Distributed and Parallel Low-Diameter Decompositions for Arbitrary and Restricted Graphs

Jinfeng Dou
Paderborn University, Germany

Thorsten Götte
University of Hamburg, Germany

Henning Hillebrandt
Paderborn University, Germany

Christian Scheideler
Paderborn University, Germany

Julian Werthmann
Paderborn University, Germany

Abstract

We consider the distributed and parallel construction of low-diameter decompositions with strong diameter. We present algorithms for arbitrary undirected, weighted graphs and also for undirected, weighted graphs that can be separated through $k \in \tilde{O}(1)$ shortest paths. This class of graphs includes planar graphs, graphs of bounded treewidth, and graphs that exclude a fixed minor K_r. Our algorithms work in the PRAM, CONGEST, and the novel HYBRID communication model and are competitive in all relevant parameters. Given $\mathcal{D} > 0$, our low-diameter decomposition algorithm divides the graph into connected clusters of strong diameter $\mathcal{D}$. For an arbitrary graph, an edge $e \in E$ of length ℓ_e is cut between two clusters with probability $O(\frac{\ell_e \cdot \log(n)}{\mathcal{D}})$. If the graph can be separated by $k \in \tilde{O}(1)$ paths, the probability improves to $O(\frac{\ell_e \cdot \log\log n}{\mathcal{D}})$. In either case, the decompositions can be computed in $\tilde{O}(1)$[1] depth and $\tilde{O}(m)$ work in the PRAM and $\tilde{O}(1)$ time in the HYBRID model. In CONGEST, the runtimes are $\tilde{O}(\mathrm{HD} + \sqrt{n})$ and $\tilde{O}(\mathrm{HD})$ respectively. All these results hold w.h.p.[2]

Broadly speaking, we present distributed and parallel implementations of sequential divide-and-conquer algorithms where we replace exact shortest paths with approximate shortest paths. In contrast to exact paths, these can be efficiently computed in the distributed and parallel setting [STOC '22]. Further, and perhaps more importantly, we show that instead of explicitly computing vertex-separators to enable efficient parallelization of these algorithms, it suffices to sample a few random paths of bounded length and the nodes close to them. Thereby, we do not require complex embeddings whose implementation is unknown in the distributed and parallel setting.

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

Keywords and phrases Distributed Graph Algorithms, Network Decomposition, Excluded Minor

Digital Object Identifier 10.4230/LIPIcs.ITCS.2025.45

Related Version *Full Version*: `https://arxiv.org/abs/2411.19859` [24]

Funding J. Dou acknowledges the support by the China Scholarship Council under grant number 202006220268. T. Götte acknowledges the support by DFG grant 491453517.

[1] $\tilde{O}(\cdot)$ hides polylogarithmic factors in n.
[2] We say an event holds w.h.p. if it occurs with probability $1 - o(n^c)$ for some $c > 1$.

1 Introduction

This paper considers the efficient construction of so-called (probabilistic) low-diameter decompositions (LDD) for general and restricted graphs in the CONGEST, PRAM, and HYBRID model. An LDD of a (weighted) graph $G := (V, E, \ell)$ with strong diameter $\mathcal{D}$ is a partition of G into disjoint connected subgraphs $\mathcal{K}(G) := K_1, \dots, K_N$ with $K_i := (V_i, E_i)$. We will refer to these subgraphs as *clusters*. We say that a cluster K_i has *strong* diameter $\mathcal{D}$ if it has a diameter of at most $\mathcal{D}$. That is, between two nodes in K_i there is a path of length (at most) $\mathcal{D}$ that only consists of nodes in K_i. In contrast, an LDD with a weak diameter creates possibly disconnected subgraphs with a diameter of $\mathcal{D}$ with respect to the original graph G. Here, for all nodes $v, w \in K_i$, there is a path of length (at most) $\mathcal{D}$ from v to w that can use nodes outside of K_i. We measure the decomposition quality by the number of edges that are *cut* between clusters. An edge $\{v, w\} \in E$ is cut if its two endpoints v and w are assigned to different clusters. Thus, a good clustering algorithm only ensures that *most* nodes are in the same cluster as their neighbors. Without this constraint, a trivial algorithm could simply remove all edges from the graph and return clusters containing one node each. There are several ways to count the edges that are cut. In this work, we consider so-called *probabilistic* decompositions. Here, the probability for an edge $z = \{v, w\}$ to be cut between two clusters K_i and K_j with $i \neq j$ that contain its respective endpoint depends on its length ℓ_z. This means that *short* edges are cut less likely than *long* edges. We use the following formal definition of LDD's in the remainder of this work:

▶ **Definition 1** (Probabilistic Low-diameter Decomposition (LDD)). *Let $G := (V, E, \ell)$ be a weighted graph and $\mathcal{D} > 0$ be a distance parameter. Then, an algorithm for computing an LDD with quality $\alpha \geq 1$ creates a series of disjoint clusters $\mathcal{K} := K_1, K_2, \dots, K_N$ with $K_i := (V_i, E_i)$ where $V_1 \sqcup \dots \sqcup V_N = V$. Futher, it holds:*

1. *Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $\mathcal{D}$.*
2. *Each edge $z := \{v, w\} \in E$ of length ℓ_z is cut between clusters with probability $O\left(\frac{\ell_z \cdot \alpha}{\mathcal{D}}\right)$.*

If we do not require each node to be in a cluster, we use the related notion of a Low-Diameter *Clustering* (LDC). In a LDC with quality (α, β) and strong diameter $\mathcal{D}$ we create a series of disjoint clusters $\mathcal{K} := K_1, K_2, \dots, K_N$ with $K_i := (V_i, E_i)$. Each cluster $K_i \in \mathcal{K}$ has a strong diameter of at most $\mathcal{D}$ and each edge $z := \{v, w\} \in E$ of length ℓ_z is cut between clusters with probability (at most) $O\left(\frac{\ell_z \cdot \alpha}{\mathcal{D}}\right)$. Further, each node is part of a cluster with probability at least β. As we will see, LDD's and LDC's are nearly equivalent as we can build an LDD from an LDC by repeatedly applying it until all nodes are clustered.

It is known that the best possible value of α is $O(\log n)$ for general graphs. However, if the graph class is restricted, better decompositions are possible. So, alongside solutions for general graphs, this work will consider the restricted class of so-called k-path separable graphs that Abraham and Gavoille introduced in [2]. Roughly speaking, a weighted graph $G = (G, E, \ell)$ is k-path separable if the *recursive* removal of k shortest paths results in connected components containing a constant fraction of the nodes. Moreover, a graph $G = (G, E)$ is *universally* k-path separable if it is k-path separable for *every* weight function ℓ. Note that this makes universal k-path separability a topological property rather than one that depends on the weight function. Abraham and Gavoille and Diot and Gavoille formalized this as follows:

▶ **Definition 2** (Universally k-Path Separarable Graphs [2, 22]). *A weighted graph $G := (V, E, \ell)$ with n vertices is* k-path separable*, if there exists a subset of vertices S, called a k-path separator, such that:*

1. $S := \mathcal{P}_0 \cup \mathcal{P}_1 \cup \mathcal{P}_2 \cup \ldots$, *where each* $\mathcal{P}_i := \{P_{i_1}, P_{i_2}, \ldots\}$ *is a set of shortest paths in* $G \setminus \bigcup_{j<i} \mathcal{P}_j$.
2. *The total number of paths in* S *is at most* k, *i.e.,* $\sum_{\mathcal{P}_i \in S} |\mathcal{P}_i| \leq k$.
3. *Each connected component in* $G \setminus S$ *contains at most* $\frac{n}{2}$ *nodes.*
4. *Either* $G \setminus S$ *is empty or* k*-path separable.*

If G *is* k*-path separable for any weight function, we call* G *universally* k*-path separable. We will sometimes abuse notation and use* $P_{i_j} \in S$ *when we refer to some path* $P_{i_j} \in \mathcal{P}_i$ *and* $\mathcal{P}_i \in S$.

In this work, we will only consider universally k-path separable graphs. Given this definition, a natural question is which graph classes are k-path separable. Clearly, all graphs that have separators that consist of at most η vertices are trivially universally η-path separable. This follows because every node can be seen as the shortest path to itself in any subgraph that it is contained in. Notably, graphs of bounded tree-width τ have separators that consist of τ nodes [53] and are therefore universally τ-path separable. Further, due to Thorup, it is known that planar graphs are universally 3-path separable [58]. More generally, Abraham and Gavoille [2] showed that every graph $G := (V, E, \ell)$ that does not include a fixed clique minor K_r is universally k-path separable where $k := f(r)$ depends only on r and not the size of G. The concrete value of $f(r)$ is based on the Structure Theorem by Robertson and Seymour [54].

These restricted graph classes have received significant attention in the context of distributed algorithms (cf. [35, 36, 37, 39, 42]). Not only do they contain graph topologies that frequently appear in practice, they also have favorable properties for distributed algorithms as they exclude pathologic worst case topologies. As k-path separable graphs are a superclass of these restricted graphs, we derive novel results for planar graphs, graphs of bounded treewidth, and most importantly, graphs that exclude a fixed minor K_r.

In a nutshell, prior works [13] and [55] have shown that computing LDD's for general graphs can be broken down into $\tilde{O}(1)$ applications of your favorite approximate shortest path algorithm in your favorite model of computation. We extend this innovative insight in two ways by improving the algorithm of [13] and showing that a) we can construct a strong diameter LDD of quality $O(\log n)$ for general graphs and b) on k-path separable graphs one can compute LDD's of quality $O(\log k + \log\log n)$. Again, our algorithms use only approximate shortest paths and are therefore applicable to various models.

Due to its many technical results, this paper takes the form of an extended abstract. In the main part of the paper, we present our algorithms, state our main lemmas, and sketch their proofs. More detailed descriptions and the full analysis, including all proofs, can be found in the full version on arXiv [24]. In the remainder of this section, we will first introduce some notations and convention in Subsection 1.1, then introduce our generalized model of computation in Subsection 1.2, and and state our main results in Subsection 1.3. Finally, we present some important applications for LDD's in Subsection 1.4 and discuss further related work (with a special focus on CONGEST algorithms) in Subsection 1.5.

1.1 Preliminaries and Notations

In this work, we only consider *undirected* graphs $G = (V, E)$ unless specifically mentioned otherwise. We further assume some basic familiarity with graph theory. For an introduction, we refer to standard textbooks, e.g., [21]. Throughout this work, we denote the distance between a node $v \in V$ and a set $S \subset V$ as $d(v, S)$. This distance is defined as the (weighted) length of the shortest path between v and the node $w \in S$ closest to v. If we consider the distance w.r.t. to a subgraph $H \subset G$, we write $d_H(\cdot, \cdot)$. Further for a subset $S \subseteq V$, a

subgraph $H \subseteq G$, and a distance δ, we define $B_H(S, \delta) = \{v \in V \mid d_H(v, S) \leq \delta\}$ to be the so-called *ball* that contains all nodes in distance (at most) δ to any node in G. Again, for $H = G$, we omit the subscript.

1.2 Model(s)

In this work, we present algorithms that can be implemented in several different models of computation for parallel and distributed systems by reducing our algorithms to (approximate) shortest-path computations. Notably, we can derive algorithms for the CONGEST and the PRAM models, which are the de facto standard models for distributed and parallel computing, respectively.

PRAM: In the PRAM model, we assume computations are executed on a machine with p processors. The processes work over a set of shared memory cells M. The input graph G is stored in these cells. In a single step of computation, they can read and write from each cell in an atomic operation. If more than one processor writes in a cell, an arbitrary processor succeeds, i.e., we consider a CRCW PRAM. The *work* of a PRAM algorithm is the total number of primitive read or write operations that the processors perform. Further, the *depth* is the length of the longest series of operations that have to be performed sequentially.

CONGEST: In the CONGEST model [52], we consider a static graph $G = (V, E)$. Each node has a unique identifier. We assume these identifiers consist of $O(\log n)$ bits, i.e., they are picked from interval $[1, \dots, n^c]$ for some $c \in \Theta(1)$. Time proceeds in synchronous rounds. In each round, nodes receive *all* messages sent in the previous round; they perform (unlimited) computation based on their internal states and the messages they have received so far. Finally, a node $v \in V$ may send a *distinct* message of size $O(\log n)$ to each neighbor in G. Thus, a message may contain a node's identifier and a few more bits of information.

HYBRID: In addition to these two *classic* models of computation, we will also use the recently established HYBRID model. The HYBRID model was introduced in [7] as a means to study distributed systems that leverage multiple communication modes. Just as in CONGEST, we assume synchronous rounds. In the HYBRID model, a node can send a message to all its neighbors in G in the so-called *local* mode and also $O(\log n)$ nodes of its choice (as long as no node receives more than $O(\log n)$ messages) in the so-called *global* mode. One typically parameterizes the size of the messages in the local mode. For a discussion of this model, we refer the interested reader to [7, 57].

Rather than exploiting these models' intricacies in bespoke algorithms, we reduce our algorithms to $\tilde{O}(1)$ applications of a $(1+\epsilon)$-approximate shortest path algorithm with $\epsilon \in O\left(1/\log^2 n\right)$ and some simple aggregations on subsets of the input graph. To be precise, our algorithms are based on approximate set-source shortest paths (SetSSP) and so-called minor aggregations. In the remainder of this section, we will clarify what exactly these operations do and how efficiently they can be implemented in the CONGEST, the PRAM, and the HYBRID model.

First, we formalize the class of approximate SetSSP algorithms. For our algorithms, we must be able to compute the shortest paths to a subset of nodes and also to virtual nodes that may not be part of the original input graphs. We define this problem as follows:

▶ **Definition 3** (Approximate SetSSP with Virtual Nodes). *Let $G := (V, E)$ be a weighted graph and let $G' := (V', E', w)$ with $V' := V \cup \{s_1, s_2, \dots\}$ be the graph that results from adding $\tilde{O}(1)$ virtual nodes to G. Each virtual node can have an edge of polynomially bounded weight to every virtual and non-virtual node in G'. Finally, let $S \subset V'$ be an arbitrary subset of virtual and non-virtual nodes. Then, an algorithm that solves $(1+\epsilon)$-approximate set-source shortest path with virtual nodes computes the following:*

- *Each node $v \in V' \setminus S$ learns a predecessor $p_v \in N(v)$ on a path of length at most $(1+\epsilon)d(v,S)$ to some node in S and marks the edge $\{v, p_v\}$. Together, all the marked edges imply an approximate shortest path tree T[3] rooted in set S.*
- *Each node $v \in V'$ learns its distance $d_T(v,S) \leq (1+\epsilon)d(v,S)$ to S in tree T, i.e., its exact distance to S in T and its $(1+\epsilon)$-approximate distance to S in G'.*

Note that, if we can compute SetSSP on a graph G in τ time/depth, we can also compute it on any set of disjoint subgraphs of G in τ time/depth. To be precise, let $C_1, \dots, C_N$ with $C_i = (V_i, E_i)$ be a set of disjoint subgraphs of G and let $S_1, \dots, S_N$ with $S_i \subset V_i$ be subsets of nodes from each C_i. Suppose, we want to compute an SetSSP from each S_i in C_i. To this end, we simply assign a prohibitively high length to an edge between two subgraphs $C_i \neq C_j$. Then, we compute a SetSSP from $\mathcal{S} = \{S_1, \dots, S_N\}$ that results in a tree T. Due to their length, no (approximate) shortest path will ever contain an edge between subgraphs and so, the closest node to any node in C_i must be from S_i. Thus, the tree restricted to C_i is an approximate shortest path tree for S_i.

Our second building block are so-called *minor aggregations*, which were first introduced in [35, 36]. Consider a network $G = (V, E)$ and a (possibly adversarial) partition of vertices into disjoint subsets $V_1, V_2, \dots, V_N \subset V$, each of which induces a *connected* subgraph $G[V_i]$. We will call these subsets *minors*. Further, let each node $v \in V$ have private input x_v of length $\tilde{O}(1)$, i.e., a value that can be sent along an edge in $\tilde{O}(1)$ rounds. Finally, we are given an aggregation function $\bigotimes$ like `SUM`, `MIN`, `AVG`, ..., which we want to compute in each part. Then a *minor aggregation* computes these functions in all minors $G[V_i]$. Note that the diameter of these minors might be (much) larger than the diameter of G. Therefore, the corresponding algorithm has to use edges outside of each minor to be efficient.

Both approximate shortest paths and minor aggregation can be efficiently implemented in arbitrary and restricted graphs in all of our models. Note that we are mainly interested in algorithms that require $\tilde{O}(1)$ aggregations and $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in \Omega(1/\log^c n)$ as these algorithms can then be efficiently implemented in all three models. Note that *efficient* means different things in different models. While in the PRAM, we hope for a polylogarithmic depth using a linear number of processors and in HYBRID we aim for polylogarithmic runtime, we have to manage our expectations for CONGEST. In the CONGEST model, the best we can hope for is $O(\mathrm{HD})$, where HD denotes the length of the longest shortest path between two nodes *if we ignore the weights*, the so-called hop diameter. The hop-diameter is a natural lower bound for most *global* problems in CONGEST as it describes the precise time in which a message starting from a node $v \in V$ can (theoretically) reach every node in the network. While this can be up to n, it may be much smaller in many practical graphs.

To be precise, it holds:

▶ **Theorem 4.** *Let $G := (V, E, \ell)$ be a (weighted) graph and let $\mathcal{A}$ be an algorithm that can be broken down into the following two types of operations:*

1. *A $(1+\epsilon)$-approximate SetSSP on a set of disjoint connected subgraphs $C_1, \dots, C_N$ with $C_i = (V_i, E_i)$ from a set of sources $S_1, \dots, S_N$ with $S_i \subset V_i$.*
2. *A minor aggregation on a set of disjoint connected subgraphs $C_1, \dots, C_N$ with $C_i = (V_i, E_i)$.*

[3] Technically, for a set S with $|S| \geq 2$, the algorithm produces a forest with the individual tree rooted in the nodes S. However, we slightly abuse notation and refer to it as a forest.

The subgraphs $C_1, \dots, C_N$ may differ in each step. Suppose that $\mathcal{A}$ can be broken down into τ_s steps of $(1+\epsilon)$-approximate SetSSP computations and τ_m minor aggregations. Then, it holds:

- *In* CONGEST*, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot (HD + \sqrt{n})\right)$ time, w.h.p. If graph G is (universally) k-path separable, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot k \cdot HD\right)$ time, w.h.p.*
- *In the* PRAM *model with $\tilde{O}(m)$ processors, $\mathcal{A}$ can be executed with $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot m\right)$ work and $\tilde{O}\left(\tau_s + \tau_m\right)$ depth, w.h.p.*
- *In the* HYBRID *model, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m)\right)$ time, w.h.p., using $O(n)$ bits per message in the local mode. If graph G is (universally) k-path separable, $\mathcal{A}$ can be executed in $\tilde{O}\left((\epsilon^{-2} \cdot \tau_s + \tau_m) \cdot k\right)$ time, w.h.p., using $O(\log n)$ bits per message in the local mode*

For general graphs, the bounds for approximate SetSSP in CONGEST and PRAM follow from a recent breakthrough result [56]. Similarly, the runtimes for minor aggregation in CONGEST and PRAM are given in [41] and [56], respectively. Finally, [57] showed the bounds for both SetSSP and minor aggregation in the HYBRID model. For the runtime on k-path separable graphs in CONGEST, we use that k-path separable graphs exclude K_{4k+1} as minor [22]. If the graph excludes a fixed clique minor K_r, the runtime of a minor aggregation is $\tilde{O}(r \cdot \mathrm{HD})$ [37] and $(1+\epsilon)$-approximate SetSSP can be solved in $\tilde{O}(\epsilon^{-2} \cdot r \cdot \mathrm{HD})$ [56]. Note that for CONGEST, all these bounds are optimal as neither SetSSP nor the aggregation can be computed faster [41].

1.3 Our Contributions

In the following, we give a formal statement of our contributions. First, we show that we can achieve the (asymptotically) best possible quality of $O(\log n)$ for general through approximate SetSSP and minor aggregation. Our first main theorem is the following.

▶ **Theorem 5** (LDDs for General Graphs)**.** *Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) undirected graph. Then, there is an algorithm that creates an LDD of G with strong diameter $\mathcal{D}$ and quality $O(\log n)$. The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O\left(1/\log^2 n\right)$.*

By Theorem 4, this implies that the algorithm can, w.h.p., be implemented in $\tilde{O}(\mathrm{HD} + \sqrt{n})$ time in CONGEST, and $\tilde{O}(1)$ depth in the PRAM and $\tilde{O}(1)$ time in HYBRID. Thus, our algorithm is currently the best randomized LDD construction in CONGEST for general weighted graphs. It has same runtime as [55] and [13], creates clusters of *strong* diameter, and has (asymptotically) optimal quality of $O(\log n)$. We achieve this improvement through a more fine-grained analysis of the well-known exponential-delay clustering first used by Miller et al. [50] that was already used by Becker, Lenzen, and Emek in [13]. In fact, we only present a little addition to their existing algorithm to ensure the strong diameter.

Second, for k-path separable graphs with $k \in \tilde{O}(1)$, we present an algorithm with almost exponentially better quality. More precisely, our second main theorem is the following.

▶ **Theorem 6** (LDDs for k-Path Seperable Graphs)**.** *Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) $\tilde{O}(1)$-path separable graph. Then, there is an algorithm that creates an LDD of G with strong diameter $\mathcal{D}$ and quality $O(\log \log n)$. The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O\left(1/\log^2 n\right)$.*

Thus, by the same reasoning as above, the algorithm can, w.h.p., be implemented in $\tilde{O}(\mathrm{HD})$ time in CONGEST, and $\tilde{O}(1)$ depth in the PRAM and $\tilde{O}(1)$ time in HYBRID.

Our main technical novelty for proving Theorem 6 is a weaker form of a path separator that can be constructed without computing an embedding of G. Prior algorithms for k-path separable graphs, e.g. [2], often employ a so-called shortest path decomposition where we recursively compute k-path separators until the graph is empty. However, computing these separators requires precomputing a complex embedding, which we cannot afford in a distributed or parallel setting. Instead, we take a different approach to completely avoid the (potentially expensive) computation of an embedding. We show that by sampling approximate shortest paths (in a certain way), we can obtain what we will call a weak $\mathcal{D}$-separator. A weak $\mathcal{D}$-separator S' only ensures that in the graph $G \setminus S$, all nodes have at most a constant fraction of nodes in distance $\mathcal{D}$. In particular, in contrast to a *classical* separator, the graph $G \setminus S'$ might still be connected. We strongly remark that, in considering weak separators, we sacrifice some of the useful properties of a traditional seperator. Nevertheless, this only results in some additional polylogarithmic factors in the runtime of our applications.

1.4 Applications of LDDs

In this section, we discuss the natural question of why we consider LDD's in the first place. Simply put, LDD's are part of an algorithm designer's toolkit for developing efficient divide-and-conquer algorithms, as they offer a generic way to decompose a given graph. As such, creating LDD's with low edge-cutting probability plays a significant role in numerous algorithmic applications. In the following, we present a comprehensive list of applications (which we do not claim to be complete):

Tree Embeddings. As a first example, LDD's can be used to embed graphs into trees, i.e., constructing so-called metric tree embeddings (MTE) and low-stretch spanning trees (LSSP). In both, we map a given graph $G := (V, E, \ell)$ into a randomly sampled tree $T = (V, E_T, \ell_T)$ in a way that preserves the path lengths of the original graph G on expectation. In an LSSP, it must hold $E_T \subset E$, i.e., we compute a subtree of G, while a MTE can use virtual edges not present in the original graph. LSSPs and MTEs have proven to be a helpful design paradigm for efficient algorithms as many computational problems are significantly easier to solve on trees than on general graphs. For example, they have been used to construct distributed and parallel solvers for certain important classes of LPs [6, 14, 20, 38], which can then be used solve MaxFlow and other complex problems. Nearly all constructions use LDD's (or variants thereof) as subroutines; see [1, 5, 12, 25] for the construction of LSSPs and [8, 9, 10, 31] for MTEs.

Light Spanners. Another interesting avenue for LDD's are so-called *light spanners* first introduced by Elkin, Neimann, and Solomon in [30]. Given a weighted graph $G = (V, E, \ell)$, a t-spanner is a subgraph $H \subset G$ that approximately preserves the distances between the nodes of the original graph by a factor t, i.e., for all pairs of node $v, w \in V$, it holds $d_H(v, w) \leq t \cdot d_G(v, w)$. A spanner's lightness L_H is the ratio between the weight of all its edges and the MST of graph G, i.e., it holds $L_H = \sum_{e \in H} \ell_e / \sum_{e \in MST(G)} \ell_e$. Thus, compared to the LSSPs and MTEs, light spanners have more freedom in choosing their edges as they are not required to be trees. The lightness is (arguably) a natural quality measure for a spanner, especially for distributed computing. Consider, for example, an efficient broadcast scheme in which a node wants to spread a message to every other node in G. Further, suppose the edges' length as the cost of using this edge to transmit a message. If we send the messages along

the edges of the light spanner H of G, the total communication cost can be bound by its lightness, and the stretch bounds the cost of the path between any two nodes. Despite this, there are almost no distributed algorithms that compute t-spanners with an upper bound on the lightness with [26] being a notable exception of stretch $O(k)$ with lightness $\tilde{O}(n^{\frac{1}{k}})$. The complicating factor in the distributed construction is that lightness is a global measure. Many previously known distributed spanner constructions like [11] or [50] are very local in their computations as they only consider each node's t-neighborhood with $t \in O(\log n)$. Therefore, the resulting spanners' lightness is unbounded despite having few edges. LDD's are connected to light spanners through an observation by Neiman and Filtser in [33]: They show that with black-box access to an LDD algorithm with quality α, one can construct an $O(\alpha)$-spanner with lightness $\tilde{O}(\alpha)$ for any weighted graph $G = (V, E, \ell)$. In addition to an algorithm that creates LDD's for geometrically increasing diameters, they only require so-called (α, β)-nets for which they already provided a distributed implementation in [26]. Thus, finding better distributed algorithms LDDs, especially for restricted graphs, drectly improves the distributed construction of light spanners.

Compact Routing Schemes. *Routing schemes* are distributed algorithms that manage the forwarding of data packets between network devices. Thus, developing efficient routing schemes is essential to enhance communication among multiple parties in distributed systems and, therefore, is well researched [15, 18, 27, 28, 42, 43, 44, 45, 46, 48]. In particular, it is known that LDD's can be used to construct routing schemes with routing paths that are not significantly longer than the shortest paths in G. An LDD with strong diameter and quality α directly enables us to construct a routing scheme where all paths only differ by a factor of $O(\alpha)$ from the true shortest paths. This factor is typically referred to as the stretch of the routing scheme. The core idea is to create LDD's with diameter $O(\alpha\mathcal{D}_i)$ for geometrically increasing distance scales $\mathcal{D}_i = 2^i$. Then, for the right choice of parameters hidden in the O-notation, for any two nodes in distance $[\mathcal{D}_i, 2\mathcal{D}_i]$ there is a cluster of diameter $O(\alpha\mathcal{D})$ containing them both with constant probability. Using techniques presented in [23], this is sufficient to efficiently compute a routing scheme with stretch $O(\alpha)$ in CONGEST, the PRAM, and the HYBRID model. However, this stretch can likely be improved using more sophisticated algorithms.

1.5 Related Work

In this section, we present some related decomposition schemes. We focus on the CONGEST model as it provides the greatest variety of decomposition schemes. We present a compact overview in Table 1.

Despite the vast number of applications for LDD's on *weighted* graphs, research on LDD's in a distributed setting has mostly focused on the unweighted case, producing many efficient algorithms in this regime. Two notable exceptions, however, explicitly consider weighted graphs and are closely related to our results: The work of Becker, Emek, and Lenzen [13] creates LDD's of quality $O(\log n)$ with weak diameter[4] for general graphs. We note that $O(\log n)$ is the best quality we can hope for in an LDD due to a result by Bartal [8]. The algorithm requires $\tilde{O}(\mathrm{HD} + \sqrt{n})$ time in the CONGEST model, which is optimal as each distributed LDD construction in a weighted graph requires $\Omega(\mathrm{HD} + \sqrt{n})$ time [40]

[4] Actually, [13] proves a stronger property of the diameter. Although the diameter is weak, the number of nodes outside of a cluster that are part of the shortest path between two nodes of a cluster is limited. For many applications, this is sufficient and just as good as a strong diameter.

as we can derive approximate shortest paths from it. Just as our algorithm, [13] consists of $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$. Further, [55], makes two significant improvements compared to [13]. They present a decomposition with strong diameter (instead of weak), and their construction is deterministic (instead of randomized). Here, they do not bound the probability that a specific edge of length ℓ is cut, but instead – since the algorithm is deterministic – count the overall number of edges of length ℓ that are cut. We say that a deterministic decomposition has quality α, if the number of edges of length ℓ with endpoint in different clusters is bound by $m \cdot \frac{\alpha \cdot \ell}{\mathcal{D}}$. Note that $m \cdot \frac{\alpha \cdot \ell}{\mathcal{D}}$ is exactly the *expected* number of cut edges in a probabilistic LDD of quality α. Thus, with constant probability, the probabilistic version roughly cuts the same number of edges. However, without further information about the specific random choices made by the corresponding algorithm, a probabilistic LDD could cut many more edges. While this can be mitigated with standard probability amplification techniques, i.e., executing the algorithm $O(\log n)$ times and picking the iteration that cuts the fewest edges, a deterministic LDD does not have this problem in the first place. In a deterministic LDD, we have the guarantee that no matter what happens in the execution of the algorithm, less than $m \cdot \frac{\alpha \cdot \ell}{\mathcal{D}}$ edges of length ℓ are cut. This makes these decompositions very useful when we do not have the time or resources for $O(\log n)$ repetitions. A typical example is distributed algorithms for local problems, where we aim for sublogarithmic runtimes. That being said, they have a slightly worse quality of only $O(\log^3 n)$. Again, the algorithm consists of $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$.

For restricted graphs like planar, bounded treewidth, or minor-free graphs, we are unaware of a distributed/parallel algorithm explicitly designed for weighted graphs. The research [16, 17, 29, 34, 47, 19] in the CONGEST model focused on so-called *network decompositions* for unweighted graphs that enable fast algorithms for local problems like MIS, Coloring, or Matching. In principle, these algorithms could also be applied to weighted graphs and produce LDD's. However, their runtime depends on the diameter $\mathcal{D}$ of the resulting clusters, which might be much larger than the hop diameter (or even n) if the graph is weighted. We further remark that some authors use slightly different notations when describing LDD's. While our definition above focuses on the clusters' diameter, some works emphasize the number of cut edges. That is, they define it as a decomposition that cuts $\epsilon \cdot m$ edges and creates a cluster of diameter $\mathcal{D} := O(\epsilon^{-1})$. The quality is then described by the blowup in the diameter. Notably, Chang and Su [17] show that a low-diameter decomposition with $\mathcal{D} = O(r\epsilon^{-1})$ can be computed in $\epsilon^{-O(1)} \log^{O(1)} n$ rounds with high probability and $\epsilon^{-O(1)} 2^{O(\sqrt{\log n \log\log n})}$ rounds deterministically in the CONGEST model for any K_r-free graph. Chang [16] later improved the deterministic runtime to $O(\mathcal{D} \log^* n + \mathcal{D}^5)$. In particular, if we translate this to our notion of quality, Chang [16] for unweighted graphs that present LDD's in K_r-free graphs with quality $O(r)$ in the CONGEST model. Therefore, our algorithm has worse a worse quality of $O(\log\log n)$ but is faster for large diameters $\mathcal{D}$, which arguably trades off its worse cutting probability. Also note that our techniques are entirely different from those of [16]. We elaborate more on this in the full version [24].

1.6 Structure of This Paper

The main part of the paper is structured as follows: In Section 2 we present the algorithm behind Theorem 5. In doing so, we also present several useful intermediate results we reuse later. Then, in Section 3 we present our novel technique to compute weak separators. In Section 4, we combine our insights into clustering algorithms from Section 2 and findings on the construction of separators from Section 3 to develop the algorithm behind Theorem 6.

Table 1 An overview of the related work on LDD's for various graph families in the CONGEST model.

Ref.	Quality	Diam.	Runtime	Weights	Comment
[19]	$O\left(\mathcal{D}^{1/c}\right)$	Strong	$O\left(\mathcal{D}^{O(1)}\right)$	×	Planar
[47]	$O\left(\mathcal{D}^{1/c}\right)$	Strong	$\tilde{O}\left(\mathcal{D}^{O(1)}\right)$	×	K_r-free
[16]	$O(r)$	Strong	$\tilde{O}\left(\mathcal{D}^{O(1)}\right)$	×	K_r-free
Thm. 6 + [22]	$O(\log\tau + \log\log n)$	Strong	$\tilde{O}(\tau^2\text{HD})$	✓	Treewidth τ
Thm. 6 + [58]	$O(\log\log n)$	Strong	$\tilde{O}(\text{HD})$	✓	Planar
Thm. 6 + [2]	$O(\log f(r) + \log\log n)$	Strong	$\tilde{O}(r \cdot f(r) \cdot \text{HD})$	✓	K_r-free
[29]	$O(\log n)$	Strong	$\tilde{O}(\mathcal{D})$	×	
[13]	$O(\log n)$	Weak	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	
[55]	$O\left(\log^3 n\right)$	Strong	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	
Thm. 5	$O(\log n)$	Strong	$\tilde{O}(\text{HD} + \sqrt{n})$	✓	

2 Strong Diameter LDDs for General Graphs

In this section, we present the algorithm behind Theorem 5. The description is divided into three sections/subroutines. First, in Subsection 2.1, we present a generic clustering algorithm. Second, in Subsection 2.2 we combine this algorithm with a technique from [13] that cuts all edges with the correct probability. However, we may not add each node to a cluster. Finally, in Subsection 2.3, we present a generic technique that recursively applies the algorithm from Subsection 2.2 to cluster all nodes while asymptotically preserving the cut probability of an edge.

2.1 Pseudo-Padded Decompositions Using Approximate Shortest Paths

We begin with a crucial technical theorem that will build the foundation of most of our results. A key component in this construction is the use of truncated exponential variables. In particular, we will consider exponentially distributed random variables truncated to the $[0, 1]$-interval. Loosely speaking, a variable is truncated by resampling it until the outcome is in the desired interval. In the following, we will always talk about variables that are *truncated to the* $[0, 1]$*-interval* when we talk about truncated variables. The density function for a truncated exponential distribution with parameter $\lambda > 1$ is defined as follows:

▶ **Definition 7** (Truncated Exponential Distribution). *We say a random variable X is truncated exponentially distributed with parameter λ if and only if its density function is $f(x) := \frac{\lambda \cdot e^{-x\lambda}}{1-e^{-\lambda}}$. We write $X \sim \mathsf{Texp}(\lambda)$. Further, if $X \sim \mathsf{Texp}(\lambda)$ and $Y := \mathcal{D} \cdot X$, we write $Y \sim \mathcal{D} \cdot \mathsf{Texp}(\lambda)$.*

The truncated exponential distribution is a useful tool for decompositions that has been extensively used in the past [4, 32, 51]. Using a truncated exponential distribution and $(1+\epsilon)$-approximate SetSSP computations, we prove a helpful auxiliary result, namely:

▶ **Theorem 8** (Pseudo-Padded Decomposition for General Graphs). *Let $\mathcal{D} > 0$ be a distance parameter, ϵ be an error parameter, $G := (V, E, \ell)$ a (possibly weighted) undirected graph, and $\mathcal{X} \subseteq V$ be a set of possible cluster centers. Suppose that for each node $v \in V$, the following two properties hold:*

- ***Covering Property:*** *There is at least one $x \in \mathcal{X}$ with $d_G(v, x) \leq \mathcal{D}$.*
- ***Packing Property:*** *There are at most τ centers $x' \in \mathcal{X}$ with $d_G(v, x') \leq 6\mathcal{D}$.*

Then, for $\epsilon \in o(1/\log \tau)$ there is an algorithm that computes a series of connected clusters $\mathcal{K} = K_1, \dots, K_N$ with strong diameter $4(1+\epsilon)\mathcal{D}$ where for all nodes $v \in V$ and all $\epsilon \leq \gamma \leq \frac{1}{32}$, it holds:

$$\mathbf{Pr}[B(v, \gamma\mathcal{D}) \subset K(v)] \geq e^{-\Theta((\gamma+\epsilon)\log \tau)} - O(1/n^c).$$

Here, $K(v)$ denotes the cluster that contains v. The algorithm can be implemented with one $(1+\epsilon)$ approximate SetSSP computation and $\tilde{O}(1)$ minor aggregations.

Technically, this algorithm is a generalization of the algorithm in [32] that is based on **exact** shortest path computations. This algorithm is itself derived from [50]. Our algorithm replaces all these exact computations through $(1+\epsilon)$-approximate calculations. The main analytical challenge is carrying the resulting error ϵ through the analysis to obtain the bounds in the theorem. The same approach was already used by Becker, Lenzen, and Emek [13]. However, our analysis is more fine-grained w.r.t. to the impact of the approximation parameter ϵ.

In the following, we give the high-level idea behind the construction. An intuitive way to think about the clustering process from [32, 50] is as follows: Each center x draws value $\delta_x \sim \mathcal{D} \cdot \mathsf{Texp}(2 + 2\log \tau)$ and wakes up at time $\mathcal{D} - \delta_x$. Then, it begins to broadcast its identifier. The spread of all centers is done in the same unit tempo. A node v joins the cluster of the *first* identifier that reaches it, breaking ties consistently. If we had $O(\mathcal{D})$ time, we could indeed implement it exactly like this. However, this approach is infeasible for a general $\mathcal{D} \in \tilde{\Omega}(1)$ in weighted graphs as $\mathcal{D}$ may be arbitrarily large. Instead, we will model this intuition using a virtual super source s and shortest path computations. For each center $x \in \mathcal{X}$, we independently draw a value $\delta_x \sim \mathcal{D} \cdot \mathsf{Texp}(2 + 2\log \tau)$ from the truncated exponential distribution with parameter $2 + 2\log(\tau)$. We assume that τ (or some upper bound thereof) is known to each node. Then, we add a virtual source s with a weighted virtual edge (s, x) to each center $x \in \mathcal{X}$ with weight $w_x := (\mathcal{D} - \delta_x)$. Then, we compute a $(1+\epsilon)$-approximate SetSSP from s with $\epsilon \leq \frac{1}{40 \log \tau}$. Any node joins the cluster of the *last* center on its (approximate) shortest path to s.

With exact shortest paths, this construction would preserve the intuition. Any node whose shortest path to s contains center x as its last center on the path to s would have been reached by x's broadcast first. The statement is not that simple with approximate distances as the approximation may introduce a *detour*, so the center that minimizes $(\mathcal{D} - \delta_x) + d(x, v)$ may not actually cluster a node $v \in V$. In particular, two endpoints of a short edge $\{v, w\}$ may end up in different clusters although the same center minimizes the distance to both. The approximation error for v could be large, while for w, it is very low, which can cause undesired behavior. The nodes will only be added to the same cluster, if its center's head start (which is determined by the random distance to the source) is large enough to compensate for the *approximation errors*. This, however, depends on the approximation and *not* an the length of the edge we consider. Nevertheless, if the error ϵ is small enough, we get similar clustering guarantees that are good enough for our purposes. In particular, if the difference in distance between the two closest centers is larger than $O(\epsilon\mathcal{D})$, the clustering behaves very similar to its counterpart with exact distances. In our analysis, we show this by carefully carrying the approximation error through the analysis. We do not introduce any fundamental new techniques, but simply carry the approximation error all the way through the analysis of the corresponding clustering process with exact distances presented in [32].

In addition to Theorem 8 itself, this also allows us to observe the positive correlation between nodes that are on the shortest path to some cluster center. This closer look also allows us to prove the following technical fact about the pseudo-padded decompositions:

▶ **Lemma 9** (Well-Separated Nodes). *Let $\mathcal{K}$ be a partition computed by the algorithm from Theorem 8. Then, there a constant $c \geq 1$ such that for given node $u \in K_i$ with probability $(1/2)$ the following holds: There is path $P_u := (u = v_1, \dots, v_N = x_i)$ of length at most $2(1+\frac{1}{40 \log \tau})\mathcal{D}$ to the cluster center x_i of K_i where for each $v_j \in P_u$, it holds $B\left(v_j, \frac{\mathcal{D}}{c \log \tau}\right) \subseteq K_i$.*

This lemma states that if a single node $v \in V$ is padded, all nodes on a short path to its cluster center are likely padded as well. While this initially sounds very technical, it roughly translates to the following: Consider a clustering $\mathcal{K} = K_1, \dots, K_N$. Then for a suitably chosen $\rho := \frac{\mathcal{D}}{c \cdot \log \tau}$, in each cluster $\mathcal{K}_i = (V_i, E_i)$ there is a constant fraction of nodes $V_i' \subseteq V_i$ in the distance ρ to their closest node in a neighboring cluster $K_j \neq K_i$ **and** for any two $v, w \in V_i'$ there is a path of length $6\mathcal{D}$ (via the cluster center) that only consists of nodes in V_i'. This insight will be *crucial* for our next algorithm. A more detailed description and the full proof of both lemmas can be found in the full version [24].

2.2 Strong LDCs from Pseudo-Padded Decompositions

In this section, we present a generic algorithm that creates an LDC with strong diameter of quality $(O(\log \tau), 1/2)$ if the number of nodes that can be centers of clusters has been sufficiently *sparsed out*. In particular, we suppose that each node can only be in one of τ clusters. For this case, we show that it holds:

▶ **Theorem 10** (A Generic Clustering Theorem). *Let $\mathcal{D} > 0$ be a distance parameter, $G := (V, E, \ell)$ a (possibly weighted) undirected graph, and $\mathcal{X} \subseteq V$ be a set of marked nodes. Suppose that for each node $v \in V$, the following two properties hold:*

- ***Covering Property:*** *There is at least one $x \in \mathcal{X}$ with $d(v, x) \leq \mathcal{D}$.*
- ***Packing Property:*** *There are at most τ centers $x' \in \mathcal{X}$ with $d(v, x') \leq 6\mathcal{D}$.*

Then, there is an algorithm that creates an LDC of strong diameter $8\mathcal{D}$ with quality $(O(\log \tau), 1/2)$. The algorithm can be implemented with $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O(1/\log^2 n)$.

The algorithm uses the so-called *blurry ball growing* (BBG) technique. This is perhaps the key technical result that Becker, Emek, and Lenzen introduced in [13]. It provides us with the following guarantees:

▶ **Lemma 11** (Blurry Ball Growing (BBG), cf. [13, 55]). *Given a subset $S \subseteq V$ and an arbitrary parameter $\rho > 0$, an algorithm for BBG outputs a superset $S' \supseteq S$ with the following properties*

1. *An edge $\{v, w\} \in E$ of length $\ell_{(v,w)}$ is cut with probability $\mathbf{Pr}[v \in S', w \notin S'] \leq O\left(\frac{\ell_{(v,w)}}{\rho}\right)$.*
2. *For each node $v \in S'$, it holds $d(v, S) \leq \frac{\rho}{1-\alpha} \leq 2\rho$ where $\alpha \in O(\log \log n / \log n)$.*

BBG can be implemented using $\tilde{O}(1)$ $(1+\epsilon)$ approximate SetSSP computations with $\epsilon \in O\left((\log \log n / \log n)^2\right)$.

This technique allows us to create clusters with a low probability of cutting an edge while only having access to approximate shortest paths. Note that in [55] the dependency on ϵ was improved to $O(1/\log n)$. The paper also presents a deterministic version of blurry ball growing. However, we will only use the probabilistic version introduced above.

Given this definition, we now give an overview of the algorithm that creates an LDC: In the following, define $\rho := \frac{\mathcal{D}}{c \cdot \log \tau}$ where c is the constant from Lemma 9. The algorithm first computes a pseudo-padded decomposition $\mathcal{K} = K_1, \dots, K_N$ of strong diameter $8\mathcal{D}$ using the algorithm of Theorem 8. To do this, we choose the given set $\mathcal{X}$ of marked nodes as the set of cluster centers.

The following steps are executed in each cluster K_i in parallel. Within each cluster K_i, we determine a so-called *inner cluster* $K_i' \subseteq K_i$ of strong diameter $6\mathcal{D}$ where each node $v \in V_i'$ has distance (at least) $\rho' := \rho/2$ to the closest node in different cluster $K_j \neq K_i$. These inner clusters can be determined via two (approximate) SetSSP computations: First, all nodes calculate the 2-approximate distance to the closest node in a different cluster. To do this, we first create a virtual supersource s, and all nodes $v \in K_i$ with an edge to a node $w \in K_j$ create a virtual edge of length $\ell_{(v,w)}$ to s. If they have several such neighbors, they choose the shortest edge. Then, we perform a 2-approximate SetSSP from s on the resulting virtual graph that consists of all edges within a cluster and the virtual edges to s. In other words, the edges $\{v, w\}$ between clusters are not considered.

We then mark all nodes where this 2-approximate distance exceeds $2\rho'$ as *active nodes*. Next, we consider the graph G' induced by the active nodes. Using a $(1+\frac{1}{40\log\tau})$-approximate SetSSP calculation, the active nodes compute if they have a path of length at most $3\mathcal{D}$ to their cluster center in G'. If so, they add themselves to the inner cluster K_i' of K_i. Recall that their exact distance to the next cluster is at least ρ' as the paths are 2-approximate. Further, for any two nodes $v, w \in K_i'$ there is a path of length at most $6\mathcal{D}$ via the cluster center and so the inner cluster has a strong diameter of $6\mathcal{D}$. Thus, this procedure *always* results in an inner cluster K_i' with the desired properties. In the third and final stage of the algorithm, the actual clusters are computed. To this end, the *inner clusters* $\mathcal{K}' := \bigcup_{i=1}^{N} K_i'$ are used as the input set to the BBG procedure from Lemma 11. In particular, we choose the parameter for the BBG to be $\rho'' := \rho'/2$ and compute the superset $S(\mathcal{K}') := \mathsf{blur}(\mathcal{K}', \rho'')$. Now define final clustering $\mathcal{C} = C_1, \dots, C_N$ where $C_i := K_i \cap S(\mathcal{K}')$. In other words, the cluster C_i contains the inner cluster K_i' and all nodes from K_i added by the BBG process.

The resulting clustering fulfills all three properties required by Theorem 10. For the cutting probability, note that the distance from an inner cluster to a different cluster is ρ', and BBG only adds nodes in distance smaller than $2\rho'' = \rho'$. Thus, all edges with an endpoint in cluster K_i are only cut by the blurring process and not by the initial pseudo-padded decomposition. Thus, the choice of ρ'' implies that edges are cut with correct probability of $O(\ell/\rho'') = O(\frac{\ell \cdot \log\tau}{\mathcal{D}})$ by Lemma 11. The (strong) diameter of the cluster is at most $8\mathcal{D}$ as each node is in the distance at most $\rho \leq \mathcal{D}$ to a node from an inner cluster, and all nodes in an inner cluster are in the distance $6\mathcal{D}$ to one another. Therefore, it only remains to show that a nodes is clustered with prob. $1/2$. We can prove this via Lemma 9. For all nodes $v \in V$ the following holds with probability $1/2$: Node $v \in K_i$ **and** all nodes on the path P_v of length $2(1+\frac{1}{40\log\tau})\mathcal{D}$ to its cluster center x_i are in distance at least ρ to the closest node in a different cluster. Therefore, all these nodes will mark themselves active. As the 2-approximate SetSSP only overestimates, these nodes compute a distance more than $\rho = 2\rho'$, which causes them to be active. Moreover, this implies that the path P_v is fully contained in the graph G' induced by active nodes. As the path P_v is of length $2(1+\frac{1}{40\log\tau})\mathcal{D}$ and ends in the cluster center x_i, the $(1+\frac{1}{40\log\tau})$-approximate SetSSP computation will find a path of length at most $(1+\frac{1}{40\log\tau})2(1+\frac{1}{40\log\tau})\mathcal{D} \leq 3\mathcal{D}$ in G'. Therefore, with a probability of at least $1/2$, node v and all nodes on the path to the cluster center will be added to the inner cluster K_i'. Thus, Theorem 10 follows as half of all nodes are in an inner cluster and are therefore guaranteed to be in some cluster C_i. A more detailed description and the full proof can be found in the full version [24].

2.3 Strong LDDs from Strong LDCs

In this section, we show that we can create an LDD with quality $O(\log n)$ for any graph within $\tilde{O}(1)$ minor aggregations and $(1+\epsilon)$-approximate SetSSP's. Note that, at first glance, the theorem follows almost directly from Theorem 10. Choose $\mathcal{X} = V$ and $\mathcal{D}' = 8\mathcal{D}$ and apply

the theorem to a graph G. We obtain a clustering with diameter $\mathcal{D}'$ if each node has one node in distance $\mathcal{D}'/8$ and at most τ nodes in distance $\mathcal{D}'$. Clearly, for every possible choice of parameter $\mathcal{D}'$, each node has at least one marked node in distance $\mathcal{D}'/8$, namely itself. Further, for every possible choice of parameter $\mathcal{D}'$, each node has at most n marked node in distance $\mathcal{D}$ because there are only n nodes in total. Thus, by choosing all nodes as centers, we obtain a clustering with quality $O(\log n)$ and a diameter we can choose freely. While the diameter and edge-cutting probability of the resulting clustering are both (asymptotically) correct, we can only guarantee that at least half of the nodes are clustered on expectation. To put it simply, we do not have a partition as required. However, we can reapply our algorithm to the graph induced by the *unclustered* nodes. This clusters half of the remaining nodes on expectation. Thus, by applying Markov's inequality, a constant fraction of nodes is clustered with constant probability. It is easy to see that if we continue this for $O(\log n)$ iterations, we obtain the desired partition, w.h.p. Further, this will not significantly affect the edge-cutting probability, as each edge that is not cut will be added to a cluster with constant probability. Therefore, there cannot be too many iterations where the edge can actually be cut. Thus, applying the algorithm until all nodes are clustered intuitively produces the required LDD of quality $O(\log n)$. We formalize and prove this intuition in the following lemma.

▶ **Lemma 12.** *Let $G = (V, E, \ell)$ be a weighted graph. Let $\mathcal{A}$ be an algorithm that for each subgraph $G' \subseteq G$ can create clusters $K_1, \dots, K_N$ where each edge is cut between clusters with probability at most α, and each node is added to a cluster with probability at least β. Then, recursively applying $\mathcal{A}$ to G until all its nodes are in a cluster creates an LDD of quality $O(\alpha\beta^{-2})$. The procedure requires $O(\beta^{-1} \log n)$ applications of $\mathcal{A}$, w.h.p.*

The full proof can be found in the full version [24]. Together, Lemma 12 and Theorem 10 imply Theorem 5. Note that we state Lemma 12 more generally than needed, so we can reuse it in the next section.

3 Weak Separators from Approximate Shortest Paths

We now move from arbitrary graphs to k-path separable graphs. This section presents our main technical result and considers the distributed computation of separators for k-path separable graphs. Recall that a separator S is a *subgraph* of G such that in the induced subgraph $G \setminus S$ that results from removing S from G, every connected component only contains a constant fraction of G's nodes. Separators are a central tool in designing algorithms for restricted graph classes as they give rise to efficient divide-and-conquer algorithms. Our algorithms are no exception. In particular, we will compute a novel type of separator as the main building blocks of our clustering result. Note that the distance is taken w.r.t. to G, which could be a subgraph of a larger graph. We formally define our weak separators as follows:

▶ **Definition 13** (Weak κ-Path $(\mathcal{D}, \epsilon)$-Separator)**.** *Let $G := (V, E, \ell)$ be a weighted graph, $\mathcal{D} > 1$ be an arbitrary distance parameter, and $\epsilon > 0$ be an approximation parameter. Then, we call the set $S(\mathcal{D}, \epsilon) := (\mathcal{P}_1, \dots, \mathcal{P}_\kappa)$ with $\mathcal{P}_i := (P_i, B_i)$ a weak κ-path separator, if it holds:*

1. *Each $P_i \in \mathcal{P}_i$ is a (approximate) shortest path in $G \setminus \bigcup_{j=1}^{i-1} \mathcal{P}_j$ of length at most $4\mathcal{D}$.*
2. *Each $B_i \subseteq B_G(P_i, \epsilon\mathcal{D})$ is a set of nodes surrounding path P_i.*
3. *For all $v \in (V \setminus \bigcup_{j=1}^{\kappa} \mathcal{P}_i)$ it holds $|B_{G\setminus S}(v, \mathcal{D})| \leq (7/8) \cdot n$.*

Note that this definition is generic enough to also include vanilla k-path separators (where each set B_i only contains the path itself and no further nodes) and *traditional* vertex separators (where each path is a single node). Thus, for certain k-path separable graphs, we can already use existing algorithms to compute these separators in distributed and parallel models[5]. For general k-path separable graphs, however, we need new techniques.

Why Computing Separators for k-Path Separable Graphs is Hard

Before we go into details, let us first review some background on separators that motivates why we will mainly work with the weaker concept. To this end, we first review prior results and identify possible difficulties. Starting with positive results, it *is* possible to construct path separators for planar graphs in a distributed setting. In fact, the state-of-the-art algorithms for computing DFS trees in [39] or computing a graph's (weighted) diameter in [48] in planar graphs construct a path separator for planar graphs. The algorithm presented in these papers constructs a path separator in $\tilde{O}(\text{HD})$ time, w.h.p. However, while the existence of an efficient algorithm for a planar graph gives hope, it turns out to be very tough for other k-path separable graphs. The constructions in both [39] and [48] heavily exploit the fact that a planar embedding of a (planar) graph can be computed in $\tilde{O}(\text{HD})$ time in CONGEST due to Ghaffari and Haeupler in [36]. Given a suitable embedding, the ideas could be generalized. Thus, finding a suitable embedding could be a way to extend this algorithm to more general k-separable graphs. A good candidate for such an embedding can be found in [2]. Abraham and Gavoille present an algorithm that computes a $f(r)$-path separator for K_r-free graphs [2]. Their algorithm relies on a complex graph embedding algorithm by Seymour and Robertson [54]. On a very high level, the embedding divides G into subgraphs that can almost be embedded on a surface similar to planar graph. By *almost*, we mean that in each subgraph, there is only small number of non-embeddable parts (with special properties) that need to be handled separately. The concrete number of these parts depends only on r. Given such an embedding, Abraham and Gavoille show that one can efficiently compute k-path separator. Sadly, we have little hope that the algorithm that computes the embedding can be efficiently implemented in a distributed (or even parallel) context. It already requires $n^{O(r)}$ time in the sequential setting. Moreover, it requires sophisticated techniques that can not be trivially sped up by running them in parallel. Thus, it remains elusive (to the authors at least) how to implement this algorithm in any distributed or even parallel model.

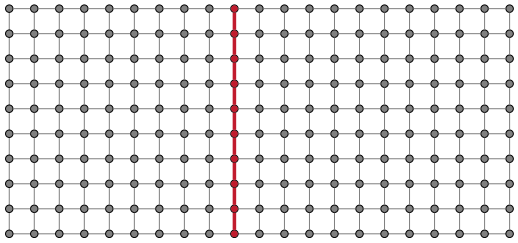
Why Computing Weak Separators for k-Path Separable Graphs is Easier

Now that we have established why it is difficult to compute k-path separators, we can proceed to the initially promised weak separators. Our main insight is that instead of computing an embedding in a distributed and parallel setting, we show that we get very similar guarantees using a weaker separator that can be computed without the embedding. Thus, we take a different approach to completely avoid the (potentially expensive) computation of an embedding. We strongly remark that, in doing so, we sacrifice some of the separator's useful properties. However, in our applications, this only results in some additional polylogarithmic factors in the runtime. To be precise, we prove the following lemma:

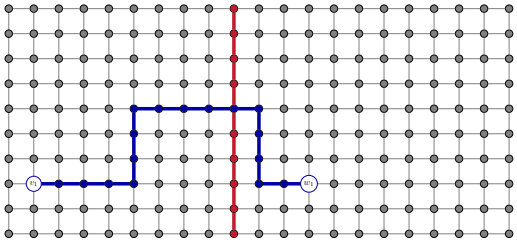
[5] For planar graphs, we can use the algorithm provided in [48] to compute a separator that consists of 4 paths. While for graphs of bounded tree-width τ, we can use [42] to obtain a separator that consists of τ nodes.

▶ **Theorem 14** (Weak Separators for k-Path Separable Graphs)**.** *Consider a weighted k-path separable graph $G := (V, E, \ell)$ with weighted diameter smaller than $\mathcal{D}$. For $\epsilon \geq 0$, there is an algorithm that constructs a weak $O(\epsilon^{-1} \cdot k \cdot \log n)$-path $(\mathcal{D}, \epsilon)$-separator, w.h.p. The algorithm uses $O(\epsilon^{-1} \cdot k \cdot \log n)$ minor aggregations and $O(\epsilon^{-1} \cdot k \cdot \log n)$ 2-approximate shortest path computations, w.h.p.*

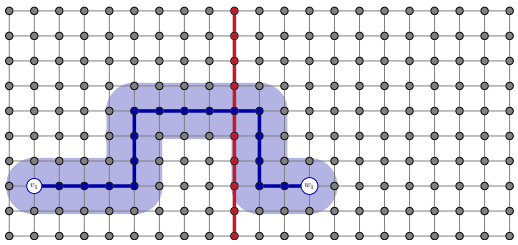
Surprisingly, the algorithm behind the lemma can be stated in just a few sentences. The core idea is to iteratively sample approximate shortest paths and nodes close to these paths and remove them from the graph until all remaining nodes have few nodes in distance $\mathcal{D}$, w.h.p. More precisely, we start with a graph $G_0 = G$. The algorithm proceeds in $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ steps where in step t, we consider graph $(V_t, E_t) := G_t \subset G_{t-1}$. A single step t works as follows: First, we pick a node $v \in V_t$ uniformly at random. Then, we compute a 2-approximate shortest path tree T_v from v. Let $W_v \subset V_t$ be set of all nodes in distance at most $4 \cdot \mathcal{D}$ to v in T_v. We pick a node $w \in W_v$ uniformly at random and consider the corresponding path $P_t = (v, \dots, w)$. Using P_t as a set-source, we compute a 2-approximate shortest path tree T_{P_t} from P_t. Now denote B_{P_t} to be the nodes in the distance at most $\epsilon \cdot \mathcal{D}$ to P_t in G (and **not** G_t). Finally, we remove P_t and B_{P_t} from G_t to obtain G_{t+1}. Note that, due to the approximation guarantee, this removes all nodes in distance at least $\epsilon/2 \cdot \mathcal{D}$ from P_t in G. Further, all steps can be computed using 2-approximate paths (and few aggregation to pick the random nodes) and are completely devoid of any complex embedding. A visualization of the algorithm can be found in Figure 1.



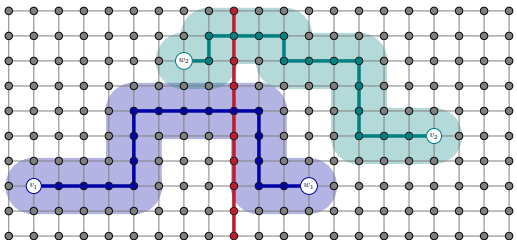
(a) Marked in red is a separator path unknown to the algorithm.



(b) When sampling a random path of bounded length, the probability of crossing the separator is constant.



(c) We remove nodes in distance $\epsilon\mathcal{D}$ around the path. This also removes an $\epsilon\mathcal{D}$-fraction of the separator path.



(d) If the residual graph is not well-separated, the probability of crossing (what remains of) the separator is still constant.

Figure 1 We choose a two-dimensional mesh for illustration. While the operations in Figures **(b)** and **(c)** are straightforward to prove, the core of our analysis we will be the claim we make in Figure **(d)**.

Why does this work? As our main analytical tool, we consider the following potential:

$$\Phi_t := \left|\left\{v \in V_t \;\middle|\; |B_{G_t}(v, 2\mathcal{D})| \geq (7/8) \cdot n\right\}\right|$$

Further, for brevity, we write $\Phi_t(v) = 1$ if and only if $|B_{G_t}(v, 2\mathcal{D})| \geq (7/8) \cdot n$ and $\Phi_t(v) = 0$, otherwise. Note that $\Phi_t := \sum_{v \in V} \Phi_t(v)$.

We make the following important observation: As soon as this potential drops below $(7/8) \cdot n$, we have more than $(1/8) \cdot n$ nodes with less than $(7/8) \cdot n$ nodes in distance $2\mathcal{D}$. This implies that no node can have more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. Suppose for contradiction that there is a node $v \in V_t$ with more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. By the triangle inequality, all these nodes would be in distance $2\mathcal{D}$ to each other. Thus, the potential would be bigger than $(7/8) \cdot n$ as there are more than $(7/8) \cdot n$ nodes with more than $(7/8) \cdot n$ nodes in distance $\mathcal{D}$. This is a contradiction and therefore the nodes sampled until this point are the desired weak separator.

Therefore, we want to bound the number of steps until the potential drops. For this, we will first show the following useful fact about k-path separators, namely

▶ **Lemma 15.** *Let $G := (V, E, w)$ be a weighted k-path separable graph of n nodes with (weighted) diameter $\mathcal{D}$. Then, there exists a set $\mathcal{B} = \{P_1, \ldots, P_\kappa\}$ with $\kappa \leq k$ simple paths of length at most $32\mathcal{D}$ such that for all $v \in V$, it holds $B_{G \setminus \mathcal{B}}(v, 4\mathcal{D}) \leq (3/4) \cdot n$.*

In other words, a subset of bounded length paths intersects with a constant fraction of all paths our algorithm can potentially sample. The proof requires copious use of the triangle inequality and the pigeonhole principle and can be found in the full version [24].

Now, consider the event that the algorithm samples a path P_t that *crosses* some path of $\mathcal{B}$, i.e., the path we sample contains (at least) one node from one of the κ paths in $\mathcal{B}$. In this case, we remove a subpath of length $(\epsilon/2)\mathcal{D}$ from this path when we remove the set B_{P_t} around the P_t. This follows because we determine the set B_t with respect to the original graph G and not G_t. Thus, after sampling $O(\epsilon^{-1} \cdot \mathcal{D})$ paths that cross $\mathcal{B}$, we must have completely removed $\mathcal{B}$ from G. If $\mathcal{B}$ is removed, each node has at most $(3/4) \cdot n$ nodes in distance $\mathcal{D} \leq 4\mathcal{D}$ per definition and we are done. Thus, we will bound the time until **either** the potential drops **or** a total of $O(\epsilon^{-1} \cdot k)$ paths cross $\mathcal{B}$. Recall that in each step t, we sample a path from v_t to w_t. In the following, denote the event that the path from v_t to w_t crosses $\mathcal{B}$ as $v_t \rightsquigarrow_{\mathcal{B}} w_t$. By definition of $\mathcal{B}$, for each node $v \in V$, the set of nodes where all paths of length smaller than $4 \cdot \mathcal{D}$ contain a node of $\mathcal{B}$ is of size at least $(1/4) \cdot n$. Thus, for all nodes $v \in V_t$ with $\Phi_t(v) = 1$, at least $(1/4) \cdot n - (1/8) \cdot n \geq (1/8) \cdot n$ of these nodes must still be in distance $2\mathcal{D}$. All these nodes might be chosen as the path's endpoint P_t if we pick v as the starting point as the 2-approximate distance is at most $4\mathcal{D}$. So, if we sample one of them, we cross $\mathcal{B}$ as all paths of length at most $4\mathcal{D}$ cross $\mathcal{B}$. Thus, in a configuration with potential $\Phi_t \geq (7/8) \cdot n$, the probability to sample a path that crosses $\mathcal{B}$ is at least

$$\begin{aligned}\mathbf{Pr}[v_t \rightsquigarrow_{\mathcal{B}} w_t \mid \Phi_t \geq (7/8) \cdot n] &\geq \mathbf{Pr}[\Phi_t(v_t) = 1 \mid \Phi_t \geq (7/8) \cdot n] \cdot \mathbf{Pr}[v_t \rightsquigarrow_{\mathcal{B}} w_t \mid \Phi_t(v_t) = 1] \\ &\geq \frac{1}{n} \cdot \Phi_t \cdot \frac{1}{n} \cdot (1/8) \cdot n \geq \frac{7}{64} \geq \frac{1}{16}\end{aligned}$$

Thus, if the potential has not dropped after T steps, the expected number of crossings is $(1/16) \cdot T$. Using standard Chernoff-like tail estimates, we can show that for any $T \in \Omega(\log n)$, this would imply that at least $\frac{T}{32}$ paths crossed $\mathcal{B}$, w.h.p. Thus, within $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ steps, we either reached a step with low potential or had sufficiently many crossings to remove $\mathcal{B}$. In either case, we are done as each node had less than $7/8 \cdot n$ nodes in distance $\mathcal{D}$. As up until this point we sampled $T \in O(\epsilon^{-1} \cdot k \cdot \log n)$ paths of length $4\mathcal{D}$ and all nodes in distance $\epsilon\mathcal{D}$ around these paths, this set must be the desired weak separator.

4 Low-Diameter Decompositions for k-path Separable Graphs

In this section, we prove Theorem 6. Our algorithm will combine the insights we gathered for LDD's and LDC's in general graphs with our novel separator construction. Our main technical result is the following proposition:

▶ **Proposition 16** (A Clustering Theorem for Restricted Graphs). *Let $\mathcal{D} > 0$ be an arbitrary distance parameter and $G := (V, E, \ell)$ be a (possibly weighted) $\tilde{O}(1)$-path separable graph. Then, there is an algorithm that creates an LDC of strong diameter $\mathcal{D}$ with quality $(O(\log k + \log\log n), 1/2)$. The algorithm can be implemented in $\tilde{O}(k)$ minor aggregations and $\tilde{O}(k)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$.*

Crucially, the proposition is sufficient to prove Theorem 6 for $\tilde{O}(1)$-separable graphs: For $k \in \tilde{O}(1)$ one application of the algorithm creates a clustering with strong diameter $\mathcal{D}$ and cutting probability $O\left(\frac{\ell_z \cdot (\log\log n)}{\mathcal{D}}\right)$. Now recall that universally k-path separable graphs are closed under minor-taking. Therefore, the graph $G \setminus \mathcal{K}$ is also k-path separable, and we can apply the algorithm to $G \setminus \mathcal{K}$ with the same guarantees to cluster at least half of the remaining nodes. By Lemma 12, after $O(\log n)$ recursive applications, all nodes are clustered, and we obtain an LDD with quality $O(\log\log n)$ as required. Further, for $k \in \tilde{O}(1)$ one iteration of the algorithm can, w.h.p., be implemented in $\tilde{O}(1)$ minor aggregations and $\tilde{O}(1)$ $(1+\epsilon)$-approximate SetSSP computations with $\epsilon \in O\left(1/\log^2 n\right)$. Thus, $O(\log n)$ recursive applications are within the required time complexity of Theorem 6.

Thus, for the remainder, we will focus on Proposition 16. From a high-level perspective, the algorithm behind Proposition 16 proceeds in two distinct phases: the *backbone phase* and the *refinement phase*. In the backbone phase, we create a special partition $\mathcal{K}[\mathcal{B}] := K(\mathcal{B}_1), \dots, K(\mathcal{B}_N)$ of G. Each node $v \in K(\mathcal{B}_i)$ is in distance at most $\mathcal{D}_{BC} < \mathcal{D}$ to some subset $\mathcal{B}_i \subset K(\mathcal{B}_i)$ that consist of $\tilde{O}(k)$ (possibly disjoint) paths of bounded length of $O\left(\mathcal{D}\log^2\right)$. These paths may not necessarily be short**est** paths in G or even $K(\mathcal{B}_i)$, only their length is bounded. We call these subsets $\mathcal{B}_1, \dots, \mathcal{B}_N$ the *backbones* of the clusters. In contrast to an LDD, the clusters do **not** have simple nodes $v \in V$ as centers. As a result, the diameter of the resulting clusters might be much larger than $\mathcal{D}$. Moreover, a single cluster might not even be connected. In the second phase, we turn the resulting clustering into an LDC with connected clusters of strong diameter $\mathcal{D}$. We do so by computing an LDC $\mathcal{K}_i = K_{i_1}, \dots, K_{i_N}$ of strong diameter $\mathcal{D}$ in each cluster $K(\mathcal{B}_i)$. Here, we exploit the special structure of the backbone cluster and choose a carefully chosen subset of nodes from each backbone path as possible cluster centers such that no more than $\tilde{O}(k)$ centers can cluster a given node. These centers will then be used as an input to Theorem 10 to refine each backbone cluster to a set of connected clusters of strong diameter $\mathcal{D}$.

This algorithm is inspired by the works of Abraham, Gavoille, Gupta, Neiman, and Talwar [3, 4] and the subsequent improvement of their algorithm by Filtser [32]. Surprisingly, we do not require fundamental changes to obtain an efficient distributed algorithm. Both algorithms essentially follow the same two-phase structure sketched above. They first constructed a partition around paths sampled in a certain way, then refined each partition individually to obtain proper clusters centered around nodes. For reference, Abraham, Gavoille, Gupta, Neiman, and Talwar dubbed these subsets around which the partitions are constructed *skeletons* while Filtser used the term *r-core*. However, their key properties are (almost) the same. We use the different term *backbones* because we construct them slightly differently. Our contribution when comparing our algorithm to [3, 4, 32] is that we show a) that the exact shortest path can be replaced by approximate paths and the blurry ball growing procedure

from Lemma 11 and, therefore, can be implemented efficiently and b) that we can use our weak separator from Section 3 to parallelize the algorithm and achieve a logarithmic recursion depth efficiently.

In the remainder of this section, we sketch the backbone phase in Subsection 4.1 and the refinement phase in Subsection 4.2. Together, the lemmas presented in these sections will prove Proposition 16.

4.1 The Backbone Phase

In this section, we sketch the first phase of the algorithm. We begin by giving a general definition of the special type of clustering computed in this phase. Formally, we define it as follows:

▶ **Definition 17** (Backbone Clustering). *Let $\mathcal{D} > 0$ be a distance parameter and $G := (V, E, \ell)$ a (possibly weighted) graph. Then, a (α, β, κ)-backbone clustering is a series of disjoint subgraphs $K(\mathcal{B}_1), \dots, K(\mathcal{B}_N)$ with $K(\mathcal{B}_i) = (V_i, E_i)$ where $V_1 \sqcup \dots \sqcup V_N = V$. Further, the following three conditions hold:*

1. *Each node $v \in K(\mathcal{B}_i)$ is in distance at most $\mathcal{D}$ to $\mathcal{B}_i$.*
2. *An edge of length ℓ is cut between clusters with probability $O\left(\frac{\ell \cdot \alpha}{\mathcal{D}}\right)$.*
3. *Each backbone $\mathcal{B}_i$ consists of at most κ paths of length $O(\beta\mathcal{D})$.*

Given this definition, we show that we can use our algorithm for constructing weak separators to create a good backbone clustering of a graph G. To be precise, we show the following lemma:

▶ **Lemma 18** (Backbone Clustering). *Let $\mathcal{D} > 0$ be a distance parameter and $G := (V, E, \ell)$ a (possibly weighted) k-path separable graph. Then, there is an algorithm that creates an (α, β, κ)-backbone clustering with $\alpha \in O(\log(k \log n))$, $\beta \in O(\log^2 n)$, $\kappa \in O(k \log^2 n)$ and pseudo-diameter $\mathcal{D}$ for G. The algorithm can be implemented with $\tilde{O}(k)$ minor aggregations and $(1+\epsilon)$-approximate SetSSP computations where $\epsilon \in O\left(1/\log^2 n\right)$.*

The algorithm works as follows: Let $G_t \subset G$ be the graph at the beginning of the t^{th} recursive step, where initially $G_0 = G$. At the beginning of each recursive step, we create an LDD $C_1, \dots, C_{N'}$ of diameter $\mathcal{D}' \in O(\mathcal{D} \log^2 n)$ of G_t. We use the algorithm from Theorem 5 for this. The edges between two clusters C_i and C_j are removed and will not be considered in the following iterations. We denote the graph that results from removing all these edges as G'_t. Then, in each C_i, we compute a weak $\tilde{O}(k)$-path $(\mathcal{D}', \epsilon)$-separator $\mathcal{S}_i$ with $\epsilon \in O(1/\log^2 n)$ using the algorithm from Theorem 14. Recall that we defined this separator to be the union of $O(\epsilon^{-1} \cdot k)$ paths of length $4 \cdot \mathcal{D}'$ and some nodes in distance $\epsilon\mathcal{D}'$ to these paths. The $O(\epsilon^{-1} \cdot k)$ paths in $\mathcal{S}_i$ will be the backbone of a cluster. To add further nodes to the cluster, we proceed in two steps: First, we draw $X_i \sim \mathsf{Texp}(4 \log(\log n))$ and compute a $(1+\epsilon)$-approximate SetSSP from $\mathcal{S}_i$. Then, we mark all nodes at a distance at most $(1+\epsilon)X_i \cdot \mathcal{D}_{BC}/4$ from $\mathcal{S}_i$. We denote this set as $K_{\mathsf{Texp}}(\mathcal{S}_i)$. Then, we apply the BBG procedure of Lemma 11 with parameter $\rho_{\mathsf{blur}} := \mathcal{D}_{BC}/c_{\mathsf{blur}} \cdot \log\log n$ from $K_{\mathsf{Texp}}(\mathcal{S}_i)$. Here, $c_{\mathsf{blur}} > 8$ is a suitably chosen constant. This results in a superset $K_{\mathsf{blur}}(\mathcal{S}_i) \subseteq C_i$ for each subgraph C_i. Finally, we add all sets $K_{\mathsf{blur}}(\mathcal{S}_i)$ from all clusters C_i to our clustering $\mathcal{K}[\mathcal{B}]$ and remove them from the graph.

We repeat this process with the remaining graph $G_{t+1} := G'_t \setminus \bigcup K_{\mathsf{blur}}(\mathcal{S}_i)$ until all subgraphs are empty. As we remove a weak $\mathcal{D}'$-separator in each step **and** then create an LDD with strong diameter $\mathcal{D}'$, the size of the largest connected components shrinks by a constant factor in each recursion. Thus, we require $O(\log n)$ recursions overall until all components are empty.

Recall that in each cluster $K_{\mathsf{blur}}(\mathcal{S}_i)$, we define the short paths $P_1, \dots, P_\kappa$ in the separator $\mathcal{S}_i$ to be the backbone. As there are $\kappa \in O(k \cdot \log^2 n)$ paths, it remains to analyze its pseudo-diameter and edge-cutting probability to prove that the resulting clustering is a proper backbone clustering. For an appropriate choice of $\epsilon \in O(1/\log^2 n)$, all nodes in $\mathcal{S}_i$ are in distance $\epsilon\mathcal{D}' \leq \mathcal{D}_{BC}/4$ to any path. Further as $(1+\epsilon) \leq 2$ and $X_i \cdot \mathcal{D}_{BC}/4 \leq \mathcal{D}_{BC}/4$, all nodes in $K_{\mathsf{Texp}}(\mathcal{S}_i)$ are in distance $(1+\epsilon) \cdot X_i \cdot \mathcal{D}_{BC}/4 \leq \mathcal{D}_{BC}/2$ to $\mathcal{S}_i$. Finally, the distance of any node in $K_{\mathsf{blur}}(\mathcal{S}_i)$ to any node in $K_{\mathsf{Texp}}(\mathcal{S}_i)$ is at most $2\rho_{\mathsf{blur}} \leq \mathcal{D}_{BC}/4$. Thus, the distance from each node $u \in K_{\mathsf{blur}}(\mathcal{S}_i)$ to any of the paths in the separator is at most

$$\begin{aligned} d(u, \mathcal{B}_i) &\leq d(v, K_{\mathsf{Texp}}(S_i)) + \mathbf{max}_{v \in K_{\mathsf{Texp}}(S_i)} d(v, S_i) + \mathbf{max}_{w \in S_i} d(w, \mathcal{B}_i) \\ &\leq \mathcal{D}_{BC}/4 + 2 \cdot \mathcal{D}_{BC}/4 + \mathcal{D}_{BC}/4 = 4\mathcal{D}_{BC}/4 = \mathcal{D}_{BC} \end{aligned}$$

Thus, only the edge-cutting probability remains to be shown. For each edge $z \in E$, two operations can cut an edge in each iteration. The LDD from Theorem 5 and the BBG procedure from Lemma 11. First, consider the LDD. As we pick $\mathcal{D}' \in O(\mathcal{D} \log^2 n)$, a single application of Theorem 5 cuts an edge with probability $O\left(\ell_z \log n / \mathcal{D}'\right) = O\left(\ell_z / \mathcal{D} \log n\right)$. Thus, as we apply it at most $O(\log n)$ times, the probability sums up to $O\left(\ell_z / \mathcal{D}\right)$ by the union bound. The other possibility is that z is cut by the BBG. By our choice of ρ_{blur}, the probability for the ball to cut z is $O\left(\ell_z \log(k \log n) / \mathcal{D}\right)$. Therefore, we cannot simply use the union bound to sum up the probabilities over all $O(\log n)$ iterations. Here, the intermediate step where we construct $K_{\mathsf{Texp}}(\mathcal{S}_i)$ comes into play. We can exploit that $K_{\mathsf{Texp}}(\mathcal{S}_i)$ must be *close* to either endpoint of z, i.e., within distance at most $2\rho_{\mathsf{blur}}$, in order for $\mathsf{blur}(K_{\mathsf{Texp}}(\mathcal{S}_i), \rho_{\mathsf{blur}})$ to cut it. The distance between $\mathsf{Texp}(\mathcal{S}_i)$ and z is determined by the exponentially distributed variable $X_i \cdot \mathcal{D}_{BC}/4$. Using the properties of the truncated exponential distribution, we can show that the probability of $\mathsf{Texp}(\mathcal{S}_i)$ being close to z without adding the endpoints of z to $\mathsf{Texp}(\mathcal{S}_i)$ is constant for our choice of parameters. Thus, the edge is safe from ever being cut before the blur procedure is even executed. This follows from the properties of the truncated exponential distribution. Thus, on expectation, there will only be a constant number of tries before z is either cut or safe. As we will see in the full analysis, this is sufficient for our probability bound.

Finally, verifying that each algorithm step requires $\tilde{O}(k)$ approximate SetSSP computations and minor aggregations is easy. We use three subroutines from Theorem 5, Theorem 14, and Lemma 11, which all require at most $\tilde{O}(k)$ approximate SetSSP computations and minor aggregations on a k-path separable graph. In addition, the algorithm only uses one more SetSSP; thus, the runtime follows. A more detailed description and the full proof can be found in the full version [24].

4.2 The Refinement Phase

For the second phase, we show a generic lemma that allows us to turn any backbone clustering as described in Definition 17 into an LDC. Thus, this statement is independent of the algorithm that creates the backbone clustering. To be precise, it holds:

▶ **Lemma 19.** *Suppose we have an algorithm $\mathcal{A}$ that creates a (α, β, κ)-backbone clustering with pseudo-diameter $\mathcal{D}_{BC}$ of a weighted graph G. Then, there is an algorithm that creates an LDC of strong diameter $16 \cdot \mathcal{D}_{BC}$ with quality $(O(\alpha + \log \kappa\beta), 1/2)$. The algorithm requires one application of $\mathcal{A}$, $\tilde{O}(\kappa)$ minor aggregations, and $\tilde{O}(1)$ approximate SetSSP computations with $\epsilon \in O(1/\log^2 n)$.*

The algorithm begins by constructing backbone clustering $K(B_1), \dots, K(B_N)$ using algorithm $\mathcal{A}$. The following steps are executed in parallel in each cluster $K(\mathcal{B}_i)$: On each path $P \in \mathcal{B}_i$ in the backbone, mark a subset of nodes in distance (at most) $\mathcal{D}_{BC}$ to each other and use

them as potential cluster centers. This can be done with one shortest path computation and two aggregations on each path: First, we let each node compute the distance to the first node of the path $v_1 \in P$. A simple SetSSP computation can do this; as there is only one path, the algorithm returns an exact result. Then, we let each node locally compute its *distance class* $\lceil \frac{d_P(v,v_1)}{\mathcal{D}_{BC}} \rceil$. Finally, we mark the first node in each distance class. A node can determine whether it is the first node by aggregating the distance class of both its neighbors in two aggregations. Having marked the nodes, we use them as centers $\mathcal{X}_i$ in the algorithm of Theorem 10 and compute an LDC in $K(\mathcal{B}_i)$. To determine the gurantees of this clustering, we need to determine the *covering* and *packing* properties of these centers. Each node $v \in K(\mathcal{B}_i)$ has at least one marked node in distance $2\mathcal{D}_{BC}$ by construction: It must have a path $P_j \in \mathcal{B}_i$ in distance $\mathcal{D}_{BC}$ and the next marked nodes on the path can also only be in distance $\mathcal{D}_{BC}$ to the node closest to v. Further, as a path of length $\beta\mathcal{D}_{BC}$ has β distance classes, we mark at most β nodes per path. As we limit the number of paths in $\mathcal{B}_i$ to κ and the length of each path to $\beta\mathcal{D}_{BC}$, the total number of centers is limited to $\beta \cdot \kappa$. Thus, each node in $K(\mathcal{B}_i)$ is covered by one center in distance $2\mathcal{D}_{BC}$ and *packed* by at most $\beta \cdot \kappa$ centers in distance $6 \cdot (2\mathcal{D}_{BC})$. Following Theorem 10, this gives us an LDC with strong diameter $8 \cdot (2\mathcal{D}_{BC})$ and quality $O(\log \beta\kappa)$. Finally recall that an edge can either be cut by the initial construction of the backbone clustering or by this procedure. By the union bound, an edge is cut with probability $O(\frac{\alpha+\log \beta\kappa}{\mathcal{D}})$ by either stage. As the algorithm consists of computing a backbone clustering, one SetSSP and one minor aggregations in each path to mark the nodes, and an application of Theorem 10, the proclaimed runtime follows.

Combing this result with Lemma 18, this creates the desired LDC with strong diameter $16\mathcal{D}_{BC}$ of quality $O(\log k + \log\log n)$ in each cluster. In particular, an edge is cut with probability $O(\frac{\ell \cdot \log k \log n}{\mathcal{D}})$ in either of the two phases. This proves the Proposition 16 when choosing $\mathcal{D}_{BC} = \mathcal{D}/16$. A more detailed description and the full proof can be found in the full version [24].

5 Conclusion & Future Work

We presented two novel distributed decomposition schemes for arbitrary and restricted graphs that perform favorably in all relevant parameters and apply to various models. First, for arbitrary graphs, we provided a decomposition scheme that produces decompositions with strong diameter and quality $O(\log n)$. The open question remains whether our algorithm's guarantees can also be achieved deterministically. As our results are deeply connected with the properties of the (truncated) exponential distribution, this likely requires very different techniques and/or novel ideas for derandomization. Second, we presented a decomposition scheme with quality $O(\log k + \log\log n)$ for universally k-path separable graphs, a graphs class containing many practical, relevant graphs. Albeit having a decomposition quality that is exponentially smaller than the quality for general graphs, our algorithm is *not* optimal. First of all, the dependency on n could be improved in future work. Further, for specific k-path separable graphs, the dependency on k could be problematic. For example, in a K_r-free graphs, we have $k = f(r)$ where $f(r)$ depends only on r. We note that $f(r)$, albeit being a constant for a constant r, is extremely large. Its concrete value is based on the Structure Theorem of Seymour and Robinson [54]. Their analysis is quite involved and stretches over a vast series of papers and articles. Moreover, due to its complexity, it is hard even to find an asymptotic characterization of that number. In [49], it is described as a "*tower of powers of* 2 *of height at most* 5*, with* r^{1000} *on top*". We emphasize that this number very likely is too high for practical applications and, therefore, further investigation is required.

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 781–790. IEEE Computer Society, 2008. doi:10.1109/FOCS.2008.62.

2 Ittai Abraham and Cyril Gavoille. Object location using path separators. In Eric Ruppert and Dahlia Malkhi, editors, *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, pages 188–197. ACM, 2006. doi:10.1145/1146381.1146411.

3 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: padded decomposition for minor-free graphs. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 79–88. ACM, 2014. doi:10.1145/2591796.2591849.

4 Ittai Abraham, Cyril Gavoille, Anupam Gupta, Ofer Neiman, and Kunal Talwar. Cops, robbers, and threatening skeletons: Padded decomposition for minor-free graphs. *SIAM J. Comput.*, 48(3):1120–1145, 2019. doi:10.1137/17M1112406.

5 Ittai Abraham and Ofer Neiman. Using petal-decompositions to build a low stretch spanning tree. *SIAM J. Comput.*, 48(2):227–248, 2019. doi:10.1137/17M1115575.

6 Ioannis Anagnostides, Christoph Lenzen, Bernhard Haeupler, Goran Zuzic, and Themis Gouleakis. Almost universally optimal distributed laplacian solvers via low-congestion shortcuts. *Distributed Comput.*, 36(4):475–499, 2023. doi:10.1007/S00446-023-00454-0.

7 John Augustine, Kristian Hinnenthal, Fabian Kuhn, Christian Scheideler, and Philipp Schneider. Shortest paths in a hybrid network model. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1280–1299. SIAM, 2020. doi:10.1137/1.9781611975994.78.

8 Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*, pages 184–193. IEEE Computer Society, 1996. doi:10.1109/SFCS.1996.548477.

9 Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 161–168. ACM, 1998. doi:10.1145/276698.276725.

10 Yair Bartal. Graph decomposition lemmas and their role in metric embedding methods. In Susanne Albers and Tomasz Radzik, editors, *Algorithms - ESA 2004, 12th Annual European Symposium, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3221 of *Lecture Notes in Computer Science*, pages 89–97. Springer, 2004. doi:10.1007/978-3-540-30140-0_10.

11 Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007. doi:10.1002/RSA.20130.

12 Ruben Becker, Yuval Emek, Mohsen Ghaffari, and Christoph Lenzen. Distributed algorithms for low stretch spanning trees. In Jukka Suomela, editor, *33rd International Symposium on Distributed Computing, DISC 2019, October 14-18, 2019, Budapest, Hungary*, volume 146 of *LIPIcs*, pages 4:1–4:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.DISC.2019.4.

13 Ruben Becker, Yuval Emek, and Christoph Lenzen. Low Diameter Graph Decompositions by Approximate Distance Computation. In Thomas Vidick, editor, *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 50:1–50:29, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2020.50.

14 Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Near linear-work parallel sdd solvers, low-diameter decomposition, and low-stretch subgraphs. In *SPAA 2011: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures, San Jose, CA, USA, June 4-6, 2011 (Co-located with FCRC 2011)*, SPAA '11, pages 13–22, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1989493.1989496.

15 Jannik Castenow, Christina Kolb, and Christian Scheideler. A bounding box overlay for competitive routing in hybrid communication networks. In *Structural Information and Communication Complexity - 26th International Colloquium, SIROCCO 2019, L'Aquila, Italy, July 1-4, 2019, Proceedings*, pages 345–348, 2019. doi:10.1007/978-3-030-24922-9_26.

16 Yi-Jun Chang. Efficient distributed decomposition and routing algorithms in minor-free networks and their applications. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 55–66. ACM, 2023. doi:10.1145/3583668.3594604.

17 Yi-Jun Chang and Hsin-Hao Su. Narrowing the LOCAL-CONGEST gaps in sparse networks via expander decompositions. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 301–312. ACM, 2022. doi:10.1145/3519270.3538423.

18 Sam Coy, Artur Czumaj, Christian Scheideler, Philipp Schneider, and Julian Werthmann. Routing schemes for hybrid communication networks. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 317–338. Springer, 2023. doi:10.1007/978-3-031-32733-9_14.

19 Andrzej Czygrinow, Michal Hanckowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In Gadi Taubenfeld, editor, *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*, volume 5218 of *Lecture Notes in Computer Science*, pages 78–92. Springer, 2008. doi:10.1007/978-3-540-87779-0_6.

20 Tijn de Vos. Minimum cost flow in the CONGEST model. In Sergio Rajsbaum, Alkida Balliu, Joshua J. Daymude, and Dennis Olivetti, editors, *Structural Information and Communication Complexity - 30th International Colloquium, SIROCCO 2023, Alcalá de Henares, Spain, June 6-9, 2023, Proceedings*, volume 13892 of *Lecture Notes in Computer Science*, pages 406–426. Springer, 2023. doi:10.1007/978-3-031-32733-9_18.

21 Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg; New York, fourth edition, 2010.

22 Emilie Diot and Cyril Gavoille. Path separability of graphs. In Der-Tsai Lee, Danny Z. Chen, and Shi Ying, editors, *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, volume 6213 of *Lecture Notes in Computer Science*, pages 262–273. Springer, 2010. doi:10.1007/978-3-642-14553-7_25.

23 Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Brief announcement: Distributed construction of near-optimal compact routing schemes for planar graphs. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing, PODC 2023, Orlando, FL, USA, June 19-23, 2023*, pages 67–70. ACM, 2023. doi:10.1145/3583668.3594561.

24 Jinfeng Dou, Thorsten Götte, Henning Hillebrandt, Christian Scheideler, and Julian Werthmann. Distributed and parallel low-diameter decompositions for arbitrary and restricted graphs, 2024. arXiv:2411.19859.

25 Michael Elkin, Yuval Emek, Daniel A. Spielman, and Shang-Hua Teng. Lower-stretch spanning trees. *SIAM J. Comput.*, 38(2):608–628, 2008. doi:10.1137/050641661.

26 Michael Elkin, Arnold Filtser, and Ofer Neiman. Distributed construction of light networks. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, pages 483–492. ACM, 2020. doi:10.1145/3382734.3405701.

27 Michael Elkin and Ofer Neiman. On efficient distributed construction of near optimal routing schemes: Extended abstract. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 235–244. ACM, 2016. doi:10.1145/2933057.2933098.

28 Michael Elkin and Ofer Neiman. Near-optimal distributed routing with low memory. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 207–216, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3212734.3212761.

29 Michael Elkin and Ofer Neiman. Distributed strong diameter network decomposition. *Theor. Comput. Sci.*, 922:150–157, 2022. doi:10.1016/J.TCS.2022.04.019.

30 Michael Elkin, Ofer Neiman, and Shay Solomon. Light spanners. *SIAM J. Discret. Math.*, 29(3):1312–1321, 2015. doi:10.1137/140979538.

31 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. doi:10.1016/j.jcss.2004.04.011.

32 Arnold Filtser. On Strong Diameter Padded Decompositions. In Dimitris Achlioptas and László A. Végh, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019)*, volume 145 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 6:1–6:21, Dagstuhl, Germany, 2019. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.APPROX-RANDOM.2019.6.

33 Arnold Filtser and Ofer Neiman. Light spanners for high dimensional norms via stochastic decompositions. *Algorithmica*, 84(10):2987–3007, 2022. doi:10.1007/S00453-022-00994-0.

34 Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhon. Improved distributed network decomposition, hitting sets, and spanners, via derandomization. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2532–2566. SIAM, 2023. doi:10.1137/1.9781611977554.CH97.

35 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks I: planar embedding. In George Giakkoupis, editor, *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing, PODC 2016, Chicago, IL, USA, July 25-28, 2016*, pages 29–38. ACM, 2016. doi:10.1145/2933057.2933109.

36 Mohsen Ghaffari and Bernhard Haeupler. Distributed algorithms for planar networks II: low-congestion shortcuts, mst, and min-cut. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 202–219. SIAM, 2016. doi:10.1137/1.9781611974331.ch16.

37 Mohsen Ghaffari and Bernhard Haeupler. Low-congestion shortcuts for graphs excluding dense minors. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 213–221. ACM, 2021. doi:10.1145/3465084.3467935.

38 Mohsen Ghaffari, Andreas Karrenbauer, Fabian Kuhn, Christoph Lenzen, and Boaz Patt-Shamir. Near-optimal distributed maximum flow. *SIAM J. Comput.*, 47(6):2078–2117, 2018. doi:10.1137/17M113277X.

39 Mohsen Ghaffari and Merav Parter. Near-optimal distributed DFS in planar graphs. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 21:1–21:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.DISC.2017.21.

40 Mohsen Ghaffari and Goran Zuzic. Universally-Optimal Distributed Exact Min-Cut. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 281–291, Salerno Italy, July 2022. ACM. doi:10.1145/3519270.3538429.

41 Bernhard Haeupler, David Wajc, and Goran Zuzic. Universally-optimal distributed algorithms for known topologies. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 1166–1179. ACM, 2021. doi:10.1145/3406325.3451081.

42 Taisuke Izumi, Naoki Kitamura, Takamasa Naruse, and Gregory Schwartzman. Fully polynomial-time distributed computation in low-treewidth graphs. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 11–22. ACM, 2022. doi:10.1145/3490148.3538590.

43 Fabian Kuhn and Philipp Schneider. Routing Schemes and Distance Oracles in the Hybrid Model. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing (DISC 2022)*, volume 246 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:22, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.DISC.2022.28.

44 Christoph Lenzen and Boaz Patt-Shamir. Fast routing table construction using small messages: extended abstract. In *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 381–390, 2013. doi:10.1145/2488608.2488656.

45 Christoph Lenzen and Boaz Patt-Shamir. Fast partial distance estimation and applications. In Chryssis Georgiou and Paul G. Spirakis, editors, *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 21 - 23, 2015*, pages 153–162. ACM, 2015. doi:10.1145/2767386.2767398.

46 Christoph Lenzen, Boaz Patt-Shamir, and David Peleg. Distributed distance computation and routing with small messages. *Distributed Comput.*, 32(2):133–157, 2019. doi:10.1007/s00446-018-0326-6.

47 Reut Levi, Moti Medina, and Dana Ron. Property testing of planarity in the CONGEST model. *Distributed Comput.*, 34(1):15–32, 2021. doi:10.1007/S00446-020-00382-3.

48 Jason Li and Merav Parter. Planar diameter via metric compression. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 152–163. ACM, 2019. doi:10.1145/3313276.3316358.

49 Daniel Lokshtanov, Saket Saurabh, and Meirav Zehavi. *Efficient Graph Minors Theory and Parameterized Algorithms for (Planar) Disjoint Paths*, pages 112–128. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-42071-0_9.

50 Gary L. Miller, Richard Peng, Adrian Vladu, and Shen Chen Xu. Improved parallel algorithms for spanners and hopsets. In *Proc. of the 27th ACM symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 192–201, 2015. doi:10.1145/2755573.2755574.

51 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In Guy E. Blelloch and Berthold Vöcking, editors, *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, pages 196–203. ACM, 2013. doi:10.1145/2486159.2486180.

52 David Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Society for Industrial and Applied Mathematics, USA, 2000.

53 Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

54 Neil Robertson and Paul D Seymour. Graph minors. xvi. excluding a non-planar graph. *Journal of Combinatorial Theory, Series B*, 89(1):43–76, 2003. doi:10.1016/S0095-8956(03)00042-X.

55 Václav Rozhon, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 1114–1121. IEEE, 2022. doi:10.1109/FOCS54457.2022.00107.

56 Václav Rozhon, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected (1+ϵ)-shortest paths via minor-aggregates: near-optimal deterministic parallel and distributed algorithms. In Stefano Leonardi and Anupam Gupta, editors, *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*, pages 478–487. ACM, 2022. doi:10.1145/3519935.3520074.

57 Philipp Schneider. *Power and limitations of hybrid communication networks*. PhD thesis, University of Freiburg, Freiburg im Breisgau, Germany, 2023. URL: https://freidok.uni-freiburg.de/data/232804.

58 Mikkel Thorup. Compact oracles for reachability and approximate distances in planar digraphs. *J. ACM*, 51(6):993–1024, November 2004. doi:10.1145/1039488.1039493.