SIAM J. COMPUT.
Vol. 38, No. 5, pp. 1761–1781

APPROXIMATING MINIMUM MAX-STRETCH SPANNING TREES ON UNWEIGHTED GRAPHS*

YUVAL EMEK† AND DAVID PELEG†

Abstract. Given a graph G and a spanning tree T of G, we say that T is a *tree t-spanner* of G if the distance between every pair of vertices in T is at most t times their distance in G. The problem of finding a tree t-spanner minimizing t is referred to as the *Minimum Max-Stretch spanning Tree (MMST)* problem. This paper concerns the MMST problem on unweighted graphs. The problem is known to be NP-hard, and the paper presents an $O(\log n)$-approximation algorithm for it. Furthermore, it is established that unless P = NP, the problem cannot be approximated additively by any $o(n)$ term.

Key words. spanning trees, low stretch, spanners

AMS subject classifications. 05C05, 05C12, 05C85

DOI. 10.1137/060666202

1. Introduction.

1.1. The problem.

Consider a connected n-vertex graph G. Let T be a spanning tree of G and let x and y be two vertices in G. The *stretch* of x and y in T, denoted $\mathrm{str}_T(x, y)$, is the ratio of the distance between x and y in T to their distance in G. The *maximum stretch*[1] of T, denoted max-str(T), is defined as the maximum of $\mathrm{str}_T(x, y)$, taken over all pairs of vertices x, y in G. The problem of finding a spanning tree T minimizing max-str(T) is referred to as the *Minimum Max-Stretch spanning Tree (MMST)* problem. In this paper we study the MMST problem on unweighted graphs. This problem is known to be NP-hard [5], and this paper presents the first nontrivial approximation algorithm for it, achieving an approximation ratio of $O(\log n)$. Our algorithm is inspired by the algorithm presented in [16] for the *Minimum Restricted Diameter spanning Tree (MRDT)* problem. We then establish a hardness of approximation result, showing that it is NP-hard to approximate the problem additively by a term of $o(n)$.

The MMST problem finds applications in network design and, in particular, in the context of distributed systems. One such application is the *arrow distributed directory protocol* introduced in [7]. This protocol supports the location of mobile objects in a distributed network. It is implemented over a spanning tree T that spans the network, and, as shown in [19], the worst case overhead ratio of the protocol is proportional to the maximum stretch of T. Therefore, a good candidate for the backbone of the arrow protocol is a spanning tree with low maximum stretch (see also [17]).

1.2. Related work.

The notion of stretch can be defined for any spanning subgraph. Formally, given a graph G, a spanning subgraph H of G, and a pair of

*Received by the editors July 27, 2006; accepted for publication (in revised form) September 15, 2008; published electronically December 19, 2008. An extended abstract appeared in [11]. This research was supported in part by a grant from the Israel Science Foundation.
http://www.siam.org/journals/sicomp/38-5/66620.html

†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel (yuval.emek@weizmann.ac.il, david.peleg@weizmann.ac.il).

[1]In some papers the notion of stretch refers to the maximum taken over all vertex pairs. To avoid misunderstanding, we distinguish between stretch, defined in the current paper for a specific vertex pair, and maximum stretch, defined for the whole tree.

vertices x, y in G, the stretch of x and y in H is defined as the ratio of the distance between x and y in H to their distance in G. A spanning subgraph with maximum stretch t is called a *t-spanner*. Spanners for general graphs were first introduced in [22]. Sparse spanners (namely, spanners with a small number of edges) were first studied in [20], where the problem of determining for a given graph G and a positive integer m whether G has a t-spanner with at most m edges is shown to be NP-complete for $t = 2$ while a polynomial time construction is presented for a $(4t + 1)$-spanner with $O\left(n^{1+1/t}\right)$ edges for every n-vertex graph and $t \geq 1$. Simple algorithms for constructing sparse spanners for arbitrary weighted graphs are presented in [2], including the construction of a $(2t+1)$-spanner with at most $n\left\lceil n^{1/t}\right\rceil$ edges for every n-vertex graph and $t > 0$. For any fixed $t \geq 3$ the problem of determining, for an arbitrary graph G and a positive integer m, whether G has a t-spanner with at most m edges is proved to be NP-complete in [4]. A polynomial time construction for a 3-spanner with $O\left(n^{3/2}\right)$ edges is presented in [9].

Cast in this terminology, the MMST problem can therefore be redefined as the attempt to find a *tree t-spanner* minimizing t. The NP-hardness of the MMST problem, even on unweighted graphs, is established in [5], where it is proved that determining whether an arbitrary weighted (respectively, unweighted) graph has a tree t-spanner is NP-complete for every fixed $t > 1$ (respectively, $t \geq 4$). The same paper also presents a polynomial time algorithm for constructing a tree 1-spanner in a weighted graph (if such a spanner exists) and a polynomial time algorithm for constructing a tree 2-spanner in an unweighted graph (if such a spanner exists).

Low stretch spanning trees in planar graphs were first studied in [12], where it is proved that finding a spanning tree T with minimum max-str(T) is NP-hard even for unweighted planar graphs. Polynomial time algorithms are presented therein for the problem of deciding for a fixed parameter t whether a planar unweighted graph with bounded face length has a tree t-spanner and for the problem of deciding whether an arbitrary unweighted planar graph has a tree 3-spanner. A polynomial time algorithm for the MMST problem on outerplanar graphs is presented in [21].

Hardness of approximation is established in [19], where it is shown that approximating the MMST problem within a factor better than $(1 + \sqrt{5})/2$ is NP-hard. This is improved in [18] by observing that the $2 - \epsilon$ inapproximabilty result established in [14] for the *min-max strictly fundamental cycle basis* problem also holds in the context of the MMST problem on unweighted graphs. A number of papers have studied the related but easier problem of finding a spanning tree with good *average* stretch factor [1, 3, 13, 10].

Given a complete graph $G = (V(G), E(G))$ with edge weights obeying the triangle inequality and a subset $R \subseteq E(G)$ called the *requirements* of G, the MRDT problem is to find a spanning tree T of G that minimizes the *restricted diameter* of T defined as the diameter of T restricted to vertex pairs in R. The MRDT problem is presented and proved to be NP-hard in [16]. The same paper presents an $O(\log n)$-approximation algorithm for this problem. It can be shown that the MMST problem on complete graphs with edge weights arising from the distances in some unweighted graph is a special case of the MRDT problem.

1.3. A brief description of the technique. Our main result is the first nontrivial approximation algorithm for the MMST problem on unweighted graphs. This algorithm relies on a graph decomposition technique that, given an unweighted graph G of size n, breaks it into disjoint connected components, each of size at most $n/2$, by discarding the edges internal to some ball B of radius proportional to the maximum stretch ρ of an optimal solution to the MMST problem on G. The spanning trees generated by recursive invocations of the algorithm on each such connected component are combined with a single source shortest paths spanning tree of B to produce a spanning tree T of G.

Consider an arbitrary edge (x, y) in G. Our analysis relies on observing that the number of edges added to the unique path between x and y in T on each recursive level is $O(\rho)$. Since there are at most $\log n$ recursive levels (as the size of the graph decreases by a factor of 2 on each recursive level), the stretch of x and y in T is $O(\log n)$.

The technique presented in [16] for the approximation algorithm of the MRDT problem is similar to that described above. The novel approach in this paper is the adaptation of this technique to graphs which are not necessarily complete.

1.4. Outline of the paper. In section 2 we present the basic notation and definitions used throughout this paper. Two lower bounds on the minimum max-stretch of unweighted graphs are established in section 3. Our approximation algorithm, named Algorithm `Construct_Tree`, is presented in section 4. In section 5 we prove that the output of Algorithm `Construct_Tree` is a spanning tree, and in section 6 we analyze the performance guarantee of the algorithm, establishing an $O(\log n)$ upper bound on the approximation ratio. Our analysis is based on the lower bounds of section 3. In section 7 we prove that the analysis of the performance guarantee of the algorithm is tight. The hardness of approximating the MMST problem on unweighted graphs is studied in section 8, proving that unless P = NP, the problem cannot be approximated additively by a term of $o(n)$.

2. Preliminaries.

Throughout, we consider a connected unweighted undirected n-vertex graph G. Let $V(G)$ and $E(G)$ denote the vertex and edge sets of G, respectively. The *length* of a path P in the graph is the number of edges in the path, denoted by $\mathrm{len}(P)$. For two vertices u, v in $V(G)$, let $\mathrm{dist}_G(u, v)$ denote the *distance* between them in G, i.e., the length of a shortest path between u and v. The definition of distance is extended to vertex subsets as follows. Let U and W be two subsets of $V(G)$. The *distance* between U and W is the minimum distance between any pair of vertices in U and W, denoted by $\mathrm{dist}_G(U, W) = \min\{\mathrm{dist}_G(u, w) \mid u \in U \text{ and } w \in W\}$.

For a subset $U \subseteq V(G)$, let $G(U)$ denote the subgraph of G *induced* by U, that is, $V(G(U)) = U$ and $E(G(U)) = E(G) \cap (U \times U)$. Denote the set of edges *internal* to U by $E(U) = E(G(U))$.

Although the notion of stretch can be defined for every spanning subgraph, our focus in the current paper is on spanning trees only. Consider some spanning tree T of G. Denote the *stretch* of u and v in T with respect to G by

$$\mathrm{str}_{T,G}(u, v) = \frac{\mathrm{dist}_T(u, v)}{\mathrm{dist}_G(u, v)}.$$

Denote the *maximum stretch* of T with respect to G by

$$\text{max-str}(T, G) = \max_{x,y \in V(G)} \{\mathrm{str}_{T,G}(x, y)\}.$$

When the graph G is clear from the context we may omit it and write simply $\mathrm{str}_T(u, v)$ and max-str(T). Denote the *minimum max-stretch* of G by

$$\text{max-str}(G) = \min\{\text{max-str}(T, G) \mid T \text{ is a spanning tree of } G\}.$$

Consider two metrics τ and δ over the vertices V. We say that τ *dominates* δ if $\tau(u,v) \geq \delta(u,v)$ for every $u,v \in V$. We say that τ is a *tree metric*[2] if τ is induced by the distances in some weighted tree over V. We extend the definitions of stretch and maximum stretch for tree metrics as follows. For every two vertices $u,v \in V$, denote the *stretch* of u and v in τ with respect to δ by

$$\mathrm{str}_{\tau,\delta}(u,v) = \frac{\tau(u,v)}{\delta(u,v)}.$$

The *maximum stretch* of τ with respect to δ, denoted max-str(τ,δ), is defined accordingly.

Consider some metric δ over the vertices V. Given a vertex set $U \subseteq V$, we define the *diameter* of U with respect to δ as

$$\mathrm{diam}_\delta(U) = \max\{\delta(u,v) \mid u,v \in U\}.$$

Given a vertex $u \in V$ and some positive real ρ, we denote the *ball* centered at u of radius ρ with respect to δ by $B_\delta(u,\rho) = \{v \in V(G) \mid \delta(u,v) \leq \rho\}$.

A *partition* of a set S is a collection $P = \{U_1,\ldots,U_k\}$, where $U_i \cap U_j = \emptyset$ for every $1 \leq i < j \leq k$ and $\bigcup_{1\leq i\leq k} U_i = S$. Unless stated otherwise, we assume that U_i is nonempty for every $1 \leq i \leq k$. If S is the set of vertices of some graph, then the subsets $U_1,\ldots,U_k$ are called the *clusters* of the partition. (Note that our definition does not require a cluster to be connected in G.)

For a partition $P = \{U_1,\ldots,U_k\}$ of $V(G)$, let $E(P)$ denote the set of edges *internal* to the clusters of P, i.e., $E(P) = \bigcup_{U_i\in P} E(U_i)$. Denote the set of edges *external* to P, namely, the edges connecting vertices in two different clusters, by $\overline{E}(P) = E(G) - E(P)$. Every edge set $F \subseteq \overline{E}(P)$ *induces* a logical *cluster graph* on P, obtained by contracting each cluster in P into a single node and replacing each edge $(u,v) \in F$, where $u \in U_i$ and $v \in U_j$, by the edge (U_i,U_j). We say that F *induces a tree* on P if the cluster graph induced by F on P is a tree. For a subset $X \subseteq V(G)$, denote the set of clusters in P that intersect X by $\mathcal{I}(P,X) = \{U_i \in P \mid U_i \cap X \neq \emptyset\}$.

A central tool in our construction is a graph decomposition based on eliminating the edges of some ball of radius ρ. This decomposition is obtained as follows. For a metric δ over $V(G)$, a vertex $u \in V(G)$, and a positive integer ρ, erase from G the internal edges (but not the vertices) of the ball centered at u of radius ρ with respect to δ, $E(B_\delta(u,\rho))$, and let $G^1,\ldots,G^r$ be the connected components in the remaining graph. The collection $\{G^1,\ldots,G^r\}$ is referred to as the (u,ρ,δ)*-decomposition* of G. Figure 1 illustrates the $(u,2,\mathrm{dist}_G(\cdot,\cdot))$-decomposition of a graph G. We say that the vertex u is a ρ*-center* with respect to G and δ if $|V(G^i)| \leq n/2$ for every $1 \leq i \leq r$.

3. Lower bounds on the minimum max-stretch.

In this section we establish some general lower bounds on the minimum max-stretch of a graph G. These lower bounds are used in section 6 to yield the performance guarantee of our approximation algorithm.

We begin with correlating the existence of a ρ-center with the minimum max-stretch of the graph.

THEOREM 3.1. *Consider a graph G and a connected vertex induced subgraph H of G, and let δ be the restriction of* $\mathrm{dist}_G(\cdot,\cdot)$ *to the vertices of H. Then H admits a (2κ)-center with respect to δ, where* $\kappa = $ max-str(G).

[2]Our definition of a tree metric differs from the standard definition, where τ is induced by the distances between pairs of vertices from V in some weighted tree over a superset of V. Clearly, a tree metric according to our definition is also a tree metric according to the standard definition.

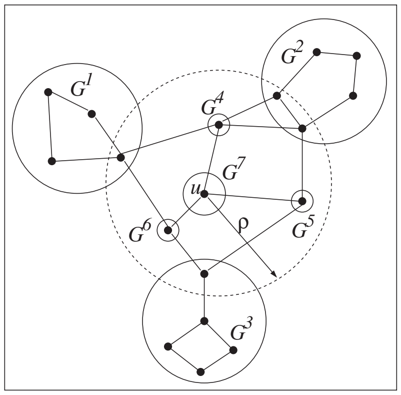

FIG. 1. *The decomposition of G with respect to* $\text{dist}_G(\cdot,\cdot)$*, u, and $\rho = 2$. The dashed circle represents* $B_{\text{dist}_G(\cdot,\cdot)}(u, 2)$.

Proof. Clearly, every spanning tree of G induces a tree metric over $V(G)$ that dominates the distances in G. In particular, there exists such a tree metric τ with $\text{max-str}(\tau, \text{dist}_G(\cdot,\cdot)) = \kappa$. In [15] it is proved that a tree metric τ over the vertex set V can be transformed into a tree metric τ' over an arbitrary vertex subset $U \subseteq V$, such that $\tau(x, y) \le \tau'(x, y) \le 8\,\tau(x, y)$ for every two vertices x, y in U. This result is improved in [16] for the special case where τ arises from the distances in an unweighted tree over V so that $\tau(x, y) \le \tau'(x, y) \le 4\,\tau(x, y)$. It follows that there exists a tree metric τ' over $V(H)$ that dominates δ such that $\text{max-str}(\tau', \delta) \le 4\kappa$. We next show that this implies that H admits a (2κ)-center with respect to δ.

Recall that in every n-vertex tree T, there exists a vertex u, named the *centroid* of T, such that the removal of all edges incident with u disconnects T to subtrees of size at most $n/2$ each. Consider the tree T over $V(H)$ that corresponds to τ', and let u be a centroid of T. (Observe that T is not necessarily a spanning tree of H.) Let $T_1, \ldots, T_k$ be the subtrees of T obtained from the removal of all edges incident with u.

Let $\{H^1, \ldots, H^r\}$ be the $(u, 2\kappa, \delta)$-decomposition of H, namely, the connected components of H after erasing the internal edges of $B_\delta(u, 2\kappa)$. We claim that for every $1 \le i \le r$, there exists some $1 \le j \le k$ such that $V(H^i) \subseteq V(T_j)$; hence u is a (2κ)-center of H and the theorem holds. Assume by way of contradiction that the claim is false, and let H^i be a subgraph of H that falsifies the claim, i.e., such that $V(H^i) \nsubseteq V(T_j)$ for any $1 \le j \le k$. Since H^i is connected, it follows that there exists an edge $(x, y) \in E(H^i)$ such that $x \in V(T_j)$ and $y \in V(T_{j'})$ for some $1 \le j < j' \le k$. The edge (x, y) was not removed by the $(u, 2\kappa, \delta)$-decomposition of H; thus $\delta(u, x) \ge 2\kappa$ and $\delta(u, y) \ge 2\kappa$, where at least one of these two inequalities is strict. Therefore, $\delta(u, x) + \delta(u, y) > 4\kappa$. Since τ' dominates δ, it follows that $\tau'(u, x) + \tau'(u, y) > 4\kappa$. But x and y are in different subtrees obtained from the removal of the edges incident with u; therefore, $\tau'(x, y) = \tau'(u, x) + \tau'(u, y) > 4\kappa$, in contradiction to the fact that $\text{max-str}(\tau', \delta) \le 4\kappa$. The theorem follows. $\square$

Consider a graph G, and let H be a connected vertex induced subgraph of G. Let $\bar{H}$ be the subgraph induced on G by $V(G) - V(H)$. Denote the set of edges that cross between H and $\bar{H}$ by $F(H) = \{(x, y) \in E(G) \mid x \in V(H) \text{ and } y \in V(\bar{H})\}$. Let $W(H)$ be the set of endpoints of edges in $F(H)$. We say that H is a *κ-outspread subgraph* of G if $F(H)$ can be partitioned into two *remote parts* F_1 and F_2 with endpoints W_1

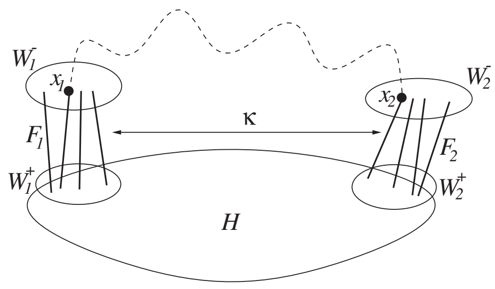

FIG. 2. *A κ-outspread subgraph H. The vertices x_1 and x_2 are connected in $\bar{H}$.*

and W_2, respectively, such that

- $\text{dist}_G(W_1, W_2) \geq \kappa$ and
- there exist two vertices $x_1 \in W_1 \cap V(\bar{H})$ and $x_2 \in W_2 \cap V(\bar{H})$ such that $\bar{H}$ admits a simple path between x_1 and x_2.

Note that $\bar{H}$ is not necessarily connected, but it is assumed to have some "connectivity" between endpoints of F_1 and endpoints of F_2. In what follows, we define $W_i^+ = W_i \cap V(H)$ and $W_i^- = W_i \cap V(\bar{H})$ for $i = 1, 2$. A κ-outspread subgraph is illustrated in Figure 2. The existence of an outspread subgraph in a graph implies a lower bound on its minimum max-stretch, as established in the following theorem.

THEOREM 3.2. *If a graph G admits a κ-outspread subgraph, then* $\text{max-str}(G) > \kappa$.

Proof. Consider a graph G, and let H be a κ-outspread subgraph of G, with remote parts F_1 and F_2 with endpoints W_1 and W_2. Let T be a spanning tree of G and suppose, toward deriving contradiction, that $\text{max-str}(T) \leq \kappa$.

We begin by strengthening the second requirement of a κ-outspread subgraph and proving that W_1^- and W_2^- are connected by a path fully contained in $T \cap \bar{H}$. To show this, we prove that T contains a subtree T' such that

- $V(T') \cap W_1^- \neq \emptyset$;
- $V(T') \cap W_2^- \neq \emptyset$;
- T' lies entirely in $\bar{H}$; and
- T' is maximal; namely, if T'' is a subtree of T and $V(T') \subset V(T'')$, then $V(T'') \cap V(H) \neq \emptyset$.

By definition, since H is a κ-outspread subgraph of G, the subgraph $\bar{H}$ admits a simple path between W_1^- and W_2^-. Let $\psi = (x_1^0, x_1^1, \ldots, x_1^s, x_2^t, x_2^{t-1}, \ldots, x_2^0)$ be the shortest such path, where $x_1^0 \in W_1^-$, $x_2^0 \in W_2^-$, $\kappa \leq \text{len}(\psi) = s + t + 1$, and $\frac{\text{len}(\psi)}{2} - 1 \leq s \leq t \leq \frac{\text{len}(\psi)}{2}$. Since ψ is a shortest path between W_1^- and W_2^-, it follows that $\text{dist}_G(x_i^j, W_i^-) = j$ for $i = 1, 2$ and every vertex x_i^j in ψ. By the definition of a κ-outspread subgraph, we have $\text{dist}_G(x_i^j, W_i^+) = j + 1$ and $\text{dist}_G(x_i^j, W_{3-i}^+) \geq \min\{\kappa + j + 1, s + t + 2 - j\}$ for $i = 1, 2$ and every x_i^j in ψ. Therefore, $\text{dist}_G(x_1^s, W_i^+) + \text{dist}_G(x_2^t, W_i^+) > \kappa$ for $i = 1, 2$. Consequently, the unique path between x_1^s and x_2^t in T does not contain any vertex in $W_1^+ \cup W_2^+$, as, otherwise, it is of length greater than κ, in contradiction to the assumption that $\text{max-str}(T) \leq \kappa$. Moreover, $\text{dist}_G(x_i^j, W_{3-i}^+) + \text{dist}_G(x_i^{j-1}, W_{3-i}^+) > \kappa$ for $i = 1, 2$ and every x_i^j and x_i^{j-1} in ψ; hence the unique path between x_i^j and x_i^{j-1} in T does not contain any vertex in W_{3-i}^+.

Let $\pi_{s,t}$ be the unique path between x_1^s and x_2^t in T. As $\pi_{s,t}$ does not contain any vertex in $W_1^+ \cup W_2^+$, it must lie entirely in $\bar{H}$. We would like to develop $\pi_{s,t}$ into a subtree T' as described above. For $i = 1, 2$, apply the following process to extend

$\pi_{s,t}$ up to a vertex in W_i^- without adding any edge of $F(H)$. Initialize the variable subtree Υ to be the path $\pi_{s,t}$. Initialize the variable integer j^{min} by $j^{min} = \min\{j \mid x_i^j \in V(\Upsilon)\}$ (j^{min} is well defined, as x_1^s and x_2^t are in Υ). If $j^{min} = 0$, then we are done since the subtree Υ now contains a W_i^- vertex and $V(\Upsilon) \subseteq V(\bar{H})$.

Otherwise, apply a "developing step" as follows. Let χ be the unique path between $x_i^{j^{min}}$ and $x_i^{j^{min}-1}$ in T. The path χ does not contain any edge in F_{3-i} since, otherwise, it also contains a vertex in W_{3-i}^+. If χ contains an edge in F_i, then it must contain a vertex in W_i^- right before that edge, in which case the subtree Υ can be extended up to a vertex in W_i^- without adding any $F(H)$ edge, and we are done. Otherwise, the path χ does not contain any $F(H)$ edge at all; thus Υ can be extended so that j^{min} decreases by a positive integral term without adding any edge in $F(H)$. We repeat the "developing step" until $j^{min} = 0$. Applying this process for $i = 1, 2$ yields a subtree T' of T such that T' contains a vertex in W_1^- and a vertex in W_2^- and $E(T') \cap F(H) = \emptyset$; hence T' lies entirely in $\bar{H}$. If T' is not maximal, then extend it by adding adjacent edges of $E(T) \cap E(\bar{H})$ as long as possible.

We now turn to label the vertices of H by their location in T with respect to the subtree T'. Pick an arbitrary vertex $r \in V(T')$ and direct the edges of T toward r. Consider a vertex v in $V(H)$, and let π_v be the unique path from v to r in T. If u is the first vertex on π_v such that $u \in V(T')$, then we say that v is *covered* by u with respect to T'. Since T' is maximal, if v is covered by u, then the edge entering u on π_v is in F and u must lie in W_i^- for some $i \in \{1, 2\}$. For $i = 1, 2$, let

$$U_i \;=\; \{v \in V(H) \mid v \text{ is covered by some vertex } u \in W_i^- \text{ with respect to } T'\}.$$

Note that $\{U_1, U_2\}$ is a partition of $V(H)$. For two vertices $x, y \in V(H)$, we say that x and y are *separated* by T' if $x \in U_1$ and $y \in U_2$ (or vice versa). Observe that this implies that the unique path between x and y in (the undirected) T contains an edge in F_1 and an edge in F_2; thus its length is at least $\kappa + 2$. It follows that for an edge (x, y) in $E(H)$, the vertices x and y cannot be separated by T' since, otherwise, we have max-str$(T) > \kappa$. But, by definition, the subgraph H is connected; therefore, $V(H) = U_i$ for some $i \in \{1, 2\}$ and $U_{3-i} = \emptyset$. Without loss of generality, suppose that $V(H) = U_1$.

Let y_2^- be a vertex in $V(T') \cap W_2^-$, and consider a neighbor $y_2^+ \in W_2^+$ of y in G. As y_2^+ is in U_1, it is covered by some vertex $y_1^- \in W_1^-$; hence $\mathrm{dist}_T(y_2^-, y_2^+) = \mathrm{dist}_T(y_2^-, y_1^-) + \mathrm{dist}_T(y_2^+, y_1^-)$. Since $\mathrm{dist}_G(W_1, W_2) \geq \kappa$, and since the distances in T dominate the distances in G, it follows that $\mathrm{dist}_T(y_2^-, y_2^+) \geq 2\kappa$, in contradiction to the assumption that max-str$(T) \leq \kappa$. The theorem follows. ◻

4. The approximation algorithm.

4.1. Overview.

The main idea behind our algorithm is that, given an n-vertex graph $\widehat{G}$ with max-str$(\widehat{G}) = \kappa$, there exists (as proved in Theorem 3.1) a vertex $u \in V(\widehat{G})$ with the property that discarding the internal edges of the ball centered at u of radius 2κ decomposes the graph into several connected components, each of size at most $n/2$, with no edges crossing between them. Using this fact, our algorithm is invoked on the integer test values $\rho \in (1, 2n]$, suspected of being 2κ. For every such test value, the algorithm tries to construct the output spanning tree $\widehat{T}$ by looking for a vertex u and a decomposition as above and merging a spanning tree T_{local} of the ball centered at u of radius $3\rho/2$ (a superset of the ball of radius ρ) with the spanning trees returned from the recursive calls made for every connected component of the decomposition. The $\rho/2$ gap between the radius of the ball used for decomposing the graph and the radius of the ball being spanned by the tree T_{local} guarantees the connectivity of the spanning tree produced by the algorithm. A detailed description of this process is presented in the following sections.

4.2. Algorithm `Construct_Tree`. We present an algorithm named Algorithm `Construct_Tree`, that, given an n-vertex graph $\widehat{G}$ and integral test value ρ, constructs a spanning tree $\widehat{T}$ of $\widehat{G}$ with maximum stretch at most $3\rho \log n$ or reports failure. In section 6 we prove that the algorithm does not fail if $\rho \geq 2\text{max-str}(\widehat{G})$. Since the minimum max-stretch of $\widehat{G}$ is an integer in $[1, n]$, a test value $\rho \leq 2\text{max-str}(\widehat{G})$ on which the algorithm does not fail can be guessed in $O(\log n)$ attempts, to yield a $(6 \log n)$-approximation algorithm.

Algorithm `Construct_Tree` works in a "divide and conquer" (recursive) approach. Throughout the execution of the algorithm, we maintain a forest (i.e., a cycle-free subgraph containing all the vertices) $\mathcal{F}$ of the input graph $\widehat{G}$. The forest is a global data structure shared by all recursive invocations of the algorithm. Initially, the forest $\mathcal{F}$ is empty (does not contain any edge), and, upon termination of the algorithm, $\mathcal{F}$ contains the spanning tree $\widehat{T}$ of $\widehat{G}$. On each recursive invocation, Algorithm `Construct_Tree` gets a vertex induced subgraph G of $\widehat{G}$ as input and adds some edges of G to $\mathcal{F}$. Upon termination of the recursive invocation on G, the vertices of $V(G)$ all belong to a single tree in $\mathcal{F}$ (each connected component in $\mathcal{F}$ is a tree).

Consider a recursive invocation of Algorithm `Construct_Tree` on the vertex induced subgraph G of $\widehat{G}$. Due to some earlier recursive invocations, the forest $\mathcal{F}$ may already contain some edges of G. Let $\mathcal{P}$ be the partition of $V(G)$ that corresponds to the connected components of $\mathcal{F}$; that is, each cluster in $\mathcal{P}$ is a subset of $V(G)$, and two vertices $x, y \in V(G)$ are in the same cluster in $\mathcal{P}$ if and only if $\mathcal{F}$ admits a path between x and y. We refer to the partition $\mathcal{P}$ as the *connectivity partition* of $V(G)$ with respect to $\mathcal{F}$. To prevent the possibility of creating cycles in $\mathcal{F}$, the algorithm will add an edge $e \in E(G)$ to $\mathcal{F}$ only if e is external to $\mathcal{P}$. Let U be an arbitrary cluster in $\mathcal{P}$. Note that the vertex induced subgraph $G(U)$ is not necessarily connected, although every two vertices in U are connected by a path in $\mathcal{F}$ (it may be the case that some of the vertices of this path are missing in G).

Throughout we denote the metric defined by distances in $\widehat{G}$ by $\widehat{\delta}$. Let δ be the restriction of $\widehat{\delta}$ to the vertices of G. Algorithm `Construct_Tree` works as follows. It first finds a ρ-center u with respect to δ and identifies the set of clusters $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$, namely, the clusters of $\mathcal{P}$ that intersect the ball centered at u of radius $3\rho/2$. If a ρ-center cannot be found, then the algorithm halts and reports "failure." Next, a subset of the external edges of the partition $\mathcal{P}$ that induces a tree (referred to as the *local tree* T_{local}) on these clusters is added to $\mathcal{F}$. The choice of the subset of edges that induces the local tree on the clusters $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$ is made by Procedure `Construct_Local_Tree`. The edges internal to the ball centered at u of radius ρ (but not the vertices) are discarded from the graph, and subsequently the graph decomposes[3] into separate connected components $G^1, \dots, G^r$. A recursive call is then made on the graph[4] $G(V(G^i))$ for every $1 \leq i \leq r$.

Let $\mathcal{F}'$ be the forest $\mathcal{F}$ after the edges of the local tree were added to it, and let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to $\mathcal{F}'$. Observe that the connec-

[3]The connected components of this graph decomposition should not be confused with the clusters of the connectivity partition $\mathcal{P}$.

[4]Observe that $G(V(G^i))$ is not necessarily identical to G^i, as some of the edges in $G(V(G^i))$ may be internal to $B_\delta(u, \rho)$. Such edges were discarded in the decomposition process, and they cannot be found in G^i.

tivity partition $\mathcal{P}'$ is obtained from the connectivity partition $\mathcal{P}$ (that corresponds to $\mathcal{F}$ before the edges of the local tree were added) by merging all clusters that intersect $B_\delta(u, 3\rho/2)$ into a single cluster, referred to as the *kernel cluster* U'. It is easy to verify (a stronger claim is proved later on) that the kernel cluster disintegrates into several connected components of the graph decomposition $G^1, \ldots, G^r$. Consequently, a pair of vertices in the kernel cluster may be *separated* on this recursive invocation. However, it is crucial for the correctness of Algorithm `Construct_Tree` that all other clusters in $\mathcal{P}'$ remain intact (see section 5). In fact, assuming that $\rho \geq 2\text{max-str}(\widehat{G})$, the $\rho/2$ gap between the radius of the ball used for decomposing the graph and that being spanned by the local tree ensures (as proved in section 6.1) that this is indeed the case.

Formally, we say that the connectivity partition $\mathcal{P}'$ is a *hub* with respect to the connected components $G^1, \ldots, G^r$ of the graph decomposition if, for every cluster $U \in \mathcal{P}' - \{U'\}$, there exists a connected component G^i, $1 \leq i \leq r$, such that $U \subseteq V(G^i)$. In other words, there is at most one cluster in $\mathcal{P}'$ that disintegrates into several connected components of the graph decomposition. (Recall that the connectivity partition is defined with respect to the global forest regardless of the connectivity in G; hence the subgraphs induced on G by its clusters are not necessarily connected to begin with. A cluster that induces a nonconnected subgraph on G may be disintegrated into several connected components even if none of its internal edges were discarded in the graph decomposition.) If $\mathcal{P}'$ is not a hub with respect to $G^1, \ldots, G^r$, then the algorithm halts and reports "failure."

The current recursive invocation of the algorithm is said to *succeed* if it manages to avoid the two failures, namely, if it finds a ρ-center and $\mathcal{P}'$ is a hub. Otherwise, we say that the current recursive invocation *fails*, in which case the algorithm should be reinvoked on $\widehat{G}$ with a larger test value ρ. A formal description of Algorithm `Construct_Tree` is given in Table 1.

TABLE 1
Algorithm `Construct_Tree`.

```
Input: A vertex induced subgraph $G$ of $\widehat{G}$.
Let $\delta$ be the restriction of $\widehat{\delta}$ to the vertices $V(G)$.
    1. If $|V(G)| = 1$, then return.
    2. Find a $\rho$-center $u$ with respect to $G$ and $\delta$.
       If none exists, then halt and report "failure."
    3. $T_{\text{local}} \leftarrow$ Construct_Local_Tree($G, u$).
    4. Set $\mathcal{F} \leftarrow \mathcal{F} \cup T_{\text{local}}$.
    5. Let $G^1, \ldots, G^r$ be the connected components of the graph remaining from $G$ after
       discarding the edges in $E(B_\delta(u, \rho))$.
    6. Let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to the (modified) forest $\mathcal{F}$.
    7. If $\mathcal{P}'$ is not a hub with respect to $G^1, \ldots, G^r$, then halt and report "failure."
    8. For every $1 \leq i \leq r$, invoke Construct_Tree($G(V(G^i))$).
```

Since the connected components $G^1, \ldots, G^r$ of the (u, ρ, δ)-decomposition of G are formed by discarding the edges internal to $B_\delta(u, \rho)$, we have the following.

OBSERVATION 4.1. *The vertex set $B_\delta(u, \rho)$ intersects G^i for every $1 \leq i \leq r$.*

4.3. Procedure `Construct_Local_Tree`. Consider a graph G. Let ξ be a metric over $V(G)$, and let $P = \{U_1, \ldots, U_k\}$ be a partition of $V(G)$. We say that the graph $\mathcal{R}$ is the *transitive graph* of G with respect to P and ξ if each cluster in P is completed

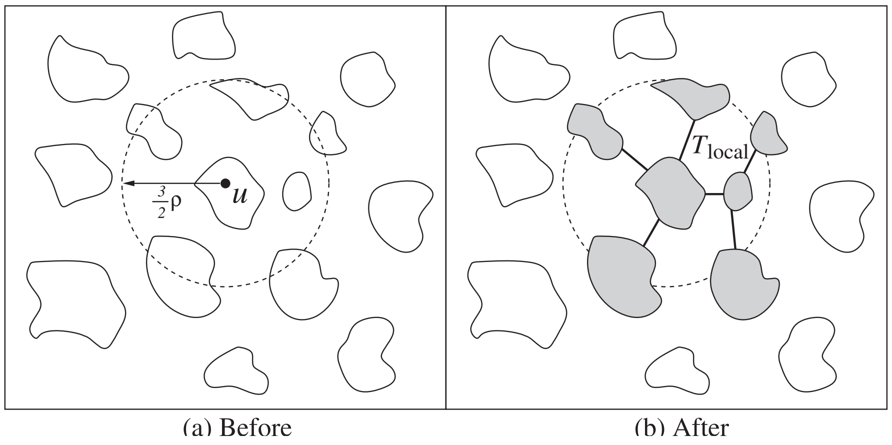

FIG. 3. *The operation of Procedure* `Construct_Local_Tree`. (a) *The clusters of $\mathcal{P}$ and the ball centered at u of radius $3\rho/2$.* (b) *T_{local} induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$. The shaded area forms the kernel cluster U'.*

into a clique in $\mathcal{R}$, namely,

- $V(\mathcal{R}) = V(G)$ and
- $E(\mathcal{R}) = \overline{E}(P) \cup \bigcup_{1 \le i \le k} (U_i \times U_i)$.

The graph $\mathcal{R}$ is weighted with edge lengths $\ell(e) = \xi(e)$ for every $e \in E(\mathcal{R})$. (Distances in a weighted graph are defined with respect to the length of the edges.)

Recall that $\mathcal{P}$ is the connectivity partition of $V(G)$ with respect to the forest $\mathcal{F}$ and that δ is the restriction of $\widehat{\delta}$ to the vertices $V(G)$. The purpose of the following procedure, named `Construct_Local_Tree`, is to find a subset T_{local} of external edges from $\overline{E}(\mathcal{P})$ that induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$. Let $\mathcal{P}'$ be the connectivity partition of $V(G)$ with respect to $\mathcal{F} \cup T_{\text{local}}$. Note that by the choice of T_{local}, it follows that $\mathcal{P}' - \mathcal{P}$ contains a single cluster, named the *kernel cluster U'*, which is the union of all the clusters in $\mathcal{P}$ that intersect $B_\delta(u, 3\rho/2)$. (See Figure 3.) The subset T_{local} should be chosen so that the diameter of the kernel cluster is not much greater than the sum of diameters of the $\mathcal{P}$-clusters it replaces (this requirement is presented formally in section 6).

In principle, this task can be achieved by using a depth-$(3\rho/2)$ shortest path tree rooted at u on the transitive graph $\mathcal{R}$. However, a naive choice of such a shortest path tree might create cycles in the cluster graph induced on $\mathcal{P}$. Procedure `Construct_Local_Tree` carefully avoids this complication. The procedure begins by constructing the transitive graph $\mathcal{R}$ of G with respect to $\mathcal{P}$ and δ. Next, the vertex set U' is initiated to consist of the cluster of u in $\mathcal{P}$, and the edge set T_{local} is initiated to be empty. The vertices of $\mathcal{R}$ are then processed in increasing order of distances from the vertex u. In section 5 we prove that whenever Procedure `Construct_Local_Tree` is invoked, the distances in $\mathcal{R}$ agree with δ; hence if the procedure processes the vertex x before it processes the vertex y, then $\delta(u, x) \le \delta(u, y)$.

Consider a vertex v when it is processed by the procedure, and let U_i be v's cluster in the partition $\mathcal{P}$. If $v \notin U'$, then the procedure adds the vertices of U_i to U' and adds the edge (w, v) to T_{local}, where w is the predecessor of v in some shortest path from u to v. Note that (w, v) is an edge in $E(G)$. This is justified since v must be the first vertex in U_i to be processed by the procedure (as otherwise, it was already in U'), and since w was processed before v (as $\text{dist}_{\mathcal{R}}(u, w) < \text{dist}_{\mathcal{R}}(u, v)$). The procedure halts when all vertices at distance at most $3\rho/2$ from u are processed and returns the edge set T_{local}. A formal description of Procedure `Construct_Local_Tree` is given in Table 2. Procedure `Construct_Local_Tree` can be implemented as a simple variant of the well-known Dijkstra algorithm for finding shortest paths from a single source [8, 6].

TABLE 2
Procedure `Construct_Local_Tree`.

```
Input: A vertex induced subgraph $G$ of $\widehat{G}$ and a vertex $u \in V(G)$.
Output: An edge set $T_{\text{local}} \subseteq \overline{E}(\mathcal{P})$ that induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, B_\delta(u, 3\rho/2))$.
    1. Construct the transitive graph $\mathcal{R}$ of $G$ with respect to $\mathcal{P}$ and $\delta$.
    2. Let $U$ be $u$'s cluster in $\mathcal{P}$.
    3. Set $U' \leftarrow U$ and $T_{\text{local}} \leftarrow \emptyset$.
    4. Let $v_1, \ldots, v_n$ be the vertices of $\mathcal{R}$ in increasing order of distances from $u$, namely,
       $v_1 = u$ and $\text{dist}_{\mathcal{R}}(v_i, u) \leq \text{dist}_{\mathcal{R}}(v_{i+1}, u)$ for every $1 \leq i < n$.
    5. For $i = 2, \ldots, n$, and as long as $\text{dist}_{\mathcal{R}}(v_i, u) \leq \frac{3\rho}{2}$, do:
        (a) Let $U_i$ be $v_i$'s cluster in $\mathcal{P}$.
        (b) If $v_i \notin U'$, then do:
              i. Set $U' \leftarrow U' \cup U_i$.
             ii. Let $\pi$ be a shortest path from $u$ to $v_i$ in $\mathcal{R}$. Let $w$ be the predecessor of $v_i$
                 in $\pi$.
            iii. Set $T_{\text{local}} \leftarrow T_{\text{local}} \cup (w, v_i)$.
    6. return $T_{\text{local}}$.
```

OBSERVATION 4.2. *The edge set T_{local} output by the procedure induces a tree on the clusters of $\mathcal{I}(\mathcal{P}, U')$.*

5. Correctness.

In this section we prove that Algorithm `Construct_Tree` generates a spanning tree of the given graph. In what follows, $\widehat{G}$ and $\widehat{T}$ stand for the unweighted n-vertex connected graph input to the first invocation of the algorithm and the subgraph stored in $\mathcal{F}$ upon termination of the algorithm, respectively. In order to analyze the recursive algorithm, we shall *label* each recursive invocation with a string in $\mathbb{N}^*$. The labeling is done in an inductive manner. The top recursive invocation (on the graph $\widehat{G}$) is labeled with the empty string ω. Consider a recursive invocation labeled with the string σ on the graph G (which is a vertex induced subgraph of $\widehat{G}$) for some $\sigma \in \mathbb{N}^*$, and let $G^1, \ldots, G^r$ be the connected components of the graph decomposition of G (see line 5 of Algorithm `Construct_Tree`)). Then the recursive invocation on the graph $G(V(G^i))$ is labeled with the string σi for every $1 \leq i \leq r$. We refer to the recursive invocation labeled with the string σ as the *σ-recursive invocation.*

We shall use subscript σ to denote the various ingredients of the σ-recursive invocation. Let G_σ, δ_σ, and u_σ denote the input graph G, the restriction of $\widehat{\delta}$ to the vertices $V(G)$, and the center vertex u, respectively, in the σ-recursive invocation. Observe that the original graph $\widehat{G}$ input to the top recursive invocation is denoted by G_ω. Let $\mathcal{F}_\sigma$ and $\mathcal{F}'_\sigma$ be snapshots of the forest $\mathcal{F}$ at the beginning of the execution of the σ-recursive invocation and after the addition of the local tree, respectively. Let $\mathcal{P}_\sigma$ and $\mathcal{P}'_\sigma$ denote the connectivity partitions of $V(G_\sigma)$ with respect to $\mathcal{F}_\sigma$ and $\mathcal{F}'_\sigma$, respectively. Let U'_σ be the single cluster in $\mathcal{P}'_\sigma - \mathcal{P}_\sigma$ (the kernel cluster). Finally, let $\mathcal{R}_\sigma$ be the transitive graph of G_σ with respect to $\mathcal{P}_\sigma$ and δ_σ.

We begin with establishing a few fundamental facts regarding the execution of our algorithm.

LEMMA 5.1. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$.*

1. *If there exists a path π between some two vertices x and y in the graph $\widehat{G}$ such that* $\mathrm{len}(\pi) > 1$ *and* $V(\pi) \cap V(G_\sigma) = \{x, y\}$*, then both x and y are in the same cluster in $\mathcal{P}_\sigma$. In fact, if $|\sigma| > 0$, then both x and y are in the same cluster in $\mathcal{P}'_{\sigma_{-1}}$, where σ_{-1} is the longest proper prefix of σ.*
2. *The distances in the transitive graph $\mathcal{R}_\sigma$ agree with δ_σ.*

Proof. We prove the two claims simultaneously by induction on the length of the string σ. On the top recursive invocation, which is labeled with the empty string ω, we have the following.

1. The graphs G_ω and $\widehat{G}$ are identical; hence such a path π does not exist and the first claim holds vacuously.
2. The graphs $\mathcal{R}_\omega$ and $\widehat{G}$ are identical and the second claim holds trivially.

Now assume that the claims hold for the σ_{-1}-recursive invocation, where σ_{-1} is the longest proper prefix of σ, and consider the σ-recursive invocation. We first prove claim 1. Consider a path π between the vertices x and y as in the claim. If all internal vertices of the path π (namely, vertices other than the endpoints x and y) were already missing on the previous recursion level, i.e., if $V(\pi) \cap V(G_{\sigma_{-1}}) = \{x, y\}$, then, due to the inductive hypothesis, x and y are in the same tree in $\mathcal{F}_{\sigma_{-1}}$; thus they remain in the same tree in $\mathcal{F}'_{\sigma_{-1}}$ (and in $\mathcal{F}_\sigma$) and the assertion holds. Otherwise, there must be some internal vertices of the path π that have existed in the graph $G_{\sigma_{-1}}$ on the previous recursion level.

We argue that vertex x must be in the kernel cluster $U'_{\sigma_{-1}}$. The same line of reasoning shows that $y \in U'_{\sigma_{-1}}$ as well; hence both x and y are in the same tree in $\mathcal{F}'_{\sigma_{-1}}$ (and they remain in the same tree in $\mathcal{F}_\sigma$) so that the assertion holds. Let w be the internal vertex of π that still existed in $G_{\sigma_{-1}}$ which is closest to x in π; that is, the vertex w satisfies $\mathrm{dist}_\pi(w, x) \le \mathrm{dist}_\pi(w', x)$ for any vertex $w' \in V(\pi) \cap V(G_{\sigma_{-1}}) - \{x, y\}$. We consider two cases (illustrated in Figure 4).

Case (a). If (x, w) is an edge in $E(\widehat{G})$, then since w does not exist in G_σ, it follows that both w and x were in $B_{\delta_{\sigma_{-1}}}(u_{\sigma_{-1}}, \rho)$; hence $\delta_{\sigma_{-1}}(u_{\sigma_{-1}}, x) \le \rho$. By the inductive hypothesis on claim 2, the distances in $\mathcal{R}_{\sigma_{-1}}$ agree with $\delta_{\sigma_{-1}}$; thus $\mathrm{dist}_{\mathcal{R}_{\sigma_{-1}}}(u_{\sigma_{-1}}, x) \le \rho$. Therefore, since Procedure `Construct_Local_Tree` halted at distance $3\rho/2$ from the center vertex $u_{\sigma_{-1}}$ (see line 5 of the procedure), it follows that x is in the kernel cluster $U'_{\sigma_{-1}}$.

Case (b). If (x, w) is not an edge in $E(\widehat{G})$, then let U_w be w's cluster in the partition $\mathcal{P}_{\sigma_{-1}}$. By the inductive hypothesis, the vertex x is in U_w as well (due to the subpath of π that starts at x and ends at w); thus w and x are in the same cluster in $\mathcal{P}_{\sigma_{-1}}$ and they remain in the same cluster $\bar{U}_w$ in $\mathcal{P}'_{\sigma_{-1}}$. Since w is not a vertex in G_σ, it follows that the cluster $\bar{U}_w$ disintegrated into different connected components in the decomposition of the graph $G_{\sigma_{-1}}$. Therefore, the cluster $\bar{U}_w$ must be the kernel cluster $U'_{\sigma_{-1}}$ since, otherwise, the recursive invocation of Algorithm `Construct_Tree` on $G_{\sigma_{-1}}$ would have failed as $\mathcal{P}'_{\sigma_{-1}}$ is not a hub (see line 7 of the algorithm).

We now turn to prove claim 2. Consider the transitive graph $\mathcal{R}_\sigma$ as in the claim. Let x and y be any two vertices in $V(G_\sigma)$, and let π be a shortest path between x and y in $\widehat{G}$. Some segments of the path π (namely, some consecutive sequences of vertices and edges between them) may be missing in the graph G_σ. Consider such a missing segment, and let x' and y' be the vertices in π right before and right after that missing segment, respectively; i.e., the path π consists of a segment between x and x', a segment between x' and y', and a segment between y' and y, where the internal vertices of the segment between x' and y' are all missing in G_σ. By claim 1, the vertices x' and y' must be in the same tree in $\mathcal{F}_\sigma$ (see Figure 5); hence they are in the same cluster in

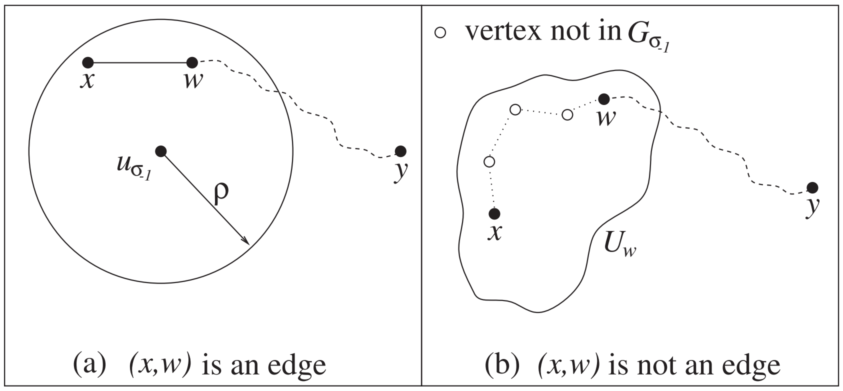

FIG. 4. *The two cases in the proof of Lemma* 5.1*'s claim* 1. *In both cases, the vertex x is in the kernel cluster* $U'_{\sigma_{-1}}$.

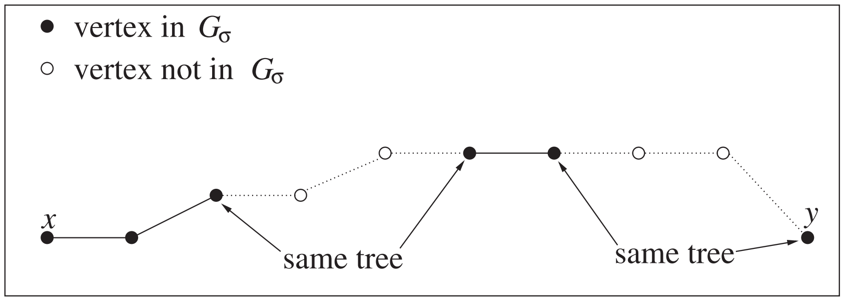

FIG. 5. *Proof of claim* 2 *of Lemma* 5.1. *The path* π *contains some missing segments.*

the partition $\mathcal{P}_\sigma$, and by the definition of the transitive graph $\mathcal{R}_\sigma$, we conclude that $\mathrm{dist}_{\mathcal{R}_\sigma}(x', y') = \delta_\sigma(x', y')$. The assertion follows as $\delta_\sigma(x, y) = \mathrm{len}(\pi)$. ◻

By claim 2 of the last lemma and since Procedure `Construct_Local_Tree` halts at distance $3\rho/2$ from the source vertex u (see line 5 of Procedure `Construct_Local_Tree`), we have the following.

COROLLARY 5.2. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$. The kernel cluster U'_σ satisfies*

$$B_{\delta_\sigma}(u_\sigma, \rho) \subseteq B_{\delta_\sigma}(u_\sigma, 3\rho/2) \subseteq U'_\sigma.$$

Next, we prove that the subgraph $\widehat{T}$ output by the algorithm is indeed a spanning tree of $\widehat{G}$.

PROPOSITION 5.3. *Consider an edge $(x, y) \in E(\widehat{G})$. There exists some string σ in $\mathbb{N}^*$ such that both x and y are in the kernel cluster U'_σ.*

Proof. Let σ be the longest string in $\mathbb{N}^*$ such that both x and y are in G_σ. This means that x and y are separated in the graph decomposition on the σ-recursive invocation; hence $x, y \in B_{\delta_\sigma}(u_\sigma, \rho)$. The assertion follows as Corollary 5.2 guarantees that $B_{\delta_\sigma}(u_\sigma, \rho) \subseteq U'_\sigma$. ◻

Since edges are added to $\mathcal{F}$ only if they are external to the connectivity partition $\mathcal{P}$ (see Procedure `Construct_Local_Tree`), it follows that $\widehat{T}$ is cycle-free. To see that $\widehat{T}$ is connected, consider an arbitrary edge $(x, y) \in E(\widehat{G})$. By Proposition 5.3, at some stage during the execution of the algorithm, both x and y belong to the kernel cluster

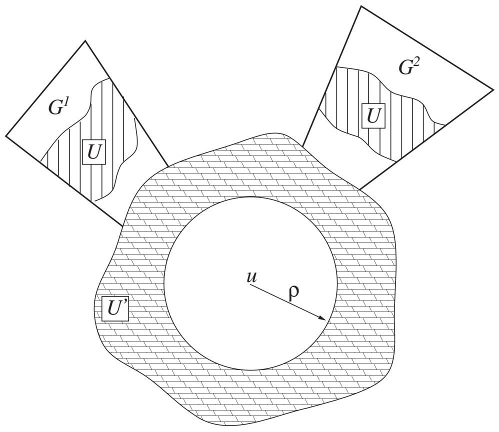

FIG. 6. *The cluster U decomposes into the connected components G^1 and G^2 of the graph decomposition.*

U'; hence the forest $\mathcal{F}$ at that stage admits a path between x and y. Therefore, $\widehat{T}$ admits a path between x and y. As $\widehat{G}$ is connected, we have the following.

THEOREM 5.4. *The graph $\widehat{T}$ output by Algorithm* `Construct_Tree` *is a spanning tree of the input graph $\widehat{G}$.*

6. Analysis.

In this section we analyze the performance of our algorithm. In section 6.1 we prove that the recursive invocations of Algorithm `Construct_Tree` on all vertex induced subgraphs of $\widehat{G}$ succeed as long as the test value ρ is at least 2max-str$(\widehat{G})$. The approximation ratio guaranteed by our algorithm is then established in section 6.2, where we prove that if the algorithm succeeds to generate a spanning tree $\widehat{T}$ with test value ρ, then max-str$(\widehat{T}) \leq 3\rho\lceil \log n \rceil$. In section 6.3 we analyze the running time of the algorithm.

6.1. Successful recursive invocation.

Consider some invocation of Algorithm `Construct_Tree` on the vertex induced subgraph G of $\widehat{G}$ with test value $\rho \geq$ 2max-str$(\widehat{G})$. The proof that a ρ-center can be found (see line 2 of the algorithm) follows directly from Theorem 3.1. Let $G^1, \ldots, G^r$ be the connected components of the graph decomposition on this recursion level. In order to prove that the connectivity partition $\mathcal{P}'$ of $V(G)$ with respect to $\mathcal{F}'$ is a hub (see line 7 of the algorithm), we have to show that the kernel cluster is the only cluster in $\mathcal{P}'$ that decomposes into several connected components.

Suppose toward deriving contradiction that there exists a cluster U in $\mathcal{P}' - \{U'\}$ that decomposes into several connected components of the graph decomposition. (The execution of the algorithm halts at that stage.) Formally, let $X_i = U \cap V(G^i)$, where without loss of generality $X_i \neq \emptyset$ for $1 \leq i \leq t$ and $X_i = \emptyset$ for $t < i \leq r$, and suppose that $t \geq 2$. Figure 6 illustrates a cluster $U \in \mathcal{P} - \{U'\}$ that decomposes into two connected components.

Let δ be the restriction of $\widehat{\delta}$ to the vertices of G, and let u be the current ρ-center. By the definition of the graph decomposition, every edge e in $E(G)$ that crosses between $V(G^i)$ and $V(G^j)$, where $1 \leq i < j \leq r$, satisfies $e \in B_\delta(u,\rho) \times B_\delta(u,\rho)$. Thus, by Observation 4.1 and Corollary 5.2, we have the following.

OBSERVATION 6.1. *Every edge e that crosses between $V(G^i)$ and $V(G^j)$, where $1 \leq i < j \leq r$, is in $U' \times U'$. Furthermore, the kernel cluster U' satisfies $U' \cap V(G^i) \neq \emptyset$*

for every $1 \leq i \leq r$.

Let T be the tree in $\mathcal{F}$ that corresponds to the cluster U, and let H be the subgraph induced on $\widehat{G}$ by $V(T)$. We prove that H is a $\rho/2$-outspread subgraph of $\widehat{G}$ (refer to section 3 for the definition of an outspread subgraph), in contradiction to Theorem 3.2.

Following the notation of section 3, let $\bar{H}$ denote the subgraph induced on $\widehat{G}$ by the vertices in $V(\widehat{G}) - V(T)$. Let $F(H)$ be the set of edges in $E(\widehat{G})$ that cross between H and $\bar{H}$, and let $W(H)$ be the set of endpoints of edges in $F(H)$. Let $F_1 = \{e \in F(H) \mid e \in E(G^1)\}$; namely, the edge set F_1 consists of the edges that cross between U vertices and other vertices in the connected component G^1 of the decomposition of G. Let $F_2 = F(H) - F_1$. By the choice of U and by Observation 6.1, the connected components $G^1, \ldots, G^t$ contain some vertices in U and some vertices not in U; thus F_1 and F_2 are nonempty.

Let W_i be the endpoints of edges in F_i for $i = 1, 2$. Let $W_i^+ = W_i \cap V(H)$ and $W_i^- = W_i \cap V(\bar{H})$. In order to prove that H is a $\rho/2$-outspread subgraph of $\widehat{G}$, we have to show that the distance between W_1 and W_2 in $\widehat{G}$ is large and to establish some connectivity properties of H and $\bar{H}$. We start with the latter.

Clearly, the vertex induced subgraph H is connected (as T is a tree in $\mathcal{F}$). For $\bar{H}$ we have the following proposition.

PROPOSITION 6.2. *There exist two vertices $x_1 \in W_1^-$ and $x_2 \in W_2^-$ such that $\bar{H}$ admits a simple path between x_1 and x_2.*

Proof. Let T' be the tree in $\mathcal{F}$ that corresponds to the cluster U'. Consider the subgraph G' induced on $\widehat{G}$ by $V(T') \cup (V(G) - U)$. This subgraph is not necessarily connected; however, the vertices of T' all belong to the same connected component G'' of G' as T' is connected by its own rights. Since $U' \subseteq V(T') \subseteq V(G'')$, it follows due to Observation 6.1 that $V(G'') \cap V(G^j) \neq \emptyset$ for every $1 \leq j \leq t$. Therefore, there exists a vertex x_i in $W_i^- \cap V(G'')$ for $i = 1, 2$. The proposition follows as G'' is a (connected) subgraph of $\bar{H}$. ∎

We now turn to prove that $\mathrm{dist}_{\widehat{G}}(W_1, W_2) \geq \rho/2$. We first argue that if two adjacent vertices were separated in the graph decomposition on some recursive invocation, then on the subsequent recursive invocations, they both lie far away from the boundary of their corresponding clusters.

PROPOSITION 6.3. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$. Let x be a vertex in $V(G_\sigma)$, and let U_x be its cluster in the connectivity partition $\mathcal{P}_\sigma$. If x admits a neighbor y in $\widehat{G}$ such that $y \notin V(G_\sigma)$, then $B_{\delta_\sigma}(x, \rho/2) \subseteq U_x$.*

Proof. Let α be the longest string in $\mathbb{N}^*$ such that both x and y are in G_α. The vertices x and y must have been separated in the graph decomposition on the α-recursive invocation (α is a proper prefix of σ). As $(x, y) \in E(G_\alpha)$, this could have happened only if both x and y were in $B_{\delta_\alpha}(u_\alpha, \rho)$. By Corollary 5.2, the kernel cluster U'_α satisfies $B_{\delta_\alpha}(u_\alpha, 3\rho/2) \subseteq U'_\alpha$; thus $w \in U'_\alpha$ for every vertex $w \in V(G_\alpha)$ such that $\mathrm{dist}_{\widehat{G}}(w, x) = \delta_\alpha(w, x) \leq \rho/2$, and w remains in x's tree in the forest from that moment on. Therefore, if such a vertex w still exists in the graph G_σ, then it must lie in U_x. ∎

Recall that the vertex set W_1 consists of the endpoints of edges in F_1. Since every edge in F_1 crosses between different clusters of the connectivity partition $\mathcal{P}$, we have the following.

Corollary 6.4. *Every vertex $x \in V(\widehat{G})$ such that $\mathrm{dist}_{\widehat{G}}(x, W_1) \leq \rho/2$ is in $V(G)$.*

Assume by way of contradiction that $\mathrm{dist}_{\widehat{G}}(W_1, W_2) < \rho/2$. Let π be a shortest path in $\widehat{G}$ between any vertex in W_1 and any vertex in W_2. Since F is a cut, it follows that π lies entirely in H or in $\bar{H}$, but it does not cross between them (as, otherwise, π is not shortest). Corollary 6.4 implies that π lies entirely in the graph G as every vertex in π is at distance less than $\rho/2$ from W_1. By Observation 6.1, every path from W_1 to W_2 in G must intersect U'. Since $V(H) \cap V(G) = U$, it follows that π cannot lie entirely in H; thus π connects between W_1^- and W_2^- in the subgraph induced by $V(G) - U$ on G (and, in particular, in $\bar{H}$).

Recall that Algorithm `Construct_Tree` employs the ball centered at u of radius ρ to decompose the graph (see line 5 of the algorithm), while the ball of radius $(3\rho/2)$ is contained in the kernel cluster U' (see Corollary 5.2). Since every vertex in W_i^- has a neighbor outside U' for $i = 1, 2$, we conclude that $\mathrm{dist}_{\widehat{G}}(W_i^-, B_\delta(u, \rho)) \geq \rho/2$. As $\mathrm{len}(\pi) < \rho/2$, π cannot contain any edge internal to $B_\delta(u, \rho)$. But this implies that the all vertices of π should have been in the same connected component of the (u, ρ, δ)-decomposition of G, in contradiction to the fact that π has one endpoint in G^1 and another in G^j for some $1 < j \leq t$. This establishes the following theorem.

Theorem 6.5. *Given an input graph $\widehat{G}$ and a test value $\rho \geq 2\text{max-str}(\widehat{G})$, Algorithm* `Construct_Tree` *succeeds on each recursive invocation.*

6.2. Approximation ratio. In this section we prove that if Algorithm `Construct_Tree` succeeds on each recursive invocation when invoked with test value ρ, then the output spanning tree $\widehat{T}$ satisfies $\mathrm{dist}_{\widehat{T}}(x, y) \leq 3\rho\lceil \log n \rceil$ for every edge (x, y) in $E(\widehat{G})$ (recall that n is the number of vertices in the input graph $\widehat{G}$). By Proposition 5.3, we know that at some stage during the execution of the algorithm, the vertices x and y are both in the kernel cluster. Consequently we would like to bound the diameter of the kernel cluster with respect to the distances in the output spanning tree $\widehat{T}$. In an attempt to establish such a bound, we will actually prove a stronger claim, stating that the sum of the diameters of many clusters, the kernel cluster being one of them, is sufficiently small.

Let $\widehat{\tau}$ be the metric defined by the distances in the output tree $\widehat{T}$. For a collection of vertex subsets $\Lambda = \{U_1, \ldots, U_k\}$, let

$$\varphi(\Lambda) \ = \ \sum_{1 \leq i \leq k} \mathrm{diam}_{\widehat{\tau}}(U_i).$$

Our subsequent analysis revolves on bounding the measure $\varphi(\mathcal{P}'_\sigma)$ as a function of the length of the string $\sigma \in \mathbb{N}^*$.

Proposition 6.6. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$. The kernel cluster U'_σ satisfies*

$$\mathrm{diam}_{\widehat{\tau}}(U'_\sigma) \ \leq \ 3\rho + \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)).$$

Proof. The execution of Procedure `Construct_Local_Tree` halts at distance $3\rho/2$ from the source vertex u_σ (see line 5 of the procedure). Therefore, the tree induced by T_{local} on the clusters of $\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$ is of depth at most $3\rho/2$ (when the cluster containing u_σ is considered to be the root). Consider some two vertices $x, y \in U'_\sigma$, and let π be the unique path between x and y in $\widehat{T}$. The path π contains at most $2 \cdot (3\rho/2) = 3\rho$ edges from the local tree T_{local}. Let $\pi' = \pi - T_{\mathrm{local}}$ be the rest of the path π. Since π' is a collection of segments internal to the $\mathcal{P}_\sigma$-clusters replaced by U'_σ, it follows that $\mathrm{len}(\pi') \le \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma))$. Therefore, $\mathrm{len}(\pi) \le 3\rho + \varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma))$, and the assertion holds. □

The next proposition enables us to bound the sum of the diameters of the clusters in $\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$.

LEMMA 6.7. *Consider the σ-recursive invocation for some string $\sigma = \alpha i$, where $\alpha \in \mathbb{N}^*$ and $i \in \mathbb{N}$. For every cluster U in $\mathcal{P}_\sigma$, there exists a cluster $\bar{U}$ in $\mathcal{P}'_\alpha$ such that $U \subseteq \bar{U}$.*

Proof. Suppose toward deriving contradiction that there exists a cluster $U \in \mathcal{P}_\sigma$ such that $U \nsubseteq \bar{U}$ for any cluster $\bar{U}$ in $\mathcal{P}'_\alpha$. This implies that $\mathcal{F}_\sigma$ contains a path π between some two vertices $x, y \in U$ that are not in the same cluster in $\mathcal{P}'_\alpha$. Let π be such a path of minimum length. The path π must satisfy $V(\pi) \cap V(G_\sigma) = \{x, y\}$, since, otherwise, there exists some subpath of π with endpoints $x', y' \in U$, where x' and y' are in different clusters in $\mathcal{P}'_\alpha$, in contradiction to π being the shortest such path.

The path π must contain some edges that do not exist in $\mathcal{F}'_\alpha$. Every such edge was added to $\mathcal{F}$ by some $\alpha j \beta$-recursive invocation, where $j \ne i$ and $\beta \in \mathbb{N}^*$. The graph $G_{\alpha j \beta}$ must contain some internal vertices of the path π; thus $\mathrm{len}(\pi) > 1$. Therefore, Lemma 5.1, when applied to the σ-recursive invocation, implies that x and y are in the same cluster in $\mathcal{P}'_\alpha$, which derives a contradiction. The assertion follows. □

We are now ready to establish the main lemma of this section.

LEMMA 6.8. *Consider the σ-recursive invocation for some string σ in $\mathbb{N}^*$. The connectivity partition $\mathcal{P}_\sigma$ satisfies*

$$\varphi(\mathcal{P}_\sigma) \;\le\; 3\rho(|\sigma| + 1).$$

Proof. We prove the assertion by induction on the length of the string σ. The only nonsingleton cluster in the connectivity partition $\mathcal{P}_\omega$ is the kernel cluster U'_ω, whose diameter with respect to the distances in $\widehat{T}$ is at most 3ρ. The diameter of a singleton cluster is 0. Therefore, the sum of the diameters of all clusters in $\mathcal{P}_\omega$ is at most 3ρ, and the assertion holds. Let $\sigma_{-1} \in \mathbb{N}^*$ be the longest proper prefix of σ, and assume that the assertion holds for σ_{-1}. Lemma 6.7 implies that for every cluster $U \in \mathcal{P}'_\sigma - \{U'_\sigma\}$, there exists a cluster $\bar{U} \in \mathcal{P}'_{\sigma_{-1}} - \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)$ such that $U \subseteq \bar{U}$. Since $\mathrm{diam}_{\widehat{T}}(U)$ is monotone under set inclusion, i.e., $U \subseteq \bar{U}$ implies $\mathrm{diam}_{\widehat{T}}(U) \le \mathrm{diam}_{\widehat{T}}(\bar{U})$, it follows that

$$\begin{aligned}\varphi(\mathcal{P}'_\sigma) - \mathrm{diam}_{\widehat{T}}(U'_\sigma) &= \varphi(\mathcal{P}'_\sigma - \{U'_\sigma\}) \le \varphi(\mathcal{P}'_{\sigma_{-1}} - \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)) \\ &= \varphi(\mathcal{P}'_{\sigma_{-1}}) - \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)).\end{aligned}$$

Thus

$$\varphi(\mathcal{P}'_\sigma) \le \mathrm{diam}_{\widehat{T}}(U'_\sigma) - \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)) + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

Another application of Lemma 6.7 guarantees that for every cluster $U \in \mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)$, there exists a cluster $\bar{U} \in \mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma)$ such that $U \subseteq \bar{U}$; hence $\varphi(\mathcal{I}(\mathcal{P}_\sigma, U'_\sigma)) \le \varphi(\mathcal{I}(\mathcal{P}'_{\sigma_{-1}}, U'_\sigma))$, and we can bound

$$\varphi(\mathcal{P}'_\sigma) \le \mathrm{diam}_{\widehat{T}}(U'_\sigma) - \varphi(\mathcal{I}(\mathcal{P}'_\sigma, U'_\sigma)) + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

Proposition 6.6 is employed to deduce that

$$\varphi(\mathcal{P}'_\sigma) \le 3\rho + \varphi(\mathcal{P}'_{\sigma_{-1}}).$$

The assertion follows by the inductive hypothesis as $|\sigma_{-1}| = |\sigma| - 1$. □

Let (x, y) be an arbitrary edge in $E(\widehat{G})$. By Proposition 5.3, there exists a string $\sigma \in \mathbb{N}^*$ such that both x and y are in the kernel cluster U'_σ. Therefore,

$$\mathrm{dist}_{\widehat{T}}(x, y) \le \mathrm{diam}_{\widehat{T}}(U'_\sigma) \le \varphi(\mathcal{P}'_\sigma).$$

By Lemma 6.8, and since the depth of the recursion is at most $\lceil \log n \rceil$ (as the size of each graph input to the recursive algorithm decreases by a factor of at least 2), we conclude that

$$\mathrm{dist}_{\widehat{T}}(x, y) \le 3\rho(|\sigma| + 1) \le 3\rho \lceil \log n \rceil.$$

As the maximum stretch of a spanning tree is always obtained on a pair of vertices that form an edge in the original graph [5], Theorems 5.4 and 6.5 imply the following.

THEOREM 6.9. *Given an n-vertex graph $\widehat{G}$, Algorithm* `Construct_Tree` *can be invoked with $O(\log n)$ different test values to generate a spanning tree $\widehat{T}$ of $\widehat{G}$ satisfying* $\text{max-str}(\widehat{T}) = O(\log n) \cdot \text{max-str}(\widehat{G})$.

6.3. Running time. We now turn to analyze the running time of Algorithm `Construct_Tree` when invoked with test value ρ on input graph $\widehat{G}$ with $\widehat{n}$ vertices and $\widehat{m}$ edges. The distance metric $\widehat{\delta}$ is constructed in a preprocessing stage in time $O(\widehat{n}\widehat{m})$ (a trivial implementation); thus in what follows we assume that $\widehat{\delta}$ and its restrictions to subsets of $V(\widehat{G})$ are known.

Consider a recursive invocation of the algorithm on vertex induced subgraph G of $\widehat{G}$, and let δ be the restriction of $\widehat{\delta}$ to the vertices $V(G)$. Denote $n = |V(G)|$ and $m = |E(G)|$. Given a vertex $u \in V(G)$, we can construct the graph G' remaining from G after the edges in $E(B_\delta(u, \rho))$ are discarded in time $O(m)$. Identifying the connected components of G' can be done in time $O(m)$ as well; hence we can decide whether u is a ρ-center with respect to G and δ in time $O(m)$. By repeating this process for every vertex $u \in V(G)$, a ρ-center is found (if one exists) in time $O(nm)$.

Procedure `Construct_Local_Tree` is merely a variant of Dijkstra's single source shortest paths algorithm on the transitive graph $\mathcal{R}$ that has $O(n^2)$ edges; hence it can be implemented to run in time $O(n^2)$. Using a disjoint-set data structure [6], connectivity queries in the forest $\mathcal{F}$ are answered in near-constant time, and the condition that the connectivity partition $\mathcal{P}'$ is a hub is verified in near-linear time. Therefore, the dominant term in the running time of the recursive invocation on G is proportional to nm. Accounting for all recursive invocations, the running time of Algorithm `Construct_Tree` is $O(\min\{\widehat{n}\widehat{m} \log \widehat{n}, \widehat{n}^3\})$.

7. Tightness of the analysis. In this section we prove that the analysis presented in section 6 is tight. Recall that our approximation algorithm invokes Algorithm `Construct_Tree` with different test values $\rho \in (1, 2n]$, finally returning the output of a successful invocation with the smallest ρ. For the sake of the analysis, in this section we assume that the approximation algorithm ignores the spanning trees output by successful invocations with larger test values, although some of these spanning trees may admit smaller maximum stretch.

LEMMA 7.1. *For every $d \in \mathbb{N}_{>0}$ there exists an unweighted graph G_d with $n(d) = \Theta(2^d)$ vertices such that* $\text{max-str}(G_d)$ *is constant while Algorithm* `Construct_Tree`*, when invoked on G_d with test value $\rho = 2$, constructs a spanning tree with maximum stretch $\Omega(\log n)$.*

Proof. Given an integer $d \ge 1$, construct the unweighted graph G_d as follows. Let T and T' be two complete binary trees of depth d, with roots r and r', respectively.

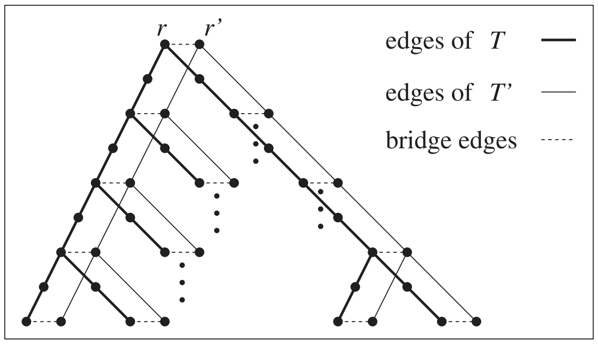

FIG. 7. G_d *for* $d = 4$.

Connect the two trees to each other by adding a *bridge* edge between every pair of corresponding vertices. Split every edge (x, y) in T by adding a new vertex z and replacing (x, y) with new edges (x, z) and (y, z). Figure 7 illustrates the construction of G_4.

Since the number of vertices in a depth d complete binary tree is $2^{d+1} - 1$ and since the number of new vertices added to T is $2^{d+1} - 2$ (equal to the number of edges in a depth d complete binary tree), it follows that

$$n(d) \;=\; |V(G_d)| \;=\; 2\,(2^{d+1}-1) \;+\; 2^{d+1}-2 \;=\; 6\cdot 2^d - 4.$$

It is easy to verify that the spanning tree obtained by removing the edges of T' and remaining with the edges of T plus the bridge edges has maximum stretch 4. On the other hand, when Algorithm `Construct_Tree` is invoked on G_d with test value $\rho = 2$, if the "original" vertices of T (those that existed in T before splitting the edges) are chosen to be the 2-centers on each recursive invocation, then the spanning tree $\widehat{T}$ returned by the algorithm is merely the union of T and T' with their roots connected by an edge, i.e., $E(\widehat{T}) = E(T) \cup E(T') \cup \{(r, r')\}$.

Let l be an arbitrary leaf in T, and let l' be its corresponding leaf in T'. The unique path between l and l' in $\widehat{T}$ goes via the edge (r, r'), and it is of length $2d+1+d = 3d+1$. Therefore, $\widehat{T}$ has maximum stretch $\Omega(d) = \Omega(\log(n(d)))$. $\square$

8. Hardness of approximation.

As mentioned in section 1.2, it is NP-hard to decide, for an arbitrary unweighted graph G, whether or not max-str$(G) \leq 4$ [5]. Moreover, since the maximum stretch of a tree on an unweighted graph is always obtained on a pair of vertices that form an edge in the original graph, it follows that max-str(G) must be an integer. Therefore, we have the following.

COROLLARY 8.1. *It is NP-hard to approximate the MMST problem on unweighted graphs by a ratio better than* $5/4$.

We show that unless P = NP, the problem cannot be approximated additively by a term of $o(n)$.

LEMMA 8.2. *It is NP-hard to distinguish between unweighted graphs with minimum max-stretch at most* $5k - 1$ *and those with minimum max-stretch at least* $6k - 1$ *for any positive integer* k.

Proof. The proof is by reduction from the problem of deciding whether an arbitrary unweighted graph has minimum max-stretch at most 4. Consider some positive integer k. Given an arbitrary unweighted graph G, construct the unweighted graph G'_k by replacing every edge $(u, v) \in E(G)$ with a unique path $P_{u,v}$ of length k between u

and v. Observe that every spanning tree T' of G'_k corresponds to a spanning tree T of G where the edge (u,v) is in $E(T)$ if and only if all the edges of $P_{u,v}$ are in $E(T')$. Moreover, if the spanning tree T' of G'_k corresponds to the spanning tree T of G, then, since T' is connected, it follows that

$$|E(T') \cap E(P_{u,v})| \;=\; \begin{cases} k & \text{if } (u,v) \in E(T), \\ k-1 & \text{otherwise} \end{cases}$$

for every $(u,v) \in E(G)$; i.e., the tree T' is missing at most a single edge from every such path $P_{u,v}$. Therefore, $\text{max-str}(T', G'_k) = k \cdot (\text{max-str}(T,G) + 1) - 1$. Thus if G has minimum max-stretch at most 4, then G'_k has minimum max-stretch at most $5k-1$. Otherwise, if G has minimum max-stretch at least 5, then G'_k has minimum max-stretch at least $6k-1$. ∎

Given an approximation algorithm A for the MMST problem on unweighted graphs and an unweighted graph G, let $A(G)$ denote the tree returned by A when invoked on G.

Corollary 8.3. *If there exist some real $\delta = o(n)$ and $\epsilon > 0$ and an approximation algorithm A for the MMST problem on unweighted graphs with performance guarantee*

$$\text{max-str}(A(G), G) \;\le\; \delta + (6/5 - \epsilon) \cdot \text{max-str}(G)$$

for every unweighted graph G, then $P = NP$.

REFERENCES

[1] N. Alon, R. M. Karp, D. Peleg, and D. West, *A graph-theoretic game and its application to the k-server problem*, SIAM J. Comput., 24 (1995), pp. 78–100.

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

[3] Y. Bartal, *On approximating arbitrary metrics by tree metrics*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 1998, pp. 161–168.

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

[5] L. Cai and D. G. Corneil, *Tree spanners*, SIAM J. Discrete Math., 8 (1995), pp. 359–387.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 2001.

[7] M. J. Demmer and M. Herlihy, *The arrow distributed directory protocol*, in Proceedings of the 12th International Symposium on Distributed Computing (DISC), 1998, Springer-Verlag, New York, pp. 119–133.

[8] E. W. Dijkstra, *A note on two problems in connexion with graphs*, Numer. Math., 1 (1959), pp. 269–271.

[9] D. Dor, S. Halperin, and U. Zwick, *All pairs almost shortest paths*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science (FOCS), IEEE Computer Society, Washington, DC, 1996, pp. 452–461.

[10] M. Elkin, Y. Emek, D.A. Spielman, and S.-H. Teng, *Lower-stretch spanning trees*, in Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2005, pp. 494–503.

[11] Y. Emek and D. Peleg, *Approximating minimum max-stretch spanning trees on unweighted graphs*, in Proceedings of the Fifteenth ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2004, pp. 261–270.

[12] S. P. Fekete and J. Kremer, *Tree spanners in planar graphs*, in Proceedings of the 24th International Workshop on Graph Theoretic Concepts in Computer Science, Springer-Verlag, London, 1998, pp. 298–309.

[13] J. Fakcharoenphol, S. Rao, and K. Talwar, *A tight bound on approximating arbitrary metrics by tree metrics*, in Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), ACM, New York, 2003, pp. 448–455.

[14] G. Galbiati, *On min-max cycle bases*, in Proceedings of the 12th International Symposium on Algorithms and Computation (ISAAC), Springer-Verlag, London, 2001, pp. 116–123.

[15] A. Gupta, *Steiner points in tree metrics don't (really) help*, in Proceedings of the Twelfth ACM-SIAM Symposium on Discrete Algorithms (SODA), ACM, New York, SIAM, Philadelphia, 2001, pp. 220–227.

[16] R. Hassin and A. Levin, *Minimum restricted diameter spanning trees*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), Springer-Verlag, New York, 2002, pp. 175–184.

[17] M. Herlihy, F. Kuhn, S. Tirthapura, and R. Wattenhofer, *Dynamic analysis of the arrow distributed protocol*, Theory Comput. Syst., 39 (2006), pp. 875–901.

[18] C. Liebchen and G. Wünsch, *The zoo of tree spanner problems*, Discrete Appl. Math., 156 (2008), pp. 569–587.

[19] D. Peleg and E. Reshef, *Low complexity variants of the arrow distributed directory*, J. Comput. System Sci., 63 (2001), pp. 474–485.

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

[21] D. Peleg and D. Tendler, *Low Stretch Spanning Trees for Planar Graphs*, Technical report MCS01-14, The Weizmann Institute of Science, Rehovot, Israel, 2001.

[22] D. Peleg and J. D. Ullman, *An optimal synchronizer for the hypercube*, SIAM J. Comput., 18 (1989), pp. 740–747.