Near-Optimal Directed
Low-Diameter Decompositions

Karl Bringmann[a] – Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Germany

Nick Fischer[b] – INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria

Bernhard Haeupler[c] – INSAIT, Sofia University "St. Kliment Ohridski", Bulgaria & ETH Zürich, Switzerland

Rustam Latypov[d] – Aalto University, Finland

Abstract

Low Diameter Decompositions (LDDs) are invaluable tools in the design of combinatorial graph algorithms. While historically they have been applied mainly to undirected graphs, in the recent breakthrough for the negative-length Single Source Shortest Path problem, Bernstein, Nanongkai, and Wulff-Nilsen [FOCS '22] extended the use of LDDs to directed graphs for the first time. Specifically, their LDD deletes each edge with probability at most $O(\frac{1}{D} \cdot \log^2 n)$, while ensuring that each strongly connected component in the remaining graph has a (weak) diameter of at most D.

In this work, we make further advancements in the study of directed LDDs. We reveal a natural and intuitive (in hindsight) connection to Expander Decompositions, and leveraging this connection along with additional techniques, we establish the existence of an LDD with an edge-cutting probability of $O(\frac{1}{D} \cdot \log n \log \log n)$. This improves the previous bound by nearly a logarithmic factor and closely approaches the lower bound of $\Omega(\frac{1}{D} \cdot \log n)$. With significantly more technical effort, we also develop two efficient algorithms for computing our LDDs: a deterministic algorithm that runs in time $\widetilde{O}(m \operatorname{poly}(D))$ and a randomized algorithm that runs in near-linear time $\widetilde{O}(m)$.

We believe that our work provides a solid conceptual and technical foundation for future research relying on directed LDDs, which will undoubtedly follow soon.

1   Introduction

In the design of combinatorial graph algorithms, decomposing graphs into smaller regions where problems become naturally easy to solve has emerged as a remarkably powerful paradigm. Two such graph decompositions stand out as particularly successful: *Low-Diameter Decompositions* (LDDs) and *Expander Decompositions* (EDs) (defined formally in Definition 1.1 and Section 3, respectively). Both decompositions remove some edges from the graph so that the remaining components either have a low diameter and thus behave nicely with respect to distance-type problems (in the case of LDDs), or are expanders which are well-suited for flow-type problems (in the case of EDs). This paradigm is particularly appealing as it typically leads to natural and efficient implementations in *parallel, distributed* and *dynamic* models, and often also to *deterministic* algorithms.

The vast majority of algorithmic results following this broader framework have been established for *undirected* graphs. However, a recently increasing number of papers have successfully applied this paradigm also to *directed* graphs [14, 16, 29, 22, 13], despite the serious technical difficulties that typically arise compared to the undirected setting. One celebrated example is the break-through *near-linear time* algorithm for negative-length Single-Source Shortest Paths by Bernstein, Nanongkai and Wulff-Nilsen [16]. In fact, their work was the first to define and apply Low-Diameter Decompositions in directed graphs.

Given these rapid and recent developments, it is to be expected that many new results will follow in this line of research. Consequently, anticipating a wave of researchers in need of directed Low-Diameter Decompositions, we initiate the systematic study of directed LDDs. Our main technical contribution is a *near-optimal* LDD—near-optimal with respect to the loss parameter and the near-linear running time. Beyond improving the state of the art, we also unveil a compelling and novel connection between directed Low-Diameter Decompositions and Expander Decompositions, bringing together two previously studied concepts that are independently well-established. We believe this connection to be the main take-away of our paper, and are confident that it will inspire further applications.

1.1   Low-Diameter Decompositions

Low-Diameter Decompositions in undirected graphs have been introduced almost 40 years ago by Awerbuch [7], and, have since evolved into an irreplaceable tool in the design of combinatorial algorithms [9, 10, 8, 32, 11, 17, 33, 34, 26, 21, 15, 27, 16]. More generally, LDDs have been utilized in the development of diverse graph-theoretic structures such as oblivious routings [38], hopsets, neighborhood covers, and notably *probabilistic tree embeddings* [11, 12, 25] (where the rough goal is to approximate an arbitrary (graph) metric by a simpler tree metric with polylogarithmic stretch) and *low-stretch spanning trees* [4, 24, 1, 31, 3, 2]. These structures in turn have led to further applications in approximation algorithms, online algorithms and network design problems [11, 18, 28].

While the precise definitions of LDDs differed at first, all of them had in common that the graph was decomposed by removing few edges (on average) so that the remaining connected components have bounded diameter. More precisely, the modern definition is that an LDD is a probability distribution over edge cuts that cut each individual edge with probability at most L/D (for some small loss factor L) so that the resulting connected components have diameter at most D. For undirected graphs, the best-possible loss factor turns out to be $L = \Theta(\log n)$ with natural matching upper and lower bounds (see e.g. [11, 25] and Footnote 1).

Only very recently, low-diameter decompositions debuted in *directed* graphs. They played a key role in the near-linear time algorithm for the Single-Source Shortest Paths problem in graphs with negative edge lengths by Bernstein, Nanongkai and Wulff-Nilsen [16] (following the paradigm

outlined before). Curiously, before their seminal work, low-diameter decompositions for directed graphs were mostly unexplored—to the extent that it was even unclear what the definition should be. An even more recent application of directed LDDs is in the context of restricted (i.e., bicriteria) shortest paths [6].

Definition 1.1 (Directed Low-Diameter Decomposition). *A directed low-diameter decomposition with* loss L *for a directed edge-weighted graph* $G = (V, E, \ell)$ *and a parameter* $D \geq 1$ *is a probability distribution over edge sets* $S \subseteq E$ *that satisfies the following two properties:*

- *For any two nodes* $u, v \in V$ *that are part of the same strongly connected component in* $G \setminus S$, *we have* $d_G(u, v) \leq D$ *and* $d_G(v, u) \leq D$.

- *For all edges* $e \in E$, *it holds that* $\Pr(e \in S) \leq \frac{\ell(e)}{D} \cdot L$.

That is, for directed graphs we require that all remaining *strongly* connected components have diameter D in the sense that each pair of nodes is at distance at most D in the original uncut graph (this property is also called a "weak" diameter guarantee; see more details in the paragraph towards the end of Section 1.1).

In light of this definition, it is natural to study directed LDDs with respect to two objectives: Minimizing the loss factor L, and *computing* the LDD efficiently. Both of these requirements typically translate immediately to algorithmic improvements (e.g., the loss factor L typically becomes a factor in the running time of algorithmic applications).

Quest 1: Minimizing the Loss L. It is easy to see that a loss factor of $L \geq 1$ is necessary: Consider a graph consisting of disjoint copies of $D+1$-cycles (of unit length, say). Any LDD is forced to cut at least one edge from every cycle, and thus some edges are deleted with probability $\frac{1}{D+1}$. In fact, from the same lower bound as for undirected graphs one can show that $L \geq \Omega(\log n)$ is necessary[1] for some directed graphs.

Conversely, Bernstein, Nanongkai and Wulff-Nilsen [16] showed that a loss of $O(\log^2 n)$ can be achieved. Their work leaves open whether this factor can be improved, possibly to $O(\log n)$ which would match the existing unconditional lower bound, or whether $\Omega(\log^2 n)$ is necessary.

Quest 2: Efficient Algorithms. To be useful in algorithmic applications, it is necessary to be able to compute the LDD (or, formally speaking, to be able to *sample* from the LDD efficiently). This is a non-negligible problem as many graph decompositions are much simpler to prove existentially than constructively and efficiently—for instance, for Expander Decompositions there is a stark contrast between the extremely simple existential proof and the involved efficient algorithms based on cut-matching games [35, 30]. The $O(\log^2 n)$-loss due to [16] is of course efficient.

Side Quest: Weak versus Strong. Another dimension is the distinction between "weak" and "strong" diameter guarantees. Specifically, Definition 1.1 requires the weak guarantee by requiring that any two nodes in the same strongly connected component are at distance at most D *in the*

---

[1]Specifically, using e.g., the probabilistic method, one can construct an undirected n-node graph G with the following two properties: (i) the graph contains cn edges (for some constant $c > 1$), and (ii) the girth of G (i.e., the length of the shortest cycle) is at least $g = \Omega(\log n)$. Such a graph constitutes an obstruction against undirected LDDs with parameter $D = \frac{g}{2} - 1$, say. Indeed, all connected components that remain after the LDD cuts edges cannot contain cycles (as any cycle contains two nodes at distance at least $\frac{g}{2} > D$). This means that all connected components are trees, and thus the remaining graph is a forest containing at most $n - 1$ edges. Therefore, the LDD has cut at least $cn - n = \Omega(n)$ edges. In particular, the per-edge cutting probability of some edges must be $\Omega(1) = \Omega(\log n/D)$. The same argument applies for directed LDDs by taking the same undirected graph and making all edges bidirected.

original graph G. The strong version instead requires that each strongly connected component in $G \setminus S$ has diameter at most D *in the graph $G \setminus S$.* While strong LDDs give a theoretically more appealing guarantee, for most algorithmic applications it turns out that weak LDDs suffice. The LDD developed by Bernstein, Nanongkai and Wulff-Nilsen [16] has a weak guarantee, but follow-up work [19] later extended their results to a strong LDD with cubic loss $O(\log^3 n)$.

In this paper, we will mostly ignore the distinction between weak and strong and follow Definition 1.1 as it is. We remark however that all of our existential results do in fact have the strong diameter guarantee.

2 Results and Technical Overview

In a nutshell, our result is that we establish a directed LDD with *near-optimal* loss $O(\log n \log \log n)$. This comes tantalizingly close to the unconditional lower bound of $\Omega(\log n)$. It resembles a similar milestone in the development of probabilistic tree embeddings [12], and also the current state of the art for low-stretch spanning trees [3]. In fact, similar $\log \log n$ barriers show up also in different contexts for directed graphs [20]. In a sense, all of these results with $\log \log n$ overhead (including ours) apply a careful recursive approach that can be traced back to work by Seymour [36] (though with strongly varying implementations depending on the setting).

We state our result in three Theorems 2.1, 2.7 and 2.8, where the first theorem is purely existential, the latter two are algorithmic, and the last has *near-linear* running time.

2.1 Near-Optimal LDDs via Expander Decompositions

The main conceptual contribution of our paper is that Low-Diameter Decompositions are closely and in a very practical way related to Expander Decompositions. Based on this insight, we prove the following theorem.

Theorem 2.1. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$.*

For the remainder of Section 2.1 we will elaborate on the proof of Theorem 2.1. It involves two separate steps: reducing the problem to *cost-minimizing* using the *Multiplicative Weight Update* method, and then showing that a so called *lopsided* expander decomposition is the desired *cost-minimizer*. We emphasize that in Section 2.1 we only focus on Quest 1 from the introduction, which is minimizing the loss L of an LDD.

Reduction to Unit Lengths. Let us assume throughout that we only deal with unit-length graphs (i.e., where $\ell(e) = 1$ for all edges). This assumption is in fact without loss of generality, as we can simply replace each edge e of length $\ell(e)$ by a path of $\ell(e)$ unit-length edges. This transformation may blow up the graph, but as we focus on the existential result first this shall not concern us.[2] Whenever the LDD cuts at least one of the path edges in the transformed graph, we imagine that the entire edge in the original graph is cut. This way, when we cut each edge with probability at most $\frac{L}{D}$, in the original graph we cut each edge with probability at most $\frac{\ell(e) \cdot L}{D}$ (by a union bound) as required.

---

[2]We are assuming throughout that all edge weights are bounded by $\mathrm{poly}(n)$, therefore this transformation leads to a graph on at most $n_0 \leq \mathrm{poly}(n)$ nodes. In particular, to achieve a loss of $L = O(\log n \log \log n)$ on the original graph it suffices to achieve a loss of $O(\log n_0 \log \log n_0)$ on the transformed graph.

Reduction to Cost-Minimization. Perhaps it appears surprising that we claim a connection between LDDs—which are inherently *probabilistic* objects—and EDs—which rather have a *deterministic* flavor. As a first step of bringing these notions together consider the following lemma based on the well-studied Multiplicative Weight Update [5] method.

Lemma 2.2 (Multiplicative Weight Update). *Let $G = (V, E)$ be a directed graph, and suppose that for all cost functions $c : E \to [|V|^{10}]$ there is a set of cut edges $S \subseteq E$ satisfying the following two properties:*

- *For any two nodes $u, v \in V$ that are part of the same strongly connected component in $G \setminus S$, we have $d_G(u, v) \leq D$ and $d_G(v, u) \leq D$.*

- $c(S) \leq c(E) \cdot \frac{L}{D}$.

Then there exists a directed LDD for G with loss $O(L)$.

Intuitively, the lemma states that in order to construct an LDD that cuts each edge with small probability L/D, it suffices to instead find a cut which *globally* minimizes the total cost of the cut edges while ensuring that every remaining strongly connected component has diameter at most D. This however comes with the price of introducing a *cost function c* to the graph, and the goal becomes to cut edges which collectively have at most an L/D fraction of the total cost. For the reader's convenience, we include a quick proof sketch of Lemma 2.2.

Proof. TOPROVE 0 □

Henceforth, we refer to the process of finding a cut with the properties stated in Lemma 2.2 as a *cost-minimizer*, which we aim to construct in the following paragraphs. That is, we now also consider a cost function c, and our goal is to cut edges S of total cost $c(S) \leq c(E) \cdot \frac{L}{D}$ (without worrying about a per-edge guarantee), while ensuring that the strongly connected components in $G \setminus S$ have diameter $\leq D$. We remark that this cost-minimization framework is quite standard.

Directed Expander Decomposition. In a leap forward, let us rename the *costs c* to *capacities c*. Our idea is simple: We want to use an Expander Decomposition to remove edges of small total capacity so that all strongly connected components in the graph become expanders and thus, in particular, have small diameter.

To make this more precise, we first introduce some (standard) notation. A node set $U \subseteq V$ naturally induces a *cut* (between U and $\overline{U} = V \setminus U$). We write $c(U, \overline{U})$ for the total capacity of edges crossing the cut from U to $\overline{U}$, and define the *volume* of U as

$$\text{vol}(U) = c(U, V) = \sum_{e \in E \cap (U \times V)} c(e),$$

and also write $\text{minvol}(U) = \min\{\text{vol}(U), \text{vol}(\overline{U})\}$. The *sparsity* of the cut U is defined by

$$\phi(U) = \frac{c(U, \overline{U})}{\text{minvol}(U)}.$$

We say that U is *ϕ-sparse* if $\phi(U) \leq \phi$, and we call a graph without ϕ-sparse cuts a *ϕ-expander*. The standard Expander Decomposition can be stated as follows.

Lemma 2.3 (Directed Expander Decomposition). *Let $\phi > 0$. For any directed graph $G = (V, E, c)$ there is an edge set $S \subseteq E$ of total capacity $c(S) \leq c(E) \cdot \phi \log c(E)$ such that every strongly connected component in the remaining graph $G \setminus S$ is a ϕ-expander.*

4

Moreover, an important property of a ϕ-expander decomposition is that the diameter of its strongly connected components depends on ϕ in the following manner.

Lemma 2.4. *Any ϕ-expander has diameter $O(\phi^{-1} \log \text{vol}(V))$.*

To understand the intuition behind Lemma 2.4, imagine that we grow a ball (i.e., a breadth-first search tree) around some node. With each step, the expansion property (related to the absence of ϕ-sparse cuts) guarantees that we increase the explored capacity by a factor of $(1 + \phi)$; thus after $O(\phi \log \text{vol}(V))$ steps we have explored essentially the entire graph.

Already at this point we have made significant progress and can recover the $O(\log^2 n)$-loss LDD: Apply the Expander Decomposition in Lemma 2.3 with parameter $\phi = \log \text{vol}(V)/D$. This removes edges with total capacity at most a $O(\log^2 \text{vol}(V)/D)$-fraction of the total capacity. In the remaining graph each strongly connected component is a ϕ-expander and thus, by Lemma 2.4, has diameter at most $O(\phi^{-1} \log \text{vol}(V)) = O(D)$ (by choosing the constants appropriately, this bound can be adjusted to $\leq D$). Finally, Lemma 2.2 turns this into an LDD with loss $O(\log^2 \text{vol}(V)) = O(\log^2 n)$.

Unfortunately, both log-factors in the Expander Decomposition (fraction of the cut capacity and diameter) are tight. Nevertheless, with some innovation we manage to bypass these bounds and improve the $O(\log^2 n)$ loss. To this end we propose a refined notion of expanders called *lopsided expanders*.

Lopsided Expander Decomposition. The notion of ϕ-sparsity defined above is oblivious to the ratio between $\text{vol}(V)$ and $\text{minvol}(U)$. For example, two cuts U and W can have the same sparsity, even though $\text{vol}(U) \gg \text{vol}(\overline{U})$ while $\text{vol}(W) \approx \text{vol}(\overline{W})$. It turns out that this leaves some unused potential, and that we should incentivize cutting cuts with large volume on both sides compared to more lopsided cuts with large volume only on one side.

Formally, we define ψ-*lopsided sparsity* of a cut U as

$$\psi(U) = \frac{c(U, \overline{U})}{\text{minvol}(U) \cdot \log \frac{\text{vol}(V)}{\text{minvol}(U)}},$$

where we include the ratio $\frac{\text{vol}(V)}{\text{minvol}(U)}$ in the denominator. Since $\text{vol}(V) > \text{minvol}(U)$, a cut can only have smaller lopsided sparsity than regular sparsity, so a graph with no sparse cuts may still have lopsided sparse cuts. A ψ-lopsided expander is defined as a graph with no ψ-lopsided sparse cuts, and can be thought of as a subclass of expanders which in addition to having no sparse cuts also has no cuts that are both sufficiently lopsided and sufficiently sparse.

The *lopsided expander decomposition* is otherwise identical to the standard expander decomposition defined previously, except that every strongly connected component is required to be a ψ-*lopsided* expander instead of a ϕ-expander. Our Lopsided Expander Decomposition has the same global "total capacity cut" guarantee as the standard expander decomposition, as stated in the following lemma coupled with a proof sketch.

Lemma 2.5 (Lopsided Expander Decomposition). *Let $\psi > 0$. For any directed graph $G = (V, E, c)$ there is an edge set $S \subseteq E$ of total capacity $c(S) \leq c(E) \cdot \psi \log c(E)$ such that every strongly connected component in the remaining graph $G \setminus S$ is a ψ-lopsided expander.*

Proof. TOPROVE 1 □

The main motivation for defining ψ-lopsided sparsity and taking the ratio $\frac{\text{vol}(V)}{\text{minvol}(U)}$ into account lies in the following lemma: Compared to standard expanders, lopsided expanders only suffer a loglog-factor in the diameter.

Lemma 2.6. *Any ψ-lopsided expander has diameter $O(\psi^{-1} \log \log \text{vol}(V) + \log \text{vol}(V))$.*

Proof. TOPROVE 2 □

Putting these two lemmas for lopsided expanders together, we indeed obtain the low-loss LDD in Theorem 2.1. Specifically, we apply the Lopsided Expander Decomposition from Lemma 2.5 with parameter $\psi = \log \log \text{vol}(V)/D$. We thereby cut only a $O(\log \text{vol}(V) \log \log \text{vol}(V)/D)$-fraction of the total capacity, and end up with a graph in which every strongly connected component is a ψ-lopsided expander. By Lemma 2.6 said components have diameter $O(\psi^{-1} \log \log \text{vol}(V)) = O(D)$ (which, again, can be made $\leq D$ by adjusting constants). Plugging this procedure into Lemma 2.2 we conclude that there is an LDD with loss $O(\log \text{vol}(V) \log \log \text{vol}(V)) = O(\log n \log \log n)$, completing the proof sketch of Theorem 2.1.

2.2 A Deterministic Algorithm

So far we have neglected Quest 2, i.e., the design of efficient algorithms. But how far from algorithmic is this approach of Section 2.1 really? It turns out that implementing this framework with some simple tricks leads to the following algorithmic result.

Theorem 2.7. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$ and support size $O(D \log n)$ that can be computed in time $\widetilde{O}(m \, \text{poly}(D))$ by a deterministic algorithm.*

This theorem comes with a strength and a weakness—the strength is that the theorem is deterministic. Note that the algorithm produces the *explicit* support of an LDD; in fact, the probability distribution is simply a uniform distribution over a given list of cut sets S. For this reason, our algorithm might lead to some derandomized applications down the road by executing an LDD-based algorithm one-by-one for all $O(D \log n)$ cut sets. Derandomizations of this sort are common and sought after, especially in distributed and parallel domains. The weakness is that the running time has an overhead of $\text{poly}(D)$. We remark that almost all algorithmic applications of LDDs (with the notable exception of [16]) set $D = \text{polylog}(n)$ or $D = n^{o(1)}$, in which case the overhead is not dramatic, and the runtime is possibly near-linear.

Proof Idea. It can be quickly verified that implementing the Multiplicative Weights Update method from Lemma 2.2 is not a problem algorithmically, and requires $O(R \cdot T)$ time, where $R = O(\log |E| \cdot D/L)$ and T is the time required by the cost-minimizer. Hence, only computing the Lopsided Expander Decomposition could be costly. Since this is a strictly stronger definition that the standard Expander Decomposition, at first glance it appears hopeless that we could achieve a simple algorithm that avoids the cut-matching games machinery. Luckily, we find yet another simple insight that allows us to find a simple and intuitive algorithm after all: While it may generally be hard to find a sparse cut (even NP-hard!), in our case we only ever need to find a sparse cut *in a graph with diameter more than D*. Indeed, if the graph has already diameter at most D, we can stop immediately (recall that we ultimately only care about the diameter and not the expander guarantee). One way to view the algorithm is that we construct a *truncated* ψ-lopsided expander decomposition, where we terminate prematurely if the diameter is small.

Fueled by this insight, consider the following two-phase process for computing a (truncated) ψ-lopsided expander decomposition:

- *Phase (I):* We repeatedly take an arbitrary node v and grow a ball around v in the hope of finding a ψ-lopsided sparse cut. By choosing $\psi = \Theta(\log \log \text{vol}(V)/D)$ we can guarantee two possible outcomes:

(a) We find a sparse cut after, say, $\frac{D}{4}$ steps, in which case we cut the edges crossing the cut and recur on both sides (as in the Lopsided Expander Decomposition).

(b) The problematic case happens: the volume of the ball reaches $\frac{3}{4} \cdot \mathrm{vol}(V)$. In this case we have spent linear time but made no progress in splitting off the graph, so we remember v and move to Phase (II).

- *Phase (II):* We compute a *buffer zone* around v, which consists of all nodes with distance $\frac{D}{2}$ to and from v. We repeatedly take an arbitrary node u from outside this buffer zone and grow a ball around u in the hope of finding a ψ-lopsided sparse cut. Because we know that the $\frac{D}{4}$-radius ball around v contains most of the volume of the graph, and node u is picked outside of the $\frac{D}{2}$-radius buffer zone, we can prove (using similar techniques to the proof sketch of Lemma 2.6) that we will always find a ψ-lopsided sparse cut U around u. Upon finding it, we cut the edges crossing the cut $(U, \overline{U})$ and start Phase (I) for the graph induced by nodes in U, while continuing Phase (II) in graph $G \setminus U$.

  When eventually there are no nodes left outside the buffer zone, we stop, since the remaining graph has diameter $\leq D$.

The correctness of the procedure above is straightforward to show: we are essentially constructing a ψ-lopsided expander decomposition from Lemma 2.5, but terminating prematurely if the diameter is small. Hence, the total capacity of the cut edges is at most the total capacity of the cut edges in a "complete" ψ-lopsided expander, which is a $O(\log \mathrm{vol}(V) \log \log \mathrm{vol}(V)/D)$-fraction of the total capacity. The diameter guarantee follows directly from the stopping condition. By plugging this procedure into the algorithmic version of Lemma 2.2 we arrive at an LDD with loss $O(\log \mathrm{vol}(V) \log \log \mathrm{vol}(V)) = O(\log n \log \log n)$ in time $\widetilde{O}(m \, \mathrm{poly}(D))$, completing the proof sketch of Theorem 2.7.

2.3 A Near-Optimal Randomized Algorithm

Our third and final contribution is achieving a near-linear running time, regardless of the magnitude of the diameter, by a randomized algorithm:

Theorem 2.8. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$ which can be computed (i.e., sampled from) in expected time $\widetilde{O}(m)$.*

In contrast to Theorems 2.1 and 2.7, our approach to Theorem 2.8 is rather technical. We try more directly to extend the ideas of Bernstein, Nanongkai and Wulff-Nilsen [16] (which in turn borrow ideas from [15]). On a very high level, their algorithm classifies nodes v *heavy* if it reaches more than $\frac{n}{2}$ nodes within distance $\frac{D}{2}$, or *light* otherwise. We can afford to cut around light nodes v (with a specifically sampled radius $r \leq D$), and recur on both sides of the cut—since v is light, the inside of the cut reduces by a constant fraction which is helpful in bounding the recursion depth. And if there are only heavy nodes in the graph then we can stop as the diameter is already at most D—indeed, the radius-$\frac{D}{2}$ balls around heavy nodes must necessarily intersect. The radius is sampled from a geometric distribution with rate $O(\log n/D)$ inspired by classical LDDs in undirected graphs [11]. The two logarithmic factors stem from (i) this $\log n$ overhead in the sampling rate, and (ii) the logarithmic recursion depth.

Our approach refines these ideas by classifying not into two classes—heavy or light—but into $\log \log n$ different levels. We essentially say that a node v is at level ℓ if it reaches roughly $n/2^{2^\ell}$ nodes within some distance roughly D. We can take advantage of this in two ways: At the one extreme, we can only cut around few nodes of small level ℓ. Indeed, when we cut around a node at level ℓ we remove roughly $n/2^{2^\ell}$ nodes from the graph, so this can be repeated at most $2^{2^\ell} \ll n$ times.

7

For these levels we can afford to sample the radius in the geometric distribution with a significantly smaller rate, $O(2^i/D)$, thereby improving (i). At the other extreme, whenever we cut around nodes with large level ℓ then in the recursive call of the inside of the cut there are only $n/2^{2^\ell} \ll n$ nodes. This effectively reduces the recursion depth of these levels to much less than $\log n$, improving (ii). In our algorithm, we find a balance between these two extreme cases.

Unfortunately, while this idea works out existentially, there are several issues when attempting to implement this approach in near-linear time. The first issue—initially classifying which nodes are at what level in time $\widetilde{O}(m)$—can be solved by an algorithm due to Cohen [23]. However, over the course of the algorithm when we repeatedly remove nodes and edges from the graphs the level of a node might *change*. Our solution involves a cute trick: First we prove that during the execution of our algorithm the level of a node can only *increase* (and never decrease). Second, instead of cutting around an arbitrary node v, we always pick a *random* node. The argument is as follows: If the classification is still correct for at least half of the nodes, then in each step we make progress with probability $\frac{1}{2}$. And otherwise we can in fact afford to reclassify by Cohen's algorithm as the level of the nodes can only increase a small number of times, leading to only few repetitions of Cohen's algorithm in total.

We omit further details here and refer to the technical Section 6.

3 Preliminaries

Before diving into the technical results, we state the basic graph notations used throughout the paper and recap the new non-standard definitions we have introduced throughout Section 2.

Graphs. Throughout we consider directed simple graphs $G = (V, E)$, where $E \subseteq V^2$, with $n = |V|$ nodes and $m = |E|$ edges. The edges of the graph can be associated with some value: a length $\ell(e)$ or a capacity/cost $c(e)$, all of which we require to be positive. For any $U \subseteq V$, we write $\overline{U} = V \setminus U$. Let $G[U]$ be the subgraph induced by U. We denote with $\delta^+(U)$ the set of edges that have their starting point in U and endpoint in $\overline{U}$. We define $\delta^-(U)$ symmetrically. We also sometimes write $c(S) = \sum_{e \in S} c(e)$ (for a set of edges S) or $c(U, W) = \sum_{e \in E \cap (U \times W)} c(e)$ and $c(U) = c(U, U)$ (for sets of nodes U, W).

The distance between two nodes v and u is written $d_G(v, u)$ (throughout we consider only the *length* functions to be relevant for distances). We may omit the subscript if it is clear from the context. The diameter of the graph is the maximum distance between any pair of nodes. For a subgraph G' of G we occasionally say that G' has *weak diameter* D if for all pairs of nodes u, v in G', we have $d_G(u, v), d_G(v, u) \le D$. A strongly connected component in a directed graph G is a subgraph where for every pair of nodes v, u there is a path from v to u and vise versa. Finally, for a radius $r \ge 0$ we write $B^+(v, r) = \{x \in V : d_G(v, x) \le r\}$ and $B^-(v, r) = \{y \in V : d_G(y, v) \le r\}$.

Polynomial Bounds. For graphs with edge lengths (or capacities), we assume that they are positive and the maximum edge length is bounded by $\mathrm{poly}(n)$. This is only for the sake of simplicity in Sections 4 and 5 (where in the more general case that all edge lengths are bounded by some threshold W some logarithmic factors in n become $\log(nW)$ instead), and is not necessary for our strongest LDD developed in Section 6.

Expander Graphs. Let $G = (V, E, \ell, c)$ be a directed graph with positive edge capacities c and positive unit edge lengths ℓ. We define the *volume* $\mathrm{vol}(U)$ by

$$\mathrm{vol}(U) = c(U, V) = \sum_{e \in E \cap (U \times V)} c(e),$$

and set $\mathrm{minvol}(U) = \min\{\mathrm{vol}(U), \mathrm{vol}(\overline{U})\}$ where $\overline{U} = V \setminus U$. A node set U naturally corresponds to a cut $(U, \overline{U})$. The *sparsity* (or *conductance*) of U is defined by

$$\phi(U) = \frac{c(U, \overline{U})}{\mathrm{minvol}(U)}.$$

In the special cases that $U = \emptyset$ we set $\phi(U) = 1$ and in the special case that $U \neq \emptyset$ but $\mathrm{vol}(U) = 0$, we set $\phi(U) = 0$. We say that U is ϕ-*sparse* if $\phi(U) \leq \phi$. We say that a directed graph is a ϕ-expander if it does not contain a ϕ-sparse cut $U \subseteq V$. We define the *lopsided sparsity* of U as

$$\psi(U) = \frac{c(U, \overline{U})}{\mathrm{minvol}(U) \cdot \log \frac{\mathrm{vol}(V)}{\mathrm{minvol}(U)}},$$

(with similar special cases), and we similarly say that U is ψ-*lopsided sparse* if $\psi(U) \leq \psi$. Finally, we call a graph a ψ-*lopsided expander* if it does not contain a ψ-lopsided sparse cut $U \subseteq V$.

4 Near-Optimal LDDs via Expander Decompositions

In this section, we show the existence of a near-optimal LDD, thereby proving our first main theorem:

Theorem 2.1. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$.*

We introduce some technical lemmas in order to build up the framework for the proof of Theorem 2.1, which can be found in the end of the section.

4.1 Reduction to Cost-Minimizers

Lemma 2.2 (Multiplicative Weight Update). *Let $G = (V, E)$ be a directed graph, and suppose that for all cost functions $c : E \to [|V|^{10}]$ there is a set of cut edges $S \subseteq E$ satisfying the following two properties:*

- *For any two nodes $u, v \in V$ that are part of the same strongly connected component in $G \setminus S$, we have $d_G(u, v) \leq D$ and $d_G(v, u) \leq D$.*

- *$c(S) \leq c(E) \cdot \frac{L}{D}$.*

Then there exists a directed LDD for G with loss $O(L)$.

Proof. TOPROVE 3 □

Lemma 2.5 (Lopsided Expander Decomposition). *Let $\psi > 0$. For any directed graph $G = (V, E, c)$ there is an edge set $S \subseteq E$ of total capacity $c(S) \leq c(E) \cdot \psi \log c(E)$ such that every strongly connected component in the remaining graph $G \setminus S$ is a ψ-lopsided expander.*

Proof. TOPROVE 4 □

To prove that lopsided expanders have small diameter, we first establish the following technical lemma.

Lemma 4.1. *Let $G = (V, E, c)$ be a directed graph and let $\psi > 0$. For any node $v \in V$ there is some radius $R = O(\psi^{-1} \log \log \mathrm{vol}(V) + \log \mathrm{vol}(V))$ such that one of the following two properties holds:*

- *$\mathrm{vol}(B^+(v, R)) \geq \frac{1}{2} \cdot \mathrm{vol}(V)$, or*

- *$\psi(B^+(v, r)) \leq \psi$ for some $0 \leq r \leq R$.*

Proof. TOPROVE 5 ☐

One can easily strengthen the lemma as follows. This insight will play a role in the next Section 5 in the construction of the deterministic algorithm.

Lemma 4.2. *Let $G = (V, E, c)$ be a directed graph and let $\psi > 0$ and $0 < \alpha < 1$. For any node $v \in V$ there is some radius $R = O(\psi^{-1} \log \log \mathrm{vol}(V) + \psi^{-1} \alpha^{-1} + \log \mathrm{vol}(V))$ such that one of the following two properties holds:*

- *$\mathrm{vol}(B^+(v, R)) \geq (1 - \alpha) \cdot \mathrm{vol}(V)$, or*

- *$\psi(B^+(v, r)) \leq \psi$ for some $0 \leq r \leq R$.*

Proof. TOPROVE 6 ☐

Lemma 2.6. *Any ψ-lopsided expander has diameter $O(\psi^{-1} \log \log \mathrm{vol}(V) + \log \mathrm{vol}(V))$.*

Proof. TOPROVE 7 ☐

Proof. TOPROVE 8 ☐

5 Near-Optimal LDDs Deterministically

In this section, we present the deterministic algorithm for computing a near-optimal LDD, thereby proving our second main theorem:

Theorem 2.7. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$ and support size $O(D \log n)$ that can be computed in time $\widetilde{O}(m \, \mathrm{poly}(D))$ by a deterministic algorithm.*

To this end, we utilize many of the same building blocks we have already introduced in Section 4. In particular, we follow the general framework of Multiplicative Weights Update to reduce the computation of an LDD to solving the cost-minimizing task. The full proof of Theorem 2.7 can be found in the end of the section. The following lemma restates the MWU method algorithmically; we omit a proof as it follows exactly the proof of Lemma 2.2 in Section 4.

Lemma 5.1 (Algorithmic Multiplicative Weight Update). *Let $G = (V, E)$ be a directed graph and let $D \geq 1$. Suppose that there is an algorithm $\mathcal{A}$ that, given G, D and a cost function $c : E \to [|V|^{10}]$, computes a set of edges $S \subseteq E$ satisfying the following properties:*

- *For any two nodes $u, v \in V$ that are part of the same strongly connected component in $G \setminus S$, we have $d_G(u, v) \leq D$ and $d_G(v, u) \leq D$.*

- *$c(S) \leq c(E) \cdot \frac{L}{D}$.*

Then there is a deterministic algorithm to compute an LDD with loss $O(L)$ for G (i.e., we compute the full support of a uniform distribution over $O(D \log n)$ cut sets). It runs in time $\widetilde{O}(mD)$ and issues $O(D \log n)$ oracle calls to $\mathcal{A}$.

For the remainder of this section we will therefore focus on the same cost minimizer setting: Given a directed graph $G = (V, E, c)$ with edge capacities (and unit lengths), the goal is select a set of cut edges $S \subseteq E$ such that $c(S) \leq c(E) \cdot O(\frac{1}{D} \cdot \log n \log \log n)$ and such that all strongly connected components in the remaining graph $G \setminus S$ have (weak) diameter at most D.

The following lemma is a consequence of the lopsided expander machinery set up before:

Lemma 5.2 (Finding Sparse Cuts). *Let $G = (V, E, c)$ be a directed graph, let $D \geq \log \mathrm{vol}(V)$ and let $v \in V$. Then there is some $\psi = O(\frac{1}{D} \cdot \log \log \mathrm{vol}(V))$ and an algorithm to determine which of the following cases applies:*

(i) *There is a radius $0 \leq r \leq D$ with $\psi(B^+(v, r)) \leq \psi$ and $c(B^+(v, r)) \leq 0.95 \cdot \mathrm{vol}(V)$.*
   (In this case the algorithm runs in linear time in the number of edges incident to $B^+(v, r)$.)

(ii) *Or, there is a radius $0 \leq r \leq D$ such that $\psi(\overline{B^-(v, r)}) \leq \psi$ and $c(B^-(v, r)) \leq 0.95 \cdot \mathrm{vol}(V)$.*
   (In this case the algorithm runs in linear time in the number of edges incident to $B^-(v, r)$.)

(iii) *Or, $c(B^+(v, D) \cap B^-(v, D)) \geq 0.9 \cdot \mathrm{vol}(V)$.*
   (In this case the algorithm runs in time $O(m)$.)

Proof. TOPROVE 9 □

Having established Lemma 5.2, now consider the algorithm in Algorithm 1. In summary, it runs in two phases. In Phase (I) we first repeatedly select a node v and attempt to cut a lopsided sparse cut around v (i.e., we cut the edges in $\delta^+(B^+(v, r))$ for some radius r). We only execute these cuts, however, until we find a node z for which $c(B^+(z, D') \cap B^-(z, D')) \geq 0.9 \cdot \mathrm{vol}(V)$—that is, both the radius-D' out- and in-balls of z make up for a big constant fraction of the entire graph. We call z a *center* node and move on to phase (II). In this phase we repeat the same steps as in Phase (I), but we only choose nodes v that have distance at least $2D'$ (in one direction or the other) to the center z. The intuition is that we can never find a second node z' which equally makes up for the entire graph, as then z and z' would have to be connected by a short path. In the remainder of this section we formally analyze Algorithm 1.

Lemma 5.3 (Total Cost of Algorithm 1). *Let $S \subseteq E$ denote the set of edges cut by Algorithm 1. Then $c(S) \leq c(E) \cdot O(\frac{1}{D} \cdot \log \mathrm{vol}(V) \log \log \mathrm{vol}(V))$.*

Proof. TOPROVE 10 □

Lemma 5.4 (Well-Definedness of Algorithm 1). *While executing Phase (II) of Algorithm 1, the subcase (iii) never happens.*

Proof. TOPROVE 11 □

Lemma 5.5 (Correctness of Algorithm 1). *Let $S \subseteq E$ denote the set of edges cut by Algorithm 1. Then for any two nodes u, w in the same strongly connected component in $G \setminus S$, it holds that $d_G(u, w) \leq D$ and $d_G(w, u) \leq D$.*

Proof. TOPROVE 12 □

Lemma 5.6 (Running Time of Algorithm 1). *Algorithm 1 runs in time $O(m \log \mathrm{vol}(V))$.*

Algorithm 1 The deterministic near-optimal LDD, see Theorem 2.7.

(I) Repeat the following steps: Take an arbitrary node $v \in V$ and apply Lemma 5.2 with parameter $D' = \lfloor \frac{D}{4} \rfloor$. Depending on the output execute the following steps:

    (i) Cut all edges in $\delta^+(B^+(v, r))$, recurse on the induced graph $G[B^+(v, r)]$, then remove all nodes in $B^+(v, r)$ from the graph.

    (ii) Cut all edges $\delta^-(B^-(v, r))$, recurse on the induced graph $G[B^-(v, r)]$, then remove all nodes in $B^-(v, r)$ from the graph.

    (iii) Remember $z \leftarrow v$ (called the *center* node) and continue with Phase (II).

(II) Compute the sets

$$X = B^+(z, D') \cap B^-(z, D'),$$
$$Y = B^+(z, 2D') \cap B^-(z, 2D').$$

Then repeat the following steps while there still exists nodes in $V \setminus Y$: Take an arbitrary node $u \in V \setminus Y$ and apply Lemma 5.2 with parameter D'. Depending on the output execute the following steps:

    (i) Cut all edges in $\delta^+(B^+(v, r))$, recurse on the induced graph $G[B^+(v, r)]$, then remove all nodes in $B^+(v, r)$ from the graph.

    (ii) Cut all edges $\delta^-(B^-(v, r))$, recurse on the induced graph $G[B^-(v, r)]$, then remove all nodes in $B^-(v, r)$ from the graph.

Proof. TOPROVE 13 □

This completes the analysis of Algorithm 1 and puts us in the position of completing the proof of Theorem 2.7.

Proof. TOPROVE 14 □

6 Near-Optimal LDDs in Near-Linear Time

In this section we show that near-optimal LDDs can even be computed in near-linear (i.e., near-optimal) running time. Formally, the goal of this section is to prove the last of our main theorems:

Theorem 2.8. *For every directed graph there exists a directed LDD with loss $O(\log n \log \log n)$ which can be computed (i.e., sampled from) in expected time $\widetilde{O}(m)$.*

We structure this section as follows: In Section 6.1 we first prove an important cutting lemma (Lemma 6.2), which we then apply in Section 6.2 to prove Theorem 2.8.

6.1 Cutting Lemma

We rely on the following result due to Cohen that we can approximate efficiently the sizes of out-balls $B^+(v, r)$ simultaneously for all nodes v. Note that we can equally approximate the size of the in-balls $B^-(v, r)$ by applying Lemma 6.1 on the reverse graph.

Lemma 6.1 (Cohen's Algorithm [23, Theorem 5.1]). *Let G be a directed weighted graph, let $r \geq 0$ and $\epsilon > 0$. There is an algorithm that runs in time $O(m\epsilon^{-2} \log^3 n)$ and computes approximations $b(v)$ satisfying with high probability that*

$$(1 - \epsilon)|B^+(v, r)| \leq b(v) \leq (1 + \epsilon)|B^+(v, r)|.$$

The goal of this subsection is to establish the following lemma:

Lemma 6.2 (Cutting Light Nodes). *Let $G = (V, E)$ be a directed weighted graph, and consider the parameters $\delta > 0$, $0 \leq r_0 < r_1$, $1 \leq s_1 < s_0 \leq m$. There is an algorithm that computes a set of cut edges $S \subseteq E$ and sets of remaining vertices $R \subseteq V$ such that:*

(i) *Each strongly connected component in $G \setminus S$ either only contains nodes from R or only nodes from $V \setminus R$. In the latter case it contains at most $\frac{3}{2} \cdot \frac{m}{s_1}$ edges.*

(ii) *For every node $v \in R$ with $|B_G^+(v, r_1)| \leq \frac{m}{s_1}$, it holds that $|B_{G[R]}^+(v, r_0)| \leq \frac{m}{s_0}$.*

(iii) *None of the edges in S has its source in R (i.e., $S \subseteq (V \setminus R) \times V$). Moreover, for each edge $e = (x, y) \in E$ it holds that:*

$$\Pr(e \in S \mid x \notin R) \leq \frac{w_e \ln(2s_0/\delta)}{r_1 - r_0}.$$

With probability at most δ the algorithm instead returns "fail", and with probability at most $\frac{1}{\text{poly}(n)}$ the algorithm returns an arbitrary answer. The running time is $O(m \log^4 n)$.

Proof. TOPROVE 15 □

6.2 Near-Linear-Time LDD

We are ready to state the LDD algorithm. Throughout this section, we fix the following parameters:

- $L = \lceil \log \log m \rceil + 1$,

- $\delta = \frac{1}{\log^{10} m}$,

- $r_0 := 0$ and $r_\ell := r_{\ell-1} + \frac{D}{2^{L-\ell+3}} + \frac{D}{4L}$ (for $1 \leq \ell \leq L$),

- $s_\ell := \min(2^{2^{L-\ell}}, m + 1)$ (for $0 \leq \ell \leq L$).

With these parameters in mind, consider Algorithm 2. The algorithm first deletes all edges with sufficiently large weight. Then it proceeds in L iterations where in each iteration we apply Lemma 6.2 to cut some edges in the graph (or reverse graph). If any of these calls returns "fail", then the entire algorithm restarts. In the following Lemmas 6.3 to 6.5 we will show that this procedure correctly implements the algorithm claimed in Theorem 2.8.

Lemma 6.3 (Correctness of Algorithm 2). *Let u, v be two nodes lying in the same strongly connected component in $G \setminus S$. Then $d_G(u, v) \leq D$.*

Proof. TOPROVE 16 □

Lemma 6.4 (Edge Cutting Probability of Algorithm 2). *For any edge $e \in E$, we have*

$$\Pr(e \in S) \leq O\left(\frac{w_e}{D} \cdot \log n \log \log n + \frac{1}{\text{poly}(n)}\right).$$

13

Algorithm 2 The near-linear-time near-optimal LDD, see Theorem 2.8.
---

1. Initially let $S \subseteq E$ be the set of edges of weight at least $\frac{D}{4L}$, and remove these edges from G

2. For $\ell \leftarrow L, \ldots, 1$:

   2.1. Run Lemma 6.2 on G with parameters $\delta, r_{\ell-1}, r_\ell, s_{\ell-1}, s_\ell$, and let S_ℓ^+, R_ℓ^+ denote the resulting sets.

   2.2. Compute the strongly connected components in $(G \setminus S_\ell^+)[V \setminus R_\ell^+]$. Recur on each such component and add the recursively computed cut edges to S.

   2.3. Update $G \leftarrow G[R_\ell^+]$ and $S \leftarrow S \cup S_\ell^+$.

   2.4. Run Lemma 6.2 on $\text{rev}(G)$ with parameters $\delta, r_{\ell-1}, r_\ell, s_{\ell-1}, s_\ell$, and let S_ℓ^-, R_ℓ^- denote the resulting sets.

   2.5. Compute the strongly connected components in $(G \setminus S_\ell^-)[V \setminus R_\ell^-]$. Recur on each such component and add the recursively computed cut edges to S.

   2.6. Update $G \leftarrow G[R_\ell^-]$ and $S \leftarrow S \cup S_\ell^-$.

3. If any of the $2L$ previous calls to Lemma 6.2 returns "fail", then we restart the entire algorithm (i.e., we reset G to be the given graph, we reset $S \leftarrow \emptyset$, and we start the execution from step 1).

Proof. TOPROVE 17 □

Lemma 6.5 (Running Time of Algorithm 2). *The expected running time is $O(m \log^5 n \log \log n)$.*

Proof. TOPROVE 18 □

We remark that we have not attempted to optimize the log-factors in the running time here and it is likely that the overhead can be reduced. An obvious improvement to shave one log-factor from the running time is to employ a faster priority queue in Dijkstra's algorithm, e.g. as developed by Thorup [37].

References

[1] Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. In *49th Annual Symposium on Foundations of Computer Science (FOCS 2008)*, pages 781–790. IEEE Computer Society, 2008. `doi:10.1109/FOCS.2008.62`.

[2] Ittai Abraham, Shiri Chechik, Michael Elkin, Arnold Filtser, and Ofer Neiman. Ramsey spanning trees and their applications. *ACM Trans. Algorithms*, 16(2):19:1–19:21, 2020. `doi:10.1145/3371039`.

[3] 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`.

[4] Noga Alon, Richard M. Karp, David Peleg, and Douglas B. West. A graph-theoretic game and its application to the k-server problem. *SIAM J. Comput.*, 24(1):78–100, 1995. `doi: 10.1137/S0097539792224474`.

[5] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(6):121–164, 2012. `doi:10.4086/toc.2012.v008a006`.

[6] Vikrant Ashvinkumar, Aaron Bernstein, and Adam Karczmarz. Faster approximation algorithms for restricted shortest paths in directed graphs. In Yossi Azar and Debmalya Panigrahi, editors, *36th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025)*, pages 5263–5277. SIAM, 2025. `doi:10.1137/1.9781611978322.180`.

[7] Baruch Awerbuch. Complexity of network synchronization. *J. ACM*, 32(4):804–823, 1985. `doi:10.1145/4221.4227`.

[8] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Fast network decomposition (extended abstract). In Norman C. Hutchinson, editor, *11th Annual ACM Symposium on Principles of Distributed Computing (PODC 1992)*, pages 169–177. ACM, 1992. `doi:10.1145/135419.135456`.

[9] Baruch Awerbuch, Andrew V. Goldberg, Michael Luby, and Serge A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science (FOCS 1989)*, pages 364–369. IEEE Computer Society, 1989. `doi:10.1109/SFCS.1989.63504`.

[10] Baruch Awerbuch and David Peleg. Routing with polynomial communication-space trade-off. *SIAM J. Discret. Math.*, 5(2):151–162, 1992. `doi:10.1137/0405013`.

[11] Yair Bartal. Probabilistic approximations of metric spaces and its algorithmic applications. In *37th Annual Symposium on Foundations of Computer Science (FOCS 1996)*, pages 184–193. IEEE Computer Society, 1996. `doi:10.1109/SFCS.1996.548477`.

[12] Yair Bartal. On approximating arbitrary metrices by tree metrics. In Jeffrey Scott Vitter, editor, *30th Annual ACM Symposium on Theory of Computing (STOC 1998)*, pages 161–168. ACM, 1998. `doi:10.1145/276698.276725`.

[13] Aaron Bernstein, Joakim Blikstad, Thatchaphol Saranurak, and Ta-Wei Tu. Maximum flow by augmenting paths in $n^{2+o(1)}$ time. In *65th Annual Symposium on Foundations of Computer Science (FOCS 2024)*. IEEE, 2024. To appear, see `https://arxiv.org/abs/2406.03648`.

[14] Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing. In Sandy Irani, editor, *61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1123–1134. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00108`.

[15] Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental SSSP in dense weighted digraphs. In Sandy Irani, editor, *61st Annual Symposium on Foundations of Computer Science (FOCS 2020)*, pages 1112–1122. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00107`.

[16] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *63rd Annual Symposium on Foundations of Computer Science (FOCS 2022)*, pages 600–611. IEEE, 2022. `doi:10.1109/FOCS54457.2022.00063`.

[17] Guy E. Blelloch, Anupam Gupta, Ioannis Koutis, Gary L. Miller, Richard Peng, and Kanat Tangwongsan. Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs. *Theory Comput. Syst.*, 55(3):521–554, 2014. `doi:10.1007/s00224-013-9444-5`.

[18] Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.

[19] Karl Bringmann, Alejandro Cassis, and Nick Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In *64th Annual Symposium on Foundations of Computer Science (FOCS 2023)*, pages 515–538. IEEE, 2023. `doi:10.1109/FOCS57990.2023.00038`.

[20] Shiri Chechik, Yang P. Liu, Omer Rotem, and Aaron Sidford. Constant girth approximation for directed graphs in subquadratic time. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *52nd Annual ACM Symposium on Theory of Computing (STOC 2020)*, pages 1010–1023. ACM, 2020. `doi:10.1145/3357713.3384330`.

[21] Shiri Chechik and Tianyi Zhang. Dynamic low-stretch spanning trees in subpolynomial time. In Shuchi Chawla, editor, *31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2020)*, pages 463–475. SIAM, 2020. `doi:10.1137/1.9781611975994.28`.

[22] Julia Chuzhoy and Sanjeev Khanna. Maximum bipartite matching in $n^{2+o(1)}$ time via a combinatorial algorithm. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *56th Annual ACM Symposium on Theory of Computing (STOC 2024)*, pages 83–94. ACM, 2024. `doi:10.1145/3618260.3649725`.

[23] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55(3):441–453, 1997. URL: `https://doi.org/10.1006/jcss.1997.1534`, `doi:10.1006/JCSS.1997.1534`.

[24] 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`.

[25] 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. URL: `https://doi.org/10.1016/j.jcss.2004.04.011`, `doi:10.1016/J.JCSS.2004.04.011`.

[26] Sebastian Forster and Gramoz Goranci. Dynamic low-stretch trees via dynamic low-diameter decompositions. In Moses Charikar and Edith Cohen, editors, *51st Annual ACM Symposium on Theory of Computing (STOC 2019)*, pages 377–388. ACM, 2019. `doi:10.1145/3313276.3316381`.

[27] Sebastian Forster, Martin Grösbacher, and Tijn de Vos. An improved random shift algorithm for spanners and low diameter decompositions. In Quentin Bramas, Vincent Gramoli, and Alessia Milani, editors, *25th International Conference on Principles of Distributed Systems (OPODIS 2021)*, volume 217 of *LIPIcs*, pages 16:1–16:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. `doi:10.4230/LIPIcs.OPODIS.2021.16`.

[28] Bernhard Haeupler, D. Ellis Hershkowitz, and Goran Zuzic. Tree embeddings for hop-constrained network design. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd*

Annual ACM Symposium on Theory of Computing (STOC 2021), pages 356–369. ACM, 2021. `doi:10.1145/3406325.3451053`.

[29] Yiding Hua, Rasmus Kyng, Maximilian Probst Gutenberg, and Zihang Wu. Maintaining expander decompositions via sparse cuts. In Nikhil Bansal and Viswanath Nagarajan, editors, *34th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2023)*, pages 48–69. SIAM, 2023. URL: `https://doi.org/10.1137/1.9781611977554.ch2`, `doi:10.1137/1.9781611977554.CH2`.

[30] Rohit Khandekar, Satish Rao, and Umesh Vazirani. Graph partitioning using single commodity flows. *J. ACM*, 56(4), jul 2009. `doi:10.1145/1538902.1538903`.

[31] Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-m log n time solver for SDD linear systems. In Rafail Ostrovsky, editor, *52th Annual Symposium on Foundations of Computer Science (FOCS 2011)*, pages 590–598. IEEE Computer Society, 2011. `doi:10.1109/FOCS.2011.85`.

[32] Nathan Linial and Michael E. Saks. Low diameter graph decompositions. *Comb.*, 13(4):441–454, 1993. `doi:10.1007/BF01303516`.

[33] 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 2013)*, pages 196–203. ACM, 2013. `doi:10.1145/2486159.2486180`.

[34] Jakub Pachocki, Liam Roditty, Aaron Sidford, Roei Tov, and Virginia Vassilevska Williams. Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2018)*, pages 1374–1392. SIAM, 2018. `doi:10.1137/1.9781611975031.91`.

[35] Harald Räcke, Chintan Shah, and Hanjo Täubig. Computing cut-based hierarchical decompositions in almost linear time. In Chandra Chekuri, editor, *25th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2014)*, pages 227–238. SIAM, 2014. `doi:10.1137/1.9781611973402.17`.

[36] Paul D. Seymour. Packing directed circuits fractionally. *Comb.*, 15(2):281–288, 1995. `doi:10.1007/BF01200760`.

[37] Mikkel Thorup. Integer priority queues with decrease key in constant time and the single source shortest paths problem. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing, June 9-11, 2003, San Diego, CA, USA*, pages 149–158. ACM, 2003. `doi:10.1145/780542.780566`.

[38] Goran Zuzic, Gramoz Goranci, Mingquan Ye, Bernhard Haeupler, and Xiaorui Sun. Universally-optimal distributed shortest paths and transshipment via graph-based ℓ_1-oblivious routing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2022)*, pages 2549–2579. SIAM, 2022. `doi:10.1137/1.9781611977073.100`.