Tree spanners of bounded degree graphs

Ioannis Papoutsakis
Kastelli Pediados, Heraklion, Crete, Greece, 700 06
February 12, 2022

Abstract

A tree \( t \)-spanner of a graph \( G \) is a spanning tree of \( G \) such that the distance between pairs of vertices in the tree is at most \( t \) times their distance in \( G \). Deciding tree \( t \)-spanner admissible graphs has been proved to be tractable for \( t < 3 \) and NP-complete for \( t > 3 \), while the complexity status of this problem is unresolved when \( t = 3 \). For every \( t > 2 \) and \( b > 0 \), an efficient dynamic programming algorithm to decide tree \( t \)-spanner admissibility of graphs with vertex degrees less than \( b \) is presented. Only for \( t = 3 \), the algorithm remains efficient, when graphs \( G \) with degrees less than \( b \log |V(G)| \) are examined.

Keywords: Tree spanner, Distance, Spanning tree, Efficient graph algorithm, Bounded degree graph.

1 Introduction

A \( t \)-spanner of a graph \( G \) is a spanning subgraph of \( G \), such that the distance between pairs of vertices in the \( t \)-spanner is at most \( t \) times their distance in \( G \). Spanners, when they have a few edges, approximate the distances in the graph, while they are sparse. Spanners of a graph that are trees attain the minimum number of edges a spanner of the graph can have. There are applications of spanners in a variety of areas, such as distributed computing [2, 28], communication networks [20, 27], motion planning and robotics [1, 9], phylogenetic analysis [3] and in embedding finite metric spaces in graphs approximately [30]. In [29] it is mentioned that spanners have applications in approximation algorithms for geometric spaces [19], various approximation algorithms [12] and solving diagonally dominant linear systems [31].

On one hand, in [4, 8, 7] an efficient algorithm to decide tree 2-spanner admissible graphs is presented, where a method to construct all the tree 2-spanners of a graph is also given. On the other hand, in [8, 7] it is proven that for each \( t \geq 4 \) the problem to decide graphs that admit a tree \( t \)-spanner is an NP-complete problem. The complexity status of the tree 3-spanner problem is unresolved. In [13], for every \( t \), an efficient algorithm to determine whether a planar graph with bounded face length admits a tree \( t \)-spanner is presented. In [14] the existence of an efficient (actually linear) algorithm for the tree spanner problem on bounded degree graphs is shown, using a theorem of Logic; while it is mentioned that: “It would be interesting to show that one could use tools that do not rely on Courcelles theorem or Bodlaenders algorithm to speed up practical implementations”. In this article, for every \( t \), an efficient dynamic programming algorithm to decide tree \( t \)-spanner admissibility of bounded degree graphs is presented (theorem 1).

Tree \( t \)-spanners (\( t \geq 3 \)) have been studied for various families of graphs. If a connected graph is a cograph or a split graph or the complement of a bipartite graph, then it admits a tree 3-spanner [7]. Also, all convex bipartite graphs have a tree 3-spanner, which can be constructed in linear time [32]. Efficient algorithms to recognize graphs that admit a tree 3-spanner have been developed for interval, permutation and regular bipartite graphs [17], planar graphs [13], directed path graphs [16], very strongly chordal graphs, 1-split graphs and chordal graphs of diameter at most 2 [6]. In [23] an efficient algorithm to decide if a graph admits a tree 3-spanner of diameter at most 5 is presented. Moreover, every strongly chordal graph admits a tree 4-spanner, which can be constructed in linear time [5]; note that, for each \( t \), there is a connected chordal graph that does not admit any tree \( t \)-spanner. The tree \( t \)-spanner problem has been studied for small diameter chordal graphs [9], diametrically uniform graphs [13], and outerplanar graphs [20]. Approximation algorithms for the tree \( t \)-spanner problem are presented in [11, 27], where in [11] a new necessary condition for a graph to have a tree \( t \)-spanner in terms of decomposition is also presented.
There are NP-completeness results for the tree $t$-spanner problem for families of graphs. In [13], it is shown that it is NP-hard to determine the minimum $t$ for which a planar graph admits a tree $t$-spanner. In [10], it is proved that, for every $t \geq 4$, the problem of finding a tree $t$-spanner is NP-complete on $K_6$-minor-free graphs. For any $t \geq 4$, the tree $t$-spanner problem is NP-complete on chordal graphs of diameter at most $t+1$, when $t$ is even, and of diameter at most $t+2$, when $t$ is odd [9]: note that this refers to the diameter of the graph not to the diameter of the spanner. In [24] it is shown that the problem to determine whether a graph admits a tree $t$-spanner of diameter at most $t+1$ is tractable, when $t \leq 3$, while it is an NP-complete problem, when $t \geq 4$. This last result is used in [25] to hint at the difficulty to approximate the minimum $t$ for which a graph admits a tree $t$-spanner.

The tree $3$-spanner problem is very interesting, since its complexity status is unresolved. In [22] it is shown that the tree $3$-spanner problem can be formulated as an integer programming optimization problem. Constraints for such a formulation appear in [22], providing certificates of tree $3$-spanner inadmissibility for some graphs.

2 Definitions

In general, terminology of [33] is used. If $G$ is a graph, then $V(G)$ is its vertex set and $E(G)$ its edge set. An edge between vertices $u, v \in G$ is denoted as $uv$. Also, $G \setminus \{uv\}$ is the graph that remains when edge $uv$ is removed from $G$. Let $v$ be a vertex of $G$, then $N_G(v)$ is the set of $G$ neighbors of $v$, while $N_G[v]$ is $N_G(v) \cup \{v\}$; in this article, graphs do not have loop edges. The degree of a vertex $v$ in $G$ is the number of edges of $G$ incident to $v$. Here, $\Delta(G)$ is the maximum degree over the vertices of $G$.

Let $G$ and $H$ be two graphs. Then, $G \setminus H$ is graph $G$ without the vertices of $H$, i.e. $V(G \setminus H) = V(G) \setminus V(H)$ and $E(G \setminus H) = \{uv \in E(G) : u \notin V(H) \text{ and } v \notin V(H)\}$. The union of $G$ and $H$, denoted as $G \cup H$, is the graph with vertex set $V(G) \cup V(H)$ and edge set $E(G) \cup E(H)$. Similarly, the intersection of $G$ and $H$, denoted as $G \cap H$, is the graph with vertex set $V(G) \cap V(H)$ and edge set $E(G) \cap E(H)$. Additionally, $G[H]$ is the subgraph of $G$ induced by the vertices of $H$, i.e. $G[H]$ contains all vertices in $V(G) \cap V(H)$ and all the edges of $G$ between vertices in $V(G) \cap V(H)$. Note that the usual definition of induced subgraph refers to $H$ being a subgraph of $G$.

The $G$ distance between two vertices $u, v \in G$ is the length of a $u, v$ shortest path in $G$, while it is infinity, when $u$ and $v$ are not connected in $G$. The definition of a tree $t$-spanner follows.

Definition 1 A graph $T$ is a $t$-spanner of a graph $G$ if and only if $T$ is a subgraph of $G$ and, for every pair $u$ and $v$ of vertices of $G$, if $u$ and $v$ are at distance $d$ from each other in $G$, then $u$ and $v$ are at distance at most $t \cdot d$ from each other in $T$. If $T$ is also a tree, then $T$ is a tree $t$-spanner of $G$.

Note that in order to check that a spanning tree of a graph $G$ is a tree $t$-spanner of $G$, it suffices to examine pairs of vertices that are adjacent in $G$ [7]. There is an additive version of a spanner as well [15] [29], which is not studied in this article. In the algorithm and in the proofs, $r$-centers are frequently used.

Definition 2 Let $r$ be an integer. Vertex $v$ of a graph $G$ is an $r$-center of $G$ if and only if for all vertices $u$ in $G$, the distance from $v$ to $u$ in $G$ is less than or equal to $r$.

To refer to all the vertices near a central vertex, the notion of a sphere is used.

Definition 3 Let $r$ be an integer and $v$ a vertex of a graph $G$. Then, the subgraph of $G$ induced by the vertices of $G$ at distance less than or equal to $r$ from $v$ is the sphere of $G$ with center $v$ and radius $r$; it is denoted as $(v, r)_G$-sphere.

Obviously, $v$ is an $r$-center of a graph $G$, if and only if the $(v, r)_G$-sphere is equal to $G$.

Let $f, g$ be functions from the set of all graphs to the non negative integers. Then, $f$ is $O(g)$ if and only if there are graph $G_0$ and integer $M$ such that $f(G) \leq Mg(G)$ for every $G$ with $|V(G)| > |V(G_0)|$. An algorithm that runs in polynomial time is called efficient.
Algorithm Find_Tree_spanner($G$, $t$)
Input: A connected nonempty graph $G$ and an integer $t > 1$.

1: $A^0_G = \emptyset$
2: For ($k = 1$ to $|V(G)|$) {
3: \hspace{1em} $A^k_G = \emptyset$
4: \hspace{1em} For (vertex $v \in G$) {
5: \hspace{2em} $S_v = \{S \subseteq G : S$ is a tree $t$-spanner of $G[S]$ and $v$ is a $\lfloor \frac{t}{2} \rfloor$-center of $S$\}
6: \hspace{1em} For ($S \in S_v$) {
7: \hspace{2em} $T^k_{v,S} = $ Find_Subtree($G$, $t$, $v$, $S$, $k$, $A^{k-1}_G$)
8: \hspace{2em} $A^k_G = A^k_G \cup \{T^k_{v,S}\}$
9: \hspace{2em} If $(V(G) = V(T^k_{v,S}))$ Return($T^k_{v,S}$) \}
10: Discard $A^{k-1}_G$
11: Return($G$ does not admit a tree $t$-spanner.)

Table 1: Algorithm Find_Tree_spanner($G$, $t$). Procedure Find_Subtree is described in Table 2.

3. Description of the algorithm.

In [21], a characterization of tree $t$-spanner admissible graphs in terms of decomposition states, generally speaking, that if a tree $t$-spanner admissible graph $G$ does not have small diameter then it is the union of two tree $t$-spanner admissible graphs whose intersection is a small diameter subgraph of $G$ (this result requires further definitions to be stated exactly and it is not used in the proofs of this article). So, it may be the case that, starting with small diameter subgraphs and adding on them partial solutions of the remaining graph, a tree $t$-spanner of the whole graph is built.

Algorithm Find_Tree_spanner in Table 1 has as input a graph $G$ and an integer $t > 1$. Its output is a tree $t$-spanner of $G$ or a message that $G$ does not admit any tree $t$-spanner. Being a dynamic programming algorithm, it grows partial solutions into final solutions starting from small subtrees of $G$. Obviously, each such subtree must be a tree $t$-spanner of the subgraph of $G$ induced by the vertices of the subtree. All these subtrees are the first partial solutions of the dynamic programming method and are generated by exhaustive search (first stage of the algorithm). Graphs of bounded degree have vertices of bounded neighborhoods; therefore, this search for small subtrees is no harm. Note that the algorithm works for all input graphs but its efficiency suffers when graphs of big degrees are examined.

In each of the next stages of this dynamic programming algorithm, each partial solution is examined and, then, if possible, it is incremented (procedure Find_Subtree in Table 2). The initial subtree of each partial solution (which was formed in the first stage) is its core. Let $T^k_{v,S}$ be a partial solution that is being examined. Removing the core of $T^k_{v,S}$, which is $S$, from $G$ creates some components. Each such component $Q$ that is not covered so far by $T^k_{v,S}$ is considered. The core of $T^k_{v,S}$ is put together with an appropriate (based on $Q$) portion of nearby partial solutions; if the resulting graph is a tree $t$-spanner of the subgraph of $G$ induced by the vertices of the resulting graph, then the partial solution under examination $T^k_{v,S}$ is incremented by the resulting graph. This increment helps $T^k_{v,S}$ to cover $Q$. If $G$ admits a tree $t$-spanner, then some of the partial solutions eventually cover $G$; if so, the algorithm outputs one of them (line 10 of Table 1). Otherwise, $|V(G)|$ stages suffice to conclude that $G$ does not admit any tree $t$-spanner (line 11 of Table 1). The description of the algorithm in the two tables has some details, which are explained in the following paragraphs.

Let us start with Table 1. Here, $A^k_G$ is set to $\emptyset$ (line 1) and its only use is to call a procedure later on correctly. To give motion to the process of growing partial solutions a main For loop is used (line 2), where variable $k$ is incremented by 1 at the end of each stage, starting from 1. Set $A^k_G$ is to store the progress on partial solutions and it is initialized to $\emptyset$ (line 3); i.e. it is a set whose elements are subtrees of $G$. The first stage ($k=1$) is different from the rest in not having previous partial solutions to merge. First, it is necessary to pick names for the primary partial solutions. For each vertex $v$ of $G$ a set $S_v$ is formed (line 5). Each subgraph $S$ of $G$ that is a tree $t$-spanner of $G[S]$ and has $v$ as a $\lfloor \frac{t}{2} \rfloor$-center becomes an element of $S_v$. This set $S_v$ can be formed by exhaustively checking all the subtrees of the sphere of $G$ with center $v$ and radius $\lfloor \frac{t}{2} \rfloor$ (see lemma 6). Note that the computations to form $S_v$ can be done only for
Procedure Find_Subtree(G, t, v, S, k, $A^k_G$)

Input: A graph $G$, an integer $t > 1$, a vertex $v \in G$, a tree $t$-spanner $S$ of $G[S]$ with $\frac{t}{2}$-center $v$, an integer $k \geq 1$, and a set $A^{k-1}_G$ of subtrees of $G$.

12: If $(k = 1)$
13: \quad $Q_{v,S} = \{Q \subseteq G : Q \text{ is a component of } G \setminus S\}$ \hfill // static
14: \quad Return($S$)
15: else {
16: \quad $T^k_{v,S} = T^k_{v,S}$ \hfill // $T^k_{v,S}$ is in $A^{k-1}_G$
17: \quad For (component $Q \in Q_{v,S}$) {
18: \quad \quad For ($T^k_{u,R} \in A^{k-1}_G$ such that $u \in N_S(v)$) {
19: \quad \quad \quad $T^k_{v,S,u,R,Q} = (T^k_{u,R} \cup R) \setminus ((R \setminus S) \setminus Q)$
20: \quad \quad \quad If ($T^k_{v,S,u,R,Q}$ is a tree $t$-spanner of $G|Q \cup S$) {
21: \quad \quad \quad \quad $T^k_{v,S} = T^k_{v,S} \cup T^k_{v,S,u,R,Q}$
22: \quad \quad \quad \quad $Q_{v,S} = Q_{v,S} \setminus \{Q\}$
23: \quad \quad \quad \quad \quad Break\}} \hfill // Stop search in $A^{k-1}_G$
24: \quad Return($T^k_{v,S}$)

Table 2: Procedure Find_Subtree($G, t, v, S, k, A^{k-1}_G$). In line 13, variable $Q_{v,S}$ has been declared as static; i.e. it is stored locally for later use, when the procedure is called again.

$k = 1$. Then, for each member $S$ of $S_v$, a partial solution is considered under the name $T^1_{v,S}$. Second, each primary partial solution must be initialized (line 7). This is a job for procedure Find_Subtree, which for $k = 1$ returns $S$; i.e. $T^1_{v,S} = S$. Of course, each newly formed primary partial solution is stored in $A^1_G$ (line 8). It may well be the case, when $G$ is a small graph, that some of these primary solutions already spans $G$ (i.e. $V(G) = V(T^1_{v,S})$); then, a tree $t$-spanner of $G$ is found. This completes the first stage of the main loop.

For $k > 1$, partial solutions are merged if possible. Again, all partial solutions are considered one by one. Procedure Find_Subtree in table 2 receives as input (among others) vertex $v$ and subtree $S \in S_v$; these two determine the name of the partial solution under examination $T^k_{v,S}$, where $k$ is just the number of the stage the algorithm is in. It also receives as input all the partial solutions formed in the previous stage of the dynamic programming method through set $A^{k-1}_G$. Procedure Find_Subtree has saved locally the set of components $Q_{v,S}$ of $G \setminus S$, when it was called during the first stage of the main algorithm ($k = 1$). Set $Q_{v,S}$ is a static variable; the content of this set changes and these changes are remembered when the procedure is called again. Another way to put it is that $Q_{v,S}$ is a global variable, which is not lost each time the procedure ends.

The central set of operations of this dynamic programming algorithm is in procedure Find_Subtree, when $k > 1$ (table 2 lines 15 to 24). First, partial solution $T^k_{v,S}$ takes the value that it had in the previous stage, which had been stored in $A^{k-1}_G$; i.e $T^k_{v,S} = T^{k-1}_{v,S}$ (line 16). Then, second, each component $Q$ in $Q_{v,S}$ is examined to check if $T^k_{v,S}$ can be extended towards $Q$ (line 17). Third, this extension will be done using other nearby partial solutions in $A^{k-1}_G$. For this, all partial solutions in $A^{k-1}_G$ that involve as central vertex a neighbor of $v$ in $S$ are considered, one at a time (line 18); the central vertex of partial solution $T^{k-1}_{u,R}$ is $u$.

Fourth, assume that the nearby partial solution $T^{k-1}_{u,R}$ is considered when component $Q$ is examined. Then, in line 19 an auxiliary graph $T^{k-1}_{v,S,u,R,Q}$ is formed (see figure 1). The fundamental part of this auxiliary graph is $T^{k-1}_{u,R}[Q \cup R]$; i.e. the restriction of the considered nearby partial solution to the component under examination (plus its core $R$). The intuition behind this operation is that $u$ may be “closer” than $v$ to $Q$ and, therefore, $T^{k-1}_{u,R}$ may have covered $Q$ in a previous stage of the dynamic programming method. For example, for $k = 2$, input graph $G$ may “end” towards the “direction” of edge $vu$; i.e. $V(Q) \subseteq V(R)$ and, therefore, $T^{k-1}_{u,R}$ covers $Q$. In most of the cases $T^{k-1}_{u,R}$ does not contain all the vertices of $Q$; then, $T^{k-1}_{u,R}[Q \cup R]$ is still meaningful, because of the slightly different than the usual definition of induced subgraph used in this article (see section 2). Having the foundation in hand, i.e. graph $T^{k-1}_{u,R}[Q \cup R]$, subtree $S$ is added to it; lets call for convenience the resulting graph $B$ (i.e.
S
Q
Of course, after all components in $Q$ to form the auxiliary graph.

Then, there is no edge of $(line 20)$ with the next component in $Q_T$ case, component

Finally, procedure $v,S,u,R,Q$ in procedure $Find_{Subtree}$ of table $2$. The left circle is $S$, which is a tree $t$-spanner of $G[S]$ and has $v$ as a $\lfloor \frac{1}{2} \rfloor$-center. Similarly, the right circle is $R$, which is a tree $t$-spanner of $G[R]$ and has $u$ as a $\lfloor \frac{1}{2} \rfloor$-center; here, $u$ is a neighbor of $v$ in $S$. Also, $Q$ is a component of $G \setminus S$. The gray area is $(R \setminus S) \setminus Q$. The vertices that correspond to this gray area are removed from $B = T_{u,R}^{-1}(Q \cup R) \cup S$ to form the auxiliary graph.

$B = T_{u,R}^{-1}(Q \cup R) \cup S$; here, $B$ is not used as a variable within table $2$. Note that $B$ may not even be a tree, because $S \cup R$ may contain a cycle. Since partial solution $T_{v,S}$ is to be incremented towards exactly $Q$, graph $(R \setminus S) \setminus Q$ (gray area in figure $1$) is removed from $B$ to form the auxiliary graph $T_{v,S,u,R,Q}$ (see lemma $1$).

Fifth, if auxiliary graph $T_{v,S,u,R,Q}$ is a tree $t$-spanner of $G[Q \cup S]$, then it is added to the partial solution under examination; i.e. $T_{v,S} = T_{v,S} \cup T_{v,S,u,R,Q}$ (line $21$). The result is again a tree $t$-spanner of the subgraph of $G$ induced by the vertices of the result (see lemma $3$ and lemma $2$). Also, in this case, component $Q$ is removed from $Q_{v,S}$ (line $22$) and the search for nearby partial solutions to grow $T_{v,S}$ towards $Q$ stops (line $23$), since $T_{v,S}$ now covers $Q$. The execution continues after the If statement (line $20$) with the next component in $Q_{v,S}$. Note that auxiliary graph $T_{v,S,u,R,Q}$ can be discarded at this point. These five steps complete the central set of operations of this dynamic programming algorithm. Of course, after all components in $Q_{v,S}$ have been examined, procedure $Find_{Subtree}$ returns $T_{v,S}^{-1}$ to the main program (line $24$).

A few comments on the algorithm follow. The algorithm works for $t = 2$ as well, although there is an efficient algorithm for this case $[3,8,7]$. Maintaining the various sets used in the algorithm is done using linked lists. This way a $For$ loop on elements of a set retrieves sequentially all elements in a linked list. Finally, procedure $Find_{Subtree}$ doesn’t need to check if its input is appropriate.

4 Proof of correctness

The following lemma is employed in various places within this section; when it is used in a proof, a footnote gives the correspondence between the variable names in the proof and the names in its statement below. It describes a basic property of spanners: vertices too far apart in a spanner of a graph cannot be adjacent in the graph. The notion of a sphere has been defined in section $2$.

Lemma 1 Let $G$ be a graph, $T$ a tree $t$-spanner of $G$, and $x$ a vertex of $G$, where $t > 1$. Let $X$ be the $(x, \lfloor \frac{1}{2} \rfloor)_t$-sphere. Let $y$ be a $T$ neighbor of $x$ and let $T_y$ be the component of $T \setminus \{xy\}$ that contains $y$. Then, there is no edge of $G$ from a vertex in $T_y \setminus X$ to a vertex in $(G \setminus T_y) \setminus X$.

Proof. Assume, towards a contradiction, that there is an edge of $G$ between a vertex $p \in T_y \setminus X$ and a vertex $q \in (G \setminus T_y) \setminus X$. Let $P_1$ be the $T$ path from $p$ to $x$. Then, all the vertices of $P_1$ but $x$ are vertices of $T_y$. Also, the length of $P_1$ is strictly greater than $\lfloor \frac{1}{2} \rfloor$, because $X$ contains all the vertices of $T$ at $T$ distance less than or equal to $\lfloor \frac{1}{2} \rfloor$ from $x$ and $p$ is out of $X$. Let $P_2$ be the $T$ path from $q$ to $x$. Then, none of the vertices of $P_2$ is a vertex of $T_y$, because $q \not\in T_y$ and $x \not\in T_y$. Also, the length of $P_2$ is strictly greater than $\lfloor \frac{1}{2} \rfloor$, because $X$ contains all the vertices of $T$ at $T$ distance less than or equal to $\lfloor \frac{1}{2} \rfloor$ from $x$ and $q$ is out of $X$. Then, the $T$ path from $p$ to $q$ has length greater than or equal to $2\lfloor \frac{1}{2} \rfloor + 2$; i.e. the $T$ distance between the endpoints of edge $pq$ of $G$ is strictly greater than $t$. This is a contradiction to $T$ being a tree $t$-spanner of $G$.

1Due to the exhaustive search for primary partial solutions, there is always some $R'$ that will do the job instead of $R$, such that $(R' \setminus S) \setminus Q = \emptyset$. But removing the gray area facilitates the proof of correctness; this way more nearby partial solutions may help the partial solution under examination to grow.
When growing a partial solution $T_1 = T_{v,S}^k$ towards a component $Q$ of $G \setminus S$, by adding an auxiliary graph $T_2 = T_{v,S,u,R,Q}^{k,l}$ (line 21 of table 2), the result $T_1 \cup T_2$ must be a tree $t$-spanner of $G[T_1 \cup T_2]$; the following lemma handles that. The vertices of forest $T_2 \setminus T_1$ are the vertices of $Q$, while the vertices of forest $T_1 \setminus T_2$ are the vertices of the components of $G \setminus S$ that have already been covered by $T_1$; since these two forests correspond to such components, there is no edge of the input graph between them. The intersection $T_1 \cap T_2$ corresponds to the core (the initial value) of partial solution $T_1$, which is $S$.

**Lemma 2** Let $G$ be a graph. Assume that $T_1$ is a tree $t$-spanner of $G[T_1]$ and that $T_2$ is a tree $t$-spanner of $G[T_2]$. Also, assume that there is no edge of $G$ between a vertex in $T_1 \setminus T_2$ and a vertex in $T_2 \setminus T_1$. If $T_1 \cap T_2$ is a nonempty tree, then $T_1 \cup T_2$ is a tree $t$-spanner of $G[T_1 \cup T_2]$.

**Proof.** Let $T_\cup = T_1 \cup T_2$ and $T_\cap = T_1 \cap T_2$. First, it is proved that $T_\cup$ is a tree. Let $Q$ be the components of $T_\cup \setminus T_\cap$. Trivially, each component in $Q$ is either an induced subgraph of $T_1 \setminus T_2$ or an induced subgraph of $T_2 \setminus T_1$.

Consider a component $Q$ in $Q$. Obviously, $T_\cup$ is a connected graph ($T_\cap$ is nonempty). So, there must be at least one edge of $T_\cap$ between $Q$ and $T_\cap$. Without loss of generality, assume that $Q$ is an induced subgraph of $T_1 \setminus T_2$. So, all the edges of $T_\cap$ between $Q$ and $T_\cap$ belong to $T_1$. Therefore, since $Q$ and $T_\cap$ are connected subgraphs of $T_1$ and $T_\cap$ is a tree, there is exactly one edge of $T_\cap$ between $Q$ and $T_\cap$.

Here, $T_\cap$ is an induced subgraph of $T_\cup$, because any extra edge would form a cycle in $T_1$ or in $T_2$. So, $T_\cup$ is a connected graph that consists of $|Q| + 1$ vertex disjoint trees plus $|Q|$ edges; therefore, it is a tree.

Second, it is proved that $T_\cap$ is a $t$-spanner of $G[T_1 \cup T_2]$. Consider an edge $vu$ of $G[T_1 \cup T_2]$. Then, $vu$ is an edge of $G[T_1]$ or an edge of $G[T_2]$, because there is no edge of $G$ between a vertex in $T_1 \setminus T_2$ and a vertex in $T_2 \setminus T_1$. Assume, without loss of generality, that $vu$ is an edge of $G[T_1]$. Since $T_1$ is a tree $t$-spanner of $G[T_1]$, the distance in $T_\cup$ between $v$ and $u$ is at most $t$.

Incrementing of partial solutions is done through a specific command within the algorithm. The following lemma examines one by one the executions of this command and confirms that the incrementing is done properly. For this, a double induction is used.

**Lemma 3** Let $G$ be a graph and $t > 1$ an integer. For every $v \in G$, for every $S \in S_v$ and for every $k$ ($1 \leq k \leq |V(G)|$), $T_{v,S}^k$ is a tree $t$-spanner of $G[T_{v,S}^k]$, where $S_v$ and $T_{v,S}^k$ are constructed in algorithm $\text{Find.Tree.spanner}$ of table 2 on input $G$, $t$).

**Proof.** Since $S \in S_v$, $T_{v,S}^1$ belongs to $A_Q$ and it is equal to $S$, which is a tree $t$-spanner of $G[S]$; so, the lemma holds for $k = 1$. Let $l_k$ be the total number of times $T_{v,S}^k$ is incremented through command

$$T_{v,S}^k = T_{v,S}^k \cup T_{v,S,u,R,Q}^{k,l} \quad (1)$$

in line 21 of table 2. Denote with $T_{v,S}^{k,l}$ the value of variable $T_{v,S}^k$, when $T_{v,S}^k$ has been incremented $l$ times, through command (1) above, where $2 \leq k \leq |V(G)|$. Then, $T_{v,S}^{2,0} = T_{v,S}^1$ and $T_{v,S}^{k,0} = T_{v,S}^{k-1,1}$, where $3 \leq k \leq |V(G)|$ (line 16 of table 2). Also, $T_{v,S}^{k,l} = T_{v,S}^{k,l-1} \cup T_{v,S,u,R,Q}^{k,l}$, where $2 \leq k \leq |V(G)|$ and $1 \leq l \leq l_k$.

First, it is proved, by induction on $l$, that for each $k$ ($2 \leq k \leq |V(G)|$) if $T_{v,S}^{k,0}$ is a tree $t$-spanner of $G[T_{v,S}^0]$, then $T_{v,S}^{k,l}$ is a tree $t$-spanner of $G[T_{v,S}^l]$, where $0 \leq l \leq l_k$. The base case ($l = 0$) holds trivially.

For the induction step ($1 \leq l \leq l_k$), $T_{v,S}^{k,l-1}$ is incremented by $T_2 = T_{v,S,u,R,Q}^{l,k,l-1}$ for some $u$, $R$ and $Q$ to become $T_{v,S}^{k,l}$, i.e. $T_{v,S}^{k,l} = T_{v,S}^{k,l} \cup T_{v,S}^{k,l}$. Then, $T_2$ must be a tree $t$-spanner of $G[Q \cup S]$ (see condition of $\text{IF}$ statement in line 20 of table 2). Also, $T_1$ is a tree $t$-spanner of $G[T_1]$, by induction hypothesis. The vertex set of $T_2$ is $V(S \cup Q)$. Also, the vertex set of $T_1$ is the vertices of $S$ union the vertices of some components of $G \setminus S$ other than $Q$, because $Q$ has not been used to increment the actual solution before. So, there is no edge of $G$ between a vertex in $T_1 \setminus T_2$ and a vertex in $T_2 \setminus T_1$ and $V(T_1 \cap T_2) = V(S)$. By construction of $T_2$, $S$ is a subtree of $T_2$. Also, $T_1$ contains $S$, because $T_1$ contains $T_{v,S,u,R,Q}^{l,k}$, which is equal to $S$. So, $T_1 \cap T_2$ is $S$, which is a nonempty tree. Here, $T_{v,S}^{k,l} = T_1 \cup T_2$. Therefore, by lemma $T_{v,S}^{k,l}$ is a tree $t$-spanner of $G[T_{v,S}^k]$.

Second, the lemma is proved by induction on $k$; i.e. it is proved that $T_{v,S}^{k,l}$ is a tree $t$-spanner of $G[T_{v,S}^k]$, where $2 \leq k \leq |V(G)|$ and $0 \leq l \leq l_k$. For the base case, $T_{v,S}^{2,0}$ is equal to $T_{v,S}^1$, which has been
shown to be a tree $t$-spanner $G[T_{v,S}^1]$. So, by the first induction, $T_{v,S}^{2,l}$ is a tree $t$-spanner of $G[T_{v,S}^{2,l}]$, for $0 \leq l \leq l_k$. For the induction step, $T_{v,S}^{k,0}$ is equal to $T_{v,S}^{k-1,l_{k-1}}$, which, by induction hypothesis, is a tree $t$-spanner $G[T_{v,S}^{k-1,l_{k-1}}]$. So, by the first induction, $T_{v,S}^{k,l}$ is a tree $t$-spanner of $G[T_{v,S}^{k,l}]$, for $0 \leq l \leq l_k$. □

Assume that a partial solution $T_{v,S}^k$ is about to grow towards a component $W$ of $G \setminus S$ with the help of a nearby partial solution $T' = T_{u,R}^{k-1}$. Then, the vertices of $R$ that are not in $S$ nor in $W$ are not needed to grow $T_{v,S}^k$. The following lemma facilitates this process.

**Lemma 4** Let $G$ be a graph and $T$ a tree $t$-spanner of $G$, where $t > 1$. Let $v$ be a vertex of $G$ and let $S$ be the $(v, \lfloor \frac{l}{2} \rfloor)_T$-sphere. Let $u$ be a $T$-neighbor of $v$ and let $R$ be the $(u, \lfloor \frac{l}{2} \rfloor)_T$-sphere. Let $W$ be a component of $G \setminus S$. Let $T'$ be a tree $t$-spanner of $G[T']$, such that $R \subseteq T'$. Let $L$ be $(R \setminus S) \cup W$. If $T'[W \cup R] \cup S$ is a tree $t$-spanner of $G[W \cup S \cup R]$, then $(T'[W \cup R] \cup S) \setminus L$ is a tree $t$-spanner of $G[W \cup S]$.

**Proof.** Obviously, $L$ corresponds to the gray area in figure [1]. Let $g$ be a vertex in $L$. Then, $g \notin S$ and $g \notin W$. So, if there is an edge of $G$ from $g$ to a vertex in $W$, then $g$ must be in $W$, a contradiction. Assume, towards a contradiction, that there is an edge $e$ of $T'[W \cup R] \cup S$ from $g$ to a vertex in $S \cup R$, such that $e \notin E(S \cup R)$. Obviously, $S \cup R$ is a connected graph, because $|\lfloor \frac{l}{2} \rfloor| > 0$. Also, $R \subseteq T'$, so all edges of $S \cup R$ are present in $T'[W \cup R] \cup S$. Therefore, there is a path in $T'[W \cup R] \cup S$ between the endpoints of $e$ that avoids $e$ (note that $L \subseteq R$). This is a contradiction to $T'[W \cup R] \cup S$ being a tree. So, since $L$ does not contain any vertex of $S$, all edges of $T'[W \cup R] \cup S$ incident to $g$ must be edges of $R$ (it was proved earlier that there is no edge of $G$ between $g$ and a vertex in $W$). Here, $g$ must be at distance exactly $|\lfloor \frac{l}{2} \rfloor|$ from $u$, because $g \in R \setminus S$ and $S$ contains all the vertices at $T$ distance less than or equal to $|\lfloor \frac{l}{2} \rfloor| - 1$ from $u$. Therefore, $g$ is incident to only one edge of $T'[W \cup R] \cup S$. So, removing $L$ from $T'[W \cup R] \cup S$ results to a tree $t$-spanner of $G[W \cup S]$. □

The main lemma in the proof of correctness of the algorithm follows. It guarantees that if the input graph admits a tree $t$-spanner, then some partial solutions can grow during some stages of the algorithm. To break its proof into small parts, minor conclusions appear as statements at the end of the paragraph that justifies them and are numbered equation like. Also, intermediate conclusions appear as numbered facts. The lemma is proved by induction on the number of stages of the algorithm (variable $k$ in the main for loop at table [1] see line [2]). Note that the algorithm is ahead of the induction, in the sense that for $k = 2$ the algorithm starts merging primary partial solutions, while the induction considers such merges for $k > \lfloor \frac{l}{2} \rfloor$, as one can see in the proof of fact [3] (for $t \leq 3$, though, the algorithm and the induction are on the same page). The intuition behind the lemma is the following. If a graph admits a tree $t$-spanner $T$, then there is a sphere $S$ of $T$ which is close to leaves of $T$, or to picture it, say that $S$ is close to an end of $T$. Then, a nearby sphere $R$ does cover some of the leaves that $S$ just misses. Here, $S$ corresponds to the partial solution that may grow, while $R$ corresponds to a nearby partial solution $T_{u,R}^k$ that may help it grow. This picture is described formally by fact [4], where $\mathcal{H} = \emptyset$ means that $S$ is close to an end of $T$. After the first steps of the induction, some partial solutions have grown. Assume now that $S$ is not close to some end of $T$. Then, $S$ is nearby to a partial solution $T_{u,R}^k$ which is closer to that end of $T$ than $S$ is. The induction hypothesis hints that, at some earlier stage, $T_{u,R}^k$ covered the part of the input graph from $R$ to that end of $T$. So, $T_{u,R}^k$ may help (see fact [4]) the partial solution that corresponds to $S$ to grow.

**Lemma 5** If $G$ admits a tree $t$-spanner $T$ ($t > 1$) for which there exists vector $(k, v, S, W, u)$ such that:

1. $1 \leq k \leq |V(G)| - 1, v \in V(G),$
2. $S$ is the $(v, \lfloor \frac{l}{2} \rfloor)_T$-sphere,
3. $u$ is a $T$-neighbor of $v$, $T_u$ is the component of $T \setminus \{uv\}$ that contains $u,$
4. $W = \{X \subseteq G : X$ is a component of $G \setminus S$ and $V(X) \subseteq V(T_u)\}$, and
5. $v$ is a $k$-center of $T_u \cup S$,

then algorithm **Find_Tree_Spanner** on input $(G, t)$ (see table [7]) returns a graph or for every component $W \in W$ there exists $R_W \subseteq G$ such that:

- $T_{u,R_w}^k$ is stored in $A_G^k$ of algorithm **Find_Tree_Spanner** on input $(G, t)$ and
\[ T_{v,S,u,R,W}^{k+1} \text{ is a tree } t\text{-spanner of } G[W \cup S], \text{ where } T_{v,S,u,R,W}^{k+1} \text{ is the graph } (T_{v,S,u,R,W}^k [W \cup R_W] \cup S) \setminus ((R_W \setminus S) \setminus W) \text{ (see line 17 of table 3 where such auxiliary graphs are constructed).} \]

**Proof.** Assume that algorithm \texttt{Find Tree spanner} on input \((G, t)\) does not return a graph; then all the stages of the main \texttt{for} loop of the algorithm are executed (line 3 of table 1). The lemma is proved by induction on \(k\). For the base case, \(k \leq \left\lfloor \frac{t}{2} \right\rfloor \). Here, \(S\) is the subgraph of \(T\) that contains all the vertices of \(T\) at \(T\) distance less than or equal to \(\left\lfloor \frac{t}{2} \right\rfloor \) from \(v\). So, all the vertices in \(\bigcup W\) have to be at \(T\) distance strictly greater than \(\left\lfloor \frac{t}{2} \right\rfloor \) from \(v\) each member of \(W\) is a component of \(G \setminus S\). Here, \(v\) is a \(k\) center of \(T_u \cup S\); so, all vertices in \(\bigcup W\) are at \(T\) distance less than or equal to \(k\) from \(v\) (the vertex set of each component in \(W\) is subset of \(T_u\)). So, \(W = \emptyset\) and the lemma holds vacuously.

**Figure 2:** The situation for \(t = 3\). Only edges of \(T\) are shown. The dashed line sets concern vertex sets involved in the induction hypothesis. Note that, when \(t = 3\), one can prove that for each \(y \in X_W\), sets \(W_y\) and \(W_y'\) coincide.

For the induction step, \(\left\lfloor \frac{t}{2} \right\rfloor + 1 \leq k \leq |V(G)| - 1\). Some definitions which are used throughout the proof are introduced in this paragraph. Let \(W\) be a component in \(W\). Let \(R\) be the \(\left\lfloor \frac{t}{2} \right\rfloor \) sphere. Let \(X_W = V(R \cap W)\). Let \(Y_W = \{y \in N_T(u) : \text{there is a vertex } x \in X_W \text{ such that the } T \text{ path from } x \text{ to } u \text{ contains } y\}\). Here, \(v\) is not in \(Y_W\), because \(V(W) \subseteq V(T_u)\). Note that, when \(t \leq 3\), \(X_W = Y_W\) (figure 2). Now, for each \(y \in Y_W\) let \(T_y\) be the component of \(T \setminus \{uy\}\) that contains \(y\). To make use of induction hypothesis appropriate sets of components are defined. For each \(y \in Y_W\) let \(W_y = \{X \subseteq G : X \text{ is a component of } G \setminus R \text{ and } V(X) \subseteq V(T_y)\}\). Since a tree \(t\) spanner for \(G[S \cup W]\) is to be constructed, only the components of each \(W_y\) that fall within \(W\) are of interest. So, for each \(y \in Y_W\), let \(W'_y = \{H \in W_y : H \subseteq W\}\). To refer to all these components, define \(H = \bigcup_{y \in Y_W} W'_y\).

A fundamental reason that the algorithm works is fact 1. It says that each component of \(G \setminus R\) falls either nicely into \(W\) or completely out of \(W\) and, therefore, the induction hypothesis becomes useful (see figure 4). On one hand, \(X_W \subseteq V(W)\), by definition of \(X_W\). Also, for every \(y \in Y_W\), every component in \(W'_y\) is a subgraph of \(W\). Therefore,

\[ V\left(\bigcup H\right) \cup X_W \subseteq V(W) \]

(2)

On the other hand, let \(p\) be a vertex in \(W\). Let \(P\) be the \(T\) path from \(p\) to \(v\) (see figure 3). Since \(V(W) \subseteq V(T_u)\), \(P\) contains \(u\) and all its vertices but \(v\) belong to \(T_u\). All the vertices of \(W\) are at \(T\) distance strictly greater than \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(v\), by the definition of \(S\) (\(W\) is a component of \(G \setminus S\)). So, \(P\) contains exactly one vertex, say vertex \(x\), at \(T\) distance exactly \(\left\lfloor \frac{t}{2} \right\rfloor + 1\) from \(v\) (\(P\) is a sub path of tree \(T\)). This means that \(x\) is at \(T\) distance exactly \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(u\). Therefore, \(x \in R\) (note that \(R\) contains all the vertices at \(T\) distance less than or equal to \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(u\)). Also, all the vertices of \(P\) from \(x\) to \(p\) are at \(T\) distance strictly greater than \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(v\); so, there is a path in \(G \setminus S\) from \(p\) to \(x\). But \(p \in W\); so, \(x \in W\) as well. Then, \(x \in X_W\), because \(X_W = V(R \cap W)\). So,

\[ p \in X_W, \text{ when } p = x \]

(3)

When \(p \neq x\), \(p\) is at \(T\) distance strictly greater than \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(u\), since \(x\) is at \(T\) distance exactly \(\left\lfloor \frac{t}{2} \right\rfloor\) from \(u\); so, \(p \not\in R\). Therefore, \(p\) is in a component, say component \(H_1\), of \(G \setminus R\) (see figure 3). Assume, towards a contradiction, that \(H_1 \not\subseteq W\). Here, \(H_1\) is a component of \(G \setminus R\), \(W\) is a component of \(G \setminus S\), and \(H_1 \cap W \neq \emptyset\). So, since \(H_1 \not\subseteq W\), there must be an edge \(e\) of \(G\) from a vertex in \(H_1 \cap W\) to a vertex in \(S \setminus R\). As it can be seen in figure 3, the \(T\) distance between the endpoints of \(e\) must be bigger than
t, which contradicts to $T$ being a tree $t$-spanner of $G$; formally, lemma $\text{[1]}$ is employed. Here, $S \setminus R$ is a subgraph of the component of $T_v \setminus \{v\}$ that contains $v$ (easily seen by the definitions of $S$ and $R$); call that component $T_v$. So, $S \setminus R \subseteq T_v \setminus R$. Also, $V(W) \subseteq V(T_v)$ and $H_1 \subseteq G \setminus R$; so, $V(H_1 \cap W) \subseteq V(T_v \setminus R)$. But $T_v \cap T_u = \emptyset$; so, $H_1 \cap W \subseteq (G \setminus T_v) \setminus R$. Therefore, $e$ is an edge of $G$ from a vertex in $T_v \setminus R$ to a vertex in $(G \setminus T_v) \setminus R$; this is a contradiction to lemma $\text{[2]}$.

So,

$$H_1 \subseteq W \quad (4)$$

Let $y$ be the neighbor of $u$ in $P$ towards $p$ (note that when $t \leq 3$, $x = y$); then, $y \in Y_W$ (since $x \in P$ and $x \in X_W$) and $p \in T_y$. Assume, towards a contradiction, that there is a vertex of $H_1$ out of $T_y$. Then, there must be an edge of $H_1$ (and of $G$ as well) from a vertex in $T_v \setminus R$ to a vertex in $(G \setminus T_y) \setminus R$, because $H_1$ is a component of $G \setminus R$. This is a contradiction to lemma $\text{[3]}$.

So,

$$V(H_1) \subseteq V(T_y) \quad (5)$$

Here, $H_1$ is a component of $G \setminus R$, such that $V(H_1) \subseteq V(T_y)$ (statement (5)) and $H_1 \subseteq W$ (statement $\text{[4]}$).

So, $H_1 \in W_y$. Therefore, since $y \in Y_W$, $H_1 \in W_y$, and $p \in H_1$, it holds that:

$$p \in V(\bigcup \mathcal{H}), \text{ when } p \neq x \quad (6)$$

Since $p$ is just any vertex in $W$, from statements $\text{(2), (3), and (6)}$, the following holds.

**Fact 1** $V(\bigcup \mathcal{H}) \cup X_W = V(W)$.

---

**Figure 3:** The shapes with fat lines correspond to $S$ (left) and to $W$ (right). The shapes with dashed lines correspond to $R$ (left) and to $H$ (right); also, $H$ may stand for $H_1$ as well, depending on the context. All the $T$ neighbors of $u$ are shown, namely, $v$, $y_1$, $y$, and $y_2$. The dark gray area is set $X_W$. The subtrees of $T$ that correspond to $T_y$ and $T_{y_2}$ are shown gray (including dark gray). Here, $y_1$ is not in $Y_W$. The gray area out of $R$ corresponds to $\bigcup_{y \in Y_W} (\bigcup W_y)$, while the gray area out of $R$ but within $W$ corresponds to $\bigcup H$. Here, $P$ is a $T$ path from $p$ to $v$; note that $P$ contains only one vertex in the dark gray area, namely $x$. Finally, the hatched area is $S \setminus R$.

Sphere $R$ is going to be the required in the conclusion of the lemma $R_W$. Note that, by removing the gray area as shown in figure $\text{[3]}$, sphere $R$ is suitable for every component in $W$, not just $W$. Trivially, $R$ is a tree $t$-spanner of $G[R]$, because it is a subtree of $T$. Also, $u$ is a $\frac{1}{2}t$-center of $R$, because $u$ is defined to be the center of sphere $R$ of radius $\frac{1}{2}t$. Therefore (line $\text{[5]}$ of table $\text{[1]}$), $R \in S_u$ and the following holds (line $\text{[6]}$ of table $\text{[1]}$).

**Fact 2** Algorithm $\text{Find.Tree.spanner}$ on input $(G, t)$ stores $T_{u,R}^k$ in $A_G^k$.

Next, it is proved that $T_{u,S_u,R,W}^{k+1}$ is a tree $t$-spanner of $G[W \cup S]$. For this, two cases are examined.

**On one hand**, consider the case that $\mathcal{H} = \emptyset$. Then, an induction hypothesis cannot be used. In this case, by fact $\text{[1]}$ $V(W) = X_W$. But, $X_W \subseteq V(R)$, by definition. So, $T_{u,R}^k[W \cup R] = T_{u,R}^k[R]$. But $R$ is equal

$^2$Vertex $u$ in the proof corresponds to vertex $x$ in the lemma, $R$ to $X$, $v$ to $y$, and $T_v$ to $T_y$.

$^3$Vertex $u$ in the proof corresponds to vertex $x$ in the lemma and $R$ to $X$. 

to $T_{u,R}^1$, because, for $k = 1$, algorithm Find_Tree_spanner on input $(G, t)$ makes the call Find_Subtree$(G, t, u, R, 1, 0)$ and procedure Find_Subtree returns $R$. Since partial solutions are never reduced during the algorithm, $R \subseteq T_{u,R}^k$. But $T_{u,R}^k$ is a tree, because of lemma \[ \text{Fact 3} \]. So, $T_{u,R}^k[W \cup R] = T_{u,R}^k[R] = R$, because the subgraph of a tree induced by the vertices of a subtree is the subtree. Therefore, $T_{v,S,u,R,W}^{k+1}$ is equal to $(R \cup S) \setminus \{(R \setminus S) \setminus W \}$. By lemma \[ \text{Fact 3} \] which “clears” the gray area in figure \[ \text{Fact 3} \] it suffices to prove that $R \cup S$ is a tree $t$-spanner of $G[W \cup S \cup R]$. Here, $R \cup S$ is a subtree of $T$, so it is a tree $t$-spanner of $G[S \cup R]$. But $V(W) \subseteq R$ in this case; so, $G[S \cup R] = G[W \cup S \cup R]$. Therefore, the following holds.

**Fact 3** $T_{v,S,u,R,W}^{k+1}$ is a tree $t$-spanner of $G[W \cup S]$, when $H = \emptyset$.

On the other hand, consider the case that $H \neq \emptyset$. To prepare for formulations of induction hypothesis, observe that $R$ is closer than $S$ to $W$. Formally, all the vertices in $T_u$ are at $T$ distance less than or equal to $k$ from $v$, because $v$ is a $k$-center of $T_u \cup S$. So, for each $y \in Y_W$, all vertices in $T_y$ are at $T$ distance less than or equal to $k - 1$ from $u$. Also, all the vertices in $R$ are at $T$ distance less than or equal to $k - 1$ from $u$. But in the induction step $k \geq \lceil \frac{k}{2} \rceil + 1$; so, for each $y \in Y_W$, all the vertices in $T_y \cup R$ are at $T$ distance less than or equal to $k - 1$ from $u$ (note that $y \in R$, so $T_y \cup R$ is connected). Therefore

For each $y \in Y_W$, vertex $u$ is a $(k - 1)$-center of $T_y \cup R$ (7)

Vector $(k - 1, u, R, W, y)$ satisfies the five conditions of the lemma for every $y \in Y_W$. To see this, first, $1 \leq \lceil \frac{k}{2} \rceil \leq k - 1 \leq |V(G)| - 2$ and $u \in G$. Second, $R$ has been defined appropriately. Third, $y$ is a $T$ neighbor of $u$ and $T_y$ has been defined appropriately. Fourth, $W_y$ has been defined appropriately. Finally, fifth, $u$ is a $(k - 1)$-center of $T_y \cup R$ (statement (7)). Therefore, since for every $y \in Y_W$ the first coordinate of vector $(k - 1, u, R, W, y)$ is strictly less than $k$, the induction hypothesis states that the conclusion of the lemma holds. Therefore, for every component in $\bigcup_{y \in Y_W} W_y$, the two statements in the conclusion of the lemma hold. But $H \subseteq \bigcup_{y \in Y_W} W_y$. Let $H$ be a component in $H$; then, $H \in W'_y$ for some $y_H \in Y_W$. Therefore, by the induction hypothesis, for $H$, there is $R_H \subseteq G$ such that (see figure 4):

- $T_{y,H,R_H}^{k-1}$ is contained in $A_{G}^{k-1}$ of algorithm Find_Tree_spanner$(G, t)$ and
- $T_{u,R,y,H,R_H}^{k}$ is a tree $t$-spanner of $G[H \cup R]$, where $T_{u,R,y,H,R_H}^{k}$ is the graph $(T_{y,H,R_H}^{k-1}[H \cup R_H] \cup R) \setminus ((R_H \setminus R) \setminus H)$.

**Figure 4**: Part of the situation in the proof of lemma \[ \text{Fact 3} \] as it is seen from two different angles. The shapes with one circular end and one rectangular end correspond to auxiliary graphs. The small rectangle is $S$ and the big rectangle is component $W$ of $G \setminus S$. The dashed circle is $R$; the dashed rectangles are components of $G \setminus R$, which are the elements of $H$ and are named as $H_1$, $H_2$, and $H_3$. Here, $V(W) = V((W \cap R) \cup H_1 \cup H_2 \cup H_3)$ (see fact \[ \text{Fact 3} \]). The induction hypothesis dictates that there are three auxiliary graphs, which are shown on the left hand side, that can help partial solution $T_{u,R}^k$ to cover $H_1$, $H_2$, and $H_3$. On the right hand side, three auxiliary graphs are also shown but are those that procedure Find_Subtree (see table 2) actually picked to increment partial solution $T_{u,R}^k$ towards $H_1$, $H_2$, and $H_3$. Note that the pairwise intersection of the auxiliary graphs on each side is $R$; for example, $T_{H_1} \cap T_{H_2} = R$. As the proof continues, $K = T_{H_1} \cup T_{H_2} \cup T_{H_3}$ becomes the key element in proving the lemma, when $H \neq \emptyset$.

Therefore, when procedure Find_Subtree is called (line 7 of table 2) with input $(G, t, u, R, k, A_{G}^{k-1})$ there is at least one entry in $A_{G}^{k-1}$ to satisfy the condition (line 20 of table 2) for incrementing

\footnote{Note that if $X$ is a connected subgraph of a tree $T$, then the $X$ distance between a pair of vertices of $X$ is equal to the $T$ distance between this pair of vertices.}
$T^k_{u,R}$ towards $H$; note that $y_H \in Y_W \subseteq N_T(u) = N_H(u)$. So (see figure 4), for some $l_H \leq k$, $T^{l_H}_{u,R}$ is incremented towards $H$ and assume that the procedure picks $T_H = T^{l_H}_{u,R;p_H;R(p_H,H),H}$ to do so, where $p_H$ is some $R$ neighbor of $u$ and $T^{l_H-1}_{p_H,R(p_H,H)}$ belongs to $A^{l_H-1}_G$; i.e., $T_H$ is a tree $t$-spanner of $G[H \cup R]$ and the command $T^l_{u,R} = T^l_{u,R} \cup T_H$ is executed (lines 20 and 21 of table 2, since $T_H \subseteq T^l_{u,R}$ and $T^l_{u,R} \subseteq T^k_{u,R}$). The following holds.

Let $K = \bigcup_{H \in \mathcal{H}} T_H$. Then, $K \subseteq T^k_{u,R}$

(8)

For each $H \in \mathcal{H}$ auxiliary graph $T_H$ is a connected graph, because it is a tree $t$-spanner of $G[H \cup R]$. Also, all these auxiliary graphs share the vertices of $R$. So, $K$ is a connected subgraph of $T^k_{u,R}$ (statement 3). But $T^k_{u,R}$ is a tree $t$-spanner of $G[T^k_{u,R}]$, because of lemma 4. So, $K$ is a tree $t$-spanner of $G[K]$. Also, $K$ coincides with $T^k_{u,R}[K]$, because it is a subtree of tree $T^k_{u,R}$. Therefore, the existence of edge $e'$ is a contradiction to lemma 2. Here, $T^k_{u,R}$ is equal to $T^k_{u,R}$, because for each $H \in \mathcal{H}$ auxiliary graph $T_H$ is a connected subgraph of $G[H \cup R]$. Therefore, by fact 1 $V(K) = V(W \cup R)$, because $X_W \subseteq V(R)$. So, $K$ is a tree $t$-spanner of $G[W \cup R]$ and coincides with $T^k_{u,R}[W \cup R]$. The following holds.

$K = T^k_{u,R}[W \cup R]$ is a tree $t$-spanner of $G[W \cup R]$  
(9)

Set $T_2 = R \cup S$. Here, $T_2$ is a connected subgraph of $T$; so, $T_2$ is a tree $t$-spanner of $G[T_2]$. Lemma 2 will be used to prove that $K \cup T_2$ is a tree $t$-spanner of $G[K \cup T_2]$, so the additional requirements for $K$ and $T_2$ are shown. Assume, towards a contradiction, that there is an edge $e'$ of $G$ between a vertex in $K \setminus T_2$ and a vertex in $T_2 \setminus K$ (a similar approach was taken in the proof of statement (3), where tree $T_2$ was defined to be the component of $T \setminus \{uv\}$ that contains $v$). Here, $V(K \setminus T_2)$ is a subset of $V(T_u \setminus R)$, because $V(W) \subseteq V(T_u)$ (here $V(K) = V(W \cup R)$ and $R \subseteq T_2$). So, since $T_u \cap T_a = \emptyset$, it holds that $V(K \setminus T_2) \subseteq V((G \setminus T_a) \setminus S)$. Also, $T_2 \setminus K$ is equal to $S \setminus R$, because $W$ being a component of $G \setminus S$ avoids $S$. Clearly, $V(S \setminus R)$ is a subset of $V(T_u \setminus R)$. Therefore, the existence of edge $e'$ is a contradiction to lemma 2. Here, $T^k_{u,R}$ is equal to $R$, which is a nonempty tree. Therefore, by lemma 2 $K \cup T_2$ is a tree $t$-spanner of $G[K \cup T_2]$. But $K \cup T_2$ is equal to $K \cup S$, because $R \subseteq K$. Therefore (see statement 9), the following holds.

$K \cup S = T^k_{u,R}[W \cup R] \cup S$ is a tree $t$-spanner of $G[W \cup R \cup S]$  
(10)

Here, $T^{k+1}_{v,S,u,R,W}$ is equal to $(T^k_{v,S} \cup R) \setminus ((R \setminus S) \setminus W)$; so, it remains to remove the gray area in figure 1. Then, by statement 10 and lemma 4 the following holds.

**Fact 4** $T^{k+1}_{v,S,u,R,W}$ is a tree $t$-spanner of $G[W \cup S]$, when $\mathcal{H} \neq \emptyset$.

By facts 2, 3, and 4 the lemma holds. □

To explain the time complexity of the algorithm, when graphs of bounded degree are inputs, the following lemma binds the size of various sets used in For loops by functions of the maximum degree of the input graph.

**Lemma 6** Let $G$ be a connected graph of maximum degree $\Delta$ and $t > 1$ an integer. Then,

1. $|S_v| \leq 2^{2^{2^{\lfloor \Delta/2 \rfloor}}},$ for every vertex $v$ of $G$,
2. $|Q_{e,S}| \leq 2^{2^{\lfloor \Delta/2 \rfloor}} + \Delta,$ for every vertex $v$ of $G$ and for every $S$ in $S_v$,
3. $|A^{k-1}_G| \leq |V(G)| \max_{x \in V(G)} |S_x|$, for every $k$ $(2 \leq k \leq |V(G)|)$,
4. the number of $T^{k-1}_{u,R} \in A^{k-1}_G$ such that $u \in N_G(v)$ is at most $\Delta \max_{x \in V(G)} |S_x|$, for every vertex $v$ of $G$, for every $S$ in $S_v$, and for every $k$ $(2 \leq k \leq |V(G)|)$.

---

5 Here, $K$ corresponds to $T_1$ of the lemma.
6 Vertex $u$ in the proof corresponds to vertex $x$ in the lemma, $R$ to $X$, $v$ to $y$, and $T_u$ to $T_v$. 

11
where, $S_v$, $Q_v,S$, $A_{G}^{k-1}$, and $T_{u,R}^{k-1}$ are constructed in algorithm Find_Tree_spanner of tables 1 and 2 on input $(G, t)$.

Proof. First, let $G_v$ be the sphere of $G$ with center $v$ and radius $\lfloor \frac{t}{2} \rfloor$. Let $S$ be a member of $S_v$. Since $v$ is a $\lfloor \frac{t}{2} \rfloor$-center of $S$ (line 7 of table 1), $S$ must be a subgraph of $G_v$. Let $L_i$ denote the number of vertices at $G$ distance exactly $i$ from $v$. Then $L_0 = 1$. Also, $L_i \leq \Delta L_{i-1}$, for $i \geq 1$, because $G$ has maximum degree $\Delta$ and each vertex at $G$ distance $i$ from $v$ must be adjacent to a vertex at $G$ distance $i - 1$ from $v$. So, $G_v$ has at most $\sum_{i=0}^{\lfloor \frac{t}{2} \rfloor} L_i$ vertices. So, simply, $G_v$ has at most $\Delta^{1+\lfloor \frac{t}{2} \rfloor} + 1$ vertices. Therefore, since $G$ has maximum degree $\Delta$, $G_v$ has at most $\Delta |V(G_v)|/2$ edges. So, simply, $G_v$ has at most $\Delta^{2^{1+\lfloor \frac{t}{2} \rfloor} + \lfloor \frac{t}{2} \rfloor}$ edges. At most $|V(G_v)| - 1$ edges can participate in $S$. So, roughly, considering the power set of $E(G_v)$, the number of elements in $S_v$ can be at most $2^{\Delta^{2^{1+\lfloor \frac{t}{2} \rfloor} + \lfloor \frac{t}{2} \rfloor}}$.

Second, any subtree $S$ in $S_v$ can have at most $|V(G_v)|$ vertices. Then, as shown earlier, $S$ can have at most $\Delta^{1+\lfloor \frac{t}{2} \rfloor} + 1$ vertices. The number of edges of $G$ with one endpoint in $S$ and the other out of $S$ can be at most $\Delta |V(S)|$, since $G$ has maximum degree $\Delta$. Then, since $G$ is connected, the number of components of $G \setminus S$ is at most $\Delta^{2^{1+\lfloor \frac{t}{2} \rfloor} + \Delta}$.

Third, for each vertex $x$ of $G$, $|S_x|$ partial solutions are in $A_{G}^{k-1}$, where $2 \leq k \leq |V(G)|$. So, $|A_{G}^{k-1}| \leq |V(G)| \max_{x \in V(G)} |S_x|$. Therefore, G admits a tree $t$-spanner.

Fourth, let $v$ be a vertex of $G$ and $S$ a member of $S_v$. Then, since $G$ has maximum degree $\Delta$, $v$ has at most $\Delta |S_v|$ neighbors in $G$. But each vertex $u$ of $G$ is a central vertex of $|S_v|$ partial solutions. Therefore, the number of $T_{u,R}^{k-1} \in A_{G}^{k-1}$ such that $u \in N_S(v)$ is at most $\Delta \max_{x \in V(G)} |S_x|$, where $2 \leq k \leq |V(G)|$. \square

Theorem 1 Let $b, t$ be positive integers. There is an efficient algorithm to decide whether any graph $G$ with $\Delta(G) \leq b$ admits a tree $t$-spanner.

Proof. If $t = 1$, then a graph admits a tree 1-spanner if and only if it is a tree. The empty graph admits a tree $t$-spanner and a disconnected graph cannot admit a tree $t$-spanner. So, it remains to check nonempty connected graphs for $t > 1$. For this, the algorithm described in this article is employed and it is proved that a nonempty connected graph $G$ admits a tree $t$-spanner if and only if algorithm Find_Tree_spanner on input $(G, t)$ returns a graph, where $t > 1$.

For the sufficiency proof, assume that algorithm Find_Tree_spanner on input $(G, t)$ returns $T_{v,S}^k$ (line 9 of table 1). Then, $V(G) = V(T_{v,S}^k)$. But $T_{v,S}^k$ is a tree $t$-spanner of $G[T_{v,S}^k]$, because of lemma 3. Therefore, $G$ admits a tree $t$-spanner.

For the necessity proof, assume that $G$ admits a tree $t$-spanner $T$. Let $v$ be a vertex of $G$ and $S$ the $(v, \lfloor \frac{t}{2} \rfloor)_t$-sphere. Then, algorithm Find_Tree_spanner on input $(G, t)$ calls procedure Find_Subtree with parameters $(G, t, v, S, 1, A_{G}^0)$ (line 7 of table 1); note that this happens even when $G$ is the one vertex graph. Then, procedure Find_Subtree returns $S$ (line 14 of table 2) which becomes $T_{v,S}^1$.

On one hand, consider the case that $Q_{v,S}$ is empty (line 13 of table 2); Then, $G$ contains no vertices out of $S$; so, the algorithm returns $T_{v,S}^1$.

On the other hand, consider the case that $Q_{v,S}$ is not empty. Assume, towards a contradiction, that algorithm Find_Tree_spanner on input $(G, t)$ does not return a graph. Then, for $k = |V(G)|$ the algorithm Find_Tree_spanner on input $(G, t)$ does not return $T_{v,S}^k$ (line 9 of table 1). This means that $T_{v,S}^k$ does not contain the vertex set of some component $W$ in $Q_{v,S}$. This happens because (lines 17 to 21 of table 2) there is no vertex $x \in N_S(v)$ and $R_W \subseteq G$, such that $T_{x,R_W}^{k-1}$ is contained in set $A_{G}^0$ and $T_{x,S,x,R_W,W}^k$ is a tree $t$-spanner of $G[W \cup S]$, where $T_{x,S,x,R_W,W}^k = (T_{x,R_W}^{k-1}, W \cup R_W \cup S) \setminus ((R_W \setminus S) \setminus W)$. But this is a contradiction to lemma 3 to see this, its five conditions are examined. First, $1 \leq k - 1 \leq |V(G)| - 1$, because $|V(G)| > 1$ in this case; also, $v \notin V(G)$. Second, $S$ is the $(v, \lfloor \frac{t}{2} \rfloor)_t$-sphere. Third, let $u$ be a $T$ neighbor of $v$ that is in a $T$ path from $W$ to $v$; also, let $T_u$ be the component of $T \setminus \{uv\}$ that contains $u$. Fourth, let $W$ be the set $\{X \subseteq G : X$ is a component of $G \setminus S$ and $V(X) \subseteq V(T_u)\}$. Finally, fifth, $v$ is a $(k - 1)$-center of $T_u$, even when $G$ is a path and $v$ an end vertex ($k = |V(G)|$ and $T_u \cup S$ is connected, because $u \in S$). Therefore, vector $(k - 1, v, S, W, u)$ satisfies the conditions of lemma 3.

It suffices to prove that $W \subseteq W$. By definition of vertex $u$, at least one vertex of $W$ belongs to $T_u$. Assume, towards a contradiction, that there is a vertex of $W$ which is not in $T_u$. Then, since $W$ is a component of $G \setminus S$, there must be an edge of $G$ from a vertex in $T_u \setminus S$ to a vertex in $(G \setminus T_u) \setminus S$. This

\[7\]This includes the $\Delta = 1$ cases, because, then, $G_v$ is the one vertex graph ($\Delta = 0$) or has at most two vertices ($\Delta = 1$).

\[8\]The algorithm works for disconnected graphs as well without increasing its time complexity. Here is the only place the connectedness of the input graph is used. This facilitates the calculations for the running time of the algorithm.

12
is a contradiction to lemma \[\text{[7]}\]. Hence, \(W \in \mathcal{W}\). Therefore, by lemma \[\text{[5]}\] there is such a vertex \(x\), namely \(u\), and such an \(R_W\); a contradiction.

It remains to prove that the algorithm runs in polynomial time, when bounded degree graphs are considered. Let \(n(G)\) be the number of vertices in \(G\). Then, checking if the input to algorithm \text{Find Tree spanner} is a connected nonempty graph \(G\) with \(\Delta(G) \leq b\) and \(t > 1\) takes \(O(n)\) time. For every vertex \(v \in G\), \(|S_v|\) is \(O(1)\), because \(\Delta(G)\) is bounded by constant \(b\) and \(t\) is a constant (see lemma \[\text{[5]}\]). So, procedure \text{Find Subtree} in table \[\text{[2]}\] is called \(O(n^2)\) times.

Consider procedure \text{Find Subtree} on input \((G, t, v, S, k, A_G^{k-1})\). The construction of \(Q_{v,S}\) takes \(O(n)\) time \((G\) has a linear number of edges) and it is done only when \(k = 1\). The commands in lines \[\text{[16]}\] to \[\text{[22]}\] of procedure \text{Find Subtree} in table \[\text{[2]}\] are executed when \(k > 1\) and are examined one by one in this paragraph. The number of partial solutions formed in the previous stage \(|A_G^{k-1}|\) is \(O(n)\) (lemma \[\text{[5]}\]), because \(|S_x|\) is \(O(1)\) for each vertex \(x \in G\). Finding \(T_{v,S}^k\) in \(A_G^{k-1}\) and doing the assignment \(T_{v,S}^k = T_{v,S}^{k-1}\) (line \[\text{[16]}\]) takes \(O(n)\) time, because \(|A_G^{k-1}|\) is \(O(n)\). Also, by lemma \[\text{[6]}\], \(|Q_{v,S}|\) is \(O(1)\), because \(\Delta(G)\) is bounded by constant \(b\) and \(t\) is a constant (line \[\text{[17]}\]). Next, it takes \(O(n)\) time to check sequentially all elements of \(A_G^{k-1}\), because \(|A_G^{k-1}|\) is \(O(n)\). Though, (line \[\text{[18]}\]) the number of \(T_{u,R}^{k-1} \in A_G^{k-1}\) such that \(u \in N_S(v)\) is \(O(1)\) (lemma \[\text{[5]}\]), because \(\Delta(G)\) is bounded by constant \(b\) and \(|S_x|\) is \(O(1)\) for each vertex \(x \in G\). Construction of \(T_{u,R}^k\) in \(Q \cup R\) in line \[\text{[19]}\] takes \(O(n)\) time. To check whether \(T_{v,S,u,R,Q}^k\) is a tree \(t\)-spanner of \(G[Q \cup S]\) in line \[\text{[20]}\] takes \(O(n)\) time, because \(\Delta(G)\) is bounded by constant \(b\) and, therefore, \(G[Q \cup S]\) has a linear number of edges. Constructing the union of \(T_{v,S}^k\) and \(T_{v,S,u,R,Q}^k\) in line \[\text{[21]}\] takes \(O(n)\) time. Finally, removing \(Q\) from \(Q_{v,S}\) in line \[\text{[22]}\] can be done in linear time. Therefore, each call of procedure \text{Find Subtree} takes \(O(n)\) time.

Returning back to algorithm \text{Find Tree spanner} (lines \[\text{[7]}\] to \[\text{[9]}\] of table \[\text{[1]}\]), inserting the output of procedure \text{Find Subtree} in \(A_G^k\) takes \(O(n)\) time. Next, checking whether \(V(G) = V(T_{v,S}^k)\) and output \(T_{v,S}^k\) take \(O(n)\) time. All these three commands are executed \(O(n^2)\) times, i.e. as many times as procedure \text{Find Subtree} is called. Therefore, algorithm \text{Find Tree spanner} takes \(O(n^3)\) time and it is efficient. \(\square\)

5 The \(t = 3\) case

Let \(G\) be a graph. When \(t = 3\), for every \(v \in G\), each \(S\) in \(S_v\) must have \(v\) as a 1-center, because \(|S| = 1\) (line \[\text{[5]}\] of table \[\text{[1]}\]). This means that \(S\) is a tree with central vertex \(v\) and all its remaining vertices are leaves. If the maximum degree of \(G\) is \(\Delta\), then \(S\) can have up to \(2^\Delta\) members. So, if graphs with maximum degree at most \(b \log n\) are considered as input to the algorithm (where \(b\) is some constant), then, for each vertex \(v\) of the input graph, the size of \(S_v\) is at most \(2^b \log n = n^b\). So, \(|S_v|\) is polynomially bounded by the number of vertices of the input graph \(n\). Also, all other sets considered in lemma \[\text{[6]}\] are polynomially bounded by \(n\). So, for \(t = 3\) and for every \(b > 0\), the algorithm runs in polynomial time and it is efficient, when graphs \(G\) with degrees less than \(b \log |V(G)|\) are examined.

As mentioned in the introduction, the problem had been solved for \(t = 2\) on general graphs. Now, for \(t > 3\), a tree in \(S_v\) may contain vertices at distance 2 from \(v\). This makes the size of \(S_v\) super-polynomial in \(n\) in the worst case, when graphs with maximum degree at most \(b \log n\) are considered. Therefore, the algorithm is not efficient in this case.

There is some possibility, though, that the \(t = 4\) case is similar to the \(t = 3\) case. The diameter of initial partial solutions (members of \(S_v\), for each vertex \(v\) of the input graph \(G\); line \[\text{[5]}\] of table \[\text{[1]}\]) is at most 2, when \(t = 3\), while it is at most 4, when \(4 \leq t \leq 5\). One can well consider initial partial solutions of diameter at most 3, when \(t = 4\). Then the tree 4-spanner admissibility of graphs with degrees less than \(b \log n\) (where \(b\) is a constant) may be decided efficiently too.

As mentioned in the proof of correctness (see figure \[\text{[2]}\]), the \(t = 3\) case exhibits some structural differences as well, compared to the \(t > 3\) cases. These differences may justify further investigation, in an attempt to resolve the complexity status of the tree 3-spanner problem.

\[^{9}\text{Vertex } v \text{ in the proof corresponds to vertex } x \text{ in the lemma, } S \text{ to } X, u \text{ to } y, \text{ and } T_u \text{ to } T_y.\]
\[^{10}\text{A diameter at most } 3 \text{ subtree of a graph } G \text{ consists of at most one central edge } e \text{ and at most } 2\Delta(G) \text{ edges sharing a vertex with } e. \text{ Therefore, the number of such subtrees with a given central edge is at most } 2^2\Delta(G).\]
6 Notes

Let us hint at the diversity of the tree spanners that the algorithm can produce, with a possible application. Assume that a tree \( t \)-spanner of a graph \( G \) is needed but it must contain certain edges of \( G \). Assume that these necessary edges form a tree \( A \) of diameter at most \( 2\lfloor \frac{t}{2} \rfloor \). Hence, there is a vertex \( a \) of \( G \) that is a \( \lfloor \frac{t}{2} \rfloor \)-center of \( A \). Because of its small diameter, \( A \) is a tree \( t \)-spanner of \( G[A] \). So, \( A \) will be in \( S_a \), when algorithm \textbf{Find Tree spanner} is run on input \((G, t)\). To output only a suitable spanner, line \( 9 \) of table \( 1 \) must be changed to

\[
\text{If } (V(G) = V(T_{v,S}^k) \text{ and } S = A) \text{ Return}(T_{v,S}^k) \}
\]

Then, the algorithm outputs a graph if and only if there is a tree \( t \)-spanner of \( G \) that contains \( A \).

The only reason that the algorithm is not efficient for general graphs is the huge size of sets \( S_v \) even for a few vertices \( v \) of the input graph \( G \). A promising research direction is to consider a family of input graphs for which one can prune these sets down to manageable sizes; i.e. when the algorithm constructs set \( S_v \) for each vertex \( v \) of input graph \( G \) (line \( 5 \) of table \( 1 \)), an efficient procedure may rule out many subtrees of \( G \) that are not needed to build a final solution, because of some properties of \( G \).

References

[1] Srinivasa Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel Smid, and Christos D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In Algorithms—ESA ’96 (Barcelona), pages 514–528. Springer, Berlin, 1996.

[2] Baruch Awerbuch. Complexity of network synchronization. Journal of the ACM, 32(4):804–823, October 1985.

[3] Hans-Jürgen Bandelt and Andreas Dress. Reconstructing the shape of a tree from observed dissimilarity data. Adv. in Appl. Math., 7(3):309–343, 1986.

[4] J. A. Bondy. Trigraphs. Discrete Mathematics, 75:69–79, 1989.

[5] Andreas Brandstädt, Victor Chepoi, and Feodor Dragan. Distance approximating trees for chordal and dually chordal graphs. J. Algorithms, 30(1):166–184, 1999.

[6] Andreas Brandstädt, Feodor F. Dragan, Hoàng-Oanh Le, and Van Bang Le. Tree spanners on chordal graphs: complexity and algorithms. Theor. Comput. Sci., 310(1-3):329–354, 2004.

[7] Leizhen Cai. Tree Spanners: Spanning trees that approximate distances. PhD thesis, Department of Computer Science, University of Toronto, 1992. (Available as Technical Report 260/92).

[8] Leizhen Cai and Derek G. Corneil. Tree spanners. SIAM J. of Discrete Mathematics, 8(3):359–378, 1995.

[9] L. Paul Chew. There are planar graphs almost as good as the complete graph. J. Comput. System Sci., 39(2):205–219, 1989. Computational geometry.

[10] Feodor F. Dragan, Fedor V. Fomin, and Petr A. Golovach. Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavík, Iceland, July 7-11, 2008, Proceedings, Part I, chapter Spanners in Sparse Graphs, pages 597–608. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[11] Feodor F. Dragan and Ekkehard Köhler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. Algorithmica, 69(4):884–905, 2014.

[12] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. Journal of Computer and System Sciences, 69(3):485 – 497, 2004. Special Issue on {STOC} 2003.

\[11\]For example, a tree spanner of the underlying graph of a communication network is needed for broadcasting but some connections (edges of the graph) are very reliable and they must be in the spanner.
[13] Sándor P. Fekete and Jana Kremer. Tree spanners in planar graphs. *Discrete Appl. Math.*, 108(1-2):85–103, 2001. International Workshop on Graph-Theoretic Concepts in Computer Science (Smolenice Castle, 1998).

[14] Fedor V. Fomin, Petr A. Golovach, and Erik Jan van Leeuwen. Spanners of bounded degree graphs. *Inf. Process. Lett.*, 111(3):142–144, 2011.

[15] Dieter Kratsch, Hoang-Oanh Le, Haiko Muller, Erich Prisner, and Dorothea Wagner. Additive tree spanners. *SIAM JOURNAL ON DISCRETE MATHEMATICS*, 17:2003, 1998.

[16] Hoàng-Oanh Le and Van Bang Le. Optimal tree 3-spanners in directed path graphs. *Networks*, 34(2):81–87, 1999.

[17] M. S. Mandanlal, G. Venkatesan, and C. Pandu Rangan. Tree 3-spanners on interval, permutation, and regular bipartite graphs. *Information Processing Letters*, 59:97–102, 1996.

[18] Paul Manuel, Bharati Rajan, Indra Rajasingh, and Amutha Alaguvel. Tree spanners, Cayley graphs, and diametrically uniform graphs. In *Graph-Theoretic Concepts in Computer Science, 29th International Workshop, WG 2003, Elspeet, The Netherlands, June 19-21, 2003, Revised Papers*, pages 334–345, 2003.

[19] Giri Narasimhan and Michiel Smid. *Geometric Spanner Networks*. Cambridge University Press, New York, NY, USA, 2007.

[20] N.S. Narayanaswamy and G. Ramakrishna. Tree-spanners in outerplanar graphs via supply demand partition. *Discrete Applied Mathematics*, 195:104 – 109, 2015. 12th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW 2013).

[21] Ioannis Papoutsakis. Two structure theorems on tree spanners. Master’s thesis, Department of Computer Science, University of Toronto, January 1999. (Available at TSpace of the University of Toronto).

[22] Ioannis Papoutsakis. *Tree Spanners of Simple Graphs*. PhD thesis, Department of Computer Science, University of Toronto, 2013. (Available at TSpace of the University of Toronto).

[23] Ioannis Papoutsakis. Tree 3-spanners of diameter at most 5. *CoRR*, abs/1402.3573:, 2014.

[24] Ioannis Papoutsakis. Tree spanners of small diameter. *CoRR*, abs/1503.06063:, 2014.

[25] Ioannis Papoutsakis. On approximating tree spanners that are breadth first search trees. *Journal of Computer and System Sciences*, 82(5):817 – 825, 2016.

[26] D. Peleg and E. Upfal. A tradeoff between space and efficiency for routing tables. In *STOC: ACM Symposium on Theory of Computing (STOC)*, 1988.

[27] David Peleg and Eilon Reshef. Low complexity variants of the arrow distributed directory. *Journal of Computer and System Sciences*, 63(3):474 – 485, 2001.

[28] David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[29] S. Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1):7:1–7:22, 2009.

[30] Y. Rabinovich and R. Raz. Lower bounds on the distortion of embedding finite metric spaces in graphs. *Discrete Comput. Geom.*, 19(1):79–94, 1998.

[31] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC ’04, pages 81–90, New York, NY, USA, 2004. ACM.

[32] G. Venkatesan, U. Rotics, M. S. Madanlal, J. A. Makowsky, and C. Pandu Rangan. Restrictions of minimum spanner problems. *Information and Computation*, 136(2):143–164, 1997.

[33] D. B. West. *Introduction to Graph Theory*. Prentice Hall, Inc., 1996.