Distributed Algorithms for Low Stretch Spanning Trees

Ruben Becker
Gran Sasso Science Institute, L’Aquila, Italy
ruben.becker@gssi.it

Yuval Emek
Technion – Israel Institute of Technology, Haifa, Israel
yemek@technion.ac.il

Mohsen Ghaffari
ETH Zürich, Switzerland
ghaffari@inf.ethz.ch

Christoph Lenzen
MPI for Informatics, Saarland Informatics Campus, Saarbrücken, Germany
clenzen@mpi-inf.mpg.de

Abstract
Given an undirected graph with integer edge lengths, we study the problem of approximating the distances in the graph by a spanning tree based on the notion of stretch. Our main contribution is a distributed algorithm in the CONGEST model of computation that constructs a random spanning tree with the guarantee that the expected stretch of every edge is $O(\log^3 n)$, where $n$ is the number of nodes in the graph. If the graph is unweighted, then this algorithm can be implemented to run in $O(D)$ rounds, where $D$ is the hop-diameter of the graph, thus being asymptotically optimal. In the weighted case, the run-time of our algorithm matches the currently best known bound for exact distance computations, i.e., $\tilde{O}(\min\{\sqrt{nD}, \sqrt{nD^{1/4}} + n^{3/5} + D\})$. We stress that this is the first distributed construction of spanning trees leading to poly-logarithmic expected stretch with non-trivial running time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Distributed algorithms

Keywords and phrases distributed graph algorithms, low-stretch spanning trees, CONGEST model, ball decomposition, star decomposition

Digital Object Identifier 10.4230/LIPIcs.DISC.2019.4

Funding Ruben Becker: This work has been partially supported by the Italian MIUR PRIN 2017 Project ALGADIMAR “Algorithms, Games, and Digital Markets”.
Yuval Emek: This work has been supported in part by an Israeli Science Foundation grant number 1016/17.

1 Introduction and Related Work

Trees are easy, general graphs are hard. This can be said as a first-order summary for a wide range of graph problems, especially in the area of approximation algorithms. Starting with the work of Alon et al. [3], there has been a beautiful line of developments that try to combat this issue and make general graphs (almost) as easy as trees, for several families of graph problems (including distances, cuts, and more) [5, 6, 8, 14, 7, 12, 29, 1, 13, 4, 26, 2]. In a very rough sense, these methods transform any general graph $G$ to a tree $T$ that approximately preserves some of the structural properties of $G$, thus opening the road for the following (generic) algorithmic approach: (1) transform the graph $G$ into a tree $T$; (2) solve the problem on $T$; and (3) project the solution in $T$ back to a solution in $G$. The quality of the obtained
solution depends on how well $T$ preserves the relevant structure of $G$. Such transformations have been a key driver in many of the algorithmic developments in the past two decades, in centralized approximation algorithms. Our focus in this paper is on distance-related graph problems and transformations of graphs into spanning trees that approximately preserve distances, based on the notion of stretch.

Spanning Trees and Stretch.

Consider some graph $G = (V, E, \ell)$, where $\ell : E \rightarrow \mathbb{R}_{>0}$ is an edge length function.$^1$ A tree $T = (V_T, E_T, \ell_T)$ is said to be a spanning tree of $G$ if (i) $V_T = V$; (ii) $E_T \subseteq E$; and (iii) $\ell_T$ is the restriction of $\ell$ to the edges in $E_T$. By definition, the distances in $T$ are at least as large as those in $G$, namely, $d_T(u, v) \geq d_G(u, v)$ for every two vertices $u, v \in V$, where the distances $d_T(\cdot, \cdot)$ and $d_G(\cdot, \cdot)$ are defined with respect to the edge length functions $\ell_T$ and $\ell$, respectively. The notion of stretch provides a bound in the converse direction: given an edge $e = (u, v) \in G$, the stretch of $e$ in $T$ is defined to be $\text{str}_T(e) = d_T(u, v)/\ell(e)$.

Ideally, we would have wanted to construct a spanning tree $T$ that admits a small stretch for every edge $e \in E$, but this is clearly hopeless, e.g., if the graph has high girth. Instead, we wish to construct a random spanning tree $T$ so that the expected stretch of every edge $e \in E$ satisfies $E_T[\text{str}_T(e)] \leq \alpha$ for some small $\alpha$ (cf. [5]). This notion is closely related (and essentially equivalent) to constructing a (deterministic) spanning tree with average stretch $\alpha$ [3]. More precisely, the per-edge expected stretch guarantee trivially leads to a bound of $O(m \cdot \alpha)$ on the expected total stretch. Using standard techniques, this leads to a distributed construction of a spanning tree whose total stretch is $O(m \cdot \alpha)$ with high probability. Therefore, the per-edge expected stretch guarantee is sufficient for the method to be functional as a subroutine.

There is an extensive literature on constructing (random) spanning trees for general graphs with low expected stretch, starting with the pioneering work of Alon et al. [3] which paved the way for the developments in [3, 12, 1, 2]. The state-of-the-art in this line of work is Abraham and Neiman’s construction of random spanning trees with expected stretch $O(\log n \log \log n)$ [2]. In a related line of work [5, 6, 8, 14, 7], it is only the distances in $G$ that matter, essentially ignoring the graph topology so that the tree $T$ can include vertices and edges that are not part of $G$, subject to the constraint that the distances in $T$ are lower-bounded by the corresponding distances in $G$. The common practice here is to think of $T$ as a dominating tree metric into which the metric space defined by the distances in $G$ can be embedded without contracting the distances. The construction of Fakcharoenphol et al. [14] (often referred to as FRT) provides an asymptotically optimal $O(\log n)$ upper bound on the expected stretch in this setting (see also [7]).

Following the influential work of Bartal [5], random dominating tree metrics with low expected stretch have contributed greatly to the design of approximation and online algorithms, for problems in which the topology of the underlying graph $G$ is abstracted away. More recently though there are new applications that require that $T$ is a subgraph of $G$ including fast solvers for symmetric diagonally dominant (SDD) linear systems [23, 21, 10, 11] and approximate max-flow and minimum cut algorithms [26, 9, 25, 31, 20], these applications point the flashlight back in the direction of low stretch spanning trees.

---

$^1$ Unless stated otherwise, all graphs in this paper are assumed to be undirected and finite.
Distributed Constructions.

Low stretch spanning trees, as well as low stretch dominating tree metrics, have also been studied and used in distributed graph algorithms. For instance, low stretch spanning trees were a key component in the max-flow algorithm of Ghaffari et al. [18] which gave the first sublinear-time distributed max-flow approximation. However, currently known distributed constructions of these trees have a suboptimal running time and/or suboptimal stretch.

Khan et al. [22] were the first to investigate distributed algorithms for random dominating tree metrics with low expected stretch, designing a distributed implementation for the FRT construction that works in $\tilde{O}(\text{SPD})$ rounds of the CONGEST model [28], where SPD denotes the shortest-path-diameter of the network, which can be as large as $\Theta(n)$, even in graphs with very small hop diameter $D$. Similar to many other graph problems, a lower bound of $\tilde{O}(D + \sqrt{n})$ rounds follows from the work of Das Sarma et al. [30] which set $\tilde{O}(D + n^{0.5})$ as the target desired round complexity. Ghaffari and Lenzen [19] provided a faster distributed construction that runs in $\tilde{O}(D + n^{0.5+\varepsilon})$ rounds and builds a random dominating tree metric with expected stretch $O(\log n/\varepsilon)$. Friedichs and Lenzen [16] further advanced this line of work by developing a distributed algorithm that outputs a random dominating tree metric with expected stretch $O(\log n)$ in $\tilde{O}((D + n^{0.5}) \cdot n^{o(1)})$ rounds.

The aforementioned distributed constructions suffer from two drawbacks: (1) the round complexity is still somewhat far from the $\tilde{O}(D + \sqrt{n})$ target; and (2) the constructed trees do not have any guarantees regarding the topology of the underlying network and in particular, they are not spanning trees of this network, thus complicating their usage in distributed settings. For the more desirable, but also more stringent, notion of low stretch spanning trees, where the tree $T$ has to be a subgraph of the $G$, the only known distributed construction is due to Ghaffari et al [18]. It gives a tree with a much worse stretch of $2^{O(\sqrt{\log n \log \log n})}$ and it runs in $\tilde{O}((D + n^{0.5}) \cdot 2\sqrt{\log n \log \log n})$ rounds, both of which are a factor of $2^{O(\sqrt{\log n \log \log n})}$ away from ideal. Indeed, this suboptimality (in both stretch and running time) is one of the two bottlenecks in turning the round complexity of the max-flow approximation to the optimal bound of $\tilde{O}(D + \sqrt{n})$.

1.1 Our Contribution

We provide a new distributed algorithm, operating in the CONGEST model, that improves the state-of-the-art for low stretch spanning trees. For unweighted graphs (i.e., graphs with unit edge lengths), our algorithm runs in asymptotically optimal $O(D)$ rounds and builds a random spanning tree with expected stretch $O(\log^3 n)$. In terms of both round complexity and stretch, this improves considerably on the algorithm of Ghaffari et al [18] which has round complexity $\tilde{O}((D + n^{0.5}) \cdot 2\sqrt{\log n \log \log n})$ and stretch $2^{O(\sqrt{\log n \log \log n})}$. Our algorithm also extends to weighted graphs with the same expected stretch guarantee, in which case the round complexity grows to $\tilde{O}(\min\{\sqrt{nD}, \sqrt{nD^{1/4}} + n^{3/5} + D\})$, i.e., boiling down to the best known complexity for exact single-source shortest path computation, which is due to Forster and Nanongkai [15]. We stress that this is the first distributed construction of spanning trees leading to poly-logarithmic expected stretch with non-trivial round complexity.

Theorem 1. A spanning tree of expected stretch $O(\log^3 n)$ for each edge can be computed in $\tilde{O}(\min\{\sqrt{nD}, \sqrt{nD^{1/4}} + n^{3/5} + D\})$ rounds w.h.p.\(^2\) If the input graph is unweighted, then

\(^2\) We say that event $A$ occurs with high probability, abbreviated w.h.p., if $\Pr(A) \geq 1 - n^{-c}$ for a constant $c$ that can be made arbitrarily large.
More generally, our method can be seen as an efficient reduction of the task of computing a low-stretch spanning tree of expected stretch $O(\log^3 n)$ to single-source shortest path computations with a virtual super-source, which is formalized in Definition 2. Any improvements in distributed algorithms for this task will thus carry over to our construction.

We note that, unfortunately, our approach cannot be used to reduce to approximate single-source shortest path computations, as the decomposition technique that we will use [27] crucially relies on the subtractive form of the triangle inequality, which fails even under small relative errors in distances.

### 1.2 Our Method In a Nutshell

The general approach taken in the current paper is very similar to the divide and conquer technique due to Elkin et al. [12]. That is, we apply a graph partitioning scheme called star decomposition (introduced formally in Section 2) that given a root or center node $x_0$, decomposes the graph into a center part $V_0$ and cone parts $V_1, \ldots, V_k$ centered at nodes $x_1, \ldots, x_k$, respectively, referred to as the cone anchors. This star decomposition has the following properties: (1) the radius of $V_i$ with respect to $x_i$, $0 \leq i \leq k$, is smaller than the radius $r$ of $G$ with respect to $x_0$ by a constant factor; and (2) each anchor node $x_i$, $i \in [k]$, is connected via a direct edge, referred to as a bridge edge, to some node $y_i \in V_0$ so that, for every cone, the distance between anchor node and center of the decomposition plus the radius of the cone is at most a factor of $1 + \varepsilon$ larger than the radius $r$ with respect to $x_0$.

The idea is then to apply such star decompositions recursively to each of the obtained parts $V_0, \ldots, V_k$, leading to spanning trees $T_0, \ldots, T_k$. The spanning tree $T$ that is returned by the algorithm is then constructed by connecting the trees $T_1, \ldots, T_k$ to the central tree $T_0$ using the bridge edges $(x_1, y_1), \ldots, (x_k, y_k)$. Clearly this approach leads to a spanning tree, however from the description so far, it is not clear why $T$ has small expected stretch.

For this, we need that the star decompositions that we construct have the additional small cut property: For each edge $e = (u, v) \in E$, the probability that $e$ is cut by the decomposition, i.e., that $u \in V_i$ and $v \in V_j$ for $i \neq j$, is at most $O(\log n \cdot \ell(e)/(\varepsilon r))$. Elkin et al. [12] use an intricate cone growing procedure in order to construct the parts $V_1, \ldots, V_k$ in a way that ensures the (deterministic counterpart of the) small cut property. It is not clear though how to implement the cone growing procedure efficiently in a distributed manner.

In this paper, we replace the cone growing process of [12] by a graph partitioning technique due to Miller et al. [27]. This technique has the desirable property that it can be implemented in the CONGEST model of computation in a straightforward way based on single source shortest path (SSSP) computations. Specifically, after constructing the center part $V_0$, we let every node $u$ that is “just outside” $V_0$ (we make this notion precise in Section 3) draw a value $\delta_u$ from an exponential distribution with mean $\beta = \Theta(\log n / \varepsilon r)$. We now conceptually start a ball growing process from all such nodes, where node $u$ joins the process at time $\delta_u$.

Miller et al. [27] have shown (for the unweighted case) that this leads to a decomposition that “cuts edges” with a probability sufficiently small for their needs (they were not concerned with star decompositions). We observe that when applied to the graph $H = G \setminus V_0$, this leads to a star decomposition that satisfies the desired small cut property, see Section 3.2. In Section 4, we furthermore show that this is sufficient for the resulting tree (after recursing on the parts of the decomposition and connecting the obtained trees using the bridge edges) to have expected stretch $O(\log^3 n)$ for every edge. Replacing the cone growing process with this decomposition technique also results in a conceptually much simpler algorithm.
for constructing spanning trees of small expected stretch in standard centralized models of computation. This can be of independent interest. Lastly, we remark that, also for the PRAM model, our technique yields a similar reduction of computing spanning trees of low expected stretch to exact SSSP with virtual super-source.

2 Preliminaries

Our algorithm runs on an undirected, connected input graph $G = (V, E, \ell)$ with positive integer edge lengths $\ell$ that are polynomially bounded, i.e., bounded by $n^c$ for some constant $c$. For graphs $H$, we denote by $d_H(u, v)$ the length of the shortest path between two nodes $u$ and $v$. If $H$ is the input graph $G$, we may omit $G$ and simply write $d(u, v)$ for $d_G(u, v)$.

For a node $u \in V$ and a radius $r > 0$, we call $B(u, r) := \{v \in V : d(u, v) \leq r\}$ the ball of radius $r$ around $u$, i.e., the set of nodes of distance at most $r$ from $u$. For any graph $G$ with node set $V$ and a node $v \in V$, we call $\text{rad}_v(G) = \max \{d_G(u, v) : v \in V\}$ the radius of $G$ with respect to $v$. For a subset of nodes $S \subseteq V$, we denote with $G[S]$ the subgraph of $G$ induced by $S$. By $W(H) = \max_{u, v \in V_H} \{d_H(u, v)\}$ we denote the weighted diameter of $H = (V_H, E_H, \ell_H)$ and by $D(H) = W(H')$ its hop diameter, where $H' = (V_H, E_H, 1)$, i.e., $H$ with all edges being assigned unit length; again, we omit $G$ from the notation in case $H = G$.

Model of Computation.

Our algorithm works in the standard CONGEST model of computation [28]. In this model, every node hosts a processor (of unlimited computational power) and is labeled by a unique $O(\log n)$-bit identifier. The computation proceeds in synchronous rounds, in each of which a node (1) performs local computations, (2) sends $O(\log n)$-bit messages to its neighbors, and (3) receives the messages that its neighbors sent. Initially, every node in the input graph $G = (V, E, \ell)$ knows its identifier and its incident edges together with their length. At the end of computation every node needs to know its part of the output. That is every node needs to know for its incident edges whether or not they belong to the output spanning tree.

Distributed SSSP Computation in Super-Source Graphs.

At its heart, our algorithm reduces the problem to a series of single source shortest path (SSSP) computations. Accordingly, we will need to compute SSSP in graphs $G_s$ that result from subgraphs of $G$ by adding a (virtual) super-source node $s \notin V$.

Definition 2 (Super-source graphs). Fix a subgraph $H = (V_H, E_H, \ell|_H)$ of $G$. Construct $G_s = (V_H \cup \{s\}, E_H \cup E_s, \ell|_{G_s})$ by choosing $E_s \subseteq V_H \times \{s\}$, picking $\ell|_{G_s}(e) \in \{1, \ldots, n^c\}$ for $e \in E_s$, and setting $\ell|_{G_s}(e) = \ell_e$ for all $e \in E_H$. We refer to $G_s$ as a super-source graph (of $G$) and to $s$ as its super-source. For distributed algorithms, we assume that each node $v \in V$ initially knows which of its incident edges in $G$ are in $V_H$, whether it is connected to $s$, and, if so, the length of edge $(s, v)$.

Although both distributed CONGEST algorithms for SSSP that we employ (for the unweighted and weighted case) assume to be run on the input graph, we observe that it is straightforward to generalize them to super-source graphs. Both considered algorithms output a tree $T$ that is a subgraph of $G_s$, where each node $v \in V_H$ learns its parent and the distance $d_T(v, s)$ from $v$ to $s$ in $T$, which is exactly the distance $d_G(v, s)$ from $v$ to $s$ in $G_s$.

Lemma 3 (folklore result). SSSP in super-source graphs can be solved in $O(W(G_s))$ rounds. In particular, if $G_s$ is unweighted (i.e., $\ell|_{G_s} = 1$), SSSP can be solved in $O(D(G_s))$ rounds.
Proof. For unweighted graphs, this is done by standard flooding to construct a BFS tree, where communication by \( s \) is simulated locally based on nodes knowing whether they are connected to \( s \) and by which length. For weighted graphs, one simulates the algorithm on the unweighted graph obtained by subdividing each edge \( e \) into \( \ell_e \) many length-1 edges. Termination is detected via the resulting spanning forest \( T \setminus \{s\} \) of \( G_s \setminus \{s\} \).

\[\text{Corollary 4} \text{ (of [15]). SSSP in super-source graphs can be solved in } \tilde{O}(\min\{\sqrt{nD}, \sqrt{nD}^{1/4} + n^{3/5} + D\}) \text{ rounds w.h.p.}\]

Comment. We comment that the algorithm of Forster and Nanongkai [15] can be directly extended to the case of super-source graphs.\(^3\) In short, the reason is as follows: there are only two differences between the case considered here and the one of [15]. (1) We cannot communicate on the virtual edges that connect \( s \) to \( V_H \), as there are no such physical edges. (2) We work on a subgraph of the base graph, whose hop diameter may be much larger than \( D \). Regarding the first point, we note that in the algorithm of [15], besides the initial coordination message from the source that can be delivered to all nodes in \( O(D) \) rounds, the source \( s \) never changes its state. Hence, it does not need to send any message to its neighbors in \( V_H \) or to receive a message from them. Regarding the second point, we note that the algorithm of [15] relies on the hop diameter \( D \) only for the purpose of global communication. In our setting, even though our computation is about a subgraph, we can still use the base graph to perform computation and in particular we can deliver any \( B \) messages to all nodes in \( O(D + B) \) rounds.

Exponential Distribution.

By \( \text{Exp}_\beta \) we denote the exponential distribution with mean \( 1/\beta \). Its density function is given by \( f_{\text{Exp}_\beta}(x) = \beta \exp(-\beta x) \cdot H(x) \), where \( H(\cdot) \) denotes the Heaviside step function and its cumulative density function by \( F_{\text{Exp}_\beta}(x) = (1 - \exp(-\beta x)) \cdot H(x) \).\(^4\) First, we observe that drawing from this distribution results in values of \( O(\log n/\beta) \) w.h.p.

\[\text{Lemma 5. For parameters } 0 < \varepsilon < 1, \beta > 0, \text{ and a sufficiently large constant } c > 0, \text{ let } t := c \log n/(4(1 + \varepsilon)\beta) \text{ and } X \sim \text{Exp}_\beta. \text{ Then } P[X \geq t] \in n^{-\Omega(c)}, i.e., X < t \text{ w.h.p.}\]

Proof. The proof is a simple calculation:

\[P[X \geq t] = \frac{\int_t^\infty e^{-\beta x} \, dx}{\int_0^\infty e^{-\beta x} \, dx} = \frac{e^{-\beta t} \int_0^\infty e^{-\beta x} \, dx}{\int_0^\infty e^{-\beta x} \, dx} \leq e^{-\Omega(c \log n)} = n^{-\Omega(c)}.\]

Intuitively, the next lemma (taken from [27]) is used as follows. Imagine that ball centers \( u \in S \subseteq V \) each grow a ball independently and in parallel, but with starting times shifted by \( -\delta_u \). Then, no matter how far exactly the ball centers are from a given edge \( e \) in the graph, the arrival times of the first and second ball differ by at least \( 2\ell_e \) with probability \( 1 - O(\beta \ell_e) \) and \( 1 - O(\frac{\log n}{\varepsilon r}) \), using \( \beta = \Theta(\frac{\log n}{\varepsilon r}) \). For any edge \( e \) of length \( \ell_e \), this means that the ball arriving first at one endpoint of the edge also is the first to arrive at the other endpoint with probability at least \( 1 - O(\frac{\ell_e \log n}{\varepsilon r}) \).

\[\text{Lemma 6} \text{ (Lemma 4.4 in [27]). Let } d_1 \leq \ldots \leq d_s \text{ be arbitrary values and } \delta_1, \ldots, \delta_s \text{ be independent random variables picked from } \text{Exp}_\beta. \text{ Then the probability that the smallest and the second smallest values of } d_i - \delta_i \text{ are within } c \text{ of each other is at most } O(\beta c).\]

\(^3\) Verified through personal communication with Sebastian Forster and Danupon Nanongkai.
\(^4\) Here the Heaviside step function is defined as \( H(x) = 0 \) if \( x < 0 \) and \( H(x) = 1 \) otherwise.
3 Computing a $(1/3, \varepsilon)$-Star Decomposition

We formally introduce $(\delta, \varepsilon)$-star decompositions following their presentation in [12].

**Definition 7.** We call a partition $V_0, \ldots, V_k$ of $V$ that satisfies

(a) for all $i \in [k] \cup \{0\}: G[V_i]$ is connected,
(b) for all $i \in [k]$ there is $e_i = (x_i, y_i) \in E$ with $x_i \in V_i$, $y_i \in V_0$

a $(\delta, \varepsilon)$-star decomposition, if, in addition,

1. $r_0 \leq (1 - \delta)r$, using the notation $r_i = \text{rad}_{x_i}(G[V_i])$ and $r = \text{rad}_{x_0}(G)$
2. for all $i \in [k]: d(x_0, x_i) \geq \delta r$
3. for all $i \in [k]: d(x_0, x_i) + r_i \leq (1 + \varepsilon)r$.

We call the nodes $x_1, \ldots, x_k$ the anchor nodes of the parts $V_1, \ldots, V_k$ and the edges $e_1, \ldots, e_k$ are called the bridge edges. We refer the reader to Figure 1 for an illustration. Note also that properties (2) and (3) imply that the radius of each of the graphs $G[V_1], \ldots, G[V_k]$ is upper bounded by $(1 + \varepsilon - \delta) \cdot r$.

![Figure 1](image-url) An illustration of a $(\delta, \varepsilon)$-star decomposition. The center part $V_0$ of radius $r_0 \leq (1 - \delta)r$ is connected to each part $V_i$ via a bridge edge $e_i = (x_i, y_i)$. The anchor nodes $x_i$ have distance at least $\delta r$ to $x_0$. Moreover, the distance of $x_0$ to $x_i$ with $i \geq 1$ plus the radius $r_i$ of the part $V_i$ is at most $(1 + \varepsilon)$ times the radius $r$ of the original graph. Note that the decomposition that we construct leads the stronger property that, for any node $v \in V_i$, the length of the path from $x_0$ to $v$ over the bridge edge $e_i$ (drawn in yellow) is at most $\varepsilon r$ longer than the shortest path $P_{x_0v}$ from $x_0$ to $v$ (drawn in green).

Given a root node $x_0 \in V$ and a radius $r_0$, let $V_0 := B(x_0, r_0)$. We let $S := \{u \in V \setminus V_0 : \exists v \in V_0, (u, v) \in E$ and $d(x_0, u) = d(x_0, v) + \ell_{(u, v)}\}$ be the so-called ball-shell of $V_0$, i.e., the nodes $u$ outside $V_0$ that have a neighbor $v$ in $V_0$ such that a shortest path from $x_0$ to $u$ passes through $v$. For a node $u \in S$, we fix $v^*_u$ to be some neighbor of $u$ in $V_0$ such that $d(x_0, u) = d(x_0, v^*_u) + \ell_{(x_0, v^*_u)}$.

Now, let $\delta : S \rightarrow \mathbb{R}_{\geq 0}$ be a function that assigns a non-negative real to every node on the ball-shell. For every node $u \in S$, we define the adjusted $\delta$-shifted distance of $u$ as

$$ad^{-\delta}_x(u) := d(x_0, u) + \max_{v \in S}\{\delta_v\} - \delta_u.$$ 

We remark that the shift by $\max_{v \in S}\{\delta_v\}$ is simply used in order to ensure non-negativity. These numbers define the delay after which $u$ starts to grow its ball (if it has not yet joined another ball), which we adjust compared to [27] by the distance of $u$ to $x_0$. This adjustment allows us to treat the general weighted case; in the unweighted setting, all of these distances
would be identical as \( u \) is a node on the ball-shell of \( V_0 \) and thus the resulting decomposition would remain unaffected by the adjustment.

We now describe Algorithm 1, which computes a \((1/3, \varepsilon)\)-star decomposition. The

\begin{algorithm}
\caption{star\_decompose\((G, x_0, \varepsilon)\)}
\begin{itemize}
\item \textbf{Input :} graph \( G = (V, E, \ell) \), node \( x_0 \in V \), \( \varepsilon > 0 \)
\item \textbf{Output :} \((1/3, \varepsilon)\)-star decomposition of \( G \) w.h.p.
\end{itemize}
\begin{algorithmic}[1]
\STATE 1 Compute \( r = \text{rad}_{x_0}(G) \).
\STATE 2 Set \( \beta := \frac{c\log n}{r} \), sample \( r_0 \in [\frac{r}{2}, \frac{2r}{3}] \) u.a.r.  // \( c \) is a suff. large constant
\STATE 3 Let \( V_0 = B(x_0, r_0) \) and \( H := G[V \setminus V_0] \).
\STATE 4 Let \( S := \{ u \in V \setminus V_0 : \exists v \in V_0, (u, v) \in E \text{ and } d(x_0, u) = d(x_0, v) + \ell(u, v) \} \).
\STATE 5 For each \( u \in S \), pick \( \delta_u \sim \text{Exp}_\beta \) independently.
\STATE 6 \( G_s := \) super-source graph obtained from \( H \) by attaching \( u \in S \) to \( s \) with length \( ad^{\delta_u}(u) \).
\STATE 7 Compute SSSP tree \( T \) of \( G_s \) rooted at \( s \).
\STATE 8 Let \( x_1, \ldots, x_k \) be the children of \( s \) in \( T \) and \( V_1, \ldots, V_k \) be the node sets of their subtrees.
\STATE 9 For each \( i \), let \( y_i = v_0^{x_i} \in V_0 \)
\STATE 10 \textbf{return} \( \{ V_0, V_1, \ldots, V_k, \text{ anchors } x_1, \ldots, x_k, \text{ nodes } y_1, \ldots, y_k \} \)
\end{algorithmic}
\end{algorithm}

algorithm starts by carving out the center ball around \( x_0 \), which has a randomized radius \( r_0 \) to ensure that edges \( e \) are cut with probability \( O(\ell_e/r) \). It then grows balls in \( G[V \setminus V_0] \) around the shell nodes \( S \), where the starting times are delayed according to random shifts \( \delta_v \sim \text{Exp}_\beta \) (this is the technique from [27]). Here, \( \beta \in \Theta(1/r) \) with \( r \) being the radius of \( G \) w.r.t. \( x_0 \), so that the probability to cut an edge \( e \) of length \( \ell_e \) is \( O(\ell_e/r) \). This is implemented by an SSSP computation with super-source \( s \), which is attached to shell node \( u \) by an edge of length \( ad^{\delta_u}(u) \), which results in the desired behavior. The subtrees rooted at children of \( s \) then correspond to the balls, and the algorithm can return the desired decomposition.

For ease of presentation, we assume in our analysis the non-integrality of the \( \delta_v \)'s is not an issue for the single source shortest path computations used; it is straightforward to use values that are rounded to integers.\footnote{This can be interpreted as distorting edge lengths by \( O(1) \) in our analysis (possibly even inconsistently in different bounds).}

\begin{corollary}
Algorithm 1 can be implemented in \( \tilde{O}(\min\{\sqrt{nD}, \sqrt{nD}^{1/4} + n^{3/5} + D\}) \) rounds w.h.p. If \( G \) is unweighted, it can be implemented in \( O(D) \) rounds w.h.p.
\end{corollary}

\textbf{Proof.} From the pseudo-code of the algorithm, it is immediate that all computations are local except (i) determining \( \text{rad}_{x_0}(G) \), (ii) finding \( B(x_0, r_0) \), (iii) determining \( T \), and (iv) determining the subtrees of \( T \setminus \{s\} \). (i) and (ii) can be performed by a call to an SSSP algorithm (a single call suffices, in fact) and making \( r_0 \) known to all nodes. The same holds true for (iii). Regarding (iv), in order to determine the connected components of \( T \setminus \{s\} \), we can invoke a variant of the minimum spanning tree algorithm by Garay, Kutten and Peleg [17, 24], which runs in \( \tilde{O}(\sqrt{n} + D) \). Applying Corollary 4, the first claim follows.

If \( G \) is unweighted, the first SSSP computation has running time \( O(D) \) by Lemma 3. The same holds for the second, provided that \( W(G_s) \in O(D) \). By Lemma 5, \( \max_{u \in S} \{ \delta_u \} \in \"}
\( O(\log n/\beta) \subset O(D) \) w.h.p. Thus,
\[
W(G_s) \leq 2 \text{rad}_s(G_s) \leq 2(\max_{u \in S}\{\delta_u\} + \text{rad}_{x_0}(G)) \in O(D)
\]

w.h.p. As the depth of \( T \) is at most \( W(G_s) \), the naive algorithm for finding the connectivity components of \( T \setminus \{s\} \) completes in \( O(D) \) rounds w.h.p. as well.

### 3.1 Correctness

We now show that Algorithm 1 indeed returns a \((1/3, \varepsilon)\)-star decomposition of \( G \) w.h.p.

**Theorem 9.** Algorithm 1 outputs a \((1/3, \varepsilon)\)-star decomposition w.h.p.

In order to prove the theorem, we examine the conditions in Definition 7. First, observe that the graphs \( G[V_i] \) are spanned by the components of \( T \setminus \{s\} \subset V \) and thus connected in \( G \). Therefore, condition (a) of the \((\delta, \varepsilon)\)-star decomposition holds by construction. In particular, \( r_i := \text{rad}_{x_0}(G[V_i]) \) is well-defined for every \( i \in [k] \). Similarly, (b) is immediate from the construction, (1) holds since \( r_0 \leq 2r/3 \), and (2) holds since \( r_0 \geq r/2 > r/3 \). We show that condition (3) holds w.h.p., the proof is based on Lemma 5.

**Lemma 10.** Let \( x_0 \in V \) be the root node given to \( G \). Moreover, let \( V_0, V_1, \ldots, V_k, S, \) and \( x_0, x_1, \ldots, x_k \) as well as \( y_1, \ldots, y_k \) be as in Algorithm 1. Let \( r = \text{rad}_{x_0}(G) \) and \( G' \) be the subgraph of \( G \) in which all edges in \( V_i \times V_j \) for \( i \neq j \in \{0, \ldots, k\} \) are deleted except for the bridge edges \((x_1, y_1), \ldots, (x_k, y_k)\). Then \( \text{rad}_{x_0}(G') \leq (1 + \varepsilon) \cdot r \) w.h.p., i.e., property (b) of Definition 7 holds.

**Proof.** For arbitrary \( v \in V \setminus V_0 \), denote by \( x_i \in S \) a shell node such that \( v \in V_i \). As \( H = G[V \setminus V_0] \) is a subgraph of \( G_s \) and \( d(x_i, v) = d_H(x_i, v) \) by construction, we have that
\[
d_{G'}(x_0, v) = d(x_0, y_i) + d_{x_0}(v) \leq d(x_0, x_i) + d_H(x_i, v) \leq d_{G_s}(x_0, v).
\]

By Lemma 5 and a union bound, \( \max_{w \in S}\{\delta_w\} < c \log n/\beta \leq \varepsilon r \) w.h.p. Let \( u \in S \) be a shell-node on a shortest path from \( x_0 \) to \( v \), i.e., \( d(x_0, v) = d(x_0, u) + d(u, v) \). Then, w.h.p.,
\[
d_{G_s}(x_0, v) \leq d_{G_s}(x_0, u) + d_{G_s}(u, v) \leq d(x_0, u) + \max_{w \in S} \{\delta_w\} - \delta_u + d_H(u, v) \leq d(x_0, u) + \varepsilon r \leq (1 + \varepsilon)r,
\]

where we again used that \( H = G[V \setminus V_0] \) is a subgraph of \( G_s \) and \( d(u, v) = d_H(u, v) \) by construction. As \( G' \) preserves distances to all nodes in \( V_0 \cup S \), the claimed bound on the radius follows. Note that property (3) of Definition 7 follows as well, as in \( G' \) the edges \((x_1, y_1), \ldots, (x_k, y_k)\) are bridges and thus
\[
\text{rad}_{x_0}(G') = \max_{i \in [k]} \{d(x_0, x_i) + \text{rad}_{x_0}(G[V_i])\}.
\]

### 3.2 Probability To Cut an Edge

Given a \((\delta, \varepsilon)\)-star decomposition \( V_0, \ldots, V_k \), we say that an edge is cut if its endpoints belong to different parts \( V_i \) of the decomposition. There are two different ways in which an edge can be cut by Algorithm 1. (1) It can be cut by the process of growing the center \( V_0 \) or (2) it is cut during the procedure in lines 5-8. We call \( E^\text{cut}_c := \{(u, v) \in E : u \in V_0, v \in V_i, i \in [k]\} \), i.e. edges between \( V_0, V_i \) for \( i \in [k] \) the edges resulting from (1) and \( E^\text{cut}_a := \{(u, v) \in E : u \in V_i, v \in V_j, i, j \notin [k]\} \), i.e. edges between \( V_i, V_j \) for \( i, j \in [k] \) the cut edges resulting from (2). Moreover we let \( E^\text{cut} = E^\text{cut}_c \cup E^\text{cut}_a \). The goal of this subsection is to prove the following lemma. Its proof is split in two lemmata, Lemma 12 and Lemma 13.
4: Distributed Low Stretch Spanning Trees

**Lemma 11.** For an edge \( e = (u, v) \in E \), it holds that \( \Pr[e \in E^{\text{cut}}] = O(\frac{\ell_e \log n}{er}) \).

It is simple to bound the probability of an edge \( e \) being in \( E^{\text{cut}}_r \).

**Lemma 12.** For an edge \( e = (u, v) \in E \), it holds that \( \Pr[e \in E^{\text{cut}}_r] \leq \frac{6\ell_e}{r} = O\left(\frac{\ell_e}{r}\right) \).

**Proof.** W.l.o.g. assume that \( d(x_0, u) \leq d(x_0, v) \). It then follows that

\[
\Pr[e \in E^{\text{cut}}_r] \leq \Pr[r_0 \in \{d(x_0, u), d(x_0, u) + \ell_e\}] = \frac{6\ell_e}{r} = O\left(\frac{\ell_e}{r}\right),
\]

since \( r_0 \) was picked u.a.r. from the interval \([r/2, 2r/3]\) of width \( r/6 \).

We now wish to bound the probability that an edge \( e = (u, v) \in E \) belongs to \( E^{\text{cut}}_{\Delta} \). Essentially, the desired bound is implicit in [27], but due to our modifications for the weighted setting we cannot apply the statements from this work as a black box entirely. However, the statement follows without much effort from Lemma 6.

**Lemma 13.** For an edge \( e = (u, v) \in E \), it holds that \( \Pr[e \in E^{\text{cut}}_{\Delta}] = O(\frac{\ell_e \log n}{\epsilon r}) \).

**Proof.** Recall that \( H = G[V \setminus V_0] \). If \((u, v) \notin E_H \), then \((u, v) \notin E^{\text{cut}}_{\Delta} \), so assume that \( e = (u, v) \in E_{H} \). Denote by \( x_u \in S \) the anchor node of the part \( u \) belongs to, i.e.,

\[
d_{G_s}(x_0, u) = d(x_0, x_u) + \max_{x \in S}\{\delta_x\} - \delta_{x_u} + d(x_u, u).
\]

We apply Lemma 6 to the values \( d_e = d(x_0, x) + d(x, u) \) and \( \delta_x \) chosen in line 5, where \( x \in S \).

This shows that with probability \( 1 - O(\beta \varepsilon) \), we have for all \( x \neq x_u \) that

\[
d(x_0, x) + d(x, u) - \delta_x > d(x_0, x_u) + d(x_u, u) - \delta_{x_u} + 2\ell_e.
\]

Adding \( \max_{x \in S}\{\delta_x\} - \ell_e \) on both sides of this inequality and applying the triangle inequality, this entails that

\[
ad_{x_u}^{\delta}(x_u) + d(x_u, v) \leq d(x_0, x_u) - \delta_{x_u} + \max_{x \in S}\{\delta_x\} + d(x_u, u) + \ell_e
\]

\[
< d(x_0, x) + d(x, u) - \delta_x - \ell_e + \max_{x \in S}\{\delta_x\} \leq ad_{x_u}^{-\delta}(x) + d(x, v)
\]

for all \( x \in S \setminus \{x_u\} \). It follows that the shortest path from \( x_0 \) to \( v \) in \( G_s \) passes through \( x_u \), i.e., \( v \) is in the same part as \( u \) and \( e \) is not cut. Accordingly, the probability that \( e \in E^{\text{cut}}_{\Delta} \) is bounded by \( O(\beta \varepsilon) = O\left(\frac{\ell_e \log n}{\epsilon r}\right) \), as claimed.

4 Building the Low-Stretch Spanning Tree

We now give an algorithm that uses our \((1/3, \varepsilon)\)-star decomposition algorithm recursively in order to compute a spanning tree of low expected stretch, see Algorithm 2. As described above, the algorithm takes as input the graph \( G = (V, E, \ell) \) with \( n \) nodes and a root node \( x_0 \in V \) (\( x_0 \) can be chosen arbitrarily) and outputs a tree \( T \) such that \( \E[\text{str}_T(e)] = O(\log^3 n) \) for any edge \( e \in E \).

Consider an invocation of Algorithm 2 on an input graph \( G \) with root node \( x_0 \in V \). Let \((H^{(0)}, x^{(0)}), (H^{(1)}, x^{(1)}), \ldots \) with \( G = H^{(0)} \) and \( x_0 = x^{(0)} \) be a sequence of graphs and root nodes corresponding to a path in the recursion tree. Moreover, let \( r^{(k)} = \text{rad}_{x^{(k)}}(H^{(k)}) \) be the radius of the \( k \)'th graph in this sequence with respect to \( x^{(k)} \).

**Corollary 14.** The recursion depth of Algorithm 2 is \( O(\log n) \) w.h.p.
As the minimum edge length is constant $c\rad$, interrupting Algorithm 2 at recursion level $R$ for some graph $G$ and partition $V_0,\ldots,V_k$ of its node set. Note that the defined sequence of $R^{(0)}(G)$ becomes sparser and sparser until the final one is the tree returned by the algorithm. As opposed to the sequence $H^{(0)},H^{(1)},\ldots$ that we considered previously (corresponding to a recursive path in the recursion tree) however, each of the graphs in $R^{(0)}(G),R^{(1)}(G),\ldots$ contains all the nodes of $G$. In fact, $R^{(0)}(G)$ is the graph that we would obtain when interrupting Algorithm 2 at recursion level $\ell$.  

\begin{algorithm}
\begin{algorithmic}
\State \textbf{Input} : graph $G = (V,E,\ell)$ with $n$ nodes, root node $x_0 \in V$
\State \textbf{Output}: spanning tree $T$ s.t. $E[\str_T(\epsilon)] = O(\log^2 n)$ for every edge $e \in E$
\State $\epsilon = \min\{\frac{\ell}{\log n}, 1\}$
\If {$|V| \leq 2$} \State return $G$
\Else
\For {$i \in [k] \cup \{0\}$} \State $T_i = \lowstretch(G[V_i],x_{i})$
\EndFor
\State return $T = \bigcup_{i \in [k] \cup \{0\}} T_i \cup \bigcup_{i \in [k]} \{(x_{i}, y_{i})\}$
\EndIf
\end{algorithmic}
\end{algorithm}

**Proof.** By Theorem 9, each call to Algorithm 1 returns a $(1/3,\epsilon)$-star decomposition w.h.p. Condition on these events. Denoting $r = \rad(G)$, from properties (1) and (2) of a $(1/3,\epsilon)$-star decomposition we get by induction that $(\frac{2}{3})^i r = \max\{1 - \delta, \delta\}^i r$ is a bound on the radius of subgraphs the algorithm is called on in recursion depth $i$. As edge lengths are polynomially bounded, we have that $r = o_n(1)$, implying that there is $\epsilon_{\max} \in O(\log n)$ so that $(\frac{2}{3})^i r < 1$.

As the minimum edge length is 1, this entails that such a graph contains no edge and the recursion stops. As the number of recursive calls in depth $i$ is trivially bounded by $n$ (the maximum number of disjoint, non-empty subgraphs of $G$), we conditioned on $O(n \log n)$ events that occur w.h.p. By a union bound, the claim follows. ◀

We will now shift focus towards the following sequence of recursively defined graphs

$$R^{(0)}(G) := G, \quad \text{and} \quad R^{(i)}(G) := \bigcup_{i \in [k] \cup \{0\}} R^{(i-1)}(G)[V_i] \cup \bigcup_{i \in [k]} \{(x_i, y_i)\} \quad \text{for } t \geq 1,$$

for some graph $G$ and partition $V_0,\ldots,V_k$ of its node set. Note that the defined sequence of $R^{(0)}(G)$ becomes sparser and sparser until the final one is the tree returned by the algorithm. As opposed to the sequence $H^{(0)},H^{(1)},\ldots$ that we considered previously (corresponding to a recursive path in the recursion tree) however, each of the graphs in $R^{(0)}(G),R^{(1)}(G),\ldots$ contains all the nodes of $G$. In fact, $R^{(0)}(G)$ is the graph that we would obtain when interrupting Algorithm 2 at recursion level $\ell$.

\begin{lemma}
For a graph $G$ and the sequence $R^{(i)}(G)$ as defined above, let $\rho^{(k)} := \rad(G^{(k)}(G))$ and let $\gamma^{(k)} = \rho^{(k)}/\rho^{(k-1)}$. Then

$$E[\gamma^{(k)}] = 1 + 2\epsilon \quad \text{and} \quad E[\rho^{(k)}] \leq (1 + 2\epsilon)^k \cdot \rad(G).$$

\end{lemma}

**Proof.** By Lemma 10, we have $\Pr[\rho^{(k)} \leq (1 + \epsilon)\rho^{(k-1)}] = \Pr[\gamma^{(k)} \leq 1 + \epsilon] \geq 1 - n^{-c}$ for a constant $c$ under our control. Choosing $c$ sufficiently large, thus

$$E[\gamma^{(k)}] \leq (1 + \epsilon) + n \cdot n^{-c} \leq 1 + 2\epsilon, \quad \text{using } \epsilon = 1/\log n \geq n^{-c+1}.$$

For the second claim, we get

$$E[\rho^{(k)}] = E\left[\rho^{(0)} \cdot \prod_{i=1}^k \gamma^{(k)}\right] = \rho^{(0)} \cdot \prod_{i=1}^k E[\gamma^{(k)}] \leq (1 + 2\epsilon)^k \cdot \rad(G),$$

since the events are independent. ◀
Lemma 16. \( E[\text{str}_T(e)] = O((\log^3 n) \text{ for the expected stretch of each edge } e=(u,v) \in E) \).

Proof. By Corollary 14, the recursion depth \( \lambda \) of Algorithm 2 satisfies \( \lambda \in O(\log n) \) w.h.p. W.l.o.g., condition on this event.\(^6\) Let us denote with \( E_k \) the set of edges being cut at recursive level \( k \) and let \( d := d_T(u,v) \) for notational convenience. Note that the event that edge \( e \) is cut in depth \( i \) of the recursion is disjoint from the event that it is cut in depth \( j \neq i \). Hence, using the law of total expectation and Lemma 11, we get that

\[
E[d] = E[d | e \in T] + \sum_{k=0}^{\lambda} E[d | e \in E_k] \cdot \Pr[e \in E_k] = \ell_e + \sum_{k=0}^{\lambda} E[d | e \in E_k] \cdot \Pr[e \in E_k].
\]

Consider the special case that \( e \in E_0 \). Clearly, \( E[d | e \in E_0] \leq E[2 \text{rad}_{x_0} R^\lambda(G)] \) (i.e., twice the radius of the computed spanning tree) in the notation of Lemma 15. By the lemma and the fact that \( \varepsilon \leq 1/\log n \), we get that

\[
E[d | e \in E_0] \leq E[2 \text{rad}_{x_0} R^\lambda(G)] \leq 2(1+2\varepsilon)^\lambda \cdot \text{rad}_{x_0}(G) \in O(\text{rad}_{x_0}(G)).
\]

Applying Lemma 11, we arrive at

\[
E[d | e \in E_0] \cdot \Pr[e \in E_0] \in O \left( \frac{\ell_e \log n}{\varepsilon} \right) = O(\ell_e \log^2 n).
\]

Now consider the case that \( e \in E_k \) for some \( k \neq 0 \). Thus, \( e \) was contained in some connected subgraph \( H \) of \( G \) that Algorithm 2 was called on recursively. The same reasoning hence shows that

\[
E[d | e \in E_k] \cdot \Pr[e \in E_k] \in O(\ell_e \log^2 n)
\]

We conclude that

\[
E[\text{str}_T(e)] = \frac{E[d]}{\ell_e} = 1 + \sum_{k=0}^{\lambda} \frac{E[d | e \in E_k] \cdot \Pr[e \in E_k]}{\ell_e} \in 1 + O(\lambda \log^2 n) \subseteq O(\log^3 n). \]

Proof of Theorem 1. By Lemma 16, Algorithm 2 achieves the stated guarantee on the stretch. By Corollary 14, its recursion depth is \( O(\log n) \) w.h.p. Each call performs a call to Algorithm 1, which can be implemented with the stated running time bound by Corollary 8. Observe that we can perform for each SSSP computation step of Algorithm 1 on recursion level \( i \) the computation for all instances on this recursion level with a single instance of the SSSP algorithm we use: we perform the computation on the union of the subgraphs induced by disjoint sets, yielding for each subgraph the correct tree \( T \) by deleting all nodes not belonging to the induced subgraph. By Corollary 8, the stated running time bounds follow, where in the unweighted case we exploit that radii (and thus diameters) decrease exponentially with the recursion depth (cf. Corollary 14).

\[
\]

References


\(^6\) As edge weights are from \( 1,\ldots,n^{O(1)} \), stretch is trivially bounded by \( n^{O(1)} \). By choosing the constant \( c \) large enough, the expected contribution of events that occur with probability at most \( 1/n^c \) can be made negligible.
Distributed Low Stretch Spanning Trees


