Low Diameter Graph Decompositions by Approximate Distance Computation

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

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

―――― **Abstract** ――――

In many models for large-scale computation, decomposition of the problem is key to efficient algorithms. For distance-related graph problems, it is often crucial that such a decomposition results in clusters of small diameter, while the probability that an edge is cut by the decomposition scales linearly with the length of the edge. There is a large body of literature on low diameter graph decomposition with small edge cutting probabilities, with all existing techniques heavily building on *single source shortest paths (SSSP)* computations. Unfortunately, in many theoretical models for large-scale computations, the SSSP task constitutes a complexity bottleneck. Therefore, it is desirable to replace exact SSSP computations with approximate ones. However this imposes a fundamental challenge since the existing constructions of low diameter graph decomposition with small edge cutting probabilities inherently rely on the subtractive form of the triangle inequality, which fails to hold under distance approximation.

The current paper overcomes this obstacle by developing a technique termed *blurry ball growing*. By combining this technique with a clever algorithmic idea of Miller et al. (SPAA 2013), we obtain a construction of low diameter decompositions with small edge cutting probabilities which replaces exact SSSP computations by (a small number of) approximate ones. The utility of our approach is showcased by deriving efficient algorithms that work in the CONGEST, PRAM, and semi-streaming models of computation. As an application, we obtain metric tree embedding algorithms in the vein of Bartal (FOCS 1996) whose computational complexities in these models are optimal up to polylogarithmic factors. Our embeddings have the additional useful property that the tree can be mapped back to the original graph such that each edge is "used" only logaritmically many times, which is of interest for capacitated problems and simulating CONGEST algorithms on the tree into which the graph is embedded.

1 Introduction

Consider an n-vertex graph $G = (V, E, \ell)$, where $\ell : E \to \mathbb{Z}_{>0}$ is an edge *length* function.[1] The *distance* between two vertices u and v in G, denoted by $\mathrm{d}_G(u, v)$, is defined to be the length with respect to ℓ of a shortest (u, v)-path in G. The *diameter* of G is the maximum distance between any two vertices, denoted by $\mathrm{diam}(G) = \max_{u,v \in V} \{\mathrm{d}_G(u, v)\}$.

A *decomposition* D of G is a partition of the vertex set V into pairwise disjoint *clusters*. Such a decomposition induces a (multiway) *cut* on G and we use $E^{\mathrm{cut}}(D)$ to denote the subset of edges that cross this cut, namely, edges whose endpoints belong to different clusters of D. The *weight* of the decomposition D is defined to be the sum $\sum_{e \in E^{\mathrm{cut}}(D)} \frac{1}{\ell_e}$ of the reciprocal lengths of the edges crossing its cut. Our focus in this paper is on the construction of decompositions whose clusters' diameter is bounded by some specified parameter r (the notion of a cluster's diameter will be made clear soon), referred to hereafter as *low diameter decompositions*. The challenging part is to keep the weight of D small.

Low diameter decompositions with small weight were first studied by Awerbuch [5] (see also [6, 4]). Bartal [7] introduced their (combinatorially equivalent) probabilistic counterpart: An (r, λ)-*decomposition* of the graph $G = (V, E, \ell)$ is a random decomposition D of G such that (1) the diameter of each cluster in D is at most r; and (2) $\Pr[e \in E^{\mathrm{cut}}(D)] \leq \frac{\lambda \ell_e}{r}$ for every edge $e \in E$. Bartal presented a method that, for a given parameter r, constructs an $(r, O(\log n))$-decomposition and proved the resulting bound on the edge cutting probabilities to be asymptotically tight.

Low diameter decompositions with small edge cutting probabilities have proven to be very useful in the algorithmic arena (see Section 7) and several different techniques have been developed over the years for constructing them [3, 7, 18, 22, 43, 24]. A common thread of all the existing techniques is that they rely heavily on making calls to a single source shortest paths (SSSP) subroutine. While we know how to solve the SSSP problem efficiently in the sequential (centralized) model of computation, the situation is much more challenging in restricted models of computation such as the CONGEST model of distributed computing, the parallel random access memory (PRAM) model, or the semi-streaming graph algorithms model. As it stands, SSSP computations are the main obstruction to designing efficient constructions of low diameter decompositions with small edge cutting probabilities in the aforementioned computational models (and related ones).

1.1 Our Contribution

In this paper, we introduce a new technique that, given a graph $G = (V, E, \ell)$ and a parameter r, constructs an $(r, O(\log n))$-decomposition of G. The crux of our construction is that it does not rely on any *exact* SSSP computations. Rather, it efficiently reduces the task to a small number of calls to an approximate SSSP subroutine. The technical challenge in this regard stems from the fact that the existing constructions of low diameter decompositions with small edge cutting probabilities crucially rely on the subtractive form of the triangle inequality, stating that $\mathrm{d}_G(u, v) \geq \mathrm{d}_G(u, w) - \mathrm{d}_G(v, w)$ for every three vertices $u, v, w \in V$. Due to the subtraction on the right hand side, the inequality fails if one replaces exact distances with approximate ones. The main technical contribution of this paper lies in overcoming this difficulty.

The approximate SSSP problem can be solved efficiently in the CONGEST [12], PRAM

---

[1] We sometimes use the shorthand ℓ_e for $\ell(e)$.

[14], and semi-streaming [12] models, hence we obtain efficient algorithms for constructing $(r, O(\log n))$-decompositions for the three computation models. These in turn can be invoked recursively to yield efficient CONGEST, PRAM, and semi-streaming constructions of path embeddable trees [16, 15] and hierarchically well-separated trees [7, 8, 9, 22] with low *stretch* — important combinatorial objects in their own right. In fact, our low diameter decompositions (and the resulting tree embeddings) admit an even stronger property.

Tree-Supported Decompositions

The notion of graph diameter naturally extends from the entire graph $G = (V, E, \ell)$ to a vertex subset $U \subseteq V$ by considering the maximum distance between any two vertices in U. This yields the following distinction: the *weak diameter* of U in G considers the distances in the underlying graph G, formally defined as $\max_{u,v \in U}\{d_G(u, v)\}$; the *strong diameter* of U in G considers the distances in the subgraph $G(U)$ induced by G on U, formally defined as $\mathrm{diam}(G(U))$.[2] In the context of low diameter graph decompositions with small edge cutting probabilities, both the weak and strong notions of the cluster diameter have been considered in the literature. As we now explain, the current paper adopts a diameter notion that falls somewhere in between the two.

For a decomposition D of the graph $G = (V, E, \ell)$, we require that each cluster $C \in D$ is associated with a tree $T_C = (U_C, F_C)$, referred to as the *supporting tree* of C, that is a subgraph of G and spans C, i.e., $C \subseteq U_C \subseteq V$ and $F_C \subseteq E$. To emphasize this requirement, we refer to the decomposition D as a *tree-supported decomposition (TSD)*. The *diameter* of a TSD D of G is then defined to be the maximum diameter of any of its supporting trees, denoted by $\mathrm{diam}(D) = \max_{C \in D}\{\mathrm{diam}(T_C)\}$.

Notice that if the supporting tree T_C of each cluster $C \in D$ is required to be a spanning tree of $G(C)$, then $\mathrm{diam}(D)$ bounds the strong diameter of D's clusters. This requirement is not imposed in the current paper, allowing T_C to use edges (and vertices) outside of $G(C)$, meaning that $\mathrm{diam}(D)$ merely bounds the weak diameter of the clusters. However, we do require that the maximum edge *load* is kept small, where the load of edge $e \in E$ in D is defined to be the number of clusters $C \in D$ such that e is included in the supporting tree of C, denoted by $\mathrm{load}_D(e) = |\{C \in D : e \in F_C\}|$. The properties of our graph decomposition construction can now be formally stated.

▶ **Theorem 1.** *There exists a (randomized) algorithm that given a graph $G = (V, E, \ell)$ and a real parameter $r > 0$, constructs a random TSD D of G with the following guarantees: (1) $\mathrm{diam}(D) \leq r$ w.h.p.;[3] (2) $\max_{e \in E}\{\mathrm{load}_D(e)\} \leq O(\log n)$ w.h.p.; and (3) $\Pr\left[e \in E^{\mathrm{cut}}(D)\right] \leq O\left(\frac{\ell_e \cdot \log n}{r}\right)$ for every edge $e \in E$. The algorithm is based on an approximate SSSP subroutine without any exact SSSP computations.*

The algorithm promised in Theorem 1 is designed by combining a novel technique termed *blurry ball growing* with the algorithmic ideas of Miller et al. [43]. As discussed earlier, this combination allows us to bypass the need for exact SSSP computations, implementing our algorithm based solely on approximate SSSP. By example of the CONGEST, PRAM, and semi-streaming models, we show that this leads to efficient implementations. We stress that what little computation is performed beyond approximate SSSP computations is very easy,

---

[2] Unless stated otherwise, the edge length function of a subgraph H of G is the restriction of ℓ to H's edge set.

[3] We say that event A occurs *with high probability*, abbreviated *w.h.p.*, if $\Pr[A] \geq 1 - n^{-c}$, where c is an arbitrarily large constant chosen upfront.

if not trivial, to implement. Accordingly, we expect the technique to carry over to further computational models.

We emphasize that our decomposition maintains a small load of $O(\log n)$ on the edges. Consequently, in many situations, our decomposition can be used in an identical way as a strong diameter decomposition with only polylogarithmic overheads. For example, although we cannot construct low average stretch spanning trees as these are required to be subgraphs of the original graph, we can construct *projected trees* (see Section 5.2), a special case of path-embeddable trees [15, 16]. Projected trees have a mapping of their edges to the original graph such that, e.g., a CONGEST algorithm on the projected tree can be simulated on the original graph with a round complexity overhead proportional to the maximum edge load. Our result is related to the low-congestion shortcuts of Ghaffari and Haeupler [28] with the following differences. In Ghaffari and Haeupler's work, the partition is chosen by an adversary and the input is restricted to unweighted graphs. In contrast, our technique constructs the partition, but weighted graphs can be treated as well. A further possible application of our projected trees is in the field of solvers for symmetric diagonally dominant linear systems, utilizing them in a similar way as low average stretch spanning trees (cf. [16, 15]). Prior algorithms for metric tree embeddings lack this property and, accordingly, cannot take this role.

1.2   Structure of this Paper

We first fix some notation and state basic facts in the preliminaries in Section 2. In Section 3, we present the blurry ball growing technique that we use in Section 4 in order to obtain the routine for computing a random TSD of low diameter, load, and edge cutting probability, as promised in Theorem 1. In Section 5, we highlight some applications of this routine: We first explain how to obtain a hierarchical decompositions by applying the method recursively (Section 5.1) and then show how to obtain random projected trees (Section 5.2) and hierarchically well-separated trees (Section 5.3) with $O(\log^2 n)$ bound on the expected stretch. We also show that this bound can be improved to $O(\log n)$ by considering the relaxed notion of p-stretch [15, 16] (Section 5.4). In Section 6, we explain how to implement our algorithms in the CONGEST, PRAM, and semi-streaming models. Further related work is reviewed in Section 7.

2   Preliminaries

We start with basic notation. We consider a weighted, undirected, connected n-vertex graph $G = (V, E, \ell)$, where $\ell : E \to \mathbb{Z}_{\geq 0}$ is an edge length function. Notice that while some of our subroutines introduce edges $e \in E$ with zero length $\ell_e = 0$, it is assumed that all edges in the original graph input to the $(r, O(\log n))$-decomposition algorithm (as well as the algorithms built on top of it) admit positive lengths. We denote the set of positive length edges by $E_{>0} = \{e \in E \mid \ell_e > 0\}$ and let $\ell_{\min} = \min\{\ell_e \mid e \in E_{>0}\}$ and $\ell_{\max} = \max\{\ell_e \mid e \in E_{>0}\}$. The ratio of $\ell_{\max}$ to $\ell_{\min}$, denoted by $\sigma = \frac{\ell_{\max}}{\ell_{\min}}$ is referred to as the *aspect ratio* of G.

For a subgraph H of G, we denote by $d_H(u, v)$ the length of the shortest path between two nodes u and v in H. If $H = G$, we may omit the subscript. For a set $B \subseteq V$ and a node $v \in V$, we use $d(B, v) := \min_{u \in B}\{d(u, v)\}$ to denote the distance of the node v to the set B. For a set of vertices $U \subseteq V$, we denote by $E^{\text{cut}}(U) := \{e = \{u, v\} \in E : u \in U, v \in V \setminus U\}$ the set of edges that are "cut" by U.

Approximate Single Source Shortest Paths

The main subroutine we use in our algorithm computes $(1+\varepsilon)$-*approximate* SSSP in undirected graphs. A $(1+\varepsilon)$-*approximate SSSP algorithm* is an algorithm that takes as input a weighted undirected graph $G = (V, E, \ell)$ and a source node $s \in V$ and returns a spanning tree T of G such that, for every node $v \in V$, the length of the path from s to v in T is at most $(1 + \varepsilon) \cdot \mathrm{d}(s, v)$, i.e., $\mathrm{d}(s, v) \leq \mathrm{d}_T(s, v) \leq (1 + \varepsilon) \cdot \mathrm{d}(s, v)$.

Super-Source Graphs

Our approach requires $(1 + \varepsilon)$-approximate SSSP computations 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 $G_s = (V_H \dot\cup \{s\}, E_H \cup E_s, \ell^{G_s})$ by choosing $E_s \subseteq V_H \times \{s\}$, picking $\ell_e^{G_s} \in \{1, \ldots, n^c\}$ for $e \in E_s$, and setting $\ell_e^{G_s} = \ell_e$ 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.

We note that one way of obtaining a super-source graph of a graph G is to contract a subset of nodes, say B, into a super-source s. In this case $V_H = V \setminus B$ and the edges E_s and their lengths result from the contraction of B into s.

Exponential Distribution

We denote the exponential distribution with mean $\frac{1}{\beta}$ by Exp_β. Using the Heaviside step function that is defined as $H(x) = 0$ if $x < 0$ and $H(x) = 1$ otherwise, the density function of the exponential distribution is given by $f_{\mathrm{Exp}_\beta}(x) = \beta \exp(-\beta x) \cdot H(x)$. Its cumulative density function is $F_{\mathrm{Exp}_\beta}(x) = (1 - \exp(-\beta x)) \cdot H(x)$. A standard result is that drawing from this distribution results in values of $O(\beta \log n)$ w.h.p.:

▶ **Lemma 3.** *For parameters $0 < \varepsilon < 1$, $\beta > 0$, and a sufficiently large constant $c > 0$, let $t := \frac{c \log n}{4(1+\varepsilon)\beta}$ and $X \sim \mathrm{Exp}_\beta$. Then $P[X \geq t] = n^{-\Omega(c)}$, i.e., $X < t$ w.h.p.*

Proof. Using the form of the density function, we get

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

We will make heavy use of the following lemma, see the paper by Miller et al. [43] for the proof. Note that in their paper they state the lemma with an upper bound of $O(\beta c)$ on the probability, although their proof in fact bounds the probability by exactly βc.

▶ **Lemma 4** (Lemma 4.4 in [43]). *Let $d_1 \leq \ldots \leq d_s$ be arbitrary values and $\delta_1, \ldots, \delta_s$ be independent random variables picked from Exp_β. Then the probability that the smallest and the second smallest values of $d_i - \delta_i$ are within c of each other is at most βc.*

Miller et al. [43] used this lemma to analyze the following ball growing technique that proceeds in time steps. Every node u in the graph grows a ball B_u independently and in parallel, but with a delay of δ_u time steps, where $\delta_u \sim \mathrm{Exp}_\beta$. Every ball increases its radius by 1 in each time step and we say that the ball B_v "arrives" at node u, if node v minimizes $\mathrm{d}(u, v) - \delta_v$ over all nodes. In this case u "gets absorbed" by v's ball B_v. The process stops when every node u is absorbed by some ball. Notice that u gets absorbed by its own ball B_u, if and only if no other ball arrives at u during the first δ_u time steps.

Now consider an arbitrary edge e in the graph and imagine it to be split into two equal length edges by a node v_e. If we let $d_1 \leq \ldots \leq d_n$ denote the n values $\mathrm{d}(u, v_e) - \delta_u$ for every $u \in V$, the above lemma shows that the arrival times of the first and second ball at node v_e differ by at least $2\ell_e$ with probability $1 - O(\beta \ell_e) = 1 - O(\frac{\ell_e \log n}{\varepsilon r})$, when choosing $\beta = \Theta(\frac{\log n}{\varepsilon r})$. Hence the lemma allows for bounding the probability of an edge being cut by such ball growing process with exponentially distributed delays.

We remark that the implementations in Section 6 draw from discrete distributions. Rounding continuous distributions to multiples of n^{-c} for sufficiently large $c \in O(1)$ yields w.h.p. the same results, but limits the number of random bits required to draw and store a random value to $O(\log n)$.

3   Blurry Ball Growing

In this section, we describe a subroutine called `blur` that takes as input a graph $G = (V, E, \ell)$ with aspect ratio $\sigma = \frac{\ell_{\max}}{\ell_{\min}} \leq \mathrm{poly}(n)$, a node set $B \subseteq V$, and a real $0 < \rho < \ell_{\max}$, and outputs a superset U of B. It guarantees that nodes in U are not too far from B, yet the probability to cut edges is small. More precisely, we establish the following theorem.

▶ **Theorem 5.** *Let $n \geq 2$. There is a routine* `blur`(G, ρ, B) *that outputs a superset U of B such that:*

1. *For every edge $e \in E$, the probability that $e \in E^{\mathrm{cut}}(U)$ is bounded by $O(\frac{\ell_e}{\rho})$.*
2. *For every $v \in U$, it holds that $\mathrm{d}(B, v) \leq \frac{\rho}{1-\alpha}$, where $\alpha = \frac{1}{2 \log n}$.*

The routine `blur`, see Algorithm 1, is based on $(1 + \varepsilon)$-approximate SSSP computations and contractions of node sets and thus can be readily parallelized. The basic idea is to grow a ball of uniformly random radius around B, where contraction of B yields the super-source of the SSSP computation. However, as approximating distances may imply that the "noise" due to the relative ε-error may cut a short edge with a comparatively large probability, the procedure is repeated with random radii drawn from uniform distributions with width that decrease by factor $\alpha = \frac{1}{2 \log n}$ in each step. To make this work, the approximation error of the SSSP algorithm must satisfy $\varepsilon \leq \alpha^2$. Accordingly, it would be desirable to chose α large for the sake of small computational costs in the approximate SSSP routine. However, it turns out that, in order to achieve Property 1 in Theorem 5, we have to set α such that $\alpha = O(\frac{\log \log n}{\log n})$. In addition, the approximate SSSP computations must respect the 0-length edges in the sense that none of the balls we grow cuts these edges. This is ensured by updating the approximate distances so that all nodes in the same connected component of the graph induced by the 0-length edges hold the same value.

Analysis

We begin with two important properties of the distance approximations $\widetilde{\mathrm{d}}^i(\cdot)$ computed in line 9.

▶ **Lemma 6.** *For every iteration i and vertex v in $G^{[i]}$, the distance approximation $\widetilde{\mathrm{d}}^i(v)$ satisfies (I) $\mathrm{d}_{G^{[i]}}(s^{[i]}, v) \leq \widetilde{\mathrm{d}}^i(v) \leq (1 + \alpha^2)\, \mathrm{d}_{G^{[i]}}(s^{[i]}, v)$; and (II) if $\{u, v\}$ is a 0-length edge in $G^{[i]}$, then $\widetilde{\mathrm{d}}^i(u) = \widetilde{\mathrm{d}}^i(v)$.*

Proof. Property (II) follows immediately from the definition of $\widetilde{\mathrm{d}}^i(v)$ (see line 9). To see that property (I) holds, notice that $\widetilde{\mathrm{d}}^i(v)$ captures the distance from $s^{[i]}$ to v in the subgraph

---

Algorithm 1: blur(G, ρ, B)

Input : graph $G = (V, E, \ell)$, real $0 < \rho < \ell_{\max}$, set $B \subset V$
Output: set $U \subseteq V$

1 $i := 0$, $B^{[0]} := B$, $\alpha := \frac{1}{2 \log n}$
2 **while** $\alpha^i \rho \geq \ell_{\min}$ **do**
3 $\quad$ $i := i + 1$, $r^{[i]} \in \mathcal{U}[0, \alpha^{i-1}\rho]$.
4 $\quad$ Obtain super-source graph $G^{[i]}$ from G by contracting $B^{[i-1]}$ into a super-source
$\quad\quad$ node $s^{[i]}$
5 $\quad$ Compute $(1 + \alpha^2)$-approximate SSSP tree $T^{[i]}$ of $G^{[i]}$
6 $\quad$ Let $G_0^{[i]}$ be the restriction of the graph $G^{[i]}$ to its 0-length edges
7 $\quad$ **for** *each* $v \in G^{[i]}$ **do**
8 $\quad\quad$ Let $W(v)$ be the connected component of v in $G_0^{[i]}$
9 $\quad\quad$ $\widetilde{\mathrm{d}}^{[i]}(v) := \min\{\mathrm{d}_{T^{[i]}}(s^{[i]}, u) \mid u \in W(v)\}$
10 $\quad$ $B^{[i]} := B^{[i-1]} \cup \{v \in G^{[i]} \mid \widetilde{\mathrm{d}}^{[i]}(v) \leq r^{[i]}\} \setminus \{s^{[i]}\}$
11 **return** $\bigcup_{j=0}^{i} B^{[j]}$

---

of $G^{[i]}$ induced by the union of the edge set of $T^{[i]}$ and the 0-length edges. The assertion follows since $T^{[i]}$ is a $(1 + \alpha^2)$-approximate SSSP tree of $G^{[i]}$. ◀

Next, we establish Property 2 of Theorem 5, which readily follows from the manner in which we sample $r^{[i]}$ from $\mathcal{U}[0, \alpha^{i-1}\rho]$.

▶ **Lemma 7.** *If* $\mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, u) = \mathrm{d}(B^{[i]}, u) \geq \frac{\alpha^i \rho}{1-\alpha}$ *for some* i, *then* $u \notin U$. *In particular, it holds that* $\mathrm{d}_G(B, v) \leq \frac{\rho}{1-\alpha}$ *for every* $v \in U$.

Proof. Any $u \in B^{[k]}$ for $k > i$ has distance to $B^{[i]}$ at most $\sum_{j \geq i+1} r^{[j]} \leq \sum_{j \geq i+1} \alpha^{j-1}\rho < \alpha^i \rho \sum_{j=0}^{\infty} \alpha^j = \frac{\alpha^i \rho}{1-\alpha}$, showing the first claim. Setting $i = 0$ yields the second claim. ◀

It remains to verify Property 1 of Theorem 5, i.e., that the probability of cutting edge $e = \{u, v\} \in E$ is bounded from above by $O(\frac{\ell_e}{\rho})$. Property (II) of Lemma 6 ensures that this bound holds if $\ell_e = 0$ as in this case, in each iteration i, either both u and v join $B^{[i]}$ in line 10 or none of them does. In the remainder of this section, we therefore focus on the edges in $E_{>0} = \{e \in E \mid \ell_e > 0\}$, starting with the following definition.

▶ **Definition 8.** *We say that edge* $\{u, v\} \in E_{>0}$ *is* safe *after step* i *of* blur(G, ρ, B) *if either* $u, v \in B^{[i]}$ *or* $\min\{\mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, u), \mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, v)\} \geq \frac{\alpha^i \rho}{1-\alpha}$.

Clearly, if $\{u, v\} \in E$ is safe after step i of blur(G, ρ, B), then $e \notin E^{\mathrm{cut}}(U)$: if $u, v \in B^{[i]}$, then $u, v \in U$ by construction; if $\min\{\mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, u), \mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, v)\} \geq \frac{\alpha^i \rho}{1-\alpha}$, then $u, v \notin U$ by Lemma 7. See Figure 1 for an illustration of these two cases. Thus, in order to bound the probability of an edge being cut, it suffices to bound the probability that an edge never becomes safe. Accordingly, we define $X_{i,e}$ to be the event that e is not safe after step i of the algorithm conditioned on the event that e was not safe after step $i - 1$ and bound $P[X_{i,e}]$.

▶ **Lemma 9.** *For each iteration* i *and* $e \in E_{>0}$, *it holds that* $\Pr[X_{i,e}] \leq \frac{5}{4} \cdot \frac{\ell_e}{\alpha^{i-1}\rho} + \alpha \cdot (1 + 4\alpha)$.
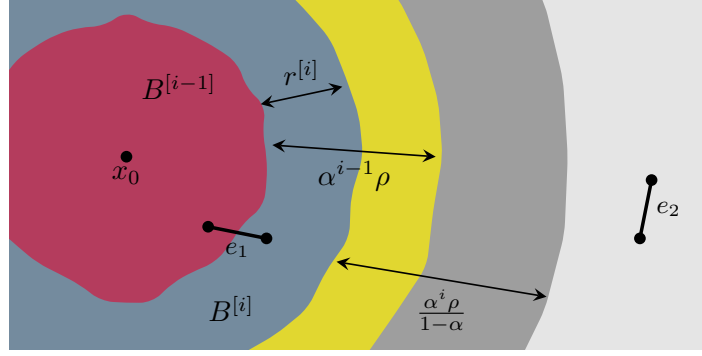
■ **Figure 1** An illustration of the blurry ball growing procedure $\texttt{blur}(G, \rho, B)$ in iteration i. The radius $r^{[i]}$ is sampled uniformly from $[0, \alpha^{i-1}\rho]$ and $B^{[i]}$ is defined as all nodes whose $(1 + \alpha^2)$-approximate distance to $B^{[i-1]}$ is at most $r^{[i]}$. Both edges e_1 and e_2 are safe from being cut after iteration i: e_1 has both endpoints in $B^{[i]} \subseteq U$ and both endpoints of e_2 are farther away from $B^{[i]}$ than $\frac{\alpha^i \rho}{1-\alpha}$, meaning that neither of them will lie in U after termination.

Proof. By Definition 8, it holds that if $e = \{u, v\} \in E_{>0}$ is not safe after step i, we must, w.l.o.g. over the choice of u, v, have $\mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, u) < \frac{\alpha^i \rho}{1-\alpha}$ and $\{u, v\} \not\subseteq B^{[i]}$. By the approximation guarantee of the SSSP algorithm and the triangle inequality, we get

$$r^{[i]} < \max\{\widetilde{\mathrm{d}}^{[i]}(u), \widetilde{\mathrm{d}}^{[i]}(v)\} \le (1 + \alpha^2) \max\{\mathrm{d}_{G^{[i]}}(s^{[i]}, u), \mathrm{d}_{G^{[i]}}(s^{[i]}, v)\}$$
$$\le (1 + \alpha^2)(\mathrm{d}_{G^{[i]}}(s^{[i]}, u) + \ell_e),$$

where the first transition follows from property (I) in Lemma 6. From the former inequality, we get that

$$\mathrm{d}_{G^{[i]}}(s^{[i]}, u) \le \mathrm{d}_{G^{[i+1]}}(s^{[i+1]}, u) + r^{[i]} < \frac{\alpha^i \rho}{1 - \alpha} + r^{[i]}, \tag{1}$$

which yields $r^{[i]} \ge \mathrm{d}_{G^{[i]}}(s^{[i]}, u) - \frac{\alpha^i \rho}{1-\alpha}$. As $r^{[i]}$ is drawn uniformly from an interval of length $\alpha^{i-1}\rho$, these lower and upper bounds on $r^{[i]}$ readily imply a bound on the probability of $X_{i,e}$:

$$\Pr[X_{i,e}] \le \Pr\left[r^{[i]} \in \left(\mathrm{d}_{G^{[i]}}(s^{[i]}, u) - \frac{\alpha^i \rho}{1 - \alpha}, (1 + \alpha^2) \cdot (\mathrm{d}_{G^{[i]}}(s^{[i]}, u) + \ell_e)\right)\right]$$
$$\le \frac{(1 + \alpha^2)\ell_e}{\alpha^{i-1}\rho} + \frac{\alpha^2 \, \mathrm{d}_{G^{[i]}}(s^{[i]}, u)}{\alpha^{i-1}\rho} + \frac{\alpha}{1 - \alpha}. \tag{2}$$

Moreover, from (1) and $r^{[i]} \le \alpha^{i-1}\rho$, we conclude that $\mathrm{d}_{G^{[i]}}(s^{[i]}, u) < \alpha^{i-1}\rho \cdot (1 + \frac{\alpha}{1-\alpha}) = \frac{\alpha^{i-1}\rho}{1-\alpha}$. Plugging into (2), with $\alpha \le \frac{1}{2}$ we get that $\Pr[X_{i,e}] \le \frac{5}{4} \cdot \frac{\ell_e}{\alpha^{i-1}\rho} + \frac{\alpha^2}{1-\alpha} + \frac{\alpha}{1-\alpha} \le \frac{5}{4} \cdot \frac{\ell_e}{\alpha^{i-1}\rho} + \alpha(1 + 4\alpha)$. ◀

Applying this lemma to all iterations in which e has a significant probability to become safe (i.e., all iterations i for which $\alpha^{i-1}\rho \ge \ell_e$), we obtain the desired bound on the probability that e is cut.

▶ **Lemma 10.** *For every edge $e \in E$, it holds that* $\Pr[e \in E^{\mathrm{cut}}(U)] = O\left(\frac{\ell_e}{\rho}\right)$.

Proof. The case where $\ell_e = 0$ has already been treated, so assume hereafter that $e \in E_{>0}$. If $\ell_e > \rho$, then trivially $\Pr[e \in E^{\mathrm{cut}}(U)] \le 1 < \frac{\ell_e}{\rho}$. Otherwise, we let $i_e \ge 1$ be the largest index

such that $\ell_e \leq \alpha^{i_e-1}\rho$. By Lemma 9, for all i, the probability that an edge that is not safe after $i-1$ steps is still not safe after step i is bounded by $\Pr[X_{i,e}] \leq \frac{5}{4} \cdot \frac{\ell_e}{\alpha^{i-1}\rho} + \alpha \cdot (1 + 4\alpha)$. Depending on the index i, we differentiate this upper bound further:

- Case $i = i_e$: As $\alpha^{i_e}\rho < \ell_e$, we get that $\alpha < \frac{\ell_e}{\alpha^{i_e-1}\rho}$. With $\alpha \leq \frac{1}{2}$, $\Pr[X_{i_e,e}] < \frac{5\ell_e}{\alpha^{i_e-1}\rho}$ follows.
- Case $i = i_e - 1$: As $\ell_e \leq \alpha^{i_e-1}\rho$, we conclude that $\frac{\ell_e}{\alpha^{i-1}\rho} \leq \frac{\alpha^{i_e-1}}{\alpha^{i_e-2}} = \alpha$, yielding with $\alpha \leq \frac{1}{2}$ that $\Pr[X_{i_e-1,e}] < 5\alpha$.
- Case $i \leq i_e - 2$: This entails that $\frac{\ell_e}{\alpha^{i-1}\rho} \leq \alpha^2$ and thus $\Pr[X_{i,e}] < 2\alpha^2 + \alpha \cdot (1 + 4\alpha) = \alpha \cdot (1 + 6\alpha)$.

Using these bounds and distinguishing cases based on i_e, we can bound the overall probability that the edge is cut.

- Case $i_e = 1$: $\Pr[e \in E^{\mathrm{cut}}(U)] \leq \Pr[X_{1,e}] = \Pr[X_{i_e,e}] < \frac{5\ell_e}{\rho}$.
- Case $i_e = 2$: $\Pr[e \in E^{\mathrm{cut}}(U)] \leq \Pr[X_{2,e}] \cdot \Pr[X_{1,e}] = \Pr[X_{i_e,e}] \cdot \Pr[X_{i_e-1,e}] < \frac{5\ell_e}{\alpha\rho} \cdot 5\alpha = \frac{25\ell_e}{\rho}$.
- Case $i_e \geq 3$: $\Pr[e \in E^{\mathrm{cut}}(U)] \leq \frac{25\ell_e}{\alpha^{i_e-2}\rho} \cdot \prod_{i \leq i_e-2} \Pr[X_{i,e}] \leq \frac{25\ell_e}{\alpha^{i_e-2}\rho} \cdot (\alpha(1 + 6\alpha))^{i_e-2} < \frac{25\ell_e}{\rho} \cdot (1 + 6\alpha)^{i_e}$.

Hence, it remains to bound $(1 + 6\alpha)^{i_e} = O(1)$. Since $\rho < \ell_{\max}$, it follows that

$$\ell_{\min} \leq \ell_e \leq \alpha^{i_e-1}\rho < \alpha^{i_e-1}\ell_{\max}.$$

Recalling that $\sigma = \frac{\ell_{\min}}{\ell_{\max}} \leq n^{O(1)}$, we conclude that $i_e = O\left(\frac{\log n}{\log(1/\alpha)}\right)$. Therefore,

$$(1+6\alpha)^{i_e} = (1+6\alpha)^{O(\log n/\log(1/\alpha))} = \left((1 + 6\alpha)^{1/(6\alpha)}\right)^{O(\alpha \log n/\log(1/\alpha))} = e^{O(\alpha \log n/\log(1/\alpha))}.$$

The assertion follows by the choice of $\alpha = \frac{1}{2\log n}$. ◀

Theorem 5 now follows from Lemmas 7 and 10.

4 Tree-Supported Decomposition

In this section, we present the construction of TSDs that admit low diameter, low load, and low edge cutting probability, establishing Theorem 1. Our method is inspired by the partition technique from [43] that allows for efficient parallel and distributed implementations. However, we seek to rely on approximate rather than on exact distance computations.

To motivate our approach, consider a naive application of the decomposition technique from [43] using approximate rather than exact distance computations. This would look as follows: One would add a super-source s to the graph, assign exponentially sampled lengths to the edges adjacent to s, compute a $(1 + \varepsilon)$-approximate SSSP tree T rooted at s for some small enough ε, and partition the node set V according to the subtrees of T rooted at the children of s. This approach certainly leads to a decomposition of G. However, a consequence of the approximate distance computation is that the probability to cut a short edge is dominated by the approximation error, which is ε times the distance to the source — an expression that may be very large compared to the length of the edge.

In order to still ensure the desired bound, we seek to employ the blurring technique from the previous section to clusters obtained as described above. This introduces the new obstacle that the clusters need to be separated from each other first, as the blurring procedure grows the clusters by a random radius. We enforce this separation by removing from each cluster every node that is too close to its boundary; Property 2 of Theorem 5, stating that

the distance of any node in the blurred cluster from the original cluster is at at most $\frac{\rho}{1-\alpha}$, determines what precisely is "too close." While this may result in a large portion of the graph not being contained in any cluster even after blurring all clusters, we can ensure that each edge is contained in some cluster with probability at least $p = \Omega(1)$ (or is very long and can be safely deleted). Hence, repeating the procedure $O(\log n)$ times completes the decomposition w.h.p.

The blurring procedure presented in Section 3 requires that the aspect ratio $\sigma = \frac{\ell_{\max}}{\ell_{\min}}$ of its input graph is bounded by poly(n) and that the parameter ρ is smaller than $\ell_{\max}$. Therefore, we have to slightly modify the graph so that it satisfies these two conditions before invoking the blurring procedure. The latter requirement is readily satisfied by deleting all edges that are sufficiently long for the edge cutting probability bound to be greater than 1 (clearly, these edges can be safely deleted). For the former requirement, we reset the length of all edges $e \in E$ that are significantly shorter than the cluster diameter bound, thus ensuring that the aspect ratio of the graph is $n^{O(1)}$. The original length of all 0-length edges is then recovered after the decomposition is constructed. Since any simple path in the graph contains at most $n-1$ short edges, it follows that the length recovery operation does not increase the diameter of any cluster in the decomposition by "too much".

Algorithm 2: ts_decompose (G, Δ)

Input : graph $G = (V, E, \ell)$ and $\Delta \in \mathbb{N}$

Output : decomposition $D = (C_1, \ldots, C_k)$ of G, trees $\mathcal{T} = (T_1, \ldots, T_k)$ of depth $\leq \frac{\Delta}{2}$
$\qquad\qquad$ s.t. T_i spans a superset of C_i

1   $\beta := \frac{3c \log n}{\Delta}$, $\varepsilon := \frac{1}{c \log^2 n}$, $D := \emptyset$, $\mathcal{T} := \emptyset$ // *c sufficiently large constant*

2   delete all edges $e \in E$ of length $\ell_e > \frac{1}{40\beta}$ // *long edges*

3   reset the length of each edge $e \in E$ with $\ell_e < \frac{\Delta}{6n}$ by setting $\ell_e = 0$ // *short edges*

4   **while** $E(G) \neq \emptyset$ **do**

     // * *initial decomposition by exponential shifts* *

5   pick $\delta_u \sim \mathrm{Exp}_\beta$ for each $u \in V$ independently

6      $G_s :=$ super-source graph of G with edges $\{u, s\}$ of length
        $\ell_{us} = 1 + \max_{v \in V}\{\delta_v\} - \delta_u$ for $u \in V$

7      $T := (1+\varepsilon)$-approximate SSSP tree for G_s with source s

8      $R :=$ roots of $T \setminus \{s\}$ and $\mathcal{V} := (V_u)_{u \in R}$, where V_u are the nodes in u's subtree

     // * *separate cells* *

9      $\partial \mathcal{V} := \bigcup_{u \in R}\{v \in V_u \mid \exists\{v, w\} \in E \colon w \notin V_u\}$

10    $G'_s :=$ super-source graph of G with edges $\{u, s\}$ of length 1 for $u \in \partial\mathcal{V}$

11    $T' := (1+\varepsilon)$-approximate SSSP tree for G'_s with source s

12   **for** *each* $u \in R$ **do**

13       $V_u^\circ := V_u \setminus \{v \in V_u \mid d_{T'}(s, v) \leq \frac{1+\varepsilon}{4\beta}\}$
        // V_u° *is the interior of cell* V_u

14       $C_u := \mathtt{blur}(G, \rho, V_u^\circ)$, where $\rho := \frac{1 - \frac{1}{2 \log n}}{4\beta}$

15   append C_u to D and the subtree of T rooted at u to $\mathcal{T}$

16       $G := G \setminus C_u$

17   recover the original length of each edge $e \in E$ with $\ell_e = 0$

18   **return** $(D, \mathcal{T})$

▶ **Remark.** The operations of resetting the lengths of the short edges (line 3) and recovering their original lengths (line 17) are necessary only if the aspect ratio of G is large and can be ignored if the aspect ratio is guaranteed to be bounded by $n^{O(1)}$. In this case, the graphs handed to the blurring procedure (line 14) have no zero length edges.

Algorithm

The pseudocode of our procedure `ts_decompose` is given in Algorithm 2. The value β chosen in Line 1 is the parameter chosen for the exponential distributions: up to normalization, the density of the distribution is $\exp(-\beta x)$. The diameter of each (initial) cluster is bounded by $\max_{v \in V}\{\delta_v\}$, which we need to be smaller than $\frac{\Delta}{2}$ w.h.p. However, the probability to cut edges increases as we make the distributions "narrower," i.e., β larger. Accordingly, we choose $\beta = \Theta\left(\frac{\log n}{\Delta}\right)$, just small enough to ensure $\delta_v \leq \frac{\Delta}{2}$ w.h.p. for all $v \in V$.

The partition from [43] can be interpreted as a Voronoi decomposition in which each cell center x_v is a virtual copy of its corresponding node $v \in V$ that is attached to v by an edge of length $\max_{w \in V}\{\delta_w\} - \delta_v$. Note that the children of the virtual node s in the (approximate) shortest path tree T are exactly the nodes which have not been "absorbed" into another node's Voronoi cell before they started to grow their own. Lines 9 to 13 remove from each cluster nodes that are in distance (roughly) $\frac{1}{4\beta}$ from the boundary of the Voronoi cell containing them. Choosing a distance of $O\left(\frac{1}{\beta}\right)$ here ensures a constant probability that edges of this length remain in a shrunk cluster; longer edges can safely be cut, as the required bound on the probability for cutting them is trivial (i.e., 1), which is why they are removed at the start of the routine. We then proceed to applying the blurring subroutine to each (remaining) shrunk cluster. Note that, as the clusters remain separated due to the choice of parameters, we can realize this step concurrently for all clusters. The algorithm iterates until all nodes are assigned to clusters, which requires $O(\log n)$ loop iterations w.h.p.

The remainder of this section is dedicated to proving Theorem 1.

Number of Iterations

We first prove the key statement that, with at least constant probability, for any node w, a ball of radius $\Theta(\frac{1}{\beta})$ around it is contained within the interior of a cell.

▶ **Lemma 11.** *Consider an iteration of the while loop of Algorithm 2 and (by slight abuse of notation) denote by $G = (V, E)$ the subgraph that remains at the beginning of the iteration. For any $w \in V$, with at least constant probability a ball of radius $\frac{1}{40\beta}$ around it is contained in the interior of a cell computed in Line 13.*

Proof. For $x \in V$, set $d_x := d_{G_s}(x, w) + 1 + \max_{y \in V}\{\delta_y\}$. Moreover, set $X_x := d_x - \delta_x = \ell_{sx} + d_{G_s}(x, w)$ for $x \in V$ and let $X^{(i)}$ be the i'th order statistic of the variables X_v (i.e., the i'th smallest element). Denote by $x_{\min} \in V$ the node for which $X_{x_{\min}} = X^{(1)}$. By Lemma 4, with constant probability $X^{(2)} - X^{(1)} \geq \frac{7}{8\beta}$. Condition on this event. Accordingly, we have for all $x \in V \setminus \{x_{\min}\}$ that $X_x - X_{x_{\min}} \geq X^{(2)} - X^{(1)} \geq \frac{7}{8\beta}$.

Denote for each $v \in V$ by x_v the child of s in T in whose subtree v is situated. Then the

assumption that $x_v \neq x_{\min}$ implies by copious use of the triangle inequality that

$$
\begin{aligned}
\mathrm{d}_T(s, v) - \mathrm{d}_{G_s}(s, v) &= \ell_{x_v s} + \mathrm{d}_T(x_v, v) - \mathrm{d}_{G_s}(s, v) \\
&\geq \ell_{x_v s} + \mathrm{d}_{G_s}(x_v, v) - \mathrm{d}_{G_s}(s, v) \\
&\geq \ell_{x_v s} + \mathrm{d}_{G_s}(x_v, w) - \mathrm{d}_{G_s}(v, w) - (\mathrm{d}_{G_s}(s, w) + \mathrm{d}_{G_s}(v, w)) \\
&\geq \ell_{x_v s} + \mathrm{d}_{G_s}(x_v, w) - (\ell_{x_{\min} s} + \mathrm{d}_{G_s}(x_{\min}, w)) - 2\,\mathrm{d}_{G_s}(v, w) \\
&= X_{x_v} - X_{x_{\min}} - 2\,\mathrm{d}_{G_s}(v, w) \geq \frac{7}{8\beta} - 2\,\mathrm{d}_{G_s}(v, w).
\end{aligned}
$$

On the other hand, the approximation guarantee of the SSSP algorithm yields that

$$
\mathrm{d}_T(s, v) - \mathrm{d}_{G_s}(s, v) \leq \varepsilon\,\mathrm{d}_{G_s}(s, v) \leq \varepsilon \ell_{vs} \leq \varepsilon \max_{x \in V}\{1 + \delta_x\}.
$$

By Lemma 3, w.h.p. $\max_{x \in V}\{\delta_x\} \leq t = \frac{c \log n}{4(1+\varepsilon)\beta}$ after sampling the δ-values in Line 5 of this iteration. Condition on this event as well. Using that $\varepsilon = \frac{1}{c \log^2 n}$ and c is sufficiently large, we get that $\mathrm{d}_T(s, v) - \mathrm{d}_{G_s}(s, v) \leq \varepsilon(1 + t) < \frac{1}{4\beta}$.

In summary, if both events on which we conditioned occur, $x_v \neq x_{\min}$ entails that

$$
\mathrm{d}_{G_s}(v, w) > \frac{5}{16\beta}. \tag{3}
$$

In particular, choosing $v = w$ yields the contradiction $0 = \mathrm{d}_{G_s}(w, w) > \frac{5}{16\beta}$, i.e., $x_w = x_{\min}$.

We proceed to show that $\mathrm{d}_G(v, w) \leq \frac{1}{40\beta}$ implies that also $v \in V_{x_{\min}}^\circ$. By a union bound over the two events on which we conditioned, this will complete the proof. To this end, observe that Inequality (3) shows that a ball of radius $\frac{5}{16\beta}$ around w in G_s is contained within $V_{x_{\min}}$. Because longer edges have been deleted, nodes in $\partial \mathcal{V}$ are connected to neighbors outside their cell by edges of length at most $\frac{1}{40\beta}$. Together with the approximation guarantee of the second SSSP computation used to compute T', it follows that nodes $v \in V$ for which $\mathrm{d}_{G_s}(v, w) \leq \frac{1/16 - 1/40 - \varepsilon}{\beta} < \frac{5}{16\beta} - \frac{(1+\varepsilon)^2}{4\beta} - \frac{1}{40\beta}$ end up in $V_{x_{\min}}^\circ$. In particular, as trivially $\mathrm{d}_{G_s}(v, w) \leq \mathrm{d}_G(v, w)$ and ε is sufficiently small, we conclude that $\mathrm{d}_G(v, w) \leq \frac{1}{40\beta}$ implies that $v \in V_{x_{\min}}^\circ$. ◀

▶ **Corollary 12.** *Algorithm 2 terminates after $O(\log n)$ iterations of the while loop w.h.p.*

Proof. Consider any edge $e \in E$ that is not deleted right away, i.e., $\ell_e \leq \frac{1}{40\beta}$. By Lemma 11, in each iteration in which e is present in the remaining subgraph of G, there is a constant probability that it is contained in V_u° for some node u. Thus, the probability that the edge remains for $c \log n$ iterations is bounded by $2^{-\Omega(c \log n)} = n^{-\Omega(c)}$. By a union bound, this implies that all edges are either cut or included in a part within $O(\log n)$ iterations w.h.p., i.e., the termination condition that $E(G)$ is empty becomes satisfied. ◀

The Diameter Bound

In order to prove that the diameter bound holds, we first show that for each iteration of the while loop of Algorithm 2 and each $u \in R$, we have that $C_u \subseteq V_u$.

▶ **Lemma 13.** *Fix any iteration of the while loop of Algorithm 2 and $u \in R$. It holds that $C_u \subseteq V_u$.*

Proof. Again, denote for simplicity the remaining subgraph at the beginning of the loop iteration by $G = (V, E)$. By the approximation guarantee of the second call to the SSSP

algorithm, $v \in V_u^\circ$ implies that $\mathrm{d}(v, \partial \mathcal{V}) \geq \frac{1}{4\beta}$. By Theorem 5, $w \in C_u$ implies that $\mathrm{d}_G(w, V_u^\circ) \leq \frac{\rho}{1 - 1/(2 \log n)} = \frac{1}{4\beta}$. Consider the node $v \in V_u^\circ$ that is closest to w and fix a shortest path from v to w. By the second bound, the path is no longer than $\frac{1}{4\beta}$, which by the first bound implies that it cannot leave V_u. Hence, $w \in V_u$, showing the claim of the lemma. ◄

We observe that the above lemma yields that the algorithm indeed outputs a partition of V, and each set in the partition is spanned by the corresponding tree in $\mathcal{T}$. We now apply the tail bound on Exp_β given in Lemma 3 to infer that the diameter of the computed parts is appropriately bounded w.h.p.

▶ **Lemma 14.** *W.h.p., each cluster in the decomposition D returned by* `ts_decompose`(G, Δ) *has weak diameter at most $\frac{\Delta}{2}$. This is witnessed by the corresponding tree in $\mathcal{T}$.*

Proof. We prove that each tree $T \in \mathcal{T}$ has diameter at most $\frac{\Delta}{3}$ assuming that the lengths of the short edges e are reset to $\ell_e = 0$ (see line 3 of `ts_decompose`(G, Δ)). This implies the desired $\frac{\Delta}{2}$-bound on the diameter of T in the original graph since each path P in T includes less than n short edges and each such short edge adds at most a $\frac{\Delta}{6n}$-term to the total length of P when recovering its original length (line 17 of `ts_decompose`(G, Δ)).

By Lemma 3 and a union bound over all nodes, w.h.p. always $\max_{v \in V}\{\delta_v\} \leq 1 + t$ for $t = \frac{c \log n}{4(1+\varepsilon)\beta}$ in Line 5 of `ts_decompose`(G, Δ). Assume that v ends up in the subtree of T rooted at the child x_v of s in G_s. From the above bound, it follows that, w.h.p.,

$$\mathrm{d}_T(x_v, v) = \mathrm{d}_T(s, v) - \ell_{x_v s} \leq (1 + \varepsilon) \cdot \mathrm{d}_{G_s}(s, v) - \ell_{x_v s} \leq (1 + \varepsilon) \cdot \ell_{vs} - \ell_{x_v s}$$
$$= \varepsilon \cdot (1 + \max_{x \in V}\{\delta_x\} - \delta_v) + \delta_{x_v} - \delta_v \leq \varepsilon \cdot (1 + t) + t = (1 + \varepsilon) \cdot t + \varepsilon.$$

Recalling that R denotes the children of the root node in T, it follows that for each $x \in R$, we have that T_x has (weighted) depth at most $(1+\varepsilon)t + \varepsilon$ w.h.p. in Line 8 of `ts_decompose`(G, Δ). We conclude that w.h.p., for all $u \in R$, it holds that the subgraph induced by V_u has diameter at most $2[(1+\varepsilon)t + \varepsilon] \leq \frac{c \log n}{2\beta} + 2\varepsilon \leq \frac{\Delta}{3}$, using that $\varepsilon = \frac{1}{c \log^2 n} \leq \frac{1}{12}$ for sufficiently large c. Using Lemma 13 concludes the proof. ◄

The Edge Cutting Probability Bound

We proceed to showing that the probability to cut an edge is sufficiently small. This follows from the analysis of Algorithm 1 and the probabilistic progress guarantee from Lemma 11.

▶ **Corollary 15.** *The probability that edge $e \in E$ is cut by* `ts_decompose`(G, Δ) *is $O\left(\frac{\ell_e \log n}{\Delta}\right)$.*

Proof. Consider edge $e = \{v, w\} \in E$. If e is deleted right away, then $\ell_e > \frac{1}{40\beta} = \Omega\left(\frac{\Delta}{\log n}\right)$ and the claim trivially holds. Accordingly, assume that $\ell_e \leq \frac{1}{40\beta}$ in the following.

As shown in Lemma 13, in each iteration the parts $(V_u^\circ)_{u \in C}$ satisfy that $V_u^\circ \subseteq V_u$. Thus, if $v \in V_x$ and $w \in V_y$ for some $x, y \in C$ after Line 8, e can be only cut by v ending up in C_x, while w does not, or w ending up in C_y, while v does not. Lemma 10 shows that the probability for either event is bounded by $O\left(\frac{\ell_e \log n}{\Delta}\right)$, independently of the subgraph the calls to Algorithm 1 are executed on.

Combining this observation with the fact that, in each iteration in which e is still present by Lemma 11 it ends up in some part with probability at least $p \in \Omega(1)$, we can bound the probability that e is cut by

$$\sum_{i=1}^{\infty} (1 - p)^{i-1} O\left(\frac{\ell_e \log n}{\Delta}\right) = O\left(\frac{\ell_e \log n}{p\Delta}\right) = O\left(\frac{\ell_e \log n}{\Delta}\right). \quad ◄$$

The Load Bound

As the trees added to the output in a single iteration are subtrees of the same shortest path tree, these trees are disjoint. Hence, the bound on the number of iterations also bounds the number of trees in which an edge may participate and thus the load of that edge in the output decomposition D. This concludes the proof of Theorem 1.

5   Sampling from Low Stretch Tree Embeddings

Consider some graph $G = (V, E, \ell)$ with positive edge lengths. We say that graph $G' = (V', E', \ell')$ with $V' \supseteq V$ *dominates* G if $\mathrm{d}_{G'}(u, v) \geq \mathrm{d}_G(u, v)$ for every two vertices $u, v \in V$. In that case, we define the stretch of edge $e = \{u, v\} \in E$ in G' to be

$$\mathrm{str}_{G'}(e) \,=\, \frac{\mathrm{d}_{G'}(u, v)}{\ell_e} \,.$$

Our goal in this section is to construct random dominating trees of a given graph $G = (V, E, \ell)$ that guarantee low expected stretch for each edge in E. The dominating trees we construct, referred to hereafter as *virtual trees*, are not spanning trees of G, because they may include vertices and edges that do not belong to V and E, respectively. Nevertheless, they admit some useful characteristics. Specifically, we consider two types of virtual (dominating) trees: *projected trees* (a special case of the path embeddable trees of [15, 16]) addressed in Section 5.2 and *hierarchically well separated trees (HSTs)* addressed in Section 5.3. In both cases, the respective constructions are based on recursive applications of the graph decomposition technique presented in Section 4, generating a *hierarchical* version of TSDs as presented in Section 5.1.

5.1   Hierarchical Decompositions

A *hierarchical tree-supported decomposition (HTSD)* $\mathbf{D}$ of a graph G is a sequence $\mathbf{D} = (D_0, D_1, \ldots, D_k)$ of TSDs that satisfies (i) $D_0 = \{V\}$; (ii) $D_k = \{\{v\} \mid v \in V\}$; and (iii) for every $1 \leq i \leq k$ and $C \in D_i$, there exists some $C' \in D_{i-1}$ such that $C \subseteq C'$. The TSDs $D_0, D_1, \ldots, D_k$ are referred to as the *levels* of $\mathbf{D}$ and the parameter k is referred to as its *depth*. The *load* of edge $e \in E$ in $\mathbf{D}$ is defined to be $\mathrm{load}_{\mathbf{D}}(e) = \sum_{i=0}^k \mathrm{load}_{D_i}(e)$.

The real sequence $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ is said to be *diameter bounding* for the HTSD $\mathbf{D}$ if $\mathrm{diam}(D_i) \leq d_i$ for every $0 \leq i \leq k$. Of particular interest are HTSDs that admit a *geometrically decreasing* diameter bounding sequence, namely a sequence $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ that satisfies $d_i \leq \alpha \cdot d_{i-1}$, $1 \leq i \leq k$, for some constant $\alpha > 1$.

Consider some HTSD $\mathbf{D} = (D_0, D_1, \ldots, D_k)$ of G with a geometrically decreasing diameter bounding sequence $\mathbf{d} = (d_0, d_1, \ldots, d_k)$. Edge $e = \{u, v\} \in E$ is said to be *decoupled* on level $0 \leq i \leq k - 1$ if u and v belong to the same cluster in level i and to different clusters in level $i + 1$, that is $e \in E^{\mathrm{cut}}(D_{i+1}) - E^{\mathrm{cut}}(D_i)$. In that case, we define the *stretch* of e in $\mathbf{D}$ with respect to $\mathbf{d}$ to be

$$\mathrm{str}_{\mathbf{D},\mathbf{d}}(e) \,=\, \frac{d_i}{\ell_e} \,.$$

▶ **Theorem 16.** *There exists a (randomized) algorithm that, given a graph $G = (V, E, \ell)$ with positive integral edge lengths and aspect ratio $\sigma = \frac{\ell_{\max}}{\ell_{\min}}$, constructs a random HTSD $\mathbf{D}$ of G with the following guarantees: (1) the depth of $\mathbf{D}$ is $n^{O(1)}$ w.h.p.; (2) $\mathbf{D}$ admits a geometrically decreasing diameter bounding sequence $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ w.h.p.; (3) $\mathrm{load}_{\mathbf{D}}(e) = O(\log \sigma)$ for every edge $e \in E$ w.h.p.; and (4) $\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D},\mathbf{d}}(e)] = O(\log^2 n)$ for every edge $e \in E$.*

Proof. We first present the construction of the (random) HTSD $\mathbf{D} = (D_0, D_1, \ldots, D_k)$ and its geometrically decreasing diameter bounding sequence $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ and then establish their desired properties.

Let $D_0 = \{V\}$. Construct a 2-approximate SSSP tree T_0 of G and set $d_0 = \mathrm{diam}(T_0)$. Assume by induction that we have already constructed the TSD D_i, $i \geq 0$, with clusters $C_i^1, \ldots, C_i^q$ and corresponding supporting trees $T_i^1, \ldots, T_i^q$. Set

$$d_i = \max \left\{ \mathrm{diam}(T_i^j) \mid j = 1, \ldots, q \right\}$$

to be the maximum diameter of the trees supporting D_i's clusters. If $d_i = 0$, i.e., all clusters in D_i are singletons, then we set $k = i$ and the construction of $\mathbf{D}$ and $\mathbf{d}$ is completed. Otherwise, we call

$$\texttt{ts\_decompose} \left(G(C_i^j), \Delta_{i+1} \right), \quad j = 1, \ldots, q,$$

where $\Delta_{i+1} = \frac{d_i}{2}$, and take D_{i+1} to be the union of the clusters returned by these q calls.

To analyze this construction, we first observe that since $d_{i+1} \leq \Delta_{i+1} \leq d_i/2$ for every $0 \leq i \leq k - 1$, it follows that

$$k \leq O(\log \sigma).$$

Moreover, each d_i corresponds to the diameter of some approximate SSSP tree w.h.p., thus it provides a constant approximation for the actual distance in G between some pair of nodes. As there are $\binom{n}{2} < n^2$ such node pairs, we conclude that the depth of $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ satisfies

$$k \leq O\left(n^2\right)$$

w.h.p.

For the expected stretch bound, consider some edge $e \in E$ of length $\ell_e \in \mathbb{Z}_{>0}$. As long as $\texttt{ts\_decompose}$ is invoked with diameter bound $\Delta_i > 6n\ell_e$, the length of e is reset (see line 3 in Algorithm 2), ensuring that e is not decoupled on that level. For each level i such that $\Delta_i \leq 6n\ell_e$, Corollary 15 guarantees that the probability that e is decoupled on level i is at most

$$O\left(\frac{\ell_e \cdot \log(n)}{\Delta_i}\right) \leq O\left(\frac{\ell_e \cdot \log(n)}{d_i}\right).$$

Taking i_e to be the smallest $i \geq 0$ such that $\Delta_i \leq 6n\ell_e$ and i_e' to be the smallest i such that $\Delta_i \leq \ell_e \log n$, we can bound the expected stretch of e in $\mathbf{D}$ with respect to $\mathbf{d}$ as

$$\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D},\mathbf{d}}(e)] \leq \sum_{i=0}^{k-1} \Pr(e \text{ is decoupled on level } i) \cdot \frac{d_i}{\ell_e}$$

$$\leq \sum_{i=i_e}^{i_e'-1} O\left(\frac{\ell_e \cdot \log(n)}{d_i}\right) \cdot \frac{d_i}{\ell_e} + \sum_{i=i_e'}^{k-1} O(1) \cdot \frac{d_i}{\ell_e}$$

$$\leq (i_e' - i_e) \cdot O(\log n) + O(1),$$

where the last transition holds as the sequence $\mathbf{d}$ is geometrically decreasing. Since $\Delta_{i+1} \leq d_i/2 \leq \Delta_i/2$ for every i, it follows, by the definitions of i_e and i_e', that $i_e' - i_e \leq O(\log n)$, thus yielding the desired bound $\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D},\mathbf{d}}(e)] \leq O(\log^2 n)$.

It remains to show that the load of every edge $e \in E$ in $\mathbf{D}$ is $O(\log \sigma)$ w.h.p. To that end, recall that in Section 4 we proved that the load on edge e in the TSD D_i is stochastically dominated by a geometric random variable with parameter $\Omega(1)$. The claim follows recalling that the depth $k \le O(\log \sigma)$ as the sum of k such random variables is $O(k)$ w.h.p.   ◀

We note that a crucial point is, of course, that the algorithm can be implemented efficiently due to relying on approximate SSSP computations only. However, as the resulting complexities are model-specific, the respective discussion is postponed to Section 6.

5.2   Embedding into a Random Projected Tree

Consider some graph $G = (V, E, \ell)$. Graph $G' = (V', E', \ell')$ with $V' \supseteq V$ is said to be a *projected* graph of G if there exists a mapping $\pi : V' \to V$ so that
(a) $\pi(v) = v$ for every $v \in V$;
(b) if $e' = \{u', v'\} \in E'$, then $\pi(e') := \{\pi(u'), \pi(v')\} \in E$; and
(c) $\ell'(e') = \ell(e)$ for every $e \in E$ and $e' \in E'$ such that $\pi(e') = e$.
The *load* of edge $e \in E$ under the projected graph G' of G (with respect to π) is defined to be the size of its preimage under π, denoted by $\mathrm{load}_{G'}(e) = |\{e' \in E' \mid \pi(e') = e\}|$. Notice that, by definition, every projected graph of G dominates G. Observe also that ℓ' is fully determined by π and ℓ, hence we may omit it from the notation in the following. Our goal in this section is to prove the following theorem.

▶ **Theorem 17.** *There exists a (randomized) algorithm that, given a graph $G = (V, E, \ell)$ with* $\mathrm{poly}(n)$*-bounded edge lengths, constructs a random projected tree T of G that satisfies the following guarantees for every edge $e \in E$: (1) $\mathrm{load}_T(e) = O(\log \sigma)$ w.h.p.; and (2) $\mathbb{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$.*

Theorem 17 is established by combining Theorem 16 with the following lemma.

▶ **Lemma 18.** *There exists an algorithm that given a graph $G = (V, E, \ell)$, a HTSD $\mathbf{D}$ of G, and a geometrically decreasing diameter bounding sequence $\mathbf{d}$ for $\mathbf{D}$, constructs a projected tree $T = (V_T, E_T, \ell_T)$ of G such that $\mathrm{load}_T(e) = \mathrm{load}_\mathbf{D}(e)$ and $\mathrm{str}_T(e) = O(\mathrm{str}_{\mathbf{D},\mathbf{d}}(e))$ for each $e \in E$.*

The rest of Section 5.2 is dedicated to proving Lemma 18. This is done by a series of graph transformations that results in the desired projected tree T. Let k be the depth of $\mathbf{D} = (D_0, D_1, \ldots, D_k)$. For $0 \le i \le k$, let $H_i = (V_i^H, E_i^H)$ be the forest obtained by taking the (graph) union over all level i supporting trees of $\mathbf{D}$, where each level i supporting tree $T_C = (U_C, F_C)$ contributes its own (distinct) copies of the vertices in U_C and edges in F_C (this means, in particular, that $|V_i^H| = \sum_{C \in D_i} |U_C|$ and $|E_i^H| = \sum_{C \in D_i} |F_C|$). Define the function $\pi_i^H : V_i^H \to V$ by mapping each vertex $v \in V_i^H$ to the vertex $\pi_i^H(v) \in V$ from which it originates, recalling that T_C is a subgraph of G. Although the preimage of vertex $u \in V$ under π_i^H may consist of several vertices, it includes exactly one vertex $v_i \in U_C$, where C is the (unique) level i cluster that contains v. We hereafter refer to this vertex v_i as the level i *clone* of v.

Recalling that the level k clusters of $\mathbf{D}$ are singletons, we identify the vertices in V_k^H with their images under (the bijection) π_k^H so that $V_k^H = V$. Let $H = (V^H, E^H)$ be the forest obtained by taking the (graph) union over $H_0, H_1, \ldots, H_k$ and let $\pi^H : V^H \to V$ be the function defined by mapping each vertex $v \in V_i^H$, $0 \le i \le k$, to $\pi^H(v) = \pi_i^H(v)$. Notice that H is a projected graph of G realized by π^H and that $\mathrm{load}_H(e) = \mathrm{load}_\mathbf{D}(e)$ for every edge $e \in E$. It remains to show that we can turn H into a projected tree $T = (V_T, E_T)$ by

connecting its connected components without increasing the load on the edges while ensuring that the stretch of every edge $e \in E$ in T is at most $O(1)$ times larger than its stretch in $\mathbf{D}$ with respect to $\mathbf{d}$.

Given a level $0 \leq i \leq k$ and a level i cluster C, we refer to the vertex with smallest ID in C as the *leader* of cluster C, denoted by $\lambda(C)$. Notice that every vertex $v \in V$ is a leader of its level k cluster and that if v is the leader of its level i cluster, then it is also the leader of its level j cluster for all $i \leq j \leq k$.

We now construct a projected tree $T = (V^T, E^T)$ of G from H in two additional steps. First, we connect each connected component T_C of H_i, $1 \leq i \leq k$, to the unique connected component $T_{C'}$ of H_{i-1} that satisfies $C' \supseteq C$. Assuming that the leader of cluster C is $v = \lambda(C)$, this connection is realized by augmenting H with a 0-length edge that connects v_i with v_{i-1}, i.e., the level i and level $i-1$ clones of v. (Note that v_{i-1} is not necessarily the leader of cluster C'.) We call this new edge connecting v_i and v_{i-1} a *vertical* edge and denote the set of all vertical edges added to H during this step of the construction by $E_\uparrow$. Observe that the graph obtained from H by augmenting it with the vertical edges is a tree denoted hereafter by $T^\uparrow = (V^H, E^H \cup E_\uparrow)$. This holds since starting from the forest H, we connected each connected component in level $1 \leq i \leq k$ to a connected component in level $i-1$ using a single vertical edge and since H_0 is a tree.

The next and final step simply contracts all vertical edges in $T^\uparrow$, resulting in the tree $T = (V^T, E^T)$. Since the vertical edge $\{v_i, v_{i-1}\} \in E^\uparrow$ connects the clones v_i and v_{i-1} of the same vertex $v \in V$, it follows that both endpoints of the vertical edge are mapped to v under π^H. Accordingly, we readily obtain a projection $\pi^T \colon V^T \to V$ from π^H by mapping each vertex $v^T \in V^T$ to $\pi^H(v')$, where $v' \in V^H$ is any node that participated in the contraction that created v^T. Finally, note that there is a natural bijection $b \colon E^H \to E^T$ between edges in H and T, as T is obtained by first augmenting H with $E^\uparrow$ of vertical edges and then contracting these edges. By construction, we have $\pi^T(b(e)) = \pi^H(e)$ for all $e \in E^H$. In particular, T is indeed a projected tree of G and $\mathrm{load}_T(e) = \mathrm{load}_H(e) = \mathrm{load}_{\mathbf{D}}(e)$ for all $e \in E$.

It remains to prove that $\mathrm{str}_T(e) = O(\mathrm{str}_{\mathbf{D},\mathbf{d}}(e))$ for every edge $e = \{x, y\} \in E$. Since T is obtained from $T^\uparrow$ by contracting 0-length edges, it follows that $\mathrm{d}_{T^\uparrow}(x, y) = \mathrm{d}_T(x, y)$, hence it suffices to prove that $\mathrm{str}_{T^\uparrow}(e) = O(\mathrm{str}_{\mathbf{D},\mathbf{d}}(e))$. To this end, fix some node $v \in V$ and let $C_i \in D_i$, $0 \leq i \leq k$, be the (unique) level i cluster that contains v. Let $\lambda(i) = \lambda(C_i)$ be the leader of C_i and denote the level j clone of $\lambda(i)$ by $\lambda_j(i)$.

▶ **Remark 19.** For every $0 \leq i \leq k$, we have $\mathrm{d}_{T^\uparrow}(v, \lambda_i(i)) \leq \sum_{j=i}^{k-1} d_j$.

Proof. By induction on i. The base case $i = k$ holds since every vertex is the leader of its (singleton) level k cluster, hence $\lambda_i(i) = v$. For the inductive step from $i+1$ to $0 \leq i \leq k-1$, we notice that

$$\mathrm{d}_{T^\uparrow}(v, \lambda_i(i)) = \mathrm{d}_{T^\uparrow}(v, \lambda_{i+1}(i+1)) + \mathrm{d}_{T^\uparrow}(\lambda_{i+1}(i+1), \lambda_i(i+1)) + \mathrm{d}_{T^\uparrow}(\lambda_i(i+1), \lambda_i(i)).$$

Recalling that $\lambda_{i+1}(i+1)$ and $\lambda_i(i+1)$ are connected in $T^\uparrow$ by a vertical edge, we conclude that $\mathrm{d}_{T^\uparrow}(\lambda_{i+1}(i+1), \lambda_i(i+1)) = 0$. Moreover, since $\lambda_i(i+1)$ and $\lambda_i(i)$ belong to the same level i cluster $C_i \in D_i$, their distance in $T^\uparrow$ is equal to their distance in the supporting tree of C_i whose diameter is bounded by d_i, hence $\mathrm{d}_{T^\uparrow}(\lambda_i(i+1), \lambda_i(i)) \leq d_i$. The assertion follows by the inductive hypothesis ensuring that $\mathrm{d}_{T^\uparrow}(v, \lambda_{i+1}(i+1)) \leq \sum_{j=i+1}^{k-1} d_j$. ◀

Now, consider some edge $e = \{u, v\} \in E$ and let $0 \leq i \leq k-1$ be the level on which e is decoupled. Let $C \in D_i$ to be the level i cluster that contains u and v and let w be the

level i clone of the leader $\lambda(C)$ of C. Remark 19 guarantees that $\mathrm{d}_{T^\uparrow}(u, w) \leq \sum_{j=i}^{k-1} d_j$ and $\mathrm{d}_{T^\uparrow}(v, w) \leq \sum_{j=i}^{k-1} d_j$, hence

$$\mathrm{d}_{T^\uparrow}(u, v) \leq 2 \sum_{j=i}^{k-1} d_j = O(d_i),$$

where the last transition holds since $\mathbf{d} = (d_0, d_1, \ldots, d_k)$ is geometrically decreasing. The proof of Lemma 18 is completed by the definitions of $\mathrm{str}_{\mathbf{D}, \mathbf{d}}(e) = \frac{d_i}{\ell_e}$ and $\mathrm{str}_{T^\uparrow}(e) = \frac{\mathrm{d}_{T^\uparrow}(u, v)}{\ell_e}$.

5.3 Embedding into a Random HST

In this section we show how to construct an embedding into a random *hierarchically 2-separated* dominating tree (HST) with small expected stretch from the projected trees constructed in the previous section.

▶ **Definition 20** (Hierarchically Separated Trees). *An* embedding *of a weighted graph $G = (V, E, \ell)$ into a (rooted) tree $T = (V^T, E^T, \ell^T)$ is given by a one-to-one mapping $\iota \colon V \to V^T$. For $k > 1$, the tree is* hierarchically k-separated, *if for each internal non-root node, the weight of edges connecting it to its children is exactly by factor k smaller than the weight of the edge connecting it to its parent. The stretch of edge $e = \{u, v\} \in E$ w.r.t. T is defined as $\mathrm{str}_T(e) := \frac{\mathrm{d}_T(\iota(u), \iota(v))}{\ell_e}$.*

We note that our definition of hierarchical well-separation is (formally) weaker than that of hierarchically well-separated trees from the literature [7], as we dropped the requirement that the tree is balanced, i.e., all leaves are in the same depth. However, this can be easily achieved, and our construction does so without modification.

Construction

We construct our HST from a projected tree (see Section 5.2). The construction of $T = (V^T, E^T, \ell^T)$ is straightforward. Let $\mathbf{D} = (D_0, \ldots, D_k)$ be the HTSD from which the projected tree was constructed. We recall that we had assigned a leader $\lambda(C)$ to each cluster C, namely the smallest ID vertex in C. We construct V^T simply as the multiset[4] of leaders of all clusters in $\mathbf{D}$. Note that the nodes constructed for level k clusters, correspond, one-to-one, to the original nodes V of the graph. This enables us to define an embedding $\iota \colon V \to V^T$ as required in Definition 20. We construct the set of edges E^T as follows: Let $\lambda \in V^T$ be a node corresponding to an arbitrary level i cluster C with $i < k$. We introduce an edge $e := \{\lambda(C), \lambda(C')\}$, for every level $i + 1$ cluster C' that cluster C decomposes into, i.e., $C' \subseteq C$. We assign length $\ell_e^T := d_i$ to such an edge e between nodes corresponding to level i and level $i + 1$ clusters. Rooting the tree at the node in V^T corresponding to the leader of the (unique) level 0 cluster V, it is clear that the resulting tree $T = (V^T, E^T, \ell^T)$ is a hierarchically 2-separated tree of depth k.

Regarding distances, we get essentially the same result as for the projected tree we could have constructed. Denote for $v \in V$ by $\lambda(i)$ the leader of the unique level i cluster $C_i \in D_i$ such that $v \in C_i$ and denote by $\lambda^T(i) \in V^T$ its copy in T corresponding to C_i.

▶ Remark 21. For every $0 \leq i \leq k$, we have $\mathrm{d}_T(v, \lambda^T(i)) = \sum_{j=i}^{k-1} d_j$.

---

[4] For each cluster C a node $v \in V$ is leader of, there is a separate copy of v.

Proof. $\mathrm{d}_T(v, \lambda^T(i)) = \sum_{j=i}^{k-1} \ell_{\{\lambda^T(j), \lambda^T(j+1)\}}^T = \sum_{j=i}^{k-1} d_j$. ◄

▶ **Corollary 22.** *T is a dominating hierarchically 2-separated tree with $\mathrm{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$ for each edge $e \in E$.*

Proof. As discussed, T is hierarchically 2-separated by construction and we have the desired embedding $\iota \colon V \to V^T$. By Remarks 19 and 21, distances between leaves of T are at least as large as in the projected tree constructed in Section 5.2, which dominates G. The stretch bound follows analogous to Section 5.2, where Remark 19 takes the place of Remark 21. ◄

We remark that this establishes a straightforward relation between our projected trees and the HSTs constructed here. The HST edges are realized by the corresponding paths in the projected tree. In particular, while the HST may incur large loads on some graph edges, the "more fine-grained" view provided by the projected tree shows that a low-load mapping of paths in the HST to the original graph is feasible. On the other hand, this relation also demonstrates that a projected tree "behaves" like an HST due to the geometrically decreasing diameter bounding sequence of the underlying HTSD.

5.4 Bounding the p-Stretch

Cohen et al. [16] introduced the notion of p-stretch.

▶ **Definition 23** (p-Stretch). *For a graph G, an embedding of G into T, and a real $p \in (0, 1]$, the p-stretch of an edge $e = \{u, v\} \in E$ is given by $\left(\frac{\mathrm{d}_T(u,v)}{\ell_e}\right)^p$. Analogously, we define the p-stretch of an HTSD for edge e as $\left(\frac{d_i}{\ell_e}\right)^p$, where i is the level on which e is decoupled.*

Note that the 1-stretch coincides with the definition of the standard stretch defined at the beginning of this section. Our constructions meet a stronger bound of $O(\log n)$ on the p-stretch for $p < 1$, owed to the fact that for $p < 1$ larger stretch is weighed less.

▶ **Lemma 24.** *For $p \in (0, 1)$, the tree embeddings presented in Sections 5.2 and 5.3 satisfy that for each edge $e \in E$ the expected p-stretch is $O(\log n)$.*

Proof. When bounding the stretch in the proof of Theorem 16, we summed over all levels of the decomposition. Recall that the probability to decouple edge e on level i is, by Corollary 15, $O\left(\frac{\ell_e \cdot \log n}{d_i}\right)$. Denote by i_e the level such that $d_{i_e} \le \ell_e < 2d_{i_e}$. If $i > i_e$, then the stretch of e w.r.t. the HTSD is smaller than 1. For $p < 1$, the sum now can thus be bounded as

$$\sum_{i=1}^{k} O\left(\frac{\ell_e \cdot \log n}{d_i}\right) \cdot \left(\frac{d_i}{\ell_e}\right)^p = O\left(1 + \log n \cdot \left(\frac{\ell_e}{d_{i_e}}\right)^{1-p} \cdot \sum_{i=1}^{i_e} \left(\frac{d_{i_e}}{d_i}\right)^{1-p}\right)$$
$$= O\left(1 + \log n \cdot \sum_{i=1}^{i_e} \left(\frac{1}{2}\right)^{(1-p)(i_e-i)}\right)$$
$$= O(\log n),$$

where the final step exploits that the sum is a geometric series due to $1 - p > 0$. ◄

6 Implementation in Different Models

In this section, we describe how to implement the above techniques in the CONGEST, PRAM, and multipass streaming models. These should be considered as exemplary computational

models and it seems likely that our techniques transfer to other models in which a discrepancy between exact and approximate SSSP computations exist. Under the CONGEST model, some effort is needed in order to transfer the $(1+\varepsilon)$-approximate SSSP result to $(1+\varepsilon)$-approximate SSSP in super-source graphs (see Definition 2); this transformation is immediate under the PRAM and multipass streaming models.

For simplicity, we assume throughout this section that the edge lengths in the input graph $G = (V, E, \ell)$ are integers in the range $[1, n^{O(1)}]$. This means, in particular, that we do not have to worry about 0-length edges in the blurring procedure (Algorithm 1).

6.1 CONGEST **Model**

In the CONGEST model of computation [44], every node is a computing unit (of unlimited computational power) and is labeled by a unique $O(\log n)$-bit identifier. 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. We note that the restriction to polynomially bounded edge lengths implies that distances can be encoded using $O(\log n)$ bits.

At termination every node needs to know its part of the output. For the task of constructing the random TSD, this means that every node $v \in V$ knows (1) the ID of its own cluster's leader (i.e., the vertex with minimum ID, see Section 5.2); (2) the ID of the leader of cluster C if $v \in U_C$, that is, if v participates in the supporting tree T_C of cluster C; and (3) its incident edges in T_C for each supporting tree T_C in which v participates. For the task of constructing the random HTSD, v should hold that knowledge for every level of the hierarchy. As discussed in Section 5, this also provides the nodes with all what they need in order to reconstruct the resulting projected tree or HST.

In order to avoid confusion with the weighted diameter $\operatorname{diam}(G)$, in what follows, we use $\operatorname{hop}(G)$ to denote the "unweighted" diameter of G, also called the *hop diameter*.

The following corollary discusses how to compute $(1 + \varepsilon)$-approximate SSSP in a super-source graph H of a graph G in the CONGEST model. We assume that each node $v \in V$ initially knows which of its incident edges in G are in H, whether it is connected to s, and, if so, the length $\ell(\{s, v\})$.

▶ **Corollary 25** (of [12]). *Let* $\varepsilon = \frac{1}{\operatorname{polylog} n}$. *Then* $(1 + \varepsilon)$-*approximate SSSP in super-source graphs can be solved in* $\tilde{O}(\sqrt{n} + \operatorname{hop}(G))$ *rounds w.h.p. in the* CONGEST *model.*

Proof. The algorithm from [12] consists of three main steps:

1. Let S be a set composed of s and $\tilde{\Theta}(\sqrt{n})$ nodes sampled uniformly at random. Let each node $v \in S$ learn a $(1 + \frac{\varepsilon}{3})$-approximation to the minimum length of $\tilde{O}(\sqrt{n})$-hop paths to each sampled node $w \in S$ (if no such node exists, any result of at least $d(v, w)$ is fine, including ∞). For each finite value, nodes on a (unique) path in G learn about them being part of this path and the next node on it.

2. Simulate a broadcast congested clique[5] $(1 + \varepsilon/3)$-approximate SSSP algorithm on the (virtual) graph on S with edge lengths given by the result from the previous step (∞ means no edge).

---

[5] The broadcast congested clique is the special case of the Congest model restricted to complete graphs and, for each round, nodes sending the same message to each of their neighbors.

3. Run $\tilde{O}(\sqrt{n})$ iterations of single source Bellman-Ford on G, where the distance values of nodes in S are initialized to the distances obtained from the previous step.

Assuming w.l.o.g. that $\varepsilon \leq 1$, this yields $(1 + \varepsilon)$-approximate distances to s. As the first step yields suitable routing information and the result of the second (i.e., an approximate SSSP-tree on the virtual graph) is global knowledge, nodes can locally determine their parent in the output tree T.

We adapt the algorithm to super-source graphs as follows.

1. The first step is based on a pipelined version of the (multi-source) Bellman-Ford algorithm that also works on directed graphs [39, Corollary 5.8]. Formally, we orient all edges of s towards it (no other change is made). Then we can easily simulate the procedure on the resulting graph, as all communication by s over one of its edges can be inferred from its length (which is known to the recipient).

Note that the result is not exactly the same as that of the first step above: all paths containing s as non-starting node have been removed. However, the decisive property of the constructed graph is that it preserves G-distances to s up to a factor of $1 + \varepsilon$. The virtual graph also needs to be undirected, which is achieved by dropping the directionality of the computed distances.

2. The simulation of the broadcast congested clique algorithm in the Congest model is based on making all communication global knowledge. Using pipelining over a BFS tree, the input of s in the virtual graph (i.e., its incident edges and their lengths) can be made global knowledge in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds. Together, this implies that all nodes can locally simulate s.

3. Simulating the communication by s in the third step of the algorithm, which is a standard Bellman-Ford computation, is straightforward.

As all steps can be adjusted preserving the guarantees of the algorithm and the asymptotic running time is increased by additive $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ only, the result now follows from [12].

◀

This leads to the following result for Algorithm 1 from Section 3. As it is basically a sequence of approximate SSSP computations, a running time bound is immediate from Corollary 25.

▶ **Corollary 26.** *Suppose $\alpha = \frac{1}{\mathrm{polylog}\, n}$ and $\rho = n^{O(1)}$. Then Algorithm 1 can be executed in the* CONGEST *model in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds w.h.p.*

Proof. The while loop terminates after at most $\lceil \log_{1/\alpha} \rho = O(\log n) \rceil$ iterations. In each iteration, $r^{[i]}$ can be chosen by an arbitrary node (e.g. the one with lowest identifier) and broadcasted via a BFS tree in $O(\mathrm{hop}(G))$ rounds. Each node then can infer from the result from the previous iteration (or the input if $i = 1$) whether it is part of $B^{[i-1]}$. Nodes adjacent to $B^{[i-1]}$ can learn about this in one communication round and infer the length of the edge connecting them to $s^{[i]}$ in $G^{[i]}$. Thus, all that remains is the approximate SSSP computation, which can be performed in the stated running time by Corollary 25 w.h.p. The $\tilde{O}$-notation absorbs the $O(\log n)$-factor from the number of loop iterations. ◀

We turn to Algorithm 2 from Section 4. As each iteration can be performed within $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds w.h.p., this implies a bound on the running time of the overall algorithm.

▶ **Corollary 27.** *There exists a (randomized)* CONGEST *algorithm that given a graph $G = (V, E, \ell)$ with* $\mathrm{poly}(n)$*-bounded edge lengths and a real parameter $\Delta > 0$, constructs a random*

TSD D of G with the following guarantees in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds w.h.p.: (1) $\mathrm{diam}(D) \leq \Delta$ w.h.p.; (2) $\max_{e \in E}\{\mathrm{load}_D(e)\} \leq O(\log n)$ w.h.p.; and (3) $\Pr[e \in E^{\mathrm{cut}}(D)] \leq O\left(\frac{\ell_e \cdot \log n}{\Delta}\right)$ for every edge $e \in E$.

Proof. All computations in Algorithm 2 with the exception of the approximate SSSP computations and the call to `blur` are local. By Corollary 26, the stated running time bound follows for a single iteration of the while loop. Here we use that the instances of `blur` can be run in parallel by Lemma 13: As $C_u \subseteq V_u$, we can delete all edges which are not connecting two nodes within the same V_u for some u and then run a single $(1 + \varepsilon)$-approximate SSSP instance, where we identify the super-sources of all calls to `blur`. Therefore, Corollary 12 and a union time bound yield the claim. ◀

We now turn to the techniques from Section 5. As the recursive calls for each level of the decomposition hierarchy when computing an HTSD can be executed concurrently with a single call to the approximate SSSP subroutine, we obtain the following result.

▶ **Corollary 28.** *There exists a (randomized) CONGEST algorithm that, given a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths, constructs a random HTSD $\mathbf{D}$ of G with the following guarantees in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds w.h.p.: (1) the depth of $\mathbf{D}$ is $O(\log n)$; (2) $\mathbf{D}$ admits a (deterministic) geometrically decreasing diameter bounding sequence $\mathbf{d}$ w.h.p.; (3) $\mathrm{load}_{\mathbf{D}}(e) = O(\log n)$ for every edge $e \in E$ w.h.p.; and (4) $\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D}, \mathbf{d}}(e)] = O(\log^2 n)$ for every edge $e \in E$.*

Proof. For each of the $O(\log n)$ levels of the decomposition, the recursive SSSP calls for each of the clusters can be merged into a single one by identifying their super-sources. The claim hence follows from Theorem 16 and Corollary 27. ◀

From the hierarchical decomposition, we obtain embeddings into projected trees and hierarchically 2-separated trees as described in Sections 5.2 and 5.3.

▶ **Corollary 29.** *There exists a (randomized) CONGEST algorithm that, given a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths, constructs a random projected tree T of G in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds that satisfies the following guarantees for every edge $e \in E$: (1) $\mathrm{load}_T(e) = O(\log n)$ w.h.p.; and (2) $\mathbb{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$.*

Proof. We obtain an HTSD using Corollary 28. Inspection of the construction in Section 5.2 reveals that all operations are local once we identify the leaders of clusters. This is, e.g., achieved by rooting all supporting trees at the respective cluster's leader, which can be done by using the Garay-Kutten-Peleg minimum spanning tree algorithm [27, 38] to compute a spanning forest of H. As the load of each edge is $O(\log n)$, the algorithm on H can be simulated at a multiplicative overhead of $O(\log n)$, resulting in running time $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$. ◀

▶ **Corollary 30.** *There exists a (randomized) CONGEST algorithm that, given a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths, constructs an embedding into a random dominating hierarchically 2-separated tree T of G in $\tilde{O}(\sqrt{n} + \mathrm{hop}(G))$ rounds with expected stretch $\mathbb{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$ for each edge $e \in E$.*

Proof. Analogous to Corollary 29. ◀

6.2 PRAM **Model**

In the PRAM model, multiple processors share a random access memory to jointly solve a computational problem. Various contention models exist for concurrent access to the same memory cell by multiple processors, but are equivalent up to small (sub-logarithmic) factors in complexity, so we assume that there is no contention. Then we can view the computation as a DAG whose nodes represent elementary computational steps and edges dependencies. The input is represented by the sources of the DAG. The crucial complexity measures are *work*, the total size of the DAG (or, equivalently, the sequential complexity of the computation) and *depth,* the maximum length of a path in the DAG (or, equivalently, the time to complete the computation with an unbounded number of processors executing steps at unit speed).

We use a result on approximate SSSP computations due to Cohen, who introduced hop sets for this purpose. Following standard notation, we use $m := |E|$, where $G = (V, E, \ell)$ is the input graph.

▶ **Corollary 31** (of [14, 20]). *Let $\varepsilon_0 > 0$ be a constant and $\varepsilon = \frac{1}{\mathrm{polylog}\, n}$. Then $(1 + \varepsilon)$- approximate SSSP in super-source graphs can be solved in $O(m^{1+\varepsilon_0})$ work and* polylog n *time w.h.p.*

We remark that the assumption that the graph G is connected implies $m^{\varepsilon_0} \geq \Omega(n^{\varepsilon_0})$ and thus the term m^{ε_0} can absorb polylog n factors.

Following the same route as for the CONGEST model, we obtain a string of corollaries. As coordination between processes is easier in the PRAM model, in most cases the results are immediate.

▶ **Corollary 32.** *Suppose $\alpha = \frac{1}{\mathrm{polylog}\, n}$, $\rho = n^{O(1)}$, and ε_0 is a constant. Then Algorithm 1 can be executed in the* PRAM *model with depth* polylog n *and work $O(m^{1+\varepsilon_0})$ w.h.p.*

▶ **Corollary 33.** *Fix any constant $\varepsilon_0 > 0$. There exists a (randomized)* PRAM *algorithm of depth* polylog n *and work $O(m^{1+\varepsilon_0})$ that, for a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths and a real parameter $\Delta > 0$, constructs a random TSD D of G with the following guarantees: (1) $\mathrm{diam}(D) \leq \Delta$ w.h.p.; (2) $\max_{e \in E}\{\mathrm{load}_D(e)\} \leq O(\log n)$ w.h.p.; and (3) $\Pr[e \in E^{\mathrm{cut}}(D)] \leq O\left(\frac{\ell_e \cdot \log n}{\Delta}\right)$ for every edge $e \in E$.*

Combining this corollary with Theorem 16, we obtain the following result.

▶ **Corollary 34.** *Fix any constant $\varepsilon_0 > 0$. There exists a (randomized)* PRAM *algorithm of depth* polylog n *and work $O(m^{1+\varepsilon_0})$ that, for a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths, constructs a random HTSD $\mathbf{D}$ of G with the following guarantees w.h.p.: (1) the depth of $\mathbf{D}$ is $O(\log n)$; (2) $\mathbf{D}$ admits a (deterministic) geometrically decreasing diameter bounding sequence $\mathbf{d}$ w.h.p.; (3) $\mathrm{load}_{\mathbf{D}}(e) = O(\log n)$ for every edge $e \in E$ w.h.p.; and (4) $\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D}, \mathbf{d}}(e)] = O(\log^2 n)$ for every edge $e \in E$.*

▶ **Corollary 35.** *Fix any constant $\varepsilon_0 > 0$. There exists a (randomized)* PRAM *algorithm of depth* polylog n *and work $O(m^{1+\varepsilon_0})$ that, given a graph $G = (V, E, \ell)$ with $\mathrm{poly}(n)$-bounded edge lengths, constructs a random projected tree T of G that satisfies the following guarantees for every edge $e \in E$: (1) $\mathrm{load}_T(e) = O(\log n)$ w.h.p.; and (2) $\mathbb{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$.*

Proof. Again, the main step after obtaining an HTSD is to identify cluster leaders. This can be easily done by pointer jumping within the stated complexity bounds. ◀

▶ **Corollary 36.** *Fix any constant $\varepsilon_0 > 0$. There exists a (randomized)* PRAM *algorithm of depth* $\mathrm{polylog}\, n$ *and work* $O(m^{1+\varepsilon_0})$ *that, given a graph $G = (V, E, \ell)$ with* $\mathrm{poly}(n)$*-bounded edge lengths, constructs an embedding into a random dominating hierarchically 2-separated tree T of G with expected stretch $\mathbb{E}_T[\mathrm{str}_T(e)] = O(\log^2 n)$ for each edge $e \in E$.*

6.3   Semi-Streaming Model

In the *streaming model* [32, 41], the input graph is given as a stream of edges without repetitions. The performance of an algorithm is measured by the space it uses, whereby space is organized in memory words of $O(\log n)$ bits. In the *multipass streaming model*, the input is presented to the algorithm in several such passes, and the goal is to keep both the number of required passes and the space consumption small. For algorithms for graph problems, it is usual to assume arbitrary arrival order of the edges. The special case where the computational problem takes an n-vertex graph as input and the amount of memory is $\tilde{O}(n)$ is also known as the *semi-streaming model* [23]. All our results in this subsection are for this setting.

▶ **Corollary 37** (of [12])**.** *In the semi-streaming model, $(1 + \varepsilon)$-approximate SSSP in super-source graphs can be solved in* $\mathrm{polylog}\, n$ *passes w.h.p. for any $\varepsilon = \frac{1}{\mathrm{polylog}\, n}$.*

All computational steps that are not SSSP computations can be either directly executed in memory (because only graphs of size $\tilde{O}(n)$ are involved) or easily performed by storing $\mathrm{polylog}\, n$ words for each node and streaming once (e.g., finding cluster leaders). Thus, corollaries analogous to the CONGEST and PRAM models are immediate.

▶ **Corollary 38.** *Suppose $\alpha = \frac{1}{\mathrm{polylog}\, n}$, $\rho = n^{O(1)}$. Then Algorithm 1 can be executed in the semi-streaming model with* $\mathrm{polylog}\, n$ *passes w.h.p.*

▶ **Corollary 39.** *There exists a (randomized) semi-streaming algorithm that given a graph $G = (V, E, \ell)$ with* $\mathrm{poly}(n)$*-bounded edge lengths and a real parameter $\Delta > 0$, constructs a random TSD D of G with the following guarantees in* $\mathrm{polylog}\, n$ *passes w.h.p.: (1) $\mathrm{diam}(D) \leq \Delta$ w.h.p.; (2) $\max_{e \in E}\{\mathrm{load}_D(e)\} \leq O(\log n)$ w.h.p.; and (3) $\Pr\left[e \in E^{\mathrm{cut}}(D)\right] \leq O\left(\frac{\ell_e \cdot \log n}{\Delta}\right)$ for every edge $e \in E$.*

▶ **Corollary 40.** *There exists a (randomized) semi-streaming algorithm that given a graph $G = (V, E, \ell)$ with* $\mathrm{poly}(n)$*-bounded edge lengths, constructs a random HTSD $\mathbf{D}$ of G with the following guarantees in* $\mathrm{polylog}\, n$ *passes w.h.p.: (1) the depth of $\mathbf{D}$ is $O(\log n)$; (2) $\mathbf{D}$ admits a (deterministic) geometrically decreasing diameter bounding sequence $\mathbf{d}$ w.h.p.; (3) $\mathrm{load}_{\mathbf{D}}(e) = O(\log n)$ for every edge $e \in E$ w.h.p.; and (4) $\mathbb{E}_{\mathbf{D}}[\mathrm{str}_{\mathbf{D},\mathbf{d}}(e)] = O(\log^2 n)$ for every edge $e \in E$.*

Corollary 40 readily implies semi-streaming algorithms that construct low stretch projected trees and dominating hierarchically 2-separated trees in $\mathrm{polylog}\, n$ passes w.h.p. However, one can, in fact, obtain such constructions in a single pass by (i) partitioning the edges to ($O(\log n)$ many) length classes; (ii) using known techniques to construct a spanner with $O(n)$ edges and $O(\log n)$ stretch for each length class; and (iii) construct the desired low stretch trees for the graph obtained from the union of these spanners in an offline fashion, observing that this graph has $O(n \log n)$ edges.[6]

---

[6] We thank an anonymous ITCS 2020 reviewer for pointing out this simple alternative construction.

7 Related Work

Low diameter graph decompositions with small edge cutting probabilities (or with small weight) play a major role in many algorithmic applications. These include the construction of low stretch spanning trees [2, 3, 4, 11, 18] and low distortion probabilistic embeddings of metric spaces into hierarchically well-separated trees [7, 8, 9, 22], fast approximate solvers of symmetric diagonally dominant linear systems [15, 36, 37, 47], constructing graph spanners [42, 45], and spectral sparsification [33, 35]. The literature in this field being vast, we can only give an incomplete review of it. We first focus on related work in distributed and parallel models of computation, as these results are closest to ours, and then turn to the related work in the streaming model. Our discussion of the related work in the former models starts with reviewing the literature on low diameter graph decompositions, and then it turns to their applications, focusing on low average stretch spanning trees and tree embeddings.

Low Diameter Graph Decompositions

In the LOCAL and CONGEST models of distributed computation[7] low diameter graph decompositions for unweighted graphs, i.e., $G = (V, E, \mathbb{1})$, play a special role as they can be leveraged to design fast algorithms for a large class of problems. More precisely, the decomposition task is complete for a certain class of local problems [30], where a problem is called local if it does not require $\Omega(\mathrm{hop}(G))$ rounds of communication (recall the definition of the hop diameter $\mathrm{hop}(G)$ from Section 6.1). Here, $\mathrm{hop}(G)$ is of relevance even in problems where the input graph is weighted, as communication over large hop distances is an inherent obstacle to small running times in distributed algorithms.

Several distributed decomposition algorithms with round complexities of $\mathrm{polylog}\, n$ and small edge cut probabilities are known for the unweighted case [19, 40, 43].[8] However, the weighted setting considered in this work is fundamentally different. A lower bound of $\mathrm{hop}(G)$ is trivial, i.e., the task is not local: intuitively, decoupling hop distance from graph distance implies that finding close-by nodes may require communication over $\mathrm{hop}(G)$ hops. In the LOCAL model, this bound is trivially tight, as nodes can learn about the entire graph in $\mathrm{hop}(G)$ rounds. In the CONGEST model, a reduction from 2-party communication complexity shows a lower bound of $\Omega\left(\frac{\sqrt{n}}{\log n}\right)$ rounds for computing an (r, λ)-decomposition for any non-trivial values of r and λ. This lower bound even holds if $\mathrm{hop}(G) = O(\log n)$ [46].[9]

Miller et al. [43] show how to compute low diameter graph decomposition with small edge cutting probabilities in unweighted graphs in the PRAM model. Their approach relies on exact SSSP computations. Given the current discrepancy in the state of the art of exact and approximate SSSP in the PRAM model, it thus cannot lead to satisfying bounds in the weighted setting.

---

[7] See Section 6.1 for the formal definition of CONGEST. The LOCAL model is identical, except that it does not restrict message sizes to $O(\log n)$.

[8] Some works only care about the chromatic number of the graph resulting from contracting clusters. However, the cited works achieve this by cutting few edges only.

[9] A low-diameter decomposition can be used to determine whether or not there is a light s-t cut in the family of lower bound graphs from [46]; s and t end up in the same cluster if and only if there is no light cut between them, as otherwise their distance is large.

Low Stretch Spanning Trees

Nevertheless, there has been some work applying decompositions in the vein of Miller et al. in order to obtain low average stretch[10] spanning trees for weighted graphs. A construction by Alon et al. [4] reduces weighted graphs to unweighted (multi)graphs. As a result Blelloch et al. [13] were able to give an efficient PRAM construction of low stretch spanning trees based on the decomposition technique by Miller et al. As shown by Blelloch et al., computation of such trees is of use for efficient PRAM solvers for symmetric diagonally dominant linear systems. A similar connection was exploited by Ghaffari et al. [29], who transferred the approach of Blelloch et al. to the CONGEST model, obtaining a low average stretch tree construction that they leveraged for approximate maximum flow computations. A downside of the aforementioned approaches is that the construction by Alon et al. suffers from a poor average stretch of $2^{\Theta(\sqrt{\log n \log \log n})}$, resulting in respective overheads in work and depth resp. round complexity in the two models when applying the computed trees in further computations.

For the CONGEST model, Becker et al. [11] gave a construction of low-average stretch spanning trees that combines the decomposition technique of Miller et al. with the star decomposition technique of Elkin et al. [18]. This approach achieves polylog n average stretch. Again, the complexity of their approach is essentially determined by an exact SSSP computation. Thus, the resulting algorithm is round-optimal up to polylogarithmic factors in the unweighted case (i.e., the running time is $\text{hop}(G) \operatorname{polylog} n$), while essentially matching the round complexity of exact SSSP in the weighted case. Exact SSSP computation in the CONGEST model is still not too well understood, with the best upper bound of $\tilde{O}(\min\{\sqrt{n \operatorname{hop}(G)}, \sqrt{n} \operatorname{hop}(G)^{1/4} + n^{3/5} + \operatorname{hop}(G)\})$ [25] still being polynomially far from the $\tilde{\Omega}(\sqrt{n} + \operatorname{hop}(G))$ lower bound.

Tree Embeddings

We apply our decomposition technique in order to obtain a metric tree embedding, following the same route as Bartal [7], obtaining the same $O(\log^2 n)$ bound on the expected stretch (note that the bound in [7] holds for any edge lengths whereas in the current paper, we make the simplifying assumption that the ratio of the maximum to minimum edge length is $\operatorname{poly}(n)$). Bartal later improved this bound to $O(\log n \log \log n)$ [8] and subsequently to asymptotically optimal $O(\log n)$ [9]. Although we cannot readily apply the same techniques, Bartal's work suggests that future improvements to our stretch bound are feasible.

Fakcharoenphol et al. [22] achieved the $O(\log n)$ stretch bound earlier, following a different approach in which the graph is not (explicitly) decomposed. However, at its core the main idea is very similar: randomization is leveraged to keep the probability of "cutting" edges proportional to their length based on the subtractive form of the triangle inequality. Also here, PRAM and CONGEST algorithms have been developed that try to mitigate the bottleneck imposed by exact SSSP computations. In the CONGEST model, it is straightforward to implement the algorithm from [22] with a round complexity that is (up to a factor of $O(\log n)$) equal to the running time of the Bellman-Ford algorithm [34]. However, shortest paths may have hop length up to $n - 1$, resulting in a running time far from the $\tilde{\Omega}(\sqrt{n} + \operatorname{hop}(G))$ lower bound. Ghaffari and Lenzen broke down shortest paths by sampling a "skeleton" of $\tilde{\Theta}(\sqrt{n})$ nodes uniformly, computing a spanner (refer to the sequel of this section for the definition of a spanner) of a graph representing the induced metric, computing a tree embedding of this

---

[10] The ratio of distance in the tree to edge length, averaged over all edges.

spanner, and finally extending this embedding to one of the original graph with modified weights via a Bellman-Ford computation. This can be seen as distorting the original distance metric such that it becomes sufficiently simple to solve exact SSSP fast, resulting in a round complexity of $\tilde{O}(n^{0.5+\varepsilon} + \mathrm{hop}(G))$ for stretch $O(\varepsilon^{-1}\log n)$. In particular, by setting $\varepsilon = \frac{1}{\log n}$, the stretch and running time bounds match our results. However, Ghaffari and Lenzen do not guarantee bounded load. We also note that their approach is inherently limited to stretch $\Omega(\log^2 n)$ when requiring a running time bound within polylog n of the lower bound, as both spanners with a near-linear number of edges and metric tree embedding must incur $\Omega(\log n)$ stretch each.

Friedrichs and Lenzen [26] provide fast PRAM and CONGEST algorithms for tree embeddings with stretch $O(\log n)$. The main difference to [31] is the use of hop sets [14] to provide "shortcuts" for distance computation that are not present in the original graph. Again, distances are then distorted by metric embeddings such that exact distance computation by a Bellman-Ford style computation becomes efficient. This leads to a $2^{O(\sqrt{\log n})}(\sqrt{n} + \mathrm{hop}(G))$-round algorithm in CONGEST and a PRAM algorithm of depth polylog n and work $O(m^{1+\varepsilon})$ (for any fixed constant $\varepsilon > 0$), where m is the number of edges and $\Omega(m)$ a trivial lower bound on the work. While the stretch guarantee is better than in our case, it should be noted that also here fundamental barriers limit this technique: lower bounds on the size of hop sets due to Abboud et al. [1] imply that any hop-set based approach must incur running time resp. work overheads of $2^{\Omega(\sqrt{\log n})}$. Although in the PRAM model we suffer the same work overhead by relying on hop-sets for the currently best known approximate SSSP algorithms [14, 20], our result shows that one can trade the additional log-factor in stretch for a logarithmic load bound that the method of Friedrichs and Lenzen cannot guarantee.

Streaming Algorithms

To the best of our knowledge, constructions of low diameter decompositions with small edge cutting probabilities have not been addressed so far in the semi-streaming literature. A related graph theoretic object whose construction has been studied in the context of streaming algorithms is *spanners*. Similarly to low (average) spanning trees, spanners also provide a sparse distance preserving representation of the graph, only that they are not required to be trees. On the other hand, their notion of distance preservation is stronger in the sense that it is required to hold in the worst case, rather than on average. Specifically, a κ-spanner of graph $G = (V, E, \ell)$ is a spanning subgraph of G that guarantees a stretch bound of at most κ for every edge in E. One is typically interested in constructing κ-spanners with a small number of edges, where $O(n^{1+2/(\kappa+1)})$ edges is the asymptotically tight bound. Streaming constructions of sparse spanners exist only for unweighted graphs [10, 17, 23], as there, the distance computations are typically restricted to the sparse subgraph maintained by the algorithm. A related notion in unweighted graphs, which has also been studied in the streaming literature [21], is an (α, β)-spanner, where the distance between vertices $u, v \in V$ in the spanner is required to be at most $\alpha \cdot \mathrm{d}_G(u, v) + \beta$ for every $u, v \in V$.

──── **References** ──

1   Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM J. Comput.*, 47(6):2203–2236, 2018.

2   Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 781–790, 2008.

3   Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. In *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC*, pages 395–406, 2012.

4   Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995.

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

6   Baruch Awerbuch and David Peleg. Sparse partitions (extended abstract). In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 503–513, 1990.

7   Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS*, pages 184–193, 1996.

8   Yair Bartal. On approximating arbitrary metrices by tree metrics. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, STOC*, pages 161–168, 1998.

9   Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In *Algorithms - ESA 2004, 12th Annual European Symposium*, pages 89–97, 2004.

10   Surender Baswana. Streaming algorithm for graph spanners - single pass and constant processing time per edge. *Inf. Process. Lett.*, 106(3):110–114, 2008.

11   Ruben Becker, Yuval Emek, Mohsen Ghaffari, and Christoph Lenzen. Distributed algorithms for low stretch spanning trees. In *33rd International Symposium on Distributed Computing, DISC*, 2019. To appear.

12   Ruben Becker, Andreas Karrenbauer, Sebastian Krinninger, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. In *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, pages 7:1–7:16, 2017.

13   Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory of Computing Systems*, 55(3):521–554, 2014.

14   Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000.

15   Michael B. Cohen, Rasmus Kyng, Gary L. Miller, Jakub W. Pachocki, Richard Peng, Anup B. Rao, and Shen Chen Xu. Solving SDD linear systems in nearly $m\log^{1/2}n$ time. In *Symposium on Theory of Computing, STOC*, pages 343–352, 2014.

16   Michael B. Cohen, Gary L. Miller, Jakub W. Pachocki, Richard Peng, and Shen Chen Xu. Stretching stretch. *CoRR*, abs/1401.2454, 2014.

17   Michael Elkin. Streaming and fully dynamic centralized algorithms for constructing and maintaining sparse spanners. *ACM Trans. Algorithms*, 7(2):20:1–20:17, 2011.

18   Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008.

19   Michael Elkin and Ofer Neiman. Distributed strong diameter network decomposition: Extended abstract. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 211–216, 2016.

20   Michael Elkin and Ofer Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 128–137, 2016.

21   Michael Elkin and Jian Zhang. Efficient algorithms for constructing (1+epsilon, beta)-spanners in the distributed and streaming models. *Distributed Computing*, 18(5):375–385, 2006.

22   Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. A tight bound on approximating arbitrary metrics by tree metrics. *J. Comput. Syst. Sci.*, 69(3):485–497, 2004.

23   Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

24 Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019.*, pages 377–388, 2019.

25 Sebastian Forster and Danupon Nanongkai. A faster distributed single-source shortest paths algorithm. In *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 686–697, 2018.

26 Stephan Friedrichs and Christoph Lenzen. Parallel metric tree embedding based on an algebraic view on moore-bellman-ford. *J. ACM*, 65(6):43:1–43:55, 2018.

27 Juan A Garay, Shay Kutten, and David Peleg. A sublinear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

28 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks ii: Low-congestion shortcuts, mst, and min-cut. In *Proceedings of the Twenty-seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 202–219, 2016.

29 Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. *SIAM Journal on Computing*, 47(6):2078–2117, 2018.

30 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the Complexity of Local Distributed Graph Problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 784–797, 2017.

31 Mohsen Ghaffari and Christoph Lenzen. Near-optimal distributed tree embedding. In *Distributed Computing - 28th International Symposium, DISC*, pages 197–211, 2014.

32 Monika Rauch Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. In *External Memory Algorithms, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, May 20-22, 1998*, pages 107–118, 1998.

33 Michael Kapralov and Rina Panigrahy. Spectral sparsification via random spanners. In *Innovations in Theoretical Computer Science 2012, Cambridge, MA, USA, January 8-10, 2012*, pages 393–398, 2012.

34 Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012.

35 Ioannis Koutis. Simple parallel and distributed algorithms for spectral graph sparsification. In *26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, Prague, Czech Republic - June 23 - 25, 2014*, pages 61–66, 2014.

36 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 590–598, 2011.

37 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD linear systems. *SIAM J. Comput.*, 43(1):337–354, 2014.

38 Shay Kutten and David Peleg. Fast Distributed Construction of Smallk-Dominating Sets and Applications. *Journal of Algorithms*, 28(1):40–66, 1998.

39 Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Computing*, 2018.

40 Nathan Linial and Michael Saks. Low diameter graph decompositions. *Combinatorica*, 13(4):441–454, 1993.

41 Andrew McGregor. Graph stream algorithms: a survey. *SIGMOD Record*, 43(1):9–20, 2014.

42 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proceedings of the 27th ACM on Symposium on Parallelism in Algorithms and Architectures, SPAA 2015, Portland, OR, USA, June 13-15, 2015*, pages 192–201, 2015.

43 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 196–203, 2013.

44   D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Discrete Mathematics and
      Applications. Society for Industrial and Applied Mathematics, 2000.

45   David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–
      116, 1989.

46   Atish Das Sarma, Stephan Holzer, Liah Kor, Amos Korman, Danupon Nanongkai, Gopal
      Pandurangan, David Peleg, and Roger Wattenhofer. Distributed verification and hardness of
      distributed approximation. *SIAM J. Comput.*, 41(5):1235–1265, 2012.

47   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 Annual ACM
      Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 81–90, 2004.