Lower-Stretch Spanning Trees

Michael Elkin*
Department of Computer Science
Ben-Gurion University of the Negev

Yuval Emek
Department of Computer Science
and Applied Mathematics
Weizmann Institute of Science

Daniel A. Spielman†
Department of Mathematics
Massachusetts Institute of Technology

Shang-Hua Teng‡
Department of Computer Science
Boston University and
Akamai Technologies Inc.

Abstract

We show that every weighted connected graph G contains as a subgraph a spanning tree into which the edges of G can be embedded with average stretch $O(\log^2 n \log\log n)$. Moreover, we show that this tree can be constructed in time $O(m \log n + n \log^2 n)$ in general, and in time $O(m \log n)$ if the input graph is unweighted. The main ingredient in our construction is a novel graph decomposition technique.

Our new algorithm can be immediately used to improve the running time of the recent solver for symmetric diagonally dominant linear systems of Spielman and Teng from

$$m 2^{\left(O(\sqrt{\log n \log\log n})\right)} \text{ to } m \log^{O(1)} n,$$

and to $O(n \log^2 n \log\log n)$ when the system is planar. Our result can also be used to improve several earlier approximation algorithms that use low-stretch spanning trees.

Categories and Subject Descriptors: F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity

General Terms: Algorithms, Theory.

Keywords: Low-distortion embeddings, probabilistic tree metrics, low-stretch spanning trees.

1 Introduction

Let $G = (V, E, w)$ be a weighted connected graph, where w is a function from E into the positive reals. We define the length of each edge $e \in E$ to be the reciprocal of its weight:

$$d(e) = 1/w(e).$$

*Part of this work was done in Yale University, and was supported by the DoD University Research Initiative (URI) administered by the Office of Naval Research under Grant N00014-01-1-0795. The work was also partially supported by the Lynn and William Frankel Center for Computer Sciences.

†Partially supported by NSF grant CCR-0324914. Part of this work was done at Yale University.

‡Partially supported by NSF grants CCR-0311430 and ITR CCR-0325630.

Given a spanning tree T of V, we define the distance in T between a pair of vertices $u, v \in V$, $\mathrm{dist}_T(u, v)$, to be the sum of the lengths of the edges on the unique path in T between u and v. We can then define the stretch[1] of an edge $(u, v) \in E$ to be

$$\mathrm{stretch}_T(u, v) = \frac{\mathrm{dist}_T(u, v)}{d(u, v)},$$

and the average stretch over all edges of E to be

$$\text{ave-stretch}_T(E) = \frac{1}{|E|} \sum_{(u,v) \in E} \mathrm{stretch}_T(u, v).$$

Alon, Karp, Peleg and West [1] proved that every weighted connected graph $G = (V, E, w)$ of n vertices and m edges contains a spanning tree T such that

$$\text{ave-stretch}_T(E) = \exp\left(O(\sqrt{\log n \log\log n})\right),$$

and that there exists a collection $\tau = \{T_1, \ldots, T_h\}$ of spanning trees of G and a probability distribution Π over τ such that for every edge $e \in E$,

$$\mathbf{E}_{T \leftarrow \Pi}\left[\mathrm{stretch}_T(e)\right] = \exp\left(O(\sqrt{\log n \log\log n})\right).$$

The class of weighted graphs considered in this paper includes multi-graphs that may contain weighted self-loops and multiple weighted-edges between a pair of vertices. The consideration of multi-graphs is essential for several results (including some in [1]).

The result of [1] triggered the study of low-distortion embeddings into *probabilistic tree metrics*. Most notable in this context is the work of Bartal [5, 6] which shows that if the requirement that the trees T be *subgraphs* of G is abandoned, then the upper bound of [1] can be improved by finding a tree whose distances approximate those in the original graph with average distortion $O(\log n \cdot \log\log n)$. On the negative side, a lower bound of $\Omega(\log n)$ is known for both scenarios [1, 5]. The gap left by Bartal was recently closed by Fakcharoenphol, Rao, and Talwar [11], who have shown a tight upper bound of $O(\log n)$.

However, some applications of graph-metric-approximation require trees that are subgraphs. Until now, no progress had been made on reducing the gap between the upper and lower bounds proved in [1] on the average stretch of subgraph spanning trees. The bound achieved in [1] for general weighted graphs had been the best bound known for *unweighted* graphs, even for *unweighted planar* graphs.

In this paper[2], we significantly narrow this gap by improving the upper bound of [1] from $\exp(O(\sqrt{\log n \log\log n}))$ to $O(\log^2 n \log\log n)$. Specifically, we give an algorithm that for every weighted connected graph $G = (V, E, w)$, constructs a spanning tree $T \subseteq E$ that satisfies $\text{ave-stretch}_T(E) = O(\log^2 n \log\log n)$. The running time of our algorithm is $O(m \log n + n \log^2 n)$ for weighted graphs, and $O(m \log n)$ for unweighted. Note that the input graph need not be simple

[1] Our definition of the stretch differs slightly from that used in [1]: $\mathrm{dist}_T(u, v)/\mathrm{dist}_G(u, v)$, where $\mathrm{dist}_G(y, v)$ is the length of the shortest-path between u and v. See Subsection 1.1 for a discussion of the difference.

[2] In a previous version of this paper, we proved the weaker bound on average stretch of $O((\log n \log\log n)^2)$. The improvement in this paper comes from re-arranging the arithmetic in our analysis. Bartal [7] has obtained a similar improvement by other means.

and its number of edges m can be much larger than $\binom{n}{2}$. However, as proved in [1], it is enough to consider graphs with at most $n(n+1)$ edges.

We begin by presenting a simpler algorithm that guarantees a weaker bound, $\text{ave-stretch}_T(E) = O(\log^3 n)$. As a consequence of the result in [1] that the existence of a spanning tree with average stretch $f(n)$ for every weighted graph implies the existence of a distribution of spanning trees in which every edge has expected stretch $f(n)$, our result implies that for every weighted connected graph $G = (V, E, w)$ there exists a probability distribution Π over a set $\tau = \{T_1, \ldots, T_h\}$ of spanning trees ($T \subseteq E$ for every $T \in \tau$) such that for every $e \in E$, $\mathbf{E}_{T \leftarrow \Pi}[\text{stretch}_T(e)] = O(\log^2 n \log\log n)$. Furthermore, our algorithm itself can be adapted to produce a probability distribution Π that guarantees a slightly weaker bound of $O(\log^3 n)$ in time $O(m \cdot \log^2 n)$. So far, we have not yet been able to verify whether our algorithm can be adapted to produce the bound of $O(\log^2 n \cdot \log\log n)$ within similar time limits.

1.1 Applications

For some of the applications listed below it is essential to define the stretch of an edge $(u, v) \in E$ as in [1], namely, $\text{stretch}_T(u, v) = \text{dist}_T(u, v)/\text{dist}_G(u, v)$. The algorithms presented in this paper can be adapted to handle this alternative definition for stretch simply by assigning new weight $w'(u, v) = 1/\text{dist}_G(u, v)$ to every edge $(u, v) \in E$ (the lengths of the edges remain unchanged). Observe that the new weights can be computed in a preprocessing stage independently of the algorithms themselves, but the time required for this computation may dominate the running time of the algorithms.

1.1.1 Solving Linear Systems

Boman and Hendrickson [8] were the first to realize that low-stretch spanning trees could be used to solve symmetric diagonally dominant linear systems. They applied the spanning trees of [1] to design solvers that run in time

$$m^{3/2} 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon),$$

where ϵ is the precision of the solution. Spielman and Teng [21] improved their results to

$$m 2^{O(\sqrt{\log n \log\log n})} \log(1/\epsilon).$$

Unfortunately, the trees produced by the algorithms of Bartal [5, 6] and Fakcharoenphol, Rao, and Talwar [11] cannot be used to improve these linear solvers, and it is currently not known whether it is possible to solve linear systems efficiently using trees that are not subgraphs.

By applying the low-stretch spanning trees developed in this paper, we can reduce the time for solving these linear systems to

$$m \log^{O(1)} n \log(1/\epsilon),$$

and to $O(n \log^2 n \log\log n \log(1/\epsilon))$ when the systems are planar. Applying a recent reduction of Boman, Hendrickson and Vavasis [9], one obtains a $O(n \log^2 n \log\log n \log(1/\epsilon))$ time algorithm for solving the linear systems that arise when applying the finite element method to solve two-dimensional elliptic partial differential equations.

1.1.2 Alon-Karp-Peleg-West Game

Alon, Karp, Peleg and West [1] constructed low-stretch spanning trees to upper-bound the value of a zero-sum two-player game that arose in their analysis of an algorithm for the k-server problem: at each turn, the *tree player* chooses a spanning tree T and the *edge player* chooses an edge $e \in E$, simultaneously. The payoff to the edge player is 0 if $e \in T$ and $\mathrm{stretch}_T(e) + 1$ otherwise. They showed that if every n-vertex weighted connected graph G has a spanning tree T of average stretch $f(n)$, then the value of this game is at most $f(n) + 1$. Our new result lowers the bound on the value of this graph-theoretical game from $\exp\left(O(\sqrt{\log n \log\log n})\right)$ to $O\left(\log^2 n \log\log n\right)$.

1.1.3 MCT Approximation

Our result can be used to improve drastically the upper bound on the approximability of the *minimum communication cost spanning tree* (henceforth, *MCT*) problem. This problem was introduced in [14], and is listed as [ND7] in [12] and [10].

The instance of this problem is a weighted graph $G = (V, E, w)$, and a matrix $\{r(u, v) \mid u, v \in V\}$ of nonnegative requirements. The goal is to construct a spanning tree T that minimizes $c(T) = \sum_{u,v \in V} r(u, v) \cdot \mathrm{dist}_T(u, v)$.

Peleg and Reshef [19] developed a $2^{O(\sqrt{\log n \cdot \log\log n})}$ approximation algorithm for the MCT problem on metrics using the result of [1]. A similar approximation ratio can be achieved for arbitrary graphs. Therefore our result can be used to produce an efficient $O(\log^2 n \log\log n)$ approximation algorithm for the MCT problem on arbitrary graphs.

1.1.4 Message-Passing Model

Embeddings into probabilistic tree metrics have been extremely useful in the context of approximation algorithms (to mention a few: buy-at-bulk network design [3], graph Steiner problem [13], covering Steiner problem [15]). However, it is not clear that these algorithms can be implemented in the message-passing model of distributed computing (see [18]). In this model, every vertex of the input graph hosts a processor, and the processors communicate over the edges of the graph.

Consequently, in this model executing an algorithm that starts by constructing a non-subgraph spanning tree of the network, and then solves a problem whose instance is this tree is very problematic, since direct communication over the links of this "virtual" tree is impossible. This difficulty disappears if the tree in this scheme is a *subgraph* of the graph. We believe that our result will enable the adaptation of the these approximation algorithms to the message-passing model.

1.2 Our Techniques

We build our low-stretch spanning trees by recursively applying a new graph decomposition that we call a *star-decomposition*. A star-decomposition of a graph is a partition of the vertices into sets that are connected into a star: a central set is connected to each other set by a single edge (see Figure 1). We show how to find star-decompositions that do not cut too many short edges and such that the radius of the graph induced by the star decomposition is not much larger than the radius of the original graph.

Our algorithm for finding a low-cost star-decomposition applies a generalization of the ball-growing technique of Awerbuch [4] to grow *cones*, where the cone at a vertex x induced by a set of vertices S is the set of vertices whose shortest path to S goes through x.

1.3 The Structure of the Paper

In Section 2, we define our notation. In Section 3, we introduce the star decomposition of a weighted connected graph. We then show how to use this decomposition to construct a subgraph spanning tree with average stretch $O(\log^3 n)$. In Section 4, we present our star decomposition algorithm. In Section 5, we refine our construction and improve the average stretch to $O\left(\log^2 n \log\log n\right)$. Finally, we conclude the paper in Section 6 and list some open questions.

2 Preliminaries

Throughout the paper, we assume that the input graph is a weighted connected multi-graph $G = (V, E, w)$, where w is a *weight* function from E to the positive reals. Unless stated otherwise, we let n and m denote the number of vertices and the number of edges in the graph, respectively. The *length* of an edge $e \in E$ is defined as the reciprocal of its weight, denoted by $d(e) = 1/w(e)$.

For two vertices $u, v \in V$, we define $\mathrm{dist}(u, v)$ to be the length of the shortest path between u and v in E. We write $\mathrm{dist}_G(u, v)$ to emphasize that the distance is in the graph G.

For a set of vertices, $S \subseteq V$, $G(S)$ is the subgraph induced by vertices in S. We write $\mathrm{dist}_S(u, v)$ instead of $\mathrm{dist}_{G(S)}(u, v)$ when G is understood. $E(S)$ is the set of edges with both endpoints in S.

For $S, T \subseteq V$, $E(S, T)$ is the set of edges with one endpoint in S and the other in T.

The *boundary* of a vertex set $S \subseteq V$, denoted $\partial(S)$, is the set of edges with exactly one endpoint in S.

For a vertex $x \in V$, $\mathrm{dist}_G(x, S)$ is the length of the shortest path between x and a vertex in S.

A *multiway partition* of V is a collection of pairwise-disjoint sets $\{V_1, \ldots, V_k\}$ such that $\bigcup_i V_i = V$. The *boundary* of a multiway partition, denoted $\partial(V_1, \ldots, V_k)$, is the set of edges with endpoints in different sets in the partition.

The *volume* of a set F of edges, denoted $\mathrm{vol}(F)$, is the size of the set $|F|$.

The *cost* of a set F of edges, denoted $\mathrm{cost}(F)$, is the sum of the weights of the edges in F.

The *volume* of a set S of vertices, denoted $\mathrm{vol}(S)$, is the number of edges with at least one endpoint in S.

The *ball* of radius r around a vertex v, denoted $B(r, v)$, is the set of vertices of distance at most r from v.

The *ball shell* of radius r around a vertex v, denoted $\mathrm{BS}(r, v)$, is the set of vertices right outside $B(r, v)$, that is, $\mathrm{BS}(r, v)$ consists of every vertex $u \in V - B(r, v)$ with a neighbor $w \in B(r, v)$ such that $\mathrm{dist}(v, u) = \mathrm{dist}(v, w) + d(u, w)$.

For $v \in V$, $\mathrm{rad}_G(v)$ is the smallest r such that every vertex of G is within distance r from v. For a set of vertices $S \subseteq V$, we write $\mathrm{rad}_S(v)$ instead of $\mathrm{rad}_{G(S)}(v)$ when G is understood.

3 Spanning-Trees of O($\log^3$n) Stretch

We present our first algorithm that generates a spanning tree with average stretch $O(\log^3 n)$. We first state the properties of the graph decomposition algorithm at the heart of our construction. We then present the construction and analysis of the low-stretch spanning trees. We defer the description of the graph decomposition algorithm and its analysis to Section 4.

3.1 Low-Cost Star-Decomposition

Definition 3.1 (Star-Decomposition). *A multiway partition $\{V_0, \ldots, V_k\}$ is a* star-decomposition *of a weighted connected graph G with center $x_0 \in V$ (see Figure 1) if $x_0 \in V_0$ and*

1. *for all $0 \le i \le k$, the subgraph induced on V_i is connected, and*
2. *for all $i \ge 1$, V_i contains an* **`anchor`** *vertex x_i that is connected to a vertex $y_i \in V_0$ by an edge $(x_i, y_i) \in E$. We call the edge (x_i, y_i) the* **`bridge`** *between V_0 and V_i.*

Let $r = \mathrm{rad}_G(x_0)$, and $r_i = \mathrm{rad}_{V_i}(x_i)$ for each $0 \le i \le k$. For $\delta, \epsilon \le 1/2$, a star-decomposition $\{V_0, \ldots, V_k\}$ is a (δ, ϵ)-star-decomposition if

a. $\delta r \le r_0 \le (1-\delta) r$, and

b. $r_0 + d(x_i, y_i) + r_i \le (1+\epsilon) r$, for each $i \ge 1$.

The `cost` *of the star-decomposition is $cost(\partial(V_0, \ldots, V_k))$.*

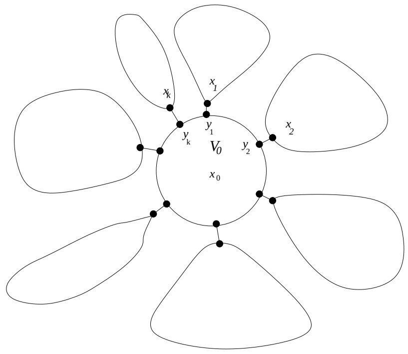

Figure 1: Star Decomposition.

Note that if $\{V_0, \ldots, V_k\}$ is a (δ, ϵ) star-decomposition of G, then the graph consisting of the union of the induced subgraphs on $V_0, \ldots, V_k$ and the bridge edges (y_k, x_k) has radius at most $(1+\epsilon)$ times the radius of the original graph.

In Section 4, we present an algorithm `StarDecomp` that satisfies the following cost guarantee. Let $\boldsymbol{x} = (x_1, \ldots, x_k)$ and $\boldsymbol{y} = (y_1, \ldots, y_k)$.

Lemma 3.2 (Low-Cost Star Decomposition). *Let $G = (V, E, w)$ be a connected weighted graph and let x_0 be a vertex in V. Then for every positive $\epsilon \le 1/2$,*

$$(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{starDecomp}(G, x_0, 1/3, \epsilon) ,$$

in time $O(m + n \log n)$, returns a $(1/3, \epsilon)$-star-decomposition of G with center x_0 of cost

$$\mathit{cost}\,(\partial\,(V_0, \ldots, V_k)) \;\leq\; \frac{6\, m \log_2(m+1)}{\epsilon \cdot \mathrm{rad}_G\,(x_0)}.$$

On unweighted graphs, the running time is $O(m)$.

3.2 A Divide-and-Conquer Algorithm

The basic idea of our algorithm is to use low-cost star-decomposition in a divide-and-conquer (recursive) algorithm to construct a spanning tree. We use $\widehat{n}$ (respectively, $\widehat{m}$) rather than n (resp., m) to distinguish the number of vertices (resp., number of edges) in the original graph, input to the first recursive invocation, from that of the graph input to the current one.

Before invoking our algorithm we apply a linear-time transformation from [1] that transforms the graph into one with at most $\widehat{n}(\widehat{n}+1)$ edges (recall that a multi-graph of $\widehat{n}$ vertices may have an arbitrary number of edges), and such that the average-stretch of the spanning tree on the original graph will be at most twice the average-stretch on this graph.

We begin by showing how to construct low-stretch spanning trees in the case that all edges have length 1. In particular, we use the fact that in this case the cost of a set of edges equals the number of edges in the set.

Fix $\alpha = (2 \log_{4/3}(\widehat{n} + 6))^{-1}$.

$T = \texttt{UnweightedLowStretchTree}(G, x_0)$.

0. If $|V| \leq 2$, return G. (If G contains multiple edges, return a single copy.)
1. Set $\rho = \mathrm{rad}_G\,(x_0)$.
2. $(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{StarDecomp}(G, x_0, 1/3, \alpha)$
3. For $0 \leq i \leq k$,
   set $T_i = \texttt{UnweightedLowStretchTree}(G(V_i), x_i)$
4. Set $T = \bigcup_i T_i \cup \bigcup_i (y_i, x_i)$.

Theorem 3.3 (Unweighted)**.** *Let $G = (V, E)$ be an unweighted connected graph and let x_0 be a vertex in V. Then*

$$T = \texttt{UnweightedLowStretchTree}(G, x_0)\ ,$$

in time $O(\widehat{m} \log \widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T\,(x_0) \leq \sqrt{e} \cdot \mathrm{rad}_G\,(x_0) \tag{1}$$

and

$$\mathit{ave\text{-}stretch}_T(E) \leq O(\log^3 \widehat{m})\ . \tag{2}$$

Proof. For our analysis, we define a family of graphs that converges to T. For a graph G, we let

$$(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{StarDecomp}(G, x_0, 1/3, \alpha)$$

and recursively define

$$R_0(G) \;=\; G \quad \text{and} \qquad R_t(G) \;=\; \bigcup_i (y_i, x_i) \cup \bigcup_i R_{t-1}(G(V_i)) \; .$$

The graph $R_t(G)$ is what one would obtain if we forced `UnweightedLowStretchTree` to return its input graph after t levels of recursion.

Because for all $\widehat{n} \geq 0$, $(2\log_{4/3}(\widehat{n}+6))^{-1} \leq 1/12$, we have $(2/3+\alpha) \leq 3/4$. Thus, the depth of the recursion in `UnweightedLowStretchTree` is at most $\log_{4/3}\widehat{n}$, and we have $R_{\log_{4/3}\widehat{n}}(G) = T$. One can prove by induction that, for every $t \geq 0$,

$$\mathrm{rad}_{R_t(G)}\left(x_0\right) \leq (1+\alpha)^t \mathrm{rad}_G\left(x_0\right).$$

The claim in (1) now follows from $(1+\alpha)^{\log_{4/3}\widehat{n}} \leq \sqrt{e}$. To prove the claim in (2), we note that

$$\begin{aligned}
\sum_{(u,v)\in\partial(V_0,\ldots,V_k)} \mathrm{stretch}_T(u,v) \;&\leq\; \sum_{(u,v)\in\partial(V_0,\ldots,V_k)} \left(\mathrm{dist}_T(x_0,u) + \mathrm{dist}_T(x_0,v)\right) && (3)\\
&\leq\; \sum_{(u,v)\in\partial(V_0,\ldots,V_k)} 2\sqrt{e}\cdot \mathrm{rad}_G\left(x_0\right), \text{ by } (1) && (4)\\
&\leq\; 2\sqrt{e}\cdot \mathrm{rad}_G\left(x_0\right) \left(\frac{6\,\widehat{m}\log_2(\widehat{m}+1)}{\alpha\cdot \mathrm{rad}_G\left(x_0\right)}\right), \text{ by Lemma 3.2.}
\end{aligned}$$

Applying this inequality to all graphs at all $\log_{4/3}\widehat{n}$ levels of the recursion, we obtain

$$\sum_{(u,v)\in E} \mathrm{stretch}_T(u,v) \;\leq\; 24\sqrt{e}\,\widehat{m}\log_2\widehat{m}\log_{4/3}\widehat{n}\log_{4/3}(\widehat{n}+6) \;=\; O(\widehat{m}\log^3\widehat{m}) \; .$$

(To justify the last inequality we remark that each edge ends up in $\partial\left(V_0,\ldots,V_k\right)$ on one of the levels of recursion, or it survives all the way down to a component of size two. In the latter case its contribution to the stretch is 1.) □

We now extend our algorithm and proof to general weighted connected graphs. We begin by pointing out a subtle difference between general and unit-weight graphs. In our analysis of `UnweightedLowStretchTree`, we used the facts that $\mathrm{rad}_G\left(x_0\right) \leq n$ and that each edge length is 1 to show that the depth of recursion is at most $\log_{4/3} n$. In general weighted graphs, the ratio of $\mathrm{rad}_G\left(x_0\right)$ to the length of the shortest edge can be arbitrarily large. Thus, the recursion can be very deep. To compensate, we will contract all edges that are significantly shorter than the radius of their component. In this way, we will guarantee that each edge is only active in a logarithmic number of iterations.

Let $e = (u,v)$ be an edge in $G = (V,E,w)$. The contraction of e results in a new graph by identifying u and v as a new vertex whose neighbors are the union of the neighbors of u and v. All self-loops created by the contraction are discarded. We refer to u and v as the preimage of the new vertex.

We now state and analyze our algorithm for general weighted graphs.

Fix $\beta = (2\log_{4/3}(\widehat{n}+32))^{-1}$.

$T = \texttt{LowStretchTree}(G = (V, E, w), x_0)$.

0. If $|V| \leq 2$, return G. (If G contains multiple edges, return the shortest copy.)
1. Set $\rho = \mathrm{rad}_G(x_0)$.
2. Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be the graph obtained by contracting all edges in G of length less than $\beta\rho/\widehat{n}$.
3. $\left(\left\{\widetilde{V}_0, \ldots, \widetilde{V}_k\right\}, \widetilde{\boldsymbol{x}}, \widetilde{\boldsymbol{y}}\right) = \texttt{StarDecomp}(\widetilde{G}, x_0, 1/3, \beta)$.
4. For each i, let V_i be the preimage under the contraction of step 2 of vertices in $\widetilde{V}_i$, and $(x_i, y_i) \in V_0 \times V_i$ be the edge of shortest length for which x_i is a preimage of $\widetilde{x}_i$ and y_i is a preimage of $\widetilde{y}_i$.
5. For $0 \leq i \leq k$, set $T_i = \texttt{LowStretchTree}(G(V_i), x_i)$
6. Set $T = \bigcup_i T_i \cup \bigcup_i (y_i, x_i)$.

In what follows, we refer to the graph $\widetilde{G}$, obtained by contracting some of the edges of the graph G, as the *edge-contracted graph.*

Theorem 3.4 (Low-Stretch Spanning Tree)**.** *Let $G = (V, E, w)$ be a weighted connected graph and let x_0 be a vertex in V. Then*

$$T = \texttt{LowStretchTree}(G, x_0) ,$$

in time $O(\widehat{m}\log\widehat{n} + \widehat{n}\log^2\widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T(x_0) \leq 2\sqrt{e} \cdot \mathrm{rad}_G(x_0) \tag{5}$$

and

$$\textit{ave-stretch}_T(E) = O(\log^3 \widehat{m}) \tag{6}$$

Proof. We first establish notation similar to that used in the proof of Theorem 3.3. Our first step is to define a procedure, SD, that captures the action of the algorithm in steps 2 through 4. We then define $R_0(G) = G$ and

$$R_t(G) = \bigcup_i (y_i, x_i) \cup \bigcup_i R_{t-1}(G(V_i)),$$

where

$$(\{V_0, \ldots, V_k\}, x_1, \ldots, x_k, y_1, \ldots, y_k) = \texttt{SD}(G, x_0, 1/3, \beta).$$

We now prove the claim in (5). Let $\rho = \mathrm{rad}_G(x_0)$. Let $t = 2\log_{4/3}(\widehat{n}+32)$ and let $\rho_t = \mathrm{rad}_{R_t(G)}(x_0)$. Each contracted edge is of length at most $\beta\rho/\widehat{n}$, and every path in the graph $G(V_i)$ contains at most n contracted edges, hence the insertion of the contracted edges to $G(V_i)$ increases its radius by an additive factor of at most $\beta\rho$. Since $(2\log_{4/3}(\widehat{n}+32))^{-1} \leq 1/24$ for every $n \geq 0$, it follows that $2/3 + 2\beta \leq 3/4$. Therefore, following the proof of Theorem 3.3, we can show that ρ_t is at most $\sqrt{e} \cdot \mathrm{rad}_G(x_0)$.

We know that each component of G that remains after t levels of the recursion has radius at most $\rho(3/4)^t \leq \rho/\widehat{n}^2$. We may also assume by induction that for the graph induced on each of these components, `LowStretchTree` outputs a tree of radius at most $2\sqrt{e}(\rho/\widehat{n}^2)$. As there are at most n of these components, we know that the tree returned by the algorithm has radius at most

$$\sqrt{e}\,\rho + n \times 2\sqrt{e}(\rho/\widehat{n}^2) \;\leq\; 2\sqrt{e}\,\rho\ ,$$

for $\widehat{n} \geq 2$.

We now turn to the claim in (6), the bound on the stretch. In this part, we let $E_t \subseteq E$ denote the set of edges that are present at recursion depth t. That is, their endpoints are not identified by the contraction of short edges in step 2, and their endpoints remain in the same component. We now observe that no edge can be present at more than $\log_{4/3}((2\widehat{n}/\beta)+1)$ recursion depths. To see this, consider an edge (u,v) and let t be the first recursion level for which the edge is in E_t. Let ρ_t be the radius of the component in which the edge lies at that time. As u and v are not identified under contraction, they are at distance at least $\beta\rho_t/\widehat{n}$ from each other. (This argument can be easily verified, although the condition for edge contraction depends on the length of the edge rather than on the distance between its endpoints.) If u and v are still in the same graph on recursion level $t + \log_{4/3}((2\widehat{n}/\beta)+1)$, then the radius of this graph is at most $\rho_t/((2\widehat{n}/\beta)+1)$, thus its diameter is strictly less than $\beta\rho_t/\widehat{n}$, in contradiction to the distance between u and v.

Similarly to the way that $\sum_{(u,v)\in\partial(V_0,\ldots,V_k)} \mathrm{stretch}_T(u,v)$ is upper-bounded in inequalities (3)-(4) in the proof of Theorem 3.3, it follows that the total contribution to the stretch at depth t is at most

$$O(\mathrm{vol}\,(E_t)\log^2\widehat{m}).$$

Thus, the sum of the stretches over all recursion depths is

$$\sum_t O\left(\mathrm{vol}\,(E_t)\log^2\widehat{m}\right) = O\left(\widehat{m}\log^3\widehat{m}\right).$$

We now analyze the running time of the algorithm. On each recursion level, the dominant cost is that of performing the `StarDecomp` operations on each edge-contracted graph. Let V_t denote the set of vertices in all edge-contracted graphs on recursion level t. Then the total cost of the `StarDecomp` operations on recursion level t is at most $O(|E_t| + |V_t|\log|V_t|)$. We will prove soon (Lemma 3.5) that $\sum_t |V_t| = O(\widehat{n}\log\widehat{n})$, and as $\sum_t |E_t| = O(\widehat{m}\log\widehat{m})$, it follows that the total running time is $O(\widehat{m}\log\widehat{n} + \widehat{n}\log^2\widehat{n})$. □

The following lemma shows that even though the number of recursion levels can be very large, the overall number of vertices in edge-contracted graphs appearing on different recursion levels is at most $O(\widehat{n}\log\widehat{n})$. This lemma is used only for the analysis of the running time of our algorithm; a reader interested only in the existential bound can skip it.

Lemma 3.5. *Let V_t be the set of vertices appearing in edge-contracted graphs on recursion level t. Then $\sum_t |V_t| = O(\widehat{n}\log\widehat{n})$.*

Proof. Consider an edge-contracted graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ on recursion level t and let x be a vertex in $\widetilde{V}$. The vertex x was formed as a result of a number of edge contractions (maybe 0). Consequently, x can be viewed as a set of all original vertices that were identified together to form it, i.e., x can be viewed as a *super-vertex*. Let $\chi(x)$ denote the set of original vertices that were identified together to form the super-vertex $x \in \widetilde{V}$.

We claim that for every super-vertex $x \in V_{t+1}$, there exist a super-vertex $y \in V_t$ such that $\chi(x) \subseteq \chi(y)$. To prove it, note that every graph on recursion level $t+1$ corresponds to a single component of a star decomposition on recursion level t. Moreover, an edge that was contracted on recursion level $t+1$ must have been contracted on recursion level t as well. Therefore we can consider a directed forest $\mathcal{F}$ in which every node at depth t corresponds to some super-vertex in V_t, and an edge leads from a node y at depth t to a node x at depth $t+1$, if $\chi(x) \subseteq \chi(y)$. Note that the roots of $\mathcal{F}$ correspond to super-vertices on recursion level 0 and the leaves of $\mathcal{F}$ correspond to the original vertices of the graph G.

In the proof of Theorem 3.4, we showed that every edge is present on $O(\log \widehat{n})$ recursion levels. Following a similar line of arguments, one can show that every super-vertex is present on $O(\log \widehat{n})$ (before it decomposes to smaller super-vertices, each contains a subset of its vertices). Since there are $\widehat{n}$ vertices in the original graph G, there are $\widehat{n}$ leaves in $\mathcal{F}$. Therefore the overall number of nodes in the directed forest $\mathcal{F}$ is $O(\widehat{n} \log \widehat{n})$, and the bound on $\sum_t |V_t|$ holds.

□

4 Star decomposition

Our star decomposition algorithm exploits two algorithms for growing sets. The first, `BallCut`, is the standard ball growing technique introduced by Awerbuch [4], and was the basis of the algorithm of [1]. The second, `ConeCut`, is a generalization of ball growing to cones. So that we can analyze this second algorithm, we abstract the analysis of ball growing from the works of [16, 2, 17]. Instead of nested balls, we consider *concentric systems*, which we now define.

Definition 4.1 (Concentric System). *A* concentric system *in a weighted graph $G = (V, E, w)$ is a family of vertex sets $\mathcal{L} = \{L_r \subseteq V : r \in \mathbb{R}^+ \cup \{0\}\}$ such that*

1. *$L_0 \neq \emptyset$,*
2. *$L_r \subseteq L_{r'}$ for all $r < r'$, and*
3. *if a vertex $u \in L_r$ and (u, v) is an edge in E then $v \in L_{r+d(u,v)}$.*

For example, for any vertex $x \in V$, the set of balls $\{B(r, x)\}$ is a concentric system. The *radius* of a concentric system $\mathcal{L}$ is $radius(\mathcal{L}) = \min\{r : L_r = V\}$. For each vertex $v \in V$, we define $\|v\|_{\mathcal{L}}$ to be the smallest r such that $v \in L_r$.

Lemma 4.2 (Concentric System Cutting). *Let $G = (V, E, w)$ be a connected weighted graph and let $\mathcal{L} = \{L_r\}$ be a concentric system. For every two reals $0 \leq \lambda < \lambda'$, there exists a real $r \in [\lambda, \lambda')$ such that*

$$cost(\partial(L_r)) \;\leq\; \frac{\mathrm{vol}(L_r) + \tau}{\lambda' - \lambda} \max\left[1, \log_2\left(\frac{m + \tau}{\mathrm{vol}(E(L_\lambda)) + \tau}\right)\right],$$

where $m = |E|$ and

$$\tau = \begin{cases} 1 & \text{if } \mathrm{vol}(E(L_\lambda)) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Proof. Note that rescaling terms does not effect the statement of the lemma. For example, if all the weights are doubled, then the costs are doubled but the distances are halved. Moreover, we may assume that $\lambda' \leq radius(\mathcal{L})$, since otherwise, choosing $r = radius(\mathcal{L})$ implies that $\partial(L_r) = \emptyset$ and the claim holds trivially.

Let $r_i = \|v_i\|$, and assume that the vertices are ordered so that $r_1 \leq r_2 \leq \cdots \leq r_n$. We may now assume without loss of generality that each edge in the graph has minimal length. That is, an edge from vertex i to vertex j has length $|r_i - r_j|$. The reason we may make this assumption is that it only increases the weights of edges, making our lemma strictly more difficult to prove. (Recall that the weight of an edge is the reciprocal of its length.)

Let $B_i = L_{r_i}$. Our proof will make critical use of a quantity μ_i, which is defined to be

$$\mu_i \;=\; \tau + \mathrm{vol}\left(E(B_i)\right) + \sum_{(v_j,v_k)\in E : j \leq i < k} \frac{r_i - r_j}{r_k - r_j}.$$

That is, μ_i sums the edges inside B_i, proportionally counting edges that are split by the boundary of the ball. The two properties of μ_i that we exploit are

$$\mu_{i+1} = \mu_i + \mathrm{cost}\left(\partial\left(B_i\right)\right)(r_{i+1} - r_i), \tag{7}$$

and

$$\tau + \mathrm{vol}\left(E(B_i)\right) \leq \mu_i \leq \tau + \mathrm{vol}\left(B_i\right). \tag{8}$$

The equality (7) follows from the definition by a straight-forward calculation, as

$$\mu_i \;=\; \tau + \mathrm{vol}\left(E(B_{i+1})\right) - \mathrm{vol}\left(\{(v_j, v_{i+1}) \in E \mid j \leq i\}\right) + \sum_{(v_j,v_k)\in E : j \leq i < k} \frac{r_i - r_j}{r_k - r_j}$$

and

$$\mathrm{cost}\left(\partial\left(B_i\right)\right)(r_{i+1} - r_i) = \sum_{(v_j,v_k)\in E : j \leq i < k} \frac{r_{i+1} - r_i}{r_k - r_j} \; .$$

Choose a and b so that $r_{a-1} \leq \lambda < r_a$ and $r_b < \lambda' \leq r_{b+1}$. Let $\nu = \lambda' - \lambda$. We first consider the trivial case in which $b < a$. (That is, $b = a - 1$.) In that case, there is no vertex with norm between λ and λ'. Thus every edge crossing $L_{(\lambda+\lambda')/2}$ has length at least ν, and therefore cost at most $1/\nu$. Therefore, by setting $r = (\lambda + \lambda')/2$, we obtain

$$\mathrm{cost}\left(\partial\left(L_r\right)\right) \leq \mathrm{vol}\left(\partial\left(L_r\right)\right)\frac{1}{\nu} \leq \mathrm{vol}\left(L_r\right)\frac{1}{\nu} \; ,$$

establishing the lemma in this case.

We now define

$$\eta = \log_2\left(\frac{m + \tau}{\mathrm{vol}\left(E(B_{a-1})\right) + \tau}\right) \; .$$

Note that $B_{a-1} = L_\lambda$, by the choice of a. A similarly trivial case is when $[a, b]$ is non-empty, and, in addition, there exists an $i \in [a-1, b]$ such that

$$r_{i+1} - r_i \geq \frac{\nu}{\eta} \; .$$

In this case, every edge in $\partial(B_i)$ has cost at most η/ν, and by choosing r to be $\max\{r_i, \lambda\}$, we satisfy

$$\text{cost}\left(\partial\left(L_r\right)\right) \;\leq \text{vol}\left(\partial\left(L_r\right)\right)\frac{\eta}{\nu} \leq \text{vol}\left(L_r\right)\frac{\eta}{\nu}\;,$$

hence, the lemma is established in this case.

In the remaining case that the set $[a, b]$ is non-empty and for all $i \in [a-1, b]$,

$$r_{i+1} - r_i < \frac{\nu}{\eta}\;, \tag{9}$$

we will prove that there exists an $i \in [a-1, b]$ such that

$$\text{cost}\left(\partial\left(B_i\right)\right) \leq \frac{\mu_i \eta}{\nu}\;,$$

hence, by choosing $r = \max\{r_i, \lambda\}$, the lemma is established due to (8).

Assume by way of contradiction that

$$\text{cost}\left(\partial\left(B_i\right)\right) > \mu_i \eta/\nu$$

for all $i \in [a-1, b]$. It follows, by (7), that

$$\mu_{i+1} > \mu_i + \mu_i(r_{i+1} - r_i)\eta/\nu$$

for all $i \in [a-1, b]$, which implies

$$\begin{aligned}
\mu_{b+1} &> \mu_{a-1}\prod_{i=a-1}^{b}\left(1 + (r_{i+1} - r_i)\eta/\nu\right) \\
&\geq \mu_{a-1}\prod_{i=a-1}^{b} 2^{((r_{i+1}-r_i)\eta/\nu)}\;, \text{ by (9) and since } 1 + x \geq 2^x \text{ for every } 0 \leq x \leq 1 \\
&= \mu_{a-1} \cdot 2^{(r_{b+1} - r_{a-1})\eta/\nu} \\
&\geq \mu_{a-1} \cdot ((m+\tau)/(\text{vol}\left(E(B_{a-1})\right) + \tau)) \\
&\geq m + \tau\;, \text{ by (8),}
\end{aligned}$$

which is a contradiction. □

An analysis of the following standard ball growing algorithm follows immediately by applying Lemma 4.2 to the concentric system $\{B(r, x)\}$.

$r = \texttt{BallCut}(G, x_0, \rho, \delta)$
1. Set $r = \delta\rho$.
2. While $\text{cost}\left(\partial\left(B(r, x_0)\right)\right) > \frac{\text{vol}(B(r,x_0))+1}{(1-2\,\delta)\rho}\log_2(m+1)$,   a. Find the vertex $v \notin B(r, x_0)$ that minimizes $\text{dist}(x_0, v)$ and set $r = \text{dist}(x_0, v)$.

Corollary 4.3 (Weighted Ball Cutting)**.** *Let $G = (V, E, w)$ be a connected weighted graph, let $x \in V$, $\rho = \mathrm{rad}_G(x)$, $r = \mathtt{BallCut}(G, x_0, \rho, 1/3)$, and $V_0 = B(r, x)$. Then $\rho/3 \leq r < 2\rho/3$ and*

$$\mathit{cost}(\partial(V_0)) \leq \frac{3(\mathrm{vol}(V_0)+1)\log_2(|E|+1)}{\rho}.$$

We now examine the concentric system that enables us to construct $V_1, \ldots, V_k$ in Lemma 3.2.

Definition 4.4 (Ideals and Cones)**.** *For any weighted graph $G = (V, E, w)$ and $S \subseteq V$, the set of* forward edges induced by S *is*

$$F(S) = \{(u \to v) : (u, v) \in E, \mathit{dist}(u, S) + d(u, v) = \mathit{dist}(v, S)\}.$$

For a vertex $v \in V$, the ideal of v induced by S, *denoted $I_S(v)$, is the set of vertices reachable from v by directed edges in $F(S)$, including v itself.*

For a vertex $v \in V$, the cone of width l around v induced by S, *denoted $C_S(l, v)$, is the set of vertices in V that can be reached from v by a path, the sum of the lengths of whose edges e that do not belong to $F(S)$ is at most l. Clearly, $C_S(0, v) = I_S(v)$ for all $v \in V$.*

That is, $I_S(v)$ is the set of vertices that have shortest paths to S that intersect v. Also, $u \in C_S(l, v)$ if there exist $a_0, \ldots, a_{k-1}$ and $b_1, \ldots, b_k$ such that $a_0 = v$, $b_k = u$, $b_{i+1} \in I_S(a_i)$, $(b_i, a_i) \in E$, and

$$\sum_i d(b_i, a_i) \leq l.$$

We now establish that these cones form concentric systems.

Proposition 4.5 (Cones are concentric)**.** *Let $G = (V, E, w)$ be a weighted graph and let $S \subseteq V$. Then for all $v \in V$, $\{C_S(l, v)\}_l$ is a concentric system in G.*

Proof. Clearly, $C_S(l, v) \subseteq C_S(l', v)$ if $l < l'$. Moreover, suppose $u \in C_S(l, v)$ and $(u, w) \in E$. Then if $(u \to w) \in F$, then $w \in C_S(l, v)$ as well. Otherwise, the path witnessing that $u \in C_S(l, v)$ followed by the edge (u, w) to w is a witness that $w \in C_S(l + d(u, w), v)$. □

$r = \mathtt{ConeCut}(G, v, \lambda, \lambda', S)$

1. Set $r = \lambda$ and if $\mathrm{vol}(E(C_S(\lambda, v))) = 0$,
   Set $\mu = (\mathrm{vol}(C_S(r, v)) + 1)\log_2(m+1)$
   otherwise,
   Set $\mu = \mathrm{vol}(C_S(r, v))\log_2(m/\mathrm{vol}(E(C_S(\lambda, v))))$
2. While $\mathrm{cost}(\partial(C_S(r, v))) > \mu/(\lambda' - \lambda)$,
   a.Find the vertex $w \notin C_S(r, v)$ minimizing $\mathrm{dist}(w, C_S(r, v))$ and set $r = r + \mathrm{dist}(w, C_S(r, v))$.

Corollary 4.6 (Cone Cutting)**.** *Let $G = (V, E, w)$ be a connected weighted graph, let v be a vertex in V and let $S \subseteq V$. Then for any two reals $0 \leq \lambda < \lambda'$, $\mathtt{ConeCut}(G, v, \lambda, \lambda', S)$ returns a real $r \in [\lambda, \lambda')$ such that*

$$\mathit{cost}(\partial(C_S(r, v))) \leq \frac{\mathrm{vol}(C_S(r, v)) + \tau}{\lambda' - \lambda}\max\left[1, \log_2\frac{m + \tau}{\mathrm{vol}(E(C_S(\lambda, v))) + \tau}\right],$$

where $m = |E|$, *and*

$$\tau = \begin{cases} 1, & \textit{if } \mathrm{vol}\,(E(C_S(\lambda, v))) = 0 \\ 0, & \textit{otherwise.} \end{cases}$$

We will use two other properties of the cones $C_S(l, v)$: that we can bound their radius (Proposition 4.7), and that their removal does not increase the radius of the resulting graph (Proposition 4.9).

Proposition 4.7 (Radius of Cones)**.** *Consider a connected weighted graph* $G = (V, E, w)$*, a vertex subset* $S \subseteq V$ *and let* $\psi = \max_{v \in V} dist(v, S)$*. Then, for every* $x \in S$*,*

$$\mathrm{rad}_{C_S(l,x)}(x) \leq \psi + 2l.$$

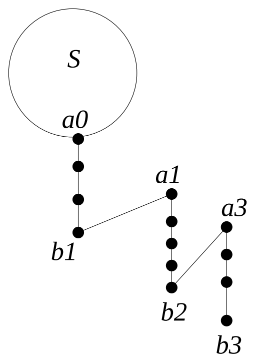

Figure 2: An illustration of a cone.

Proof. Let u be a vertex in $C_S(l, x)$, and let $a_0, \ldots, a_{k-1}$ and $b_1, \ldots, b_k$ be vertices such that $a_0 = x$, $b_k = u$, $b_{i+1} \in I_S(a_i)$, $(b_i, a_i) \in E$, and

$$\sum_i d(b_i, a_i) \leq l.$$

These vertices provide a path connecting x to u inside $C_S(l, x)$ of length at most

$$\sum_i d(b_i, a_i) + \sum_i \mathrm{dist}(a_i, b_{i+1}).$$

As the first term is at most l, we just need to bound the second term by $\psi + l$. To do this, consider the distance of each of these vertices from S. We have the relations

$$\mathrm{dist}(b_{i+1}, S) = \mathrm{dist}(a_i, S) + \mathrm{dist}(a_i, b_{i+1}) \tag{10}$$
$$\mathrm{dist}(a_i, S) \geq \mathrm{dist}(b_i, S) - d(b_i, a_i), \tag{11}$$

which imply that

$$\psi \;\geq\; \mathrm{dist}(b_k, S) \geq \sum_i \mathrm{dist}(a_i, b_{i+1}) - d(b_i, a_i) \;\geq\; \left(\sum_i \mathrm{dist}(a_i, b_{i+1}) \right) - l\ ,$$

as desired. ∎

In our proof, we actually use Proposition 4.8 which is a slight extension of Proposition 4.7. Its proof is similar.

Recall that for a vertex x and a real r, $BS(r,x)$ consists of every vertex $u \in V - B(r,v)$ with a neighbor $w \in B(r,v)$ such that $\mathrm{dist}(v,u) = \mathrm{dist}(v,w) + d(u,w)$.

Proposition 4.8 (Radius of Cones, II)**.** *Consider a connected weighted graph $G = (V,E,w)$, a vertex $x_0 \in V$ and let $\rho = \mathrm{rad}_G(x_0)$. Consider a real $r_0 < \rho$ and let $V_0 = B(r_0, x_0)$, $V' = V - V_0$ and $S = \mathrm{BS}(r_0, x_0)$. Consider a vertex $x_1 \in S$ and let $\psi = \rho - dist_G(x_0, x_1)$. Then the cones $C_S(l, x_1)$ in the graph $G(V')$ satisfy*

$$\mathrm{rad}_{C_S(l,x_1)}(x_1) \leq \psi + 2l.$$

The only difference in the proof is that in (10) and (11) one should replace S with x_1.

Proposition 4.9 (Deleting Cones)**.** *Consider a connected weighted graph $G = (V,E,w)$, a vertex subset $S \subseteq V$, a vertex $x \in S$ and a real $l \geq 0$ and let $V' = V - C_S(l,x)$, $S' = S - C_S(l,x)$ and $\psi = \max_{v \in V} dist(v,S)$. Then*

$$\max_{v \in V'} dist_{V'}(v, S') \ \leq \ \psi \ .$$

Proof. Consider some $v \in V'$, and let P be a shortest path between S and v. Note that all the edges of P are forward edges, and so if P intersects $C_S(l,x)$, then $v \in C_S(l,x)$. So, the shortest path from v to S in V must lie entirely in V'. $\square$

The basic idea of `StarDecomp` is to first use `BallCut` to construct V_0 and then repeatedly apply `ConeCut` to construct $V_1, \ldots, V_k$.

$(\{V_0, \ldots, V_k, \boldsymbol{x}, \boldsymbol{y}\}) = \texttt{StarDecomp}(G = (V,E,w), x_0, \delta, \epsilon)$
1. Set $\rho = \mathrm{rad}_G(x_0)$; Set $r_0 = \texttt{BallCut}(G, x_0, \rho, \delta)$ and $V_0 = B(r_0, x_0)$;
2. Let $S = \mathrm{BS}(r_0, x_0)$;
3. Set $G' = (V', E', w') = G(V - V_0)$, the weighted graph induced by $V - V_0$;
4. Set $(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ConeDecomp}(G', S, \epsilon\rho/2)$;
5. For each $i \in [1:k]$, set y_i to be a vertex in V_0 such that $(x_i, y_i) \in E$ and y_i is on a shortest path from x_0 to x_i. Set $\boldsymbol{y} = (y_1, \ldots, y_k)$.
$(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ConeDecomp}(G, S, \Delta)$
1. Set $G_0 = G$, $S_0 = S$, and $k = 0$.
2. while S_k is not empty a. Set $k = k+1$; Set x_k to be a vertex of S_{k-1}; Set $r_k = \texttt{ConeCut}(G_{k-1}, x_k, 0, \Delta, S_{k-1})$ b. Set $V_k = C_{S_{k-1}}(r_k, x_k)$; Set $G_k = G(V - \cup_{i=1}^k V_k)$ and $S_k = S_{k-1} - V_k$.
3. Set $\boldsymbol{x} = (x_1, \ldots, x_k)$.

Proof of Lemma 3.2. Let $\rho = \mathrm{rad}_G(x_0)$. By setting $\delta = 1/3$, Corollary 4.3 guarantees $\rho/3 \leq r_0 \leq (2/3)\rho$. Applying $\Delta = \epsilon\rho/2$ and Propositions 4.7 and 4.9, we can bound for every i, $r_0 + d(x_i, y_i) +$

$r_i \le \rho + 2\Delta = \rho + \epsilon\rho$. Thus `StarDecomp`$(G, x_0, 1/3, \epsilon)$ returns a $(1/3, \epsilon)$-star-decomposition with center x_0.

To bound the cost of the star-decomposition that the algorithm produces, we use Corollaries 4.3 and 4.6.

$$\mathrm{cost}\left(\partial\left(V_0\right)\right) \le \frac{3\left(1+\mathrm{vol}\left(V_0\right)\right)\log_2(m+1)}{\rho}, \quad \text{and}$$
$$\mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \le \frac{2\left(1+\mathrm{vol}\left(V_j\right)\right)\log_2(m+1)}{\epsilon\rho}$$

for every $1 \le j \le k$, thus

$$\begin{aligned}\mathrm{cost}\left(\partial\left(V_0, \ldots, V_k\right)\right) &\le \sum_{j=0}^{k} \mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \\ &\le \frac{2\,\log_2(m+1)}{\epsilon\rho} \sum_{j=0}^{k} (\mathrm{vol}\left(V_j\right)+1) \\ &\le \frac{6\,m\log_2(m+1)}{\epsilon\rho}\,.\end{aligned}$$

To implement `StarDecomp` in $O(m + n\log n)$ time, we use a Fibonacci heap to implement steps (2) of `BallCut` and `ConeCut`. If the graph is unweighted, this can be replaced by a breadth-first search that requires $O(m)$ time. □

5 Improving the Stretch

In this section, we improve the average stretch of the spanning tree to $O\left(\log^2 n \log\log n\right)$ by introducing a procedure `ImpConeDecomp` which refines `ConeDecomp`. This new cone decomposition trades off the volume of the cone against the cost of edges on its boundary (similar to Seymour [20]). Our refined star decomposition algorithm `ImpStarDecomp` is identical to algorithm `StarDecomp`, except that it calls

$$(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ImpConeDecomp}(G', S, \epsilon\rho/2, t, \widehat{m})$$

at Step 4, where t is a positive integer parameter whose role that will be described soon.

$(\{V_1, \ldots, V_k, \boldsymbol{x}\}) = \texttt{ImpConeDecomp}(G, S, \Delta, t, \widehat{m})$

1. Set $G_0 = G$, $S_0 = S$ and $j = 0$;
2. while S_j is not empty
   - a. Set $j = j + 1$, set x_j to be a vertex of S_{j-1}, and set $p = t - 1$;
   - b. while $p > 0$
     - i. $r_j = \texttt{ConeCut}(G_{j-1}, x_j, \frac{(t-p-1)\Delta}{t}, \frac{(t-p)\Delta}{t}, S_{j-1})$;
     - ii. **if** vol $\left(E(C_{S_{j-1}}(r_j, x_j))\right) \le \frac{m}{2^{\log^{p/t}\widehat{m}}}$ **then exit** the loop; **else** $p = p - 1$;
   - c. Set $V_j = C_{S_{j-1}}(r_j, x_j)$ and set $G_j = G(V - \cup_{i=1}^{j} V_i)$ and $S_j = S_{j-1} - V_j$;
3. Set $\boldsymbol{x} = (x_1, \ldots, x_k)$;

Lemma 5.1 (Improved Low-Cost Star Decomposition)**.** *Let G, x_0 and ϵ be as in Lemma 3.2, t be a positive integer control parameter, and $\rho = \mathrm{rad}_G(x_0)$. Then*

$$(\{V_0, \ldots, V_k\}, \boldsymbol{x}, \boldsymbol{y}) = \texttt{ImpStarDecomp}(G, x_0, 1/3, \epsilon, t, \widehat{m}) \ ,$$

in time $O(m + n \log n)$, returns a $(1/3, \epsilon)$-star-decomposition of G with center x_0 that satisfies

$$\mathit{cost}(\partial(V_0)) \ \leq \ \frac{6 \,\mathrm{vol}(V_0) \log_2(\widehat{m}+1)}{\rho} \ ,$$

and for every index $j \in \{1, 2, \ldots, k\}$ there exists $p = p(j) \in \{0, 1, \ldots, t-1\}$ such that

$$\mathit{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \leq t \cdot \frac{4 \,\mathrm{vol}(V_j) \log^{(p+1)/t}(\widehat{m}+1)}{\epsilon\rho} \ , \tag{12}$$

and unless $p = 0$,

$$\mathrm{vol}(E(V_j)) \ \leq \ \frac{m}{2^{\log^{p/t} \widehat{m}}} \ . \tag{13}$$

Proof. In what follows, we call $p(j)$ the *index-mapping* of the vertex set V_j. We begin our proof by observing that $0 \leq r_j < \epsilon\rho/2$ for every $1 \leq j \leq k$. We can then show that $\{V_0, \ldots, V_k\}$ is a $(1/3, \epsilon)$-star decomposition as we did in the proof of Lemma 3.2.

We now bound the cost of the decomposition. Clearly, the bound on $\mathrm{cost}(\partial(V_0))$ remains unchanged from that proved in Lemma 3.2 (because the algorithm that computes V_0 did not change), but here we bound $\mathrm{vol}(V_0) + 1$ by $2 \cdot \mathrm{vol}(V_0)$.

Below we will use $\Delta = \epsilon\rho/2$ as specified in the algorithm.

Fix an index $j \in \{1, 2, \ldots, k\}$, and let $p = p(j)$ be the final value of variable p in the loop above (that is, the value of p when the execution left the loop while constructing V_j). Observe that $p \in \{0, 1, \ldots, t-1\}$, and that unless the loop is aborted due to $p = 0$, we have $\mathrm{vol}(E(V_j)) \leq \frac{m}{2^{\log^{p/t} \widehat{m}}}$ and inequality (13) holds.

For inequality (12), we split the discussion to two cases. First, consider the case $p = t-1$. Then the inequality $\mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \leq (\mathrm{vol}(V_j) + 1) \log(\widehat{m}+1)(t/\Delta)$ follows directly from Corollary 4.6, and inequality (12) holds.

Second, consider the case $p < t-1$ and let r_j' be the value of the variable r_j at the beginning of the last iteration of the loop (before the last invocation of Algorithm `ConeCut`). In this case, observe that at the beginning of the last iteration, $\mathrm{vol}\left(E(C_{S_{j-1}}(r_j', x_j))\right) > \frac{m}{2^{\log^{(p+1)/t} \widehat{m}}}$ (as otherwise the loop would have been aborted in the previous iteration). By Corollary 4.6,

$$\mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \ \leq \ \frac{\mathrm{vol}(V_j)}{\Delta/t} \ \times \max\left[1, \log_2\left(\frac{m}{\mathrm{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right)}\right)\right],$$

where $V_j = C_{S_{j-1}}(r_j, x_j)$. Since

$$\frac{(t-p-2)\Delta}{t} \leq r_j' < \frac{(t-p-1)\Delta}{t},$$

it follows that

$$\mathrm{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right) \ \geq \ \mathrm{vol}\left(E(C_{S_{j-1}}(r_j', x_j))\right) \ > \ \frac{m}{2^{\log^{(p+1)/t} \widehat{m}}} \ .$$

Therefore

$$\log^{(p+1)/t} \widehat{m} \;\geq\; \max \left[1, \log_2 \left(\frac{m}{\mathrm{vol}\left(E\left(C_{S_{j-1}}\left(\frac{(t-p-1)\Delta}{t}, x_j\right)\right)\right)} \right) \right] ,$$

and

$$\mathrm{cost}\left(E\left(V_j, V - \cup_{i=0}^{j} V_i\right)\right) \;\leq\; \frac{\mathrm{vol}\left(V_j\right) \log^{(p+1)/t} \widehat{m}}{\Delta/t} \;=\; t \cdot \frac{2\,\mathrm{vol}\left(V_j\right) \log^{(p+1)/t} \widehat{m}}{\epsilon\rho} .$$

□

Our improved algorithm `ImpLowStretchTree`$(G, x_0, t, \widehat{m})$, is identical to `LowStretchTree` except that in Step 3 it calls `ImpStarDecomp`$(\widetilde{G}, x_0, 1/3, \beta, t, \widehat{m})$, and in Step 5 it calls `ImpLowStretchTree`$(G(V_i), x_i, t, \widehat{m})$. We set $t = \log\log n$ throughout the execution of the algorithm.

Theorem 5.2 (Lower–Stretch Spanning Tree)**.** *Let $G = (V, E, w)$ be a connected weighted graph and let x_0 be a vertex in V. Then*

$$T = \texttt{ImpLowStretchTree}(G, x_0, t, \widehat{m}) ,$$

in time $O(\widehat{m} \log \widehat{n} + \widehat{n} \log^2 \widehat{n})$, returns a spanning tree of G satisfying

$$\mathrm{rad}_T\left(x_0\right) \leq 2\sqrt{e} \cdot \mathrm{rad}_G\left(x\right)$$

and

$$\textit{ave-stretch}_T(E) = O\left(\log^2 \widehat{n} \log\log \widehat{n}\right) .$$

Proof. The bound on the radius of the tree remains unchanged from that proved in Theorem 3.4.

We begin by defining a system of notations for the recursive process, assigning for every graph $G = (V, E)$ input to some recursive invocation of Algorithm `ImpLowStretchTree`, a sequence $\sigma(G)$ of non-negative integers. This is done as follows. If G is the original graph input to the first invocation of the recursive algorithm, than $\sigma(G)$ is empty. Assuming that the halt condition of the recursion is not satisfied for G, the algorithm continues and some of the edges in E are contracted. Let $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ be the resulting graph. (Recall that we refer to $\widetilde{G}$ as the edge-contracted graph.) Let $\{\widetilde{V}_0, \widetilde{V}_1, \ldots, \widetilde{V}_k\}$ be the star decomposition of $\widetilde{G}$. Let $V_j \subseteq V$ be the preimage under edge contraction of $\widetilde{V}_j \subseteq \widetilde{V}$ for every $0 \leq j \leq k$. The graph $G(V_j)$ is assigned the sequence $\sigma(G(V_j)) = \sigma(G) \cdot j$. Note that $|\sigma(G)| = h$ implies that the graph G is input to the recursive algorithm on recursion level h. We warn the reader that the edge-contracted graph obtained from a graph assigned with the sequence σ may have fewer edges than the edge-contracted graph obtained from the graph assigned with the sequence $\sigma \cdot j$, because the latter may contain edges that were contracted out in the former.

We say that the edge e is *present* at recursion level h if e is an edge in $\widetilde{G}$ which is the edge-contracted graph obtained from some graph G with $|\sigma(G)| = h$ (that is, it was not contracted out). An edge e *appears* at the first level h at which it is present, and it *disappears* at the level at which it is present and its endpoints are separated by the star decomposition, or, if this never happens, then at the level of the leaf component of the recursion tree in which e appears. If an edge appears at recursion level h, then a path connecting its endpoints was contracted on every recursion level smaller than h, and no such path will be contracted on any recursion level greater than h. Moreover, an edge is never present at a level after it disappears. We define $h(e)$ and $h'(e)$ to be the recursion levels at which the edge e appears and disappears, respectively.

For every edge e and every recursion level i at which it is present, we let $U(e,i)$ denote the set of vertices $\widetilde{V}$ of the edge-contracted graph containing its endpoints. If $h(e) \le i < h'(e)$, then we let $W(e,i)$ denote the set of vertices $\widetilde{V}_j$ output by `ImpStarDecomposition` that contains the endpoints of e.

Recall that $p(j)$ denotes the index-mapping of the vertex set V_j in the star decomposition. For each index $i \in \{0,1,\ldots,t-1\}$, let $I_i = \{j \in \{1,2,\ldots,k\} \mid p(j) = i\}$. For a vertex subset $U \subseteq V$, let $\mathrm{AS}(U)$ denote the average stretch that the algorithm guarantees for the edges of $E(U)$. Let $\mathrm{TS}(U) = \mathrm{AS}(U) \cdot |E(U)|$. ($TS$ stands for the "total stretch".) Then by Lemma 5.1, the following recursive formula applies.

$$\begin{aligned}
\mathrm{TS}(V) \;\le\; & \left(\sum_{j=0}^{k} \mathrm{TS}(\widetilde{V}_j)\right) \\
& + \; 4\sqrt{e} \times \left(6\,\log(\widehat{m}+1)\cdot \mathrm{vol}\left(\widetilde{V}_0\right) \;+\; 4\,\frac{t}{\beta}\sum_{p=0}^{t-1} \log^{(p+1)/t}(\widehat{m}+1) \sum_{j\in I_p} \mathrm{vol}\left(\widetilde{V}_j\right)\right) \\
& + \; \left(\sum_{e\in E-\widetilde{E}} \mathrm{stretch}_T(e)\right) \\
= \; & \left(\sum_{j=0}^{k} \mathrm{TS}(V_j)\right) \\
& + \; 4\sqrt{e} \times \left(6\,\log(\widehat{m}+1)\cdot \mathrm{vol}\left(\widetilde{V}_0\right) \;+\; 4\,\frac{t}{\beta}\sum_{p=0}^{t-1} \log^{(p+1)/t}(\widehat{m}+1) \sum_{j\in I_p} \mathrm{vol}\left(\widetilde{V}_j\right)\right) \;, \qquad (14)
\end{aligned}$$

where we recall $\beta = \epsilon = (2\log_{4/3}(n+32))^{-1}$.

For every edge e and for every $h(e) \le i < h'(e)$, let $\pi_i(e)$ denote the index-mapping of the component $W(e,i)$ in the invocation of Algorithm `ImpConeDecomp` on recursion level i. For every index $p \in \{0,\ldots,t-1\}$, define the variable $l_p(e)$ as follows

$$l_p(e) \;=\; \left|\{h(e) \le i < h'(e) \mid \pi_i(e) = p\}\right| \;.$$

For a fixed edge e and an index $p \in \{0,1,\ldots,t-1\}$, every $h(e) \le i < h'(e)$ such that $\pi_i(e) = p$ reflects a contribution of $O(t/\beta)\cdot \log^{(p+1)/t}(\widehat{m}+1)$ to the right term in (14). Summing p over $\{0,1,\ldots,t-1\}$, we obtain

$$\sum_{p=0}^{t-1} O(t/\beta) l_p(e) \log^{(p+1)/t}(\widehat{m}+1).$$

In a few moments, we will prove that

$$\sum_{e}\sum_{p=0}^{t-1} l_p(e) \log^{p/t}(\widehat{m}+1) \le O(\widehat{m}\log_2 \widehat{m}), \qquad (15)$$

which implies that the sum of the contributions of all edges e in levels $h(e) \le i < h'(e)$ to the right term in (14) is

$$O\left(\frac{t}{\beta} \cdot \widehat{m} \log^{1+1/t} \widehat{m}\right) = O\left(t \cdot \widehat{m} \log^{2+1/t} \widehat{m}\right)$$

As $\mathrm{vol}\,(V_j)$ counts the internal edges of V_j as well as its boundary edges, we must also account for the contribution of each edge e at level $h'(e)$. At this level, it will be counted twice—once in each component containing one of its endpoints. Thus, at this stage, it contributes a factor of at most $O((t/\beta) \cdot \log \widehat{m})$ to the sum $\mathrm{TS}(V)$. Therefore all edges $e \in E$ contribute an additional factor of $O(t \cdot \widehat{m} \log^2 \widehat{m})$. Summing over all the edges, we find that all the contributions to the right term in (14) sum to at most

$$O\left(t \cdot \widehat{m} \log^{2+1/t} \widehat{m}\right).$$

Also, every $h(e) \le i < h'(e)$ such that the edge e belongs to the central component $\widetilde{V}_0$ of the star decomposition, reflects a contribution of $O(\log \widehat{m})$ to the left term in (14). Since there are at most $O(\log \widehat{m})$ such is, it follows that the contribution of the left term in (14) to $\mathrm{TS}(V)$ sums up to an additive term of $O(\log^2 \widehat{m})$ for every single edge, and $(\widehat{m} \log^2 \widehat{m})$ for all edges.

It follows that $\mathrm{TS}(V) = O(t \cdot \widehat{m} \log^{2+1/t} \widehat{m})$. This is optimized by setting $t = \log\log \widehat{m}$, obtaining the desired upper bound of $O(\log^2 \widehat{n} \cdot \log\log \widehat{n})$ on the average stretch $\mathrm{AS}(V)$ guaranteed by Algorithm `ImpLowStretchTree`.

We now return to the proof of (15). We first note that $l_0(e)$ is at most $O(\log \widehat{m})$ for every edge e. We then observe that for each index $p > 0$ and each $h(e) \le i < h'(e)$ such that $\pi_i(e) = p$, $\mathrm{vol}\,(E(U(e,i)))\,/\mathrm{vol}\,(E(W(e,i)))$ is at least $2^{\log^{p/t} \widehat{m}}$ (by Lemma 5.1, (13)).

For $h(e) \le i < h'(e)$, let $g_i(e) = \mathrm{vol}\,(E(U(e,i+1)))\,/\mathrm{vol}\,(E(W(e,i)))$ be the ratio between the volume of the set $W(e,i)$ after and before the re-appearance of contracted edges. We then have

$$\prod_{1 \le p \le t-1} \left(2^{\log^{p/t} \widehat{m}}\right)^{l_p(e)} \ \le\ \widehat{m} \prod_{h(e) \le i < h'(e)} g_i(e)\ ,$$

hence $\sum_{p=1}^{t-1} l_p(e) \log^{p/t} \widehat{m} \le \log \widehat{m} + \sum_{h(e) \le i < h'(e)} \log g_i(e)$. We will next prove that

$$\sum_{e} \sum_{h(e) \le i < h'(e)} \log g_i(e) \le \widehat{m} \log \widehat{m}, \tag{16}$$

which implies (15).

Let E_i denote the set of edges present at recursion level i. For every edge $e \in E_i$ such that $i < h'(e)$, we have

$$\sum_{e' \in E(W(e,i))} g_i(e') = g_i(e) \mathrm{vol}\,(E(W(e,i))) = \mathrm{vol}\,(E(U(e,i+1)))\,,$$

and so $\sum_{e \in E_i : i < h'(e)} g_i(e) = \mathrm{vol}\,(E_{i+1})$. As each edge is present in at most $O(\log \widehat{m})$ recursion depths, $\sum_i \mathrm{vol}\,(E_i) \le \widehat{m} \log \widehat{m}$, which proves (16). ∎

6 Conclusion

At the beginning of the paper, we pointed out that the definition of stretch used in this paper differs slightly from that used by Alon, Karp, Peleg and West [1]. If one is willing to accept a longer running time, then this problem is easily remedied as shown in Subsection 1.1.

If one is willing to accept a bound of $O(\log^3 n)$ on the stretch, then one can extend our analysis to show that the natural randomized variant `LowStretchTree`, in which one chooses the radii of the balls and cones at random, works.

A natural open question is whether one can improve the stretch bound from $O\left(\log^2 n \log\log n\right)$ to $O(\log n)$. Algorithmically, it is also desirable to improve the running time of the algorithm to $O(m \log n)$. If we can successfully achieve both improvements, then we can use the Spielman-Teng solver to solve planar diagonally dominant linear systems in $O(n \log n \log(1/\epsilon))$ time.

As the average stretch[3] of any spanning tree in a weighted connected graph is $\Omega(1)$, our low-stretch tree algorithm also provides an $O\left(\log^2 n \log\log n\right)$-approximation to the optimization problem of finding the spanning tree with the lowest average stretch. It remains open (a) whether our algorithm has a better approximation ratio and (b) whether one can in polynomial time find a spanning tree with better approximation ratio, e.g., $O(\log n)$ or even $O(1)$.

7 Acknowledgments

We are indebted to David Peleg, who was offered co-authorship on this paper.

References

[1] Noga Alon, Richard M. Karp, David Peleg, and Douglas West. A graph-theoretic game and its application to the k-server problem. *SIAM Journal on Computing*, 24(1):78–100, February 1995.

[2] Yonatan Aumann and Yuval Rabani. An o(log k) approximate min-cut max-flow theorem and approximation algorithm. *SIAM J. Comput.*, 27(1):291–301, 1998.

[3] B. Awerbuch and Y. Azar. Buy-at-bulk network design. In *Proceedings of the 38th IEEE FOCS*, pages 542–547, 1997.

[4] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985.

[5] Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of the 37th IEEE FOCS*, pages 184–193, 1996.

[6] Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the 30th ACM STOC*, pages 161–168, 1998.

[7] Yair Bartal. Personal Communication, 2005.

[3] In the context of the optimization problem of finding a spanning tree with the lowest average stretch, the stretch is defined as in [1].

[8] Erik Boman and Bruce Hendrickson. On spanning tree preconditioners. Manuscript, Sandia National Lab., 2001.

[9] Erik Boman, Bruce Hendrickson, and Stephen Vavasis. Solving elliptic finite element systems in near-linear time with support preconditioners. Manuscript, Sandia National Lab. and Cornell, http://arXiv.org/abs/cs/0407022.

[10] P. Crescenzi and V. Kann. A compendium of NP-hard problems. Available online at http://www.nada.kth.se/theory/compendium, 1998.

[11] Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. In *Proceedings of the 35th ACM STOC*, pages 448–455, 2003.

[12] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to Theory of NP-Completeness*. 1979.

[13] N. Garg, G. Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group steiner tree problem. In *Proceedings of the 9th ACM-SIAM SODA*, pages 253–259, 1998.

[14] T.C. Hu. Optimum communication spanning trees. *SIAM Journal on Computing*, pages 188–195, 1974.

[15] G. Konjevod and R. Ravi. An approximation algorithm for the covering Steiner problem. In *Proceedings of the 11th ACM-SIAM SODA*, pages 338–344, 2000.

[16] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

[17] Nathan Linial, Eran London, and Yuri Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15:215–245, 1995.

[18] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. 2000. SIAM Philadelphia PA.

[19] D. Peleg and E. Reshef. Deterministic polylogarithmic approximation for minimum communication spanning trees. In *Proc. 25th International Colloq. on Automata, Languages and Programming*, pages 670–681, 1998.

[20] P. D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.

[21] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the 36th ACM STOC*, pages 81–90, 2004.