On approximating tree spanners that are breadth first search trees

Ioannis Papoutsakis
Kastelli Pediados, Heraklion, Crete, Greece, 700 06

December 29, 2015

Abstract

A tree \( t \)-spanner \( T \) of a graph \( G \) is a spanning tree of \( G \) such that the distance in \( T \) between every pair of vertices is at most \( t \) times the distance in \( G \) between them. There are efficient algorithms that find a tree \( t \cdot O(\log n) \)-spanner of a graph \( G \), when \( G \) admits a tree \( t \)-spanner. In this paper, the search space is narrowed to \( v \)-concentrated spanning trees, a simple family that includes all the breadth first search trees starting from vertex \( v \). In this case, it is not easy to find approximate tree spanners within factor almost \( o(\log n) \). Specifically, let \( m \) and \( t \) be integers, such that \( m > 0 \) and \( t \geq 7 \). If there is an efficient algorithm that receives as input a graph \( G \) and a vertex \( v \) and returns a \( v \)-concentrated tree \( t \cdot o((\log n)^{m/(m+1)}) \)-spanner of \( G \), when \( G \) admits a \( v \)-concentrated tree \( t \)-spanner, then there is an algorithm that decides 3-SAT in quasi-polynomial time.

Keywords. tree spanner, low stretch, hardness of approximation, spanning tree, distance

1 Introduction

A tree \( t \)-spanner \( T \) of a graph \( G \) is a spanning tree of \( G \) such that the distance between every pair of vertices in \( T \) is at most \( t \) times the distance between them in \( G \). There are applications of spanners in a variety of areas, such as distributed computing \([2,19]\), communication networks \([17,18]\), motion planning and robotics \([1,7]\), phylogenetic analysis \([3]\), and in embedding finite metric spaces in graphs approximately \([21]\). In \([20]\) it is mentioned that spanners have applications in approximation algorithms for geometric spaces \([13]\), various approximation algorithms \([10]\) and solving diagonally dominant linear systems \([22]\).

On one hand, in \([4,6]\) an efficient algorithm to decide tree 2-spanner admissible graphs is presented. On the other hand, in \([6]\) it is proved 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.
There are NP-completeness results for the tree $t$-spanner problem for families of graphs. In [11], it is shown that it is NP-hard to determine the minimum $t$ for which a planar graph admits a tree $t$-spanner. 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 [5]; note that this refers to the diameter of the graph not to the diameter of the spanner. In [15] (which is based on a chapter of [14]) 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 $t \geq 4$. The reduction in this last NP-completeness proof is used as a building block for the reduction in this article (see subsection 3.1).

In [11], for every $t$, an efficient algorithm to determine whether a planar graph with bounded face length admits a tree $t$-spanner is presented. Using a theorem of Logic, the existence of an efficient algorithm to decide bounded degree graphs that admit a tree $t$-spanner appears in [12]. Also, for every $t$, an efficient dynamic programming algorithm to decide tree $t$-spanner admissibility of bounded degree graphs appears in [16].

The first non trivial approximation algorithm appears in [9]. There, an efficient algorithm that finds a tree $t \cdot O(\log n)$-spanner, when the input graph admits a tree $t$-spanner, is presented. In [8] a different efficient algorithm achieving similar approximation ratio is presented, using chordal graphs; it is also given a necessary condition for a graph to admit a tree $t$-spanner.

An alternative definition of the problem of deciding tree $t$-spanner admissible graphs is the following. Let $T$ be a spanning tree of a graph $G$. The stretch of a pair of vertices $u, v \in G$ is the ratio of the distance between them in $T$ to the distance between them in $G$. The maximum stretch of $T$ is the maximum stretch over all pairs of vertices of $G$. The Minimum Max-stretch spanning Tree problem (MMST) is finding a spanning tree of minimum maximum stretch; i.e. finding a tree $t$-spanner of a given unweighted graph $G$, such that $G$ does not admit a tree $(t - 1)$-spanner. In [18] it is proved that approximating the MMST problem within a factor better than $1 + \sqrt{2}$ is NP-hard; note that this holds for big values of minimum maximum stretch. In [9], it is also shown that, for sufficiently big $t$, it is hard to find a tree $(t + o(n))$-spanner of a given graph $G$, when $G$ admits a tree $t$-spanner; note that in this case the minimum maximum stretch is approximated additively.

An approximation algorithm has to find a good enough spanning tree of the input graph. In this article, the search space is restricted to $v$-concentrated spanning trees of the input graph $G$, where $v \in G$ (see definition 2). The family of $v$-concentrated spanning trees of a graph $G$ is simple, easy to decide, and contains all the breadth first search spanning trees of $G$ with single source vertex $v$. In this case it is not easy to find approximate tree spanners within factor almost $o(\log n)$. Specifically, let $m$ and $t$ be integers, such that $m > 0$ and $t \geq 7$. Unless there is a quasi-polynomial time algorithm for 3-SAT, there is no efficient algorithm that receives as input a graph $G$ and a vertex $v$ and returns a $v$-concentrated tree $t \cdot o((\log n)^{m/(m+1)})$-spanner of $G$, when $G$ admits
a $v$-concentrated tree $t$-spanner (theorem 1).

## Definitions and lemmas

In general, terminology of [23] 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 $uv$. If $H$ is a subgraph of $G$, then $G[H]$ is the subgraph of $G$ induced by the vertices of $H$, i.e. $G[H]$ contains exactly all the vertices of $H$ and all the edges of $G$ between vertices of $H$.

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 paper we consider graphs without loop edges.

The $G$ distance between two vertices $u, v$ of a connected graph $G$, denoted as $d_G(u, v)$, is the length of a $u,v$ shortest path in $G$. The $G$ distance between a subgraph $X$ of $G$ and a vertex $v$ of $G$ is $\min_{x \in X} d_G(x, v)$ and it is denoted as $d_G(X, v)$. Finally, the $i$th neighborhood of a vertex $v$ of a graph $G$ is defined as $N^i_G[v] = \{x \in V(G) : d_G(v, x) \leq i\}$. The definition of a tree $t$-spanner follows.

**Definition 1** A graph $T$ is a tree $t$-spanner of a graph $G$ if and only if $T$ is a subgraph of $G$ that is a tree 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 $T$.

Note that in order to check whether a spanning tree of a graph $G$ is a tree $t$-spanner of $G$, it suffices to examine pairs of adjacent in $G$ vertices.

To apply the technique introduced in this article, the search space of spanning trees (towards finding a tree $t$-spanner) must be narrowed. It seems that the broadest family of spanning trees this technique can capture is the following.

**Definition 2** Let $G$ be a graph and $v$ one of its vertices. A spanning tree $T$ of $G$ is $v$-concentrated if and only if for every $i$, $T[N^i_G[v]]$ is a connected graph.

Clearly, a breadth first search spanning tree of a graph starting from a vertex $v$ is $v$-concentrated. Also, there can be many $v$-concentrated spanning trees that are not breadth first search spanning trees starting from $v$. Moreover, note that one can prove the following:

**Proposition 1** Let $G$ be a graph that admits a tree $t$-spanner $T$, where $t \geq 1$. For every vertex $v \in G$ and for every $d \geq 0$, the vertices in $N^d_G[v]$ are in the same component of $T[N^d_G[v]]$, where $d' = d + \lceil \frac{t-1}{2} \rceil$.

This proposition hints that every tree $t$-spanner is loosely “concentrated” around each vertex $v$.

---

1 For example clique $K_n$ has only one breadth first search tree starting from a vertex $v$ of $K_n$ but it has super-polynomially on $n$ many $v$-concentrated spanning trees.

2 A $T$ path joining the endpoints of an edge of $G$ between a vertex at distance $d$ ($d > 0$) from $v$ and a vertex at distance $d - 1$ from $v$ can stretch up to $G$ distance $d'$ away from $v$ before returning back.
An instance of 3-SAT is a set of clauses, where each clause is the disjunction of exactly 3 distinct literals; a literal is a boolean variable or its negation. The 3-SAT problem is to decide whether there is a truth assignment to the variables of a given instance, such that all its clauses are satisfied. Note that if a clause contains less than 3 variables, then both a variable and its negation appear in the clause; so, the clause is satisfied by every truth assignment. Therefore, it suffices to examine instances for which each clause contains exactly 3 variables. In this article, it is assumed that each clause of an instance of 3-SAT contains exactly 3 distinct variables.

Let $f$ and $g$ be functions from the set of graphs to the set of non negative integers. Then, $f$ is $O(g)$ if and only if there exist graph $G_0$ and integer $C$, such that $f(G) \leq Cg(G)$, for every graph $G$ with $|V(G)| > |V(G_0)|$. Also, $f$ is $o(g)$ if and only if for every $\epsilon > 0$ there is a graph $G_\epsilon$ such that $f(G) < \epsilon g(G)$ for every graph $G$ with $|V(G)| > |V(G_\epsilon)|$.

To define the running time of an algorithm, assume that the algorithm is implemented by a deterministic Turing machine. For this, objects, such as instances of problems or outputs of algorithms, are encoded as 0-1 strings. For example, instances of 3-SAT can be encoded as 0-1 strings; then, the size of an instance of 3-SAT is the length of its encoding. An algorithm runs in time $f(n)$ if there is a deterministic Turing machine $M$ that implements the algorithm and the time required by $M$ on each input of length $n$ is at most $f(n)$. If an algorithm runs in polynomial time, then the algorithm is called efficient.

3 Description of the reduction

Algorithm reduction is presented in figure 3; it takes as input an instance $\phi$ of 3-SAT and an integer $h > 1$, while it returns a graph $G$. Here, $h$ is a parameter set in the proof of theorem[4] and depends on the number of variables of $\phi$; its choice is crucial for relating the finding of a not too bad approximate tree spanner of $G$ to a low enough running time for deciding satisfiability of $\phi$ upon such a tree spanner. Given $\phi$, graphs are constructed by calling function get_bb in figure[4] which become the building blocks of the final graph $G$. These building blocks are put together in a tree like structure of height $h$.

3.1 Relation to a known NP-complete problem

In [15] it is proved that it is an NP-complete problem to decide whether a graph admits a tree $t$-spanner of diameter at most $t + 1$, for $t \geq 4$. The reduction there is from 3-SAT. It turns out that for $t = 7$, graphs being built for the sake of this NP-completeness reduction can be stacked one on top of the other like building blocks. Then, a final graph $G$ is constructed by stacking building blocks, starting with a path having a central vertex $v$. This way, the difficulty of finding a tree 7-spanner locally propagates, creating a chasm; in any easily[3]

\[\text{Meaning a tree spanner that does not solve the difficult tree 7-spanner problem locally.}\]
found tree spanner of $G$ that is also concentrated around $v$, some two vertices high in a stack are adjacent in $G$ but far apart in the tree spanner.

Note that in [15] it is essential to prove the fact that if a graph $G$ admits a tree $t$-spanner of diameter at most $t + 1$, then $G$ admits a tree $t$-spanner that is a breadth first search tree. For bigger diameters, this fact does not hold; so, the search space for tree spanners must somehow be narrowed to spanning trees that are concentrated around a central vertex.

### 3.2 Formation of building block

Function `get_bb` in figure 1 receives as input an instance $\phi$ of 3-SAT and two integers $i, j$ and constructs a graph. Integers $i, j$ become labels of vertices of the output graph in order to distinguish them among copies of this graph; also, the output graph is denoted as $G_{i,j}$ by the main function that calls `get_bb`. A part of graph $G_{i,j}$ is shown in figure 2.

**Function `get_bb(\phi, i, j)`**

**Input.** A nonempty instance $\phi$ of 3SAT and two integers $i, j$.

$$V_{i,j} = \{v_{i,j}^{\oplus}, v_{i,j}^{\ominus}\}$$

for (variable $x$ of $\phi$) $V_{i,j} = V_{i,j} \cup \{x_{i,j}\}$

for (clause $c$ in $\phi$) {
  for (variable $x$ of $c$) $V_{i,j} = V_{i,j} \cup \{x_{c,j}\}$
  for ($r = 1$ to 8) $V_{i,j} = V_{i,j} \cup \{q_{i,j}^{r,c}\}$
}

$$E_{i,j} = \emptyset$$

for (variable $x$ of $\phi$) $E_{i,j} = E_{i,j} \cup \{x_{i,j}v_{i,j}^{\oplus}, x_{i,j}v_{i,j}^{\ominus}\}$

for (clause $c$ in $\phi$) {
  Let $y, z,$ and $w$ be the variables of $c$
  $g = [y_{i,j}, y_{i,j}^{\oplus}, z_{i,j}, z_{i,j}^{\ominus}, w_{i,j}, w_{i,j}^{\ominus}]$
  for ($k = 1$ to 6) for ($l = 1$ to 8)
    if ($M[k,l] = 1$) $E_{i,j} = E_{i,j} \cup \{g[k]q_{i,j}^{l,c}\}$ /*(1)
  for ($x$ in $\{y, z, w\}$)
    if ($x$ appears positive in $c$) $E_{i,j} = E_{i,j} \cup \{x_{c,j}v_{i,j}^{\ominus}\}$
    else $E_{i,j} = E_{i,j} \cup \{x_{c,j}v_{i,j}^{\oplus}\}$
}

return ($V_{i,j}, E_{i,j}$)

**Figure 1:** Function `get_bb(\phi, i, j)` that forms a graph given an instance $\phi$ of 3-SAT, which becomes a building block of the final graph. Parameters $i, j$ are used to label each building block. Matrix $M$ used in line (1) is defined outside of the function and is presented in section 3.2 (equation 1). Elements in array $g$ and matrix $M$ are numbered starting from 1, not 0. Note that it is essential, as pointed out in section 2, that each clause in $\phi$ contains exactly 3 variables.
The vertex set of $G_{i,j}$ is generated. First, two distinct vertices $v_{i,j}^{\oplus}$ and $v_{i,j}^{\ominus}$ are placed into the vertex set of $G_{i,j}$; these vertices will be used by the reduction algorithm to glue the new building block to the existing construction. Also, $v_{i,j}^{\oplus}$ will “attract” the positive standings of variables in clauses (similarly, $v_{i,j}^{\ominus}$ the negative). Second, each Boolean variable of $\phi$ gives rise to a vertex of $G_{i,j}$; for each variable $x$ of $\phi$ vertex $x_{i,j}$ of $G_{i,j}$ is generated.

Third, for each clause $c$ in $\phi$, 11 new vertices of $G_{i,j}$ are generated. Specifically, 3 vertices are for the presence of each of the 3 variables in $c$ and are distinct from the vertices generated for the variables of $\phi$; these vertices carry the subscript $c$. For example, if $c$ contains variables $y$, $z$, and $w$, then $G_{i,j}$ contains vertices $y_{i,j}^{c}$, $z_{i,j}^{c}$, and $w_{i,j}^{c}$. Additionally, the remaining 8 vertices take letter $q$, are numbered from 1 to 8, and carry the subscript $c$ as well; i.e. $G_{i,j}$ contains vertices $q_{1}^{c}, q_{2}^{c}, \ldots, q_{8}^{c}$.

Then, the edges of $G_{i,j}$ are formed. First the vertex that corresponds to each variable of $\phi$ becomes adjacent to both of the distinct vertices $v_{i,j}^{\oplus}$ and $v_{i,j}^{\ominus}$. Second, for each clause $c$ in $\phi$, edges between the 14 vertices related to $c$ are placed; 11 vertices have been generated for clause $c$ and 3 vertices correspond to variables of $\phi$ that participate in $c$. These 14 vertices are partitioned in two groups; the one group contains all 8 vertices denoted with letter $q$ and the other group the remaining 6 vertices. Each vertex in the group of 8 (the $q$ vertices) took a number when it was created; thusly, the vertices of this group are numbered from 1 to 8. The vertices in the group of 6 are placed in an array $g$ and in this way are numbered from 1 to 6 (figure 1); for example, if $c$ contains variables $y$, $z$, and $w$, then $g = [y_{i,j}^{c}, y_{i,j}^{c}, z_{i,j}^{c}, z_{i,j}^{c}, w_{i,j}^{c}, w_{i,j}^{c}]$. Having numbered the vertices within each group, the adjacencies between the two groups are determined by the following matrix:

$$M = \begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
\end{bmatrix} \quad (1)$$

Matrix $M$ has two main properties. It consists of three pairs of complementary to each other rows (for example, the first row is the complement of the second). Also, if a sub-matrix of $M$ consisting of whole rows of $M$ contains at least one 1 in each column, then the sub-matrix must contain at least one pair of complementary to each other rows.

Third, there are a few more edges incident to vertices related to $c$. To indicate the standing (negation or not) of each variable $x$ of $c$, vertex $x_{i,j}^{c}$ is made adjacent to either $v_{i,j}^{\oplus}$ or $v_{i,j}^{\ominus}$. Note that vertex $x_{i,j}^{c}$ is related only to clause $c$; in contrast, vertex $x_{i,j}$ may be related to many clauses.
Figure 2: Part of graph $G_{i,j}$, being the output of function get_bb. Let $c$ be clause $(y \lor z \lor \neg w)$. Vertices related to clause $c$ plus the distinct vertices $v^i_{\oplus}$ and $v^i_{\ominus}$ are shown. Vertices in the rectangle are the vertices in array $g$. The dashed edges are these determined by matrix $M$. All the edges of $G_{i,j}$ incident to the white vertices are shown in the figure.

Algorithm reduction$(\phi, h)$

Input. A nonempty instance $\phi$ of 3-SAT and an integer $h > 1$.

Let $G$ be the path $q_1, p_1, v, p_2, q_2$

$G^{1,1} = \text{get_bb}(\phi, 1, 1)$

$G = G \cup G^{1,1}$

identify $q_1$ with $v^{1,1}_{\oplus}$; identify $q_2$ with $v^{1,1}_{\ominus}$

$b[1] = 1; j = 0$

for $(i = 2 \text{ to } h)$

for $(s = 1 \text{ to } b[i - 1])$

for (clause $c$ in $\phi$)

for $(r = 1 \text{ to } 7)$

\[ j = j + 1 \]

$G^{i,j} = \text{get_bb}(\phi, i, j)$

$G = G \cup G^{i,j}$

identify $q^{i-1,s}_{r,c}$ with $v^i_{\ominus}$; identify $q^{i-1,s}_{r+1,c}$ with $v^i_{\oplus}$

$b[i] = j; j = 0; \]

return $G$

Figure 3: Algorithm reduction$(\phi, h)$ that forms a graph given an instance $\phi$ of 3-SAT; building blocks formed by function get_bb in figure [1] are put together in a tree like structure of height $h$. Variable $b[i]$ stores the number of building blocks in layer $i$, while variable $s$ iterates over the building blocks of layer $i - 1$. Finally, variable $r$ is used to iterate over consecutive pairs of $q$ vertices related to clause $c$ in building block $G^{i-1,s}$. 
3.3 Putting building blocks together

The construction of the final graph $G$ starts with a path of length 4, having $v$ as its central vertex. Function $\text{get\_bb}$ on input $(\phi, i, j)$ provides building block $G^{i,j}$, where $i$ and $j$ are the indexes of the block. Then, building blocks with various indexes are added in layers to the existing structure. The first index of a building block indicates the layer that the block is placed in; i.e. layer $i$ of $G$ contains exactly all graphs produced by calling $\text{get\_bb}$ on input $(\phi, i, j)$ for various $j$, while executing algorithm $\text{reduction}$ on input $(\phi, h)$.

The first building block, graph $G^{1,1}$, is attached to the endpoints of the primary path by identifying the one endpoint of the path with vertex $v^{1,1}_G$ and the other endpoint with vertex $v^{1,1}_G$. Layer 1 contains only one graph, namely $G^{1,1}$. After this first layer (command for $(i = 2$ to $h)$ in figure 3), at each step of the construction, a new building block is attached to a pair of consecutive $q$ vertices of the previous layer. Specifically, building block $G^{i,j}$ is attached to building block $G^{i-1,s}$ by identifying $q_{r,c}$ with $v^{i,j}_G$ and $q_{s+1,c}$ with $v^{i,j}_G$.

In Figure 4: Part of output graph $G$ of algorithm $\text{reduction}$ in figure 3 for $h = 2$. Only three clauses are involved here, namely $c$, $d$, and $e$, where $c$ is examined first in the for loops, $d$ second, and $e$ last. The table on the right hand side shows the vertices of $G$ that correspond to the numbered vertices in the figure; for example number 2 corresponds to 3 vertices of 3 different building blocks, which all have been identified to one vertex of $G$. Note that there is no building block attached to pair 5, 6 of $q$ vertices of $G^{1,1}$, since these are related to different clauses. The number of building blocks in layer 2 is $b[2]$.
4 Hardness of approximation

Lemma 1 For every satisfiable instance of 3-SAT $\phi$ and for every $h > 1$, graph $G$ returned by algorithm reduction in figure 3 on input $(\phi, h)$ admits a $v$-concentrated tree 7-spanner.

Proof. Let $a$ be a truth assignment that satisfies $\phi$. Let $T$ be the graph returned by algorithm tree 7-spanner in figure 5 on input $(G, \phi, a)$. Part of a building block $G^{i,j}$ of $G$ where edges of $T$ are shown appears in figure 6.

Algorithm tree_7-spanner($G$, $\phi$, $a$)
Input. A graph $G$, an instance $\phi$ of 3-SAT, and a truth assignment $a$.

1. $V = V(G)$; $E = \{q_1p_1, p_1v, vp_2, p_2q_2\}$
2. for (building block $G^{i,j}$ of $G$)
   for (variable $x$ of $\phi$)
     if ($a(x) = 1$) $E = E \cup \{x^{i,j}v^{i,j}_{\oplus}\}$
     else $E = E \cup \{x^{i,j}v^{i,j}_{\ominus}\}$
3. for (clause $c$ in $\phi$)
   for (variable $x$ of $c$)
     $E = E \cup E(G[\{q^{i,j}_{\oplus}, q^{i,j}_{\ominus}, x^{i,j}_{\oplus}\}])$ /*(1)
4. Let $z$ be a variable of $c$ that makes $c$ true through $a$
5. $Q = \bigcup_{r=1}^{8}\{q^{i,j}_{r,c}\}$
6. $E = E \cup E(G[Q \cup \{z^{i,j}_{\oplus}, z^{i,j}_{\ominus}\}])$ /*(2)
7. return $(V, E)$

Figure 5: Algorithm tree_7-spanner($G$, $\phi$, $a$) that constructs a tree 7-spanner of $G$, when $G$ is the output of algorithm reduction($\phi, h$) in figure 3 where $\phi$ is a satisfiable instance of 3-SAT; also, $a$ is a truth assignment that satisfies $\phi$. It is assumed that the building blocks of $G$ are known to this algorithm, as part of the input graph $G$. As described in the proof of lemma 1 the command in line (1) results in adding one edge to $E$, while the command in line (2) results in adding 8 edges to $E$.

An essential fact hinting that the tree 7-spanner problem is solved locally is proved first. Let $G^{i,j}$ be a building block of $G$ and $c$ a clause in $\phi$. Also, let $Q^{i,j}_{c} = \bigcup_{r=1}^{8}\{q^{i,j}_{r,c}\}$. There is one variable $z$ of $c$ which is used to form $T$ edges incident to vertices in $Q^{i,j}_{c}$ within $G^{i,j}$ (see line (2) in figure 5). Here, $z^{i,j}_{c}$ and $z^{i,j}_{c}$ correspond to complementary to each other rows of matrix $M$; so, each vertex in $Q^{i,j}_{c}$ is adjacent to one of $z^{i,j}_{c}$ or $z^{i,j}_{c}$. But $z$ is a variable that makes $c$ true through $a$; so, both of $z^{i,j}_{c}$ and $z^{i,j}_{c}$ are adjacent in $T$ to the same vertex $v^{i,j}_{\oplus}$ or $v^{i,j}_{\ominus}$ (see figure 6). Therefore,
Fact 1 For every building block $G^{i,j}$ of $G$ and for every clause $c$ in $\phi$, the $T$ distance between any two vertices in $Q^{c,i,j}$ is at most 4.

Clearly, $T$ spans $V(G)$. Let $P$ be path $q_1, p_1, v, p_2, q_2$. Let $b(i')$ be the number of building blocks in layer $i'$ of $G$. For every $i$, $0 \leq i \leq h$, let $V^i = V(P) \cup \bigcup_{j'=1}^{b(i')} V(G^{i',j})$; then, let $G^i = G[V^i]$. To picture $G^i$, it is the subgraph of $G$ induced by the first $i$ layers of $G$ plus path $P$. Clearly, $G^0 = P$ and $G^h = G$. It is proved by induction on $i$ that $T[G^i]$ is a $v$-concentrated tree 7-spanner of $G^i$. For the base case, $i = 0$, both of $T[G^0]$ and $G^0$ are equal to path $P$, which has $v$ as its central vertex.

Consider a building block $G^{i,j}$ in layer $i$, where $1 \leq i \leq h$. Let $X$ be the set of variables of $\phi$; also, let $X^{i,j} = \bigcup_{x \in X} \{x^{i,j}\}$. Then, for every $x \in X$, vertex $x^{i,j}$ is adjacent in $T$ to either $v^{i,j}_x$ or $v^{i,j}_z$ depending on the value given by $a$ for $x$. Therefore, $T[X^{i,j} \cup \{v^{i,j}_x, v^{i,j}_z\}]$ consists of exactly two trees, one containing $v^{i,j}_x$ and the other $v^{i,j}_z$. Note that both of edges $x^{i,j}v^{i,j}_x$ and $x^{i,j}v^{i,j}_z$ are edges of $G$; so, $T[X^{i,j} \cup \{v^{i,j}_x, v^{i,j}_z\}]$ is a subgraph of $G$. Towards proving that $T$ is $v$-concentrated, observe that each vertex $u$ in $X^{i,j}$ is adjacent in $T$ to a vertex (namely $v^{i,j}_x$ or $v^{i,j}_z$), which is closer than $u$ to $v$ in $G$.

Figure 6: Part of graph $G^{i,j}$, being a building block of graph $G$ returned by algorithm reduction in figure 5 on input $(\phi, h)$, where $\phi$ is a satisfiable instance of 3-SAT and $h > 1$. Let $c$ be clause $(y \lor z \lor -w)$ of $\phi$. Vertices related to clause $c$ plus the distinct vertices $v^{i,j}_x$ and $v^{i,j}_z$ are shown. Here, $a(y) = 0$, $a(z) = 1$, and $a(w) = 1$, where $a$ is a truth assignment that satisfies $\phi$. Solid edges belong to $T$, which is the tree returned by algorithm tree,7-spanner in figure 5 on input $(G, \phi, a)$; note that $z$ is the only variable that makes $c$ true through $a$. Observe that the $T$ distance between a pair of white vertices (the $q$ vertices) is at most 4.

Let $C$ be the set of clauses of $\phi$. Also, for every clause $c \in C$, let $X_c$ be the set containing the three variables of $c$. Let $X^{i,j}_c = \bigcup_{x \in X_c} \{x^{i,j}\}$. Then (see line (1) in figure 5), every vertex in $X^{i,j}_c$ is adjacent in $T$ to either $v^{i,j}_c$ or $v^{i,j}_e$ with the one and only edge of $G$ between these 3 vertices. Therefore, $T[X^{i,j} \cup \ldots]$

---

4Let $x$ be a variable that appears in $c$. Then, vertex $x^{i,j}$ is adjacent in $G$ to either $v^{i,j}_c$ or $v^{i,j}_e$, depending on the standing (negation or not) of $x$ in $c$ (see figure 5). Also, $v^{i,j}_c$ is not adjacent to $v^{i,j}_e$ in $G$. Therefore, $E(G[\{v^{i,j}_c, v^{i,j}_e, x^{i,j}\}])$ contains only one edge.
efficient algorithm that receives as input a graph \(v\) so, each vertex in \(Q\) form vertex (namely \(v\)) But \(G\) and in such cases these 2 trees have only exceptions when 2 trees are attached to the same vertex of \(T\) formed upon tree \(T\) \(v\) or \(M\) determined by matrix \(d\). Therefore, the \(T[Q]\) consists of exactly two subtrees of \(G\), one containing \(v\) and the other \(v\). Again, each vertex \(u\) in \(Q\) is adjacent in \(T\) to a vertex (namely \(v\) or \(v\)), which is closer than \(v\) to \(v\) in \(G\).

Let \(Q_{e} = \bigcup_{r=1}^{m} \{q_{r,c}\}. \) There is only one variable \(z\) in \(c\) which is used to form \(T\) edges incident to vertices in \(Q_{e}\) within \(G^{i,j}\) (see line (2) in figure 3). But \(z^{i,j}\) and \(z^{i,j}\) correspond to complementary to each other rows of matrix \(M\); so, each vertex in \(Q_{e}\) is adjacent to exactly one of \(c^{i,j}\) or \(c^{i,j}\). Also, note that \(G[Q]\) doesn’t have any edges and \(z^{i,j}\) is not adjacent to \(z^{i,j}\) in \(G\). Therefore, \(T[Q]\) consists of exactly two subtrees of \(G\), one containing \(v^{i,j}\) and the other \(v^{i,j}\), since \(V(G^{i,j}) = X^{i,j} \cup \{v^{i,j}, v^{i,j}\} \cup \bigcup_{c \in C}(X^{i,j} \cup Q^{i,j})\). Finally, each vertex \(u\) in \(Q_{e}\) is adjacent in \(T\) to a vertex (namely \(z^{i,j}\) or \(z^{i,j}\)), which is closer than \(u\) to \(v\) in \(G\).

By induction hypothesis, \(T[G^{-1}]\) is a subtree of \(G\). So, graph \(T[G]\) is formed upon tree \(T[G^{-1}]\) by attaching 0, 1, or 2 subtrees of \(G\) to each vertex of \(T[G^{-1}]\). These attached trees are vertex disjoint to each other, with only exceptions when 2 trees are attached to the same vertex \(u\) and in such cases these 2 trees have only \(u\) as a common vertex. Hence, \(T[G]\) is a subtree of \(G\).

Also, \(T[G]^{-1}\) is not only \(v\)-concentrated but a breadth first search tree of \(G^{i}\) starting from \(v\) as well; by construction of \(T[G]^{-1}\), every vertex \(u\) of \(G^{i}\) (where \(u \neq v\)) is adjacent in \(T[G]^{-1}\) to a vertex closer than \(u\) to \(v\) in \(G\).

If \(i = 1\), then \(G^{i}\) has only one layer and only one building block; in this case \(d_{T}(\{v^{i,j}, v^{i,j}\}) = 4\), because of path \(P\). If \(i > 1\), then \(G^{i, j}\) is attached to some building block \(G^{i-1,s}\), by identifying pair \(v^{i,j}, v^{i,j}\) with a pair of vertices in \(Q^{i-1,s}\), where \(c^{j}\) is a clause in \(\phi\) (similarly to set \(Q^{i, e}\) above, \(Q^{i-1,s} = \bigcup_{r=1}^{8} \{q^{i-1,s}_{r,c}\}\)). Conclusively, by fact [1] \(d_{T}(\{v^{i,j}, v^{i,j}\}) \leq 4\).

By induction hypothesis \(T[G^{-1}]\) is a 7-spanner of \(G^{-1}\). In order to prove that \(T[G]^{-1}\) is a 7-spanner of \(G^{i}\) it suffices to examine \(T\) distances between endpoints of non \(T\) edges of \(G^{i, j}\). Each vertex in \(X^{i, j}\) is at \(T\) distance 1 from \(v^{i,j}\) or \(v^{i,j}\), so the \(T\) distance between endpoints of non \(T\) edges of \(G^{i, j} [X^{i, j} \cup \{v^{i,j}, v^{i,j}\}]\) is at most 5, because \(d_{T}(v^{i,j}, v^{i,j}) \leq 4\). It remains to examine non \(T\) edges determined by matrix \(M\). Each vertex in \(\bigcup_{c \in C} Q^{i, j}\) is at \(T\) distance 2 from \(v^{i,j}\) or \(v^{i,j}\). Also, each vertex in \(X^{i, j} \cup X^{i, j}\) is at \(T\) distance 1 from \(v^{i,j}\) or \(v^{i,j}\). Therefore, the \(T\) distance between a vertex in \(\bigcup_{c \in C} Q^{i, j}\) and a vertex in \(X^{i, j} \cup X^{i, j}\) is at most 7, again because \(d_{T}(v^{i,j}, v^{i,j}) \leq 4\).

**Theorem 1** Let \(m\) and \(t\) be integers, such that \(m > 0\) and \(t \geq 7\). Also, let \(n'\) and \(f\) be functions from the set of graphs to the non negative integers, such that \(n'(G) = |V(G)|\), for every graph \(G\), and \(f\) is \(o((\log n')^{\frac{1}{m+t}})\). If there is an efficient algorithm that receives as input a graph \(G\) and a vertex \(v\) and returns a \(v\)-concentrated tree \(t \cdot f(G)\)-spanner of \(G\), when \(G\) admits a \(v\)-concentrated tree \(t\)-spanner, then there is an algorithm that decides 3-SAT in \(2^{O((\log n')^{m+1})}\) time.

**Proof.** Since \(f\) is \(o((\log n')^{\frac{1}{m+t}})\), for every \(\epsilon > 0\) there is an \(H_{\epsilon}\) such that
$f(H) < \epsilon(\log n'(H))^{\frac{m+1}{m}}$ for every $H$ with $|V(H)| > |V(H_\epsilon)|$. Let $H'$ be the graph $H_\epsilon$ that corresponds to $\epsilon = \frac{1}{\log n'}$. Let $\text{get\_spanner}$ be the approximation algorithm assumed by this theorem. Let $\phi$ be a nonempty instance of 3-SAT. It is proved that algorithm 3-SAT in figure 7 on input $\phi$ returns \text{YES} if and only if $\phi$ is satisfiable.

\begin{algorithm}
\caption{3-SAT($\phi$)}
\textbf{Input.} A nonempty instance $\phi$ of 3-SAT.
\begin{algorithmic}
\State Let $n$ be the number of variables of $\phi$
\State $G$ = reduction($\phi$, $\lceil(\log n)^m\rceil$)
\If{$(|V(G)| \leq |V(H')|)$}
\State solve $\phi$ exhaustively and return appropriately
\EndIf
\State $T$ = get\_spanner($G,v$)
\For{(building block $G_{i,j}$ of $G$)}
\For{(variable $x$ of $\phi$)}
\If{$(x^{i,j}v_{i,j}^\lor \in E(T))$} $a(x) = 1$
\Else{} $a(x) = 0$
\EndIf
\EndFor
\If{(truth assignment $a$ satisfies $\phi$)}
\State return \text{YES}
\EndIf
\EndFor
\State return \text{NO}
\end{algorithmic}
\end{algorithm}

Figure 7: Algorithm 3-SAT($\phi$) receives as input a nonempty instance $\phi$ of 3-SAT and decides whether it is satisfiable. Constant $m$ and graph $H'$ are defined outside of the algorithm; $m$ is a positive integer introduced in theorem 1 while $H'$ is given in the first paragraph of its proof. Algorithm reduction is presented in figure 3. Algorithm get\_spanner is not given explicitly but its existence is assumed by the same theorem. It is assumed that the decomposition of $G$ into building blocks is given too, when $G$ is returned by algorithm reduction.

For the necessity, algorithm 3-SAT returns \text{YES} on input $\phi$, only when it finds a truth assignment that satisfies $\phi$.

For the sufficiency, assume that $\phi$ is satisfiable. Let $n$ be the number of variables\footnote{As pointed out in section 2, each clause of an instance of 3-SAT contains exactly 3 variables.} of $\phi$. So, $\phi$ has at most $8n^3$ clauses. Set $h = \lceil(\log n)^m\rceil$. Let $G$ be the output of algorithm reduction in figure 3 on input ($\phi,h$). Note that $n \geq 3$, because $\phi$ is nonempty and each of its clauses contains 3 distinct variables; so, $h = \lceil(\log n)^m\rceil > 1$. Each building block $G_{i,j}$ of $G$ has at most $n + 88n^3$ vertices, without counting $v_{i,j}^{\lor}$ and $v_{i,j}^{\lor}$, because each variable contributes one vertex and each clause 11 vertices. To each building block in layer $i$, $1 \leq i \leq h - 1$, at most

\footnote{There is a way to encode instances of 3-SAT as 0-1 strings, such that the size of an encoding of an instance $\phi$ is polynomially bounded by the number of the variables in $\phi$. So, because of the log in the description of the running time and the $O$ notation that follows, the size of an instance of 3-SAT can be considered as the number of variables it contains.}

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Variable & Value & Symbol \hline
\end{tabular}
\caption{Table of variables and values.}
\end{table}
56n^3 building blocks are attached, because each clause contributes 7 building blocks. Let b(i) be the number of building blocks of G in layer i. Then, b(1) = 1 and b(i) ≤ 56n^3(i−1), where 2 ≤ i ≤ h. Therefore, since G has h layers, G has at most \((56n^3)^{h-1}\) building blocks. Hence, G has at most \((n + 88n^3)^{h-1} + 5\) vertices, because each block contributes at most \(n + 88n^3\) vertices; plus the 5 vertices of the starting layer. Increasing this quantity in order to make it simpler and substituting \(h\) with \([\log n]^m\), it turns out that G has at most \(2^{12(\log n)^m+1}\) vertices.

By lemma \[\] G admits a \(v\)-concentrated tree 7-spanner; so, G admits a \(v\)-concentrated tree \(t\)-spanner as well. Therefore, algorithm get_spanner on input \((G, v)\) returns a \(v\)-concentrated tree \(tf(G)\)-spanner T of G. Assume, towards a contradiction, that algorithm 3-SAT on input \(\phi\) does not return \text{YES}. Then, first, \(|V(G)| > |V(H')|\), because otherwise the exhaustive search would have found a truth assignment that satisfies \(\phi\). Second, for every building block of G truth assignment \(a\) defined upon this building block and T does not satisfy \(\phi\).

Here, \(H'\) corresponds to \(\epsilon = \frac{4}{12(h+1)}\). So, \(f(G) < \frac{4}{12(h+1)} (\log n'(G))^\frac{m}{m+1}\), because \(|V(G)| > |V(H')|\). But \(n'(G) < 2^{12(\log n)^m+1}\); therefore, \(tf(G) < 4(\log n)^m\). Hence, T is a 4h-spanner of G.

For every \(i, 1 \leq i \leq h\), there is a building block \(G_i^j\) of G in layer i, such that \(d_T(v_i^j, v_i^j) = 4i\). This is proved by induction on i. For \(i = 1\) there is only one layer in layer 1 and \(d_T(v_1^1, v_1^1) = 4\), because of path P: \(v_1^1, q_1, p_1, v, p_2, q_2 = v_1^1\). Note that T is \(v\)-concentrated; so, P is a sub-path of T.

For \(i > 1\), consider layer \(i - 1\). Then, by induction hypothesis, there is a building block \(G_i^{i-1,s}\), such that \(d_T(v_i^{i-1,s}, v_i^{i-1,s}) = 4(i-1)\). Let \(a\) be the truth assignment defined by algorithm 3-SAT upon \(G_i^{i-1,s}\) and T. Since \(a\) does not satisfy \(\phi\) there is a clause c in \(\phi\) which is not true through a.

Let \(X_c\) be the set of the 3 variables that appear in clause c. Let \(X = \bigcup_{x \in X_c} \{x_i^{i-1,s}, x_i^{i-1,s}\}\). Since T is a \(v\)-concentrated spanning tree of G, each vertex in X must be adjacent to exactly one of \(v_i^{i-1,s}\) or \(v_i^{i-1,s}\) in T. This holds because, first, \(v_i^{i-1,s}\) and \(v_i^{i-1,s}\) are the only G neighbors of vertices in X that are at G distance at most \(Td_G(X, v)\) from v (there is no edge of G between vertices in X and all vertices in X are at the same distance from v). Second, vertices within G distance \(d_G(X, v) - 1\) from v (here, \(v_i^{i-1,s}\) and \(v_i^{i-1,s}\) are at G distance \(d_G(X, v) - 1\) from v) induce a connected sub graph of T; so, a vertex in X cannot be adjacent in T to two vertices at G distance \(d_G(X, v) - 1\) from v.

Let \(Q = \bigcup_{r=1}^{\log n} Q_r\). Again, vertices in X are the only G neighbors of vertices in Q that are at G distance at most \(d_G(Q, v)\) from v (graph G[Q] has no edges and all vertices in Q are at the same distance from v). Also, vertices within G distance \(d_G(Q, v) - 1\) from v induce a connected sub graph of T. So, since T is a \(v\)-concentrated spanning tree of G, each vertex in Q is adjacent in T to exactly one vertex in X.

\[\]In a \(v\)-concentrated spanning tree T of a graph G, any vertex at G distance d (d > 0) from v must be adjacent in T to a vertex at G distance at most d from v.
The $G$ edges between $X$ and $Q$ are these determined by matrix $M$. But every sub-matrix of $M$ consisting of whole rows of $M$ must contain two complementary to each other rows of $M$ in order the sub-matrix to have a 1 in each column. Therefore, there is a variable $y$ in $X$, such that there is a vertex in $Q$ adjacent to $y^{i-1,s}$ in $T$ and another vertex in $Q$ adjacent to $y^{i-1,s}$ in $T$. Here, truth assignment $a$ does not make $c$ true; so, if $y^{i-1,s}$ is adjacent in $T$ to one of $v^{i-1,s}$ or $v^{i-1,s}$, then $y^{i-1,s}$ must be adjacent in $T$ to the other. Therefore, there is a vertex in $Q$ which is at $T$ distance 2 from $v^{i-1,s}$ and another vertex in $Q$ which is at $T$ distance 2 from $v^{i-1,s}$. But each vertex in $Q$ is at $T$ distance 2 from $v^{i-1,s}$ or $v^{i-1,s}$; so, there are two consecutive vertices in $Q$, $q^{i-1,s}$ and $q^{i-1,s}$ say (where $r_0$ is some integer from 1 to 7), such that $q^{i-1,s}$ is at $T$ distance 2 from one of $v^{i-1,s}$ or $v^{i-1,s}$ and $q^{i-1,s}$ is at $T$ distance 2 from the other.

But $d_T(v^{i-1,s}, v^{i-1,s}) = 4(i - 1)$; so, $d_T(q^{i-1,s}, q^{i-1,s}) = 4i$. To pair $q^{i-1,s}$ and $q^{i-1,s}$ is attached a building block of layer $i$, say building block $G^{i,j}$. So, $d_T(v^{i,j}, v^{i,j}) = 4i$ and the induction step holds.

Then, let $G^{h,j}$ be a building block of layer $h$ such that $d_T(v^{h,j}, v^{h,j}) = 4h$. Let $x$ be a variable of $\phi$. Then, $x^{h,j}$ is adjacent in $G$ to both of $v^{h,j}$ and $v^{h,j}$. But $x^{h,j}$ is adjacent in $T$ to only one of $v^{h,j}$ or $v^{h,j}$, because $T$ is a $v$-concentrated spanning tree of $G$. Therefore, $x^{h,j}$ is at $T$ distance $4h + 1$ from one of its $G$ neighbors $v^{h,j}$ or $v^{h,j}$, which is a contradiction, because $T$ is a $4h$-spanner of $G$.

It remains to check the time complexity of algorithm 3-SAT based on the number of variables of input. Construction of graph $G$ takes $2^{O((\log n)^{m+1})}$ time, because there are at most $2^{2(\log n)^{m+1}}$ vertices in $G$ (and even fewer building blocks in $G$) and each building block of $G$ is constructed efficiently. The exhaustive search takes place only for small values of $n$. Algorithm get_spanner is assumed to be efficient but, because of its big input, its call takes $2^{O((\log n)^{m+1})}$ time. Finally, each building block of $G$ is examined once and each such examination is done efficiently. Therefore, the for loop over building blocks of $G$ takes $2^{O((\log n)^{m+1})}$ time. □

5 Notes

The tree 7-spanner returned by algorithm tree_7-spanner in figure 5 is not only $v$-concentrated but also a breadth first search tree of $G$ starting from $v$, as pointed out in the proof of lemma 1. Moreover, restricting algorithm get_spanner to return a breadth first search tree of $G$ starting from $v$, does not affect the proof of theorem 1. Therefore, the hardness of approximation described by theorem 1 also holds for breadth first search trees starting from $v$, which is an even smaller than $v$-concentrated family of spanning trees.

Note that just one vertex of $G^{h,j}$ (other than $v^{h,j}$ or $v^{h,j}$) is needed. So, in algorithm reduction (figure 5), the last layer (layer $h$) of $G$ can be filled instead with graphs much smaller than building blocks (a path of length 2 suffices) but this does not decrease the number of vertices of $G$ dramatically.

14
A few, unrelated to each other, notes follow. First, this approach does not lead to hardness of approximating tree spanners via general spanning trees; good tree spanners are not usually breadth first search trees. Second, the result of this article holds for stretch factor $t$ greater or equal to 7; its an open problem to find low factor approximate tree $t$-spanners for $3 \leq t \leq 6$. Third, function $f = (\log n)^{\log \log \log n + \frac{1}{t+1}}$ is $o(\log n)$ but there is no $m$, such that $f$ is $o((\log n)^{\frac{m}{t+1}})$.

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

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

[7] 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.

[8] 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.

[9] Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM J. Comput.*, 38(5):1761–1781, 2008.

[10] 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] 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).
[12] Fedor V. Fomin, Petr A. Golovach, and Erik Jan van Leeuwen. Spanners of bounded degree graphs. *Inf. Process. Lett.*, 111(3):142–144, 2011.

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

[14] Ioannis Papoutsakis. *Tree Spanners of simple graphs*. PhD thesis, Department of Computer Science, University of Toronto, 2013. (Available at university T-space).

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

[16] Ioannis Papoutsakis. Tree spanners of bounded degree graphs. *CoRR*, abs/1503.06822, 2015.

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

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

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

[20] S. Pettie. Low distortion spanners. *ACM Transactions on Algorithms*, 6(1), 2009.

[21] 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.

[22] 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.

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