2-spanner admissible can be decided in linear time and that the skeleton tree of $\tilde{B}$ can be found in linear time. We also mentioned that it takes linear time to find a quasi-tree 2-spanner in B if B is triconnected. Furthermore, checking if S_B is a 2-spanner of $B[V(S_B)]$ takes linear time by Theorem 2.1(c). Since all of the remaining operations can be carried out in linear time as well, the algorithm takes linear time.

THEOREM 5.11. *A quasi-tree 2-spanner in an unweighted digraph can be found in linear time.*

6. Concluding remarks. In this paper, we introduced the notion of tree spanners and studied the theoretical and algorithmic aspects of the subject. In particular, we considered the complexity of tree spanner problems for weighted, unweighted, and directed graphs. The current complexity status of tree spanner problems is summarized in Table 2, where row "∞" indicates the complexity of finding a tree spanner (with minimum weight if G is weighted). The complexity of quasi-tree spanner problems on weighted and unweighted digraphs is the same as that of tree spanner problems on weighted and unweighted graphs, respectively.

Note that the tree 3-spanner problem on unweighted graphs and the quasi-tree 3-spanner problem on unweighted digraphs remain open. We conjecture that the tree 3-spanner problem on unweighted digraphs is NP-complete; if true this would imply the NP-completeness of the quasi-tree 3-spanner problem on unweighted digraphs.

One can also consider the tree t-spanner problem for restricted families of graphs. For partial k-trees, it is easily deduced from the results of Arnborg et al. [4] that the problem is polynomial-time solvable for any fixed t, since it is a monadic second-order problem. However, the problem is open for planar graphs, bounded degree graphs, and many other interesting families of graphs.

In terms of applications, it is desirable to construct tree spanners with small stretch factors. Is there a polynomial-time algorithm for finding a tree t-spanner such that t is close to the stretch factor of the minimum tree spanner? The notion of tree spanners can also be extended to other families of graphs. In general, given a family $\mathcal{F}$ of graphs, one can ask whether a graph G contains a spanner $H \in \mathcal{F}$. The problem is particularly interesting for families of graphs that underlie communication network structures or parallel machine architectures, since graphs that contain these graphs as spanners capture some important properties of jobs that can be carried out on these networks or machines. However, we expect that the problem is hard for most families of graphs.

Acknowledgments. Some material in this paper has previously appeared in the first author's Ph.D. dissertation under the supervision of the second author. The authors thank the referees for useful suggestions.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[2] I. Althöfer, *On optimal realization of finite metric spaces by graphs*, Discrete Comput. Geom., 3 (1988), pp. 103–122.

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

[4] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.

[5] B. Awerbuch, A. Baratz, and D. Peleg, *Efficient broadcast and light-weight spanners*, manuscript, 1992.

[6] S. Bhatt, F. Chung, F. Leighton, and A. Rosenberg, *Optimal simulations of tree machines*, in 27th IEEE Foundations of Computer Science, Toronto, 1986, pp. 274–282.

[7] J. A. Bondy, *Trigraphs*, Discrete Math., 75 (1989), pp. 69–79.

[8] J. A. Bondy and G. Fan, *Cycles in weighted graphs*, Combinatorica, 11 (1991), pp. 191–205.

[9] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, North–Holland, New York, 1976.

[10] L. Cai, *Spanning 2-trees*, manuscript, 1994.

[11] ———, *Tree Spanners: Spanning Trees that Approximate Distances*, Ph.D. thesis, University of Toronto, Toronto, Canada, 1992; Technical Report 260/92, Department of Computer

Science, University of Toronto, 1992.

[12] ———, *NP-completeness of minimum spanner problems*, Discrete Appl. Math., 48 (1994), pp. 187–194.

[13] L. Cai and D. G. Corneil, *Isomorphic tree spanner problems*, Algorithmica, to appear.

[14] L. Cai and J. M. Keil, *Computing visibility information in an inaccurate simple polygon*, Internat. J. Comput. Geom. Appl., submitted.

[15] ———, *Spanners in graphs of bounded degree*, Networks, 24 (1994), pp. 233–249.

[16] L. P. Chew, *There is a planar graph almost as good as the complete graph*, in Proc. 2nd ACM Symposium on Computational Geometry, Yorktown Heights, NY, 1986, pp. 169–177.

[17] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Assoc. Comput. Mach., 34 (1987), pp. 596–615.

[18] H. N. Gabow, Z. Galil, T. H. Spencer, and R. E. Tarjan, *Efficient algorithms for finding minimum spanning trees in undirected and directed graphs*, Combinatorica, 6 (1986), pp. 109–122.

[19] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, San Francisco, 1979.

[20] S. L. Hakimi and S. S. Yau, *Distance matrix of a graph and its realizability*, Quart. Appl. Math., 22 (1964), pp. 305–317.

[21] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[22] J. Hopcroft and R. E. Tarjan, *Dividing a graph into triconnected components*, SIAM J. Comput., 2 (1973), pp. 135–158.

[23] A. L. Liestman and T. Shermer, *Additive graph spanners*, Networks, 23 (1993), pp. 343–364.

[24] ———, *Additive spanners for hypercubes*, Parallel Process. Lett., 1 (1992), pp. 35–42.

[25] ———, *Grid spanners*, Networks, 23 (1993), pp. 123–133.

[26] D. Peleg and A. A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.

[27] D. Peleg and J. D. Ullman, *An optimal synchronizer for the hypercube*, in Proc. 6th ACM Symposium on Principles of Distributed Computing, Vancouver, 1987, pp. 77–85.

[28] D. Peleg and E. Upfal, *A tradeoff between space and efficiency for routing tables*, in Proc. 20th ACM Symposium on Theory of Computing, Chicago, 1988, pp. 43–52.

[29] D. Richards and A. L. Liestman, *Degree-constrained pyramid spanners*. Parallel Distrib. Comput., to appear.

[30] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, 1983.

[31] P. Winkler, *The complexity of metric realization*, SIAM J. Discrete Math., 1 (1988), pp. 552–559.