Parallel, Distributed, and Quantum Exact Single-Source Shortest Paths with Negative Edge Weights

Vikrant Ashvinkumar
Rutgers University, New Brunswick, NJ, USA

Aaron Bernstein
Rutgers University, New Brunswick, NJ, USA

Nairen Cao
Department of Computer Science, Boston College, Chestnut Hill, MA, USA

Christoph Grunau
ETH Zürich, Switzerland

Bernhard Haeupler
INSAIT, Sofia, Bulgaria
ETH Zürich, Switzerland

Yonggang Jiang
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Danupon Nanongkai
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Hsin-Hao Su
Department of Computer Science, Boston College, Chestnut Hill, MA, USA

Abstract

This paper presents parallel, distributed, and quantum algorithms for single-source shortest paths when edges can have negative integer weights (negative-weight SSSP). We show a framework that reduces negative-weight SSSP in all these settings to $n^{o(1)}$ calls to any SSSP algorithm that works on inputs with non-negative integer edge weights (non-negative-weight SSSP) with a virtual source. More specifically, for a directed graph with m edges, n vertices, undirected hop-diameter D, and polynomially bounded integer edge weights, we show randomized algorithms for negative-weight SSSP with

- $W_{SSSP}(m,n)n^{o(1)}$ work and $S_{SSSP}(m,n)n^{o(1)}$ span, given access to a non-negative-weight SSSP algorithm with $W_{SSSP}(m,n)$ work and $S_{SSSP}(m,n)$ span in the parallel model, and
- $T_{SSSP}(n,D)n^{o(1)}$ rounds, given access to a non-negative-weight SSSP algorithm that takes $T_{SSSP}(n,D)$ rounds in CONGEST, and
- $Q_{SSSP}(m,n)n^{o(1)}$ quantum edge queries, given access to a non-negative-weight SSSP algorithm that takes $Q_{SSSP}(m,n)$ queries in the quantum edge query model.

This work builds off the recent result of Bernstein, Nanongkai, Wulff-Nilsen [7], which gives a near-linear time algorithm for negative-weight SSSP in the sequential setting.

Using current state-of-the-art non-negative-weight SSSP algorithms yields randomized algorithms for negative-weight SSSP with

- $m^{1+o(1)}$ work and $n^{1/2+o(1)}$ span in the parallel model, and
- $(n^{2/5}D^{2/5}+\sqrt{n}+D)n^{o(1)}$ rounds in CONGEST, and
- $m^{1/2}n^{1/2+o(1)}$ quantum queries to the adjacency list or $n^{1.5+o(1)}$ quantum queries to the adjacency matrix.

Up to a $n^{o(1)}$ factor, the parallel and distributed results match the current best upper bounds for reachability [23, 12]. Consequently, any improvement to negative-weight SSSP in these models beyond the $n^{o(1)}$ factor necessitates an improvement to the current best bounds for reachability. The quantum result matches the lower bound up to an $n^{o(1)}$ factor [9].

Our main technical contribution is an efficient reduction from computing a low-diameter decomposition (LDD) of directed graphs to computations of non-negative-weight SSSP with a virtual source. Efficiently computing an LDD has heretofore only been known for undirected graphs in both the parallel and distributed models, and been rather unstudied in quantum models. The directed LDD is a crucial step of the sequential algorithm in [7], and we think that its applications to other problems in parallel and distributed models are far from being exhausted.

Other ingredients of our results include altering the recursion structure of the scaling algorithm in [7] to surmount difficulties that arise in these models, and also an efficient reduction from computing strongly connected components to computations of SSSP with a virtual source in CONGEST. The latter result answers a question posed in [6] in the negative.

32nd Annual European Symposium on Algorithms (ESA 2024).
Editors: Timothy Chan, Johannes Fischer, John Iacono, and Grzegorz Herman; Article No. 13; pp. 13:1–13:14

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2012 ACM Subject Classification Theory of computation → Shortest paths; Theory of computation → Parallel algorithms; Theory of computation → Distributed algorithms

Keywords and phrases Parallel algorithm, distributed algorithm, shortest paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.13

Related Version *Full Version*: https://arxiv.org/abs/2303.00811

Funding *Aaron Bernstein*: Supported in part by a Sloan Research Fellowship, a Google Research Fellowship, and NSF Grant 1942010.
Nairen Cao: Supported by NSF CCF-2008422.
Bernhard Haeupler: Supported by the European Research Council under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).
Hsin-Hao Su: Supported by NSF CCF-2008422.

1 Introduction

Single-source shortest paths (SSSP) is one of the most fundamental problems in graph algorithms. Given a directed graph $G = (V, E)$, an integer weight function $w : E \to \mathbb{Z}$, and a source vertex $s \in V$, we want to compute the distance from s to v for all $v \in V$.

Efficient solutions to this problem are typically better understood in the regime where edge weights are non-negative, which we denote with non-negative-weight SSSP. For example, Dijkstra's algorithm, from the 50s, requires this assumption and runs in near-linear time. The algorithms for single-source shortest paths with negative integer weights (denoted negative-weight SSSP), on the other hand, have until very recently been significantly slower. From the 50s, the classic Bellman-Ford algorithm gives an $O(mn)$ time algorithm,[1] which either computes distances from s to v or reports a negative-weight cycle. A series of improvements since then ([22, 16, 33, 2]) culminated in two recent breakthroughs: the algorithm of Chen, Kyng, Liu, Peng, Probst Gutenberg, and Sachdeva ([15]) solving transshipment and min-cost flow in time $m^{1+o(1)}$, thus implying the same runtime for negative-weight SSSP, and a parallel and independent result of Bernstein, Nanongkai, Wulff-Nilsen ([7]) giving a $\widetilde{O}(m)$ time[2] algorithm for negative-weight SSSP that uses relatively simpler techniques. Follow-up work by Bringmann, Cassis, and Fischer significantly reduces the number of log factors in the $\widetilde{O}(m)$ runtime ([10]). In this paper, we take the exploration of negative-weight SSSP to parallel, distributed, and quantum models of computation. Should there be analogous results there?

In parallel models, there has been much recent progress for the non-negative-weight SSSP problem. Rozhoň, Haeupler, Martinsson, Grunau and Zuzic ([29]) and Cao and Fineman ([11]) showed that SSSP with polynomially bounded non-negative integer edge weights can be solved with $\widetilde{O}(m)$ work and $n^{1/2+o(1)}$ depth in the parallel model. By contrast, the known bounds for negative-weight SSSP are significantly weaker: the classic Bellman-Ford algorithm solves negative-weight SSSP with $O(mn)$ work and $O(n)$ depth, and recently, Cao, Fineman and Russell ([13]) improved this to $\widetilde{O}(m\sqrt{n})$ work and $n^{5/4+o(1)}$ depth.

Similarly, in distributed models, Rozhoň et al. ([29]) and Cao and Fineman ([11]) show algorithms for SSSP with non-negative integer edge weights that take $\widetilde{O}((n^{2/5+o(1)}D^{2/5} + \sqrt{n} + D)$ rounds[3]. On the negative-weight SSSP front, the Bellman-Ford algorithm takes

[1] Here and throughout, we use n to denote the number of vertices, m to denote the number of edges of G.
[2] Here and throughout, we use the soft-O notation $\widetilde{O}$ to suppress polylogarithmic (in n) factors. Throughout the paper, we assume the maximum weight edge (in absolute value) of G is polynomially bounded.
[3] Here and throughout, we use D to denote the undirected hop-diameter of G.

$O(n)$ rounds. The current state-of-the-art by Forster, Goranci, Liu, Peng, Sun and Ye ([19]), which uses Laplacian solvers, gives an $\widetilde{O}(m^{3/7+o(1)}(n^{1/2}D^{1/4}+D))$ round algorithm for negative-weight SSSP.

In the quantum edge query model, Durr, Heiligman, Høyer, and Mhalla [17] show an algorithm for SSSP with non-negative edge weights in $O(n^{1.5})$ queries to the adjacency matrix or $O(m^{1/2}n^{1/2})$ queries to the adjacency list, which are both tight. We are not aware of any quantum edge query algorithm solving negative-weight SSSP better than the trivial $O(n^2)$ or $O(m)$ algorithm.

There is a substantial gap between the best known upper bounds for non-negative-weight SSSP and negative-weight SSSP in these models and, in fact, the number of landmark algorithms for negative-weight SSSP has been comparatively few. This begets the following question: Can we close the gap, and get parallel, distributed, and quantum algorithms for negative-weight SSSP that are nearly as efficient as the best non-negative-weight SSSP algorithms? This paper gives an answer in the affirmative.

Main Results. The main results of this paper are as follows.

▶ **Theorem 1** (Parallel SSSP reduction with negative edge-weight). *Assuming there is a parallel algorithm answering (non-negative integer weight) SSSP on directed graphs in $W(m,n)$ work and $S(m,n)$ span, then there exists a randomized algorithm that solves negative-weight SSSP on directed graphs G with polynomially bounded integer edge-weights with $O(W(m,n)(\log n)^{O(\sqrt{\log n})})$ work and $\widetilde{O}(S(m,n)2^{\sqrt{\log n}})$ span with high probability.*

Using state-of-the-art results for non-negative-weight SSSP ([29] and [11]) with Theorem 1 immediately gives a randomized parallel algorithm that solves negative-weight SSSP on directed graphs with $m^{1+o(1)}$ work and $n^{1/2+o(1)}$ span, with high probability.

▶ **Theorem 2** (Distributed SSSP reduction with negative edge-weight). *In the* CONGEST *model, assuming there is an algorithm answering (non-negative integer weight) SSSP on directed graphs in $T(n,D)$ rounds, then there exists a randomized algorithm that solves negative-weight SSSP on directed graphs G with polynomially bounded integer edge-weights and undirected hop-diameter D in $O((T(n,D)+\sqrt{n}+D)(\log n)^{O(\sqrt{\log n})})$ rounds with high probability.*

Using state-of-the-art results for non-negative-weight SSSP ([29] and [11]) with Theorem 2 immediately gives a distributed randomized algorithm that solves negative-weight SSSP on directed graphs with $O((n^{2/5+o(1)}D^{2/5}+\sqrt{n}+D)n^{o(1)})$ rounds of communication in the CONGEST model with high probability. For general graphs there is a lower bound of $T(n,D)=\Omega(\sqrt{n}+D)$ ([26]), so the factor of $\sqrt{n}+D$ in our runtime does not impact the efficiency of our reduction.

▶ **Theorem 3** (Quantum SSSP reduction with negative edge-weight). *In the quantum edge query model, assuming there is an algorithm answering (non-negative integer weight) SSSP on directed graph in $Q(m,n)$ queries, then there exists a randomized algorithm that solves negative-weight SSSP on directed graphs G with polynomially bounded integer edge-weights in $O(Q(m,n)(\log n)^{O(\sqrt{\log n})}))$ queries.*

Using the state-of-the-art results for non-negative-weight SSSP [17] with Theorem 3 immediately gives a quantum edge query algorithm that solves negative-weight SSSP on directed graphs with $n^{1.5+o(1)}$ queries to the adjacency matrix, or $m^{1/2}n^{1/2+o(1)}$ queries to the adjacency list. The upper bound is optimal up to an $n^{o(1)}$ factor by the $\Omega(n^{1.5})$ and $\Omega(\sqrt{mn})$ lower bound result [17].

We note that all of our results take the form of a general reduction from negative-weight SSSP to non-negative-weight SSSP, so any further advance in non-negative-weight SSSP immediately translates to improved bounds for negative-weight SSSP. Modulo $n^{o(1)}$ factors, the complexity of parallel and distributed algorithms for non-negative-weight SSSP match that of directed reachability [23, 12]; any improvements to negative-weight SSSP beyond the $n^{o(1)}$ factor would thus first require improvements to directed reachability in these models.

Our reductions follow the high-level framework of the recent sequential $\widetilde{O}(m)$-time algorithm of Bernstein, Nanongkai, and Wulff-Nilsen [7]. At the heart of their framework is the use of directed low-diameter decompositions (on graphs with non-negative edge weights), and one of our key technical contributions is to give algorithms for computing such a directed decomposition in parallel, distributed, and quantum models.

1.1 Our Further Contributions

On top of algorithms for negative-weight SSSP, we provide two algorithms that we believe are of independent interest. The most significant one is an efficient implementation of directed low-diameter decomposition in parallel, distributed, and quantum models. We also show an algorithm for computing strongly connected components and their topological ordering in the CONGEST model. Like Theorems 1 and 2, these results are presented as reductions to non-negative-weight SSSP. An advantage of this approach is that our results scale with non-negative-weight SSSP; if there is any progress in the upper bounds to non-negative-weight SSSP, progress to the bounds here immediately follow.

Directed Low Diameter Decomposition

Previous Work. Low-Diameter Decomposition (LDD) has long been used to design efficient algorithms for *undirected* graphs in several models of computation [1, 24, 3, 18, 21, 28, 31, 30, 7, 20]. A few recent papers developed a generalization of LDD that also applies to *directed* graphs[14, 8, 5]. Bernstein et al.[7] use directed LDD as one of the key subroutines in their sequential algorithm for negative-weight SSSP, and they present a sequential algorithm for computing directed LDD in near-linear time.

In undirected graphs, it is also known how to compute LDD efficiently in other models of computation, including parallel and distributed models; in fact, the well-known algorithm of Miller, Peng, and Xu (MPX) reduces this problem to a single shortest-path-tree computation from a dummy source s [25].

Our Results. One of our main technical contributions is showing that in several computation models, computing *directed* LDD can similarly be reduced to a small number of shortest-path-tree computations. This requires new techniques for overcoming obstacles that are unique to directed graphs; see Section 4.1 for an overview of these new techniques.

The input/output guarantees of directed LDD are stated below; they are the same as those in the sequential paper of [7]. (Note in particular that the input to LDD is a graph with *non-negative* weights.) Intuitively, for a given parameter d, the decomposition computes a small set of "bad" edges E^{rem} such that (1) Every strongly connected component in $G \setminus E^{rem}$ has weak diameter at most d and (2) Every edge of the graph is in E^{rem} with probability at most $\widetilde{O}(w(e)/d)$.

▶ **Lemma 4** (Low-Diameter Decomposition, Algorithm 1). *Let $G = (V, E, w)$ be a directed graph with a polynomially bounded weight function $w : E \to \mathbb{N}$ and let d be a positive integer. There exists a randomized algorithm* LowDiameterDecomposition(G, d) *with following guarantees:*

- *INPUT: An n-node m-edge, graph $G = (V, E, w)$ with non-negative integer edge weight and a positive integer d.*
- *OUTPUT: (proved in the full paper) a set of edges $E^{rem} \subseteq E$ satisfying:*
  - *Each SCC of the subgraph $G \setminus E^{rem}$ has weak diameter at most d in G, i.e. if u, v are two vertices in the same SCC, then $dist_G(u, v) \leq d$ and $dist_G(v, u) \leq d$.*
  - *For any $e \in E$, we have $\mathbf{Pr}[e \in E^{rem}] = O\left(\frac{w(e) \log^2 n}{d} + \frac{1}{n^8}\right)$*
- *RUNNING TIME: The algorithm is randomized and takes $\widetilde{O}(1)$ calls to (non-negative integer weight) SSSP. More specifically:*
  - *Assuming there is a parallel algorithm answering non-negative-weight SSSP in $W(m, n)$ work and $S(m, n)$ span, then LowDiameterDecomposition(G, d) takes $\widetilde{O}(W(m, n))$ work and $\widetilde{O}(S(m, n))$ span with high probability.*
  - *Assuming there exists a CONGEST algorithm answering non-negative-weight SSSP in $T(n, D)$ rounds, then LowDiameterDecomposition(G, d) takes $\widetilde{O}(T(n, D) + \sqrt{n} + D)$ rounds in the CONGEST model with high probability, where D is the undirected hop diameter.*
  - *Assuming there exists a quantum edge query algorithm answering non-negative-weight SSSP using $Q(m, n)$ queries, then LowDiameterDecomposition(G, d) takes $\widetilde{O}(Q(m, n))$ queries with high probability.*

We observe that the complexity of quantum query algorithms is typically sublinear in m, yet the output size of E^{rem} may reach up to m. Consequently, rather than directly producing E^{rem} as output, it is represented in an implicit format within $\widetilde{O}(n)$ bits. For further details, refer to the full paper.

The concept of undirected low-diameter decomposition was first introduced in the context of parallel and distributed algorithms, and some of the most important applications and use cases are in these areas. We are therefore optimistic that our directed parallel and distributed low-diameter decomposition algorithm can be applied to solve various problems in the distributed and parallel setting in the future, beyond the application of computing negative weight shortest paths addressed in this paper.

Strongly Connected Components and Their Topological Order in CONGEST

Another subroutine we need in our algorithm is finding the strongly connected components of a graph. It is known that in the parallel setting this problem reduces to single-source reachability ([32]). In the full version we show a similar reduction for the CONGEST setting; we use the same high-level framework as the parallel reduction, but this is difficult to port directly into the CONGEST model; we show that by going through the recently developed Distributed Minor-Aggregation Model, we are able to overcome this difficulty.

▶ **Lemma 5.** *There is a CONGEST algorithm that, given a directed graph $G = (V, E)$, and assuming there is an algorithm answering non-negative-weight SSSP in $T(n, D)$ rounds, outputs strongly connected components listed in a topological order. More specifically, it outputs a polynomially-bounded labelling $(r_v)_{v \in V}$ such that, with high probability*
1. *$r_u = r_v$ if and only if u and v are in the same strongly connected component;*
2. *when the SCC that u belongs to has an edge towards the SCC that v belongs to, $r_u > r_v$.*[4]
The algorithm takes $\widetilde{O}(T(n, D) + \sqrt{n} + D)$ rounds.

[4] As a matter of convenience, the labels correspond to a reverse topological order (i.e. something which appears earlier in a topological order has a larger label than something which appears later).

It is worth noting that a more careful examination gives a round complexity in terms of calls to a reachability oracle, rather than a non-negative-weight SSSP oracle. Plugging in the current state-of-the-art CONGEST algorithm for non-negative-weight SSSP ([29] and [11]) leads to a $\widetilde{O}\left(n^{1/2} + D + n^{2/5+o(1)}D^{2/5}\right)$ round algorithm, answering a question posed in [6] which asked if a lower bound of $\widetilde{\Omega}(n)$ rounds applies to the problem of finding SCCs.

1.2 Organization

In Section 2, we provide the necessary terminology and notation that will be used throughout the paper. This section can be skipped and referred back to as needed. In Section 3, we present a high-level overview of [7] and discuss the key challenges involved in adapting the results to other models. Section 4 presents our algorithm for low-diameter decomposition. Omitted results can be found in the full version of this paper, appended at the end of this submission.

2 Definitions and Preliminaries

A weighted directed graph G is a triple (V, E, w) where $w : E \to \mathbb{Z}$ is a weight function. For a weighted directed graph G, the number of vertices and edges are $|V(G)| = n$ and $|E(G)| = m$, respectively. We denote the set of negative edges by $E^{neg}(G) = \{e \in E \mid w(e) < 0\}$. For a subset $V' \subset V$, we denote the induced graph on V' by $G[V']$ and the induced edges on V' by $E(V')$. For an edge set $E' \subseteq E$, when we treat E' as a *subgraph* of G, we mean the graph $(\{u, v \mid (u, v) \in E'\}, E', w)$. A **path** is a sequence of vertices joined by edges; sometimes we refer to the path by the sequence of vertices and sometimes by the edges. A **strongly connected component** (SCC) is a set of vertices S such that for any pair of vertices $u, v \in S$, there is a path from u to v contained entirely in S.

For a path $\Gamma = \langle v_0, v_1, \ldots, v_k \rangle$, the **weight** of Γ is given by $w(\Gamma) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$, that is, the sum of the weights of the edges on the path. For a pair of nodes $u, v \in V$, the **shortest path distance** from u to v is the minimum length over all paths that start at u and end at v. We use $dist_G(u, v)$ to denote this shortest path distance with respect to the graph G. When the graph G is clear in the context, we simply write $dist(u, v)$. If there is no u-to-v path, then we define $dist(u, v) = +\infty$. Given a directed graph $G = (V, E, w)$, a vertex $s \in V$ and $d \in \mathbb{N}$, we define $\text{Ball}^{\text{in}}_G(s, d) = \{v \mid dist_G(v, s) \leq d\}$ and $\text{Ball}^{\text{out}}_G(s, d) = \{v \mid dist_G(s, v) \leq d\}$, the in or out balls centered at s with weighted radius d. For a given graph $G = (V, E, w)$ and a subset of vertices $S \subseteq V$, we define $\delta^-(S) = \{(u, v) \in E \mid u \notin S, v \in S\}$ and $\delta^+(S) = \{(u, v) \in E \mid u \in S, v \notin S\}$, the in or out edge sets crossing S .

When we say that an algorithm achieves performance $O(f(n))$ with high probability, we mean the following: for a particular choice of constant $c > 0$, with probability at least $1 - 1/n^c$ the algorithm achieves performance $O(f(n))$.

▶ **Definition 6** (Definition 2.5 of [7]). *Consider a graph $G = (V, E, w)$ and let ϕ be any function: $V \mapsto \mathbb{Z}$. Then, we define w_ϕ to be the weight function $w_\phi(u, v) = w(u, v) + \phi(u) - \phi(v)$ and we define $G_\phi = (V, E, w_\phi)$. We will refer to ϕ as a price function on V. Note that $(G_\phi)_\psi = G_{\phi+\psi}$.*

3 High Level Overview

Our results follow the framework of [7], which provides a sequential algorithm for negative-weight SSSP that takes $\widetilde{O}(m)$ time with high probability.

3.1 Overview of the Sequential Algorithm from [7]

This section provides a summary of Bernstein et al.'s approach to computing exact shortest paths on a graph with integer edge weights (both positive and negative) [7].

The final goal is to compute a price function ϕ such that all edges in G_ϕ are non-negative; since G_ϕ and G are equivalent, one can then run SSSP for non-negative weights on G_ϕ. Following the standard scaling framework, Bernstein et al.'s algorithm computes such a ϕ over multiple scaling rounds. The key component of the algorithm is a procedure ScaleDown that computes a price function that halves the minimum negative edge weight: given a weighted directed graph G where all edge weights are at least as large as $-2B$ for some non-negative parameter B, ScaleDown outputs a price function ϕ such that in G_ϕ all edge weights are at least as large as $-B$. A procedure ScaleDown with these guarantees can then easily be used to solve negative-weight SSSP, so for the rest of this section we focus exclusively on the algorithm ScaleDown.

The Algorithm ScaleDown. We now give a high-level overview of the sequential algorithm for ScaleDown in [7]. Our algorithm will follow the same general framework, but with a few key differences discussed below. See full paper for a more detailed description of our algorithm, along with pseudocode.

In order to compute the desired price function ϕ, the ScaleDown procedure consists of four phases.

- Phase 0: Run a Low Diameter Decomposition on G^B with negative-weight edges rounded up to 0 (See Lemma 4). This gives a set E^{rem} of removed edges, such that all SCCs of $G \setminus E^{rem}$ have small weak diameter. Observe that by the guarantees of directed LDD, after Phase 0 the graph will contain three types of edges: **(i)** edges within each SCC of $G \setminus E^{rem}$, **(ii)** edges that connect one component to another and are not in E^{rem}; one can intuitively think of these edges as being DAG-like, since they always go forward in the topological ordering of the SCCs of $G \setminus E^{rem}$, and **(iii)** edges from E^{rem}, where any edge $e \in E$ is in E^{rem} with probability at most $\widetilde{O}(w(e)/D)$. Phase 1 of ScaleDown addresses the first type of edge, Phase 2 the second, and Phase 3 the third.
- Phase 1: Recursively call ScaleDown on the edges inside each SCC. This finds a price function under which edges inside each SCC have non-negative weight, thus fixing the type 1 edges.
- Phase 2: Fix the edges not in E^{rem} that connect one component to another (i.e. the DAG-like edges); that is, compute a price function that makes their weight non-negative.
- Phase 3: Fix the edges of E^{rem}.

Implementing the Three Phases in the Sequential Model. Recall that the low-diameter decomposition provides two guarantees: first that the weak diameter in each SCC is bounded, and second that each edge will be in E^{rem} with probability proportional to its weight. Loosely speaking, the first guarantee ensures that in Phase 1, recursively calling ScaleDown on the SCCs is making progress, because one can show that as the diameter decreases the maximum number of negative edges on any shortest path is reduced (This is technically only true in a carefully defined auxiliary graph). Bernstein et al. show that after $O(\log n)$ recursive calls to ScaleDown, the number of negative edges on any shortest path is at most $\widetilde{O}(1)$. They then show an algorithm called ElimNeg that can efficiently compute single-source shortest paths in graphs with this property; running this algorithm from a dummy source s yields the desired price function.

For Phase 2, the focus is on DAG-like edges connecting the SCCs of $G \setminus E^{rem}$; by Phase 1, the edges in each SCC already have non-negative weights. The algorithm simply contracts each SCC into a vertex to get an acyclic graph whose edge set consists of type-2 edges. Computing a price function for these edges turns out to be very easy because the underlying graph is a DAG.

By the time the algorithm reaches Phase 3, only edges in E^{rem} can still be negative. The second guarantee of low-diameter decomposition ensures that every shortest path has few edges from E^{rem} in expectation. In other words, by the time the algorithm reaches Phase 3, the remaining graph has the following property: *on average*, the shortest path from s to any vertex v contains few negative edges. The authors of [7] then show that their subroutine ElimNeg can efficiently compute shortest distances in any graph with this property; These distances then give the desired price function.

3.2 Adapting to Other Models: Challenges & Solutions

Although this framework works well in the sequential setting, it presents additional challenges in other models. We summarize these obstacles below.

Obstacle 1: Low Diameter Decomposition. Undirected LDD in parallel and distributed models (e.g. [25],[4],[27]) is commonly solved via the following framework: each node grows a ball starting at a random "delayed" time (the distribution of the randomness is picked carefully), and the boundary of a ball stops growing once it reaches another ball. The balls create a partition of the graph, where each ball has a low diameter, and if we define E^{rem} to be the edges between different balls, then it can be shown that any particular edge e is in E^{rem} with small probability. For example, the simplest instantiation of the above approach is the well-known MPX algorithm of Miller, Peng, Xu [25]. In this algorithm, every vertex picks a random delay $\delta(v)$, and then vertex x is assigned to the ball B_y of the vertex y that minimizes $\min_{v \in V} dist(x, v) + \delta(v)$ (some of the B_y may end up empty.) One can easily compute this minimum for every x by computing a single shortest path tree from a dummy source with an edge of weight $\delta(v)$ to every vertex v.

Natural approaches: A natural way to extend the above algorithm to the directed setting is as follows: each vertex in parallel grows an outgoing ball (which means the ball growing only uses the edges going out of this ball), and the boundary stops growing once it reaches another out-ball; the edges pointing out of every ball are then included in E^{rem}. (The process should be repeated with incoming balls, but we leave this out to keep the discussions simple.) However, we can no longer argue that each edge is included in E^{rem} with a small probability. Consider the following graph: it contains a star with a middle vertex denoted by s', and $n-1$ other vertices which have edges pointing to s'. If s' is the first vertex to start growing a ball (because it ends up with the lower random delay), then the LDD algorithm will create a ball $\{s'\}$, so when every vertex on the boundary of the star later grows its own ball, the algorithm will include all the edges of the graph into E^{rem}. On the other hand, if some other vertex s in the boundary of the star starts growing a ball before s', this will result in ball $\{s, s'\}$, and when other boundary vertices start growing their own balls, all edges other than (s, s') will be added to E^{rem}.

The above example is rather naive because the input graph is not strongly connected; so, we can just return $E^{rem} = \emptyset$ (then, all the strongly connected components (SCCs) already have low diameters). But the construction can be extended to the more sophisticated example in Figure 1. The example graph in Figure 1 contains a clique of $n/2$ vertices and a directed cycle of $n/2$ vertices. An edge (s, s') is pointing from the clique to the directed cycle. We will show that in this graph, there is some particular edge that is included in E^{rem} with a constant probability, which is too high.

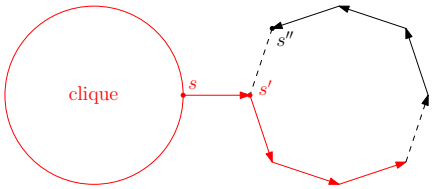

Figure 1 The figure shows the case when one vertex in the clique starts growing a ball before all vertices in the cycle and includes s, s' into the ball, while s'' is not included. s'' will be induced in another outgoing ball later. The dashed line marked the edges included in E^{rem} as a result.

The right way to compute a directed LDD on this graph is to set E^{rem} to be a single random edge on the cycle, but this is not what the parallel ball-growing approach would do. To see this, consider two cases. The first case is that the vertex s' ends up in the ball B_v of some vertex v in the clique (because v gets low random delay). The vertex s'' cannot be in this same ball B_v because the resulting out-diameter of B_v would be too large, so s'' ends up in a different ball and the edge (s'', s') is necessarily added to E^{rem}. In the second case, the vertex s' ends up in the ball B_v of some vertex v on the cycle; in this case (s, s') is added to E^{rem}. So no matter what, at least one of (s, s') or (s'', s') is added to E^{rem}, so one of these edges is added with probability at least $1/2$. (By contrast, if all edges were undirected, then a ball starting from the clique would explore the cycle in both directions up to some random threshold, and hence the edge (s', s'') would not necessarily be added to E^{rem}.)

Our approach: Our approach does not follow the random delay approach. Roughly, our algorithm simulates a variation of the directed LDD algorithm [7]. The sequential algorithm carves out the graph with disjoint balls in an arbitrary order $B_{v_1}, \ldots, B_{v_\ell}$. Doing so sequentially is inefficient in distributed models, so we instead show how to efficiently compute an index i, such that $B_{v_1} \cup \ldots \cup B_{v_{i-1}} \cup B_{v_i}$ and its complement are proportional in size. This yields a recursive algorithm with $O(\log(n))$ parallel rounds, and because our final ordering is mimicking a valid sequential ordering from [7], we are able to argue that every edge e is added to E^{rem} with a small probability. For more details, see 4.1.

Obstacle 2: Algorithms for Average Case vs Worst Case Inputs. Recall that once we reach Phase 3 of the algorithm, only the edges of E^{rem} can be negative; since the directed LDD guarantees that every edge is added to E^{rem} with small probability, this implies that every shortest path contains few negative edges *in expectation*. Bernstein et al. [7] show a simple sequential algorithm that efficiently computes shortest paths in such a graph. This algorithm works even when there are shortest paths with many negative edges, so long as the average number over shortest paths is small.

Unfortunately, such an algorithm does not seem possible in other settings. Instead, we have to settle for a weaker subroutine that requires *all* shortest paths to have few negative edges. (Technically speaking, it works in a general graph, but only returns correct distances to vertices v for which the shortest sv-path has few negative edges.) This subroutine is too weak to directly handle Phase 3 from [7]. In order to execute the framework above with our weaker subroutine, we need to introduce a more refined recursive structure for ScaleDown, which is the cause of our extra $n^{o(1)}$ factor in the time bounds.

Obstacle 3: SCCs and Their Topological Order in CONGEST. Phase 2 of our algorithm requires computing SCCs and a toplogical ordering among them. Schudy [32] gives an algorithm for computing SCCs and their topological order in the parallel model that uses

$O(\log^2 n)$ calls to non-negative-weight SSSP and yet, somewhat surprisingly, there has been no such algorithm formally written for CONGEST. Directly porting the framework of [32] to CONGEST is non-trivial; the congestion on any particular edge could be prohibitively large. We remedy this state of affairs by implementing the framework in the Distributed Minor-Aggregation Model, which abstracts away from such low-level details and can be compiled into a CONGEST algorithm.

4 Low Diameter Decomposition

Our main technical contribution is showing how to compute directed low-diameter decomposition (LDD) in various models of computation, so for the rest of the extended abstract we give an overview of our LDD algorithm, along with model-independent pseudocode (Algorithm 1). The algorithm takes as an input a directed graph with non-negative integer weights. We will defer a comprehensive discussion of the LDD algorithm, including its correctness and running time (see Lemma 4), and implementation across various models, to the full version of this paper.

4.1 Algorithm Overview

Our low diameter decomposition algorithm is presented in Algorithm 1. In this subsection, we provide an overview of Algorithm 1. The algorithm contains two phases:

Phase 1: Mark vertices as light or heavy. This phase is identical to the sequential algorithm introduced in [7]. After this phase, each vertex v will get one of the following three marks: *in-light*, *out-light*, *heavy*. It is guaranteed that w.h.p., if a vertex v is marked as (i) *in-light*, then $|\operatorname{Ball}^{\text{in}}_{\text{G}}(v, d/4)| \leq .7|V|$, (ii) *out-light*, then $|\operatorname{Ball}^{\text{out}}_{\text{G}}(v, d/4)| \leq .7|V|$, (iii) *heavy*, then $|\operatorname{Ball}^{\text{in}}_{\text{G}}(v, d/4)| > .5|V|$ and $|\operatorname{Ball}^{\text{out}}_{\text{G}}(v, d/4)| > .5|V|$. The algorithm for finding these labels can be summarized as follows. We select $\Theta(\log n)$ nodes from the graph uniformly at random and execute the SSSP algorithm starting from these nodes. The proportion of sampled nodes that are at a distance of no more than $d/4$ from a vertex v represents the size of $\operatorname{Ball}^{\text{in}}_{\text{G}}(v, d/4)$, while the proportion of nodes to which v is at a distance of no more than $d/4$ corresponds to the size of $\operatorname{Ball}^{\text{out}}_{\text{G}}(v, d/4)$. See Algorithm 1 Phase 1 for the details of how to get the marks for the proof of the guarantees.

Phase 2: Create sub-problems with small sizes. We denote the set of *in-light* vertices by V_{in}, the set of *out-light* vertices by V_{out}, and the set of *heavy* vertices by V_{heavy}. Sequentially carving our balls centered on light vertices, as in [7], would not be efficient in the models we consider. We would like to find sets which make for an efficient recursion. To this end, we first apply subroutine *FindBalancedSet* (see below for more details) on V_{in}, V_{out}. *FindBalancedSet* on V_{in} (or V_{out}) will create a random vertex set A_{in} (or A_{out}) having the following properties:

1. *(Light boundary)* It is guaranteed that each edge e is included in $\delta^-(A_{in})$ (or $\delta^+(A_{out})$) with probability $O(w(e)\log(n)/d)$. Note that this differs from Lemma 4 by a $\log n$ factor.
2. *(Balanced or contains V_*)* For $* \in \{in, out\}$, we have (i) $|A_*| \leq .9|V|$ and (ii) either $|A_*| \geq .1|V|$ or $V_* \subseteq A_*$. If $.1|V| \leq |A_*| \leq .9|V|$, we say A_* is balanced. In other words, the only case that A_* is not balanced (too small) is that V_* is completely contained in A_*.

   Now we consider two cases.

Case 1: A_{in} or A_{out} is balanced. For convenience, we only consider the case when A_{in} is balanced, i.e. $.1|V| \le |A_{in}| \le .9|V|$. The case where A_{out} is balanced is similar. In this case, we recursively call $E_1^{rem} \leftarrow LowDiameterDecomposition(G[A_{in}], d)$ and $E_2^{rem} \leftarrow LowDiameterDecomposition(G[V \backslash A_{in}], d)$, and return $\delta^-(A_{in}) \cup E_1^{rem} \cup E_2^{rem}$ as E^{rem}. Now, we verify the output guarantees.

1. *(Time cost)* Since each recursion layer decreases the size of the graph by a constant factor, the depth of the recursion tree is bounded by $O(\log n)$.
2. *(Low diameter)* Consider an SCC C of the subgraph $E - E^{rem}$. Since $\delta^-(A_{in}) \subseteq E^{rem}$, it must be the case that $C \subseteq A_{in}$ or $C \subseteq V \backslash A_{in}$. In both cases, C is included in a recursive call.
3. *(E^{rem} guarantee)* Each edge e is included in $\delta^-(A_{in})$ with probability $O(w(e)\log(n)/d)$. Each edge e can also be included in the returned edge set of a recursive call. The depth of the recursion tree is bounded by $O(\log n)$, therefore, an edge is included in E^{rem} with probability $O(w(e)\log^2 n/d)$.

Case 2: Both A_{in}, A_{out} are not balanced. In this case, we have $V_{in} \subseteq A_{in}, V_{out} \subseteq A_{out}$. We call $E_1^{rem} \leftarrow LowDiameterDecomposition(G[A_{in}], d)$, and $E_2^{rem} \leftarrow LowDiameterDecomposition(G[A_{out} \backslash A_{in}], d)$, then return $\delta^-(A_{in}) \cup \delta^+(A_{out}) \cup E_1^{rem} \cup E_2^{rem}$ as E^{rem}. Now we verify the output guarantees.

1. *(Time cost)* Notice that $|A_{in} \cup A_{out}| \le .2|V|$, thus, each recursion layer decreases the size of the graph by a constant factor; the depth of the recursion tree is bounded by $O(\log n)$.
2. *(Low diameter)* Consider an SCC C of the subgraph $E - E^{rem}$. Since $\delta^-(A_{in}), \delta^+(A_{out}) \subseteq E^{rem}$, it must be the case that $C \subseteq A_{in}$ or $C \subseteq A_{out} \backslash A_{in}$ or $C \subseteq V \backslash (A_{in} \cup A_{out}) \subseteq V_{heavy}$. In both the first two cases, C is included in a recursive call. In the third case, remember that each vertex $v \in V_{heavy}$ has the property that $|\operatorname{Ball}_{\mathrm{G}}^{\mathrm{in}}(v, d/4)| > .5|V|$ and $|\operatorname{Ball}_{\mathrm{G}}^{\mathrm{out}}(v, d/4)| > .5|V|$. Thus, any two vertices in V_{heavy} have mutual distance at most $d/2$ and so C has weak diameter at most d.
3. *(E^{rem} guarantee)* Each edge e is included in $\delta^-(A_{in})$ or $\delta^+(A_{out})$ with probability $O(w(e)\log(n)/d)$. Each edge e can also be included in the returned edge set of a recursive call. The depth of the recursion tree is bounded by $O(\log n)$, therefore, an edge is included in E^{rem} with probability $O(w(e)\log^2 n/d)$.

4.1.1 Overview of FindBalancedSet

Remember that FindBalancedSet takes V_{in} or V_{out} as input and outputs a set A_{in} or A_{out} that satisfies properties *light boundary* and *balanced* described above. For convenience, we only consider the case when V_{in} is the input. Write $V_{in} = \{v_1, v_2, ..., v_\ell\}$ (an arbitrary order).

The algorithm contains two steps.

Step 1. For each $i \in [\ell]$, sample an integer d_i following a certain geometric distribution. The detailed definition is given in the full paper. For now, we can think of the distribution in the following way: suppose a player is repeating identical independent trials, where each trial succeeds with probability $\Theta(\frac{\log n}{d})$, then d_i is the number of failed trails before the first success.

Step 2. Find the smallest $i \in [\ell]$ such that $\left|\cup_{j \le i} \operatorname{Ball}_{\mathrm{G}}^{\mathrm{in}}(v_j, d_j)\right| > 0.1|V|$, denoted as k. If k does not exist, i.e. $\left|\cup_{j \in [\ell]} \operatorname{Ball}_{\mathrm{G}}^{\mathrm{in}}(v_j, d_j)\right| \le 0.1|V|$, set $k = \ell$. Note that for a fixed i, we can compute $\left|\cup_{j \le i} \operatorname{Ball}_{\mathrm{G}}^{\mathrm{in}}(v_j, d_j)\right|$ by a single SSSP call (as opposed to computing each ball sequentially, which is inefficient); we can then binary search to find k. Return $\cup_{j \le k} \operatorname{Ball}_{\mathrm{G}}^{\mathrm{in}}(v_j, d_j)$ as A.

Algorithm 1 $E^{rem} \leftarrow LowDiameterDecomposition(G, d)$.

Input: Non-negative weighted directed graph $G = (V, E, w)$, an integer d.
Output: A random set of edges $E^{rem} \subseteq E$. (See Lemma 4 for the properties of the output.)

If G is an empty graph, return $\emptyset$;
Let n and c be defined as in full paper; // Phase 1: mark vertices as light or heavy
Sample $\lceil c \log n \rceil$ vertices in V uniformly at random, denoted as S;
For each $v \in S$, use $\mathcal{O}^{NN-SSSP}(G, v)$ to find $\text{Ball}^{\text{in}}_{\text{G}}(v, d/4)$ and $\text{Ball}^{\text{out}}_{\text{G}}(v, d/4)$;
For each $v \in V$, compute $\text{Ball}^{\text{in}}_{\text{G}}(v, d/4) \bigcap S$ and $\text{Ball}^{\text{out}}_{\text{G}}(v, d/4) \bigcap S$ using Line 4;
foreach $v \in V$ **do**
    If $|\text{Ball}^{\text{in}}_{\text{G}}(v, d/4) \bigcap S| \leq .6|S|$, mark v *in-light* // whp $|\text{Ball}^{\text{in}}_{\text{G}}(v, d/4)| \leq .7|V(G)|$
    Else if $|\text{Ball}^{\text{out}}_{\text{G}}(v, d/4) \bigcap S| \leq .6|S|$, mark v *out-light* // whp $|\text{Ball}^{\text{out}}_{\text{G}}(v, d/4)| \leq .7|V(G)|$
    Else mark v *heavy* // whp $|\text{Ball}^{\text{in}}_{\text{G}}(v, d/4)| > .5|V(G)|$ and $|\text{Ball}^{\text{out}}_{\text{G}}(v, d/4)| > .5|V(G)|$
// Phase 2: creates sub-problems with small sizes
Denote the set of *in-light* vertices by V_{in}, the set of *out-light* vertices by V_{out};
$A_{in} \leftarrow FindBalancedSet(G, V_{in}, d, in)$, $E^{rem}_{in} \leftarrow \delta^-(A_{in})$;
$A_{out} \leftarrow FindBalancedSet(G, V_{out}, d, out)$, $E^{rem}_{out} \leftarrow \delta^+(A_{out})$;
// Case 1: One of A_{in}, A_{out} is balanced.
if A_* *(* $*$ *can be in or out) has size between* $.1|V|$ *and* $.9|V|$ **then**
    $E^{rem}_1 \leftarrow LowDiameterDecomposition(G[A_*], d)$;
    $E^{rem}_2 \leftarrow LowDiameterDecomposition(G[V \backslash A_*], d)$;
    return $E^{rem}_* \bigcup E^{rem}_1 \bigcup E^{rem}_2$;
// Clean up: Check that $V \backslash (A_{in} \bigcup A_{out})$ have small weak diameter.
Pick an arbitrary vertex $u \in V \backslash (A_{in} \cup A_{out})$. Use $\mathcal{O}^{NN-SSSP}(G, u)$ to find $\text{Ball}^{\text{in}}_{\text{G}}(u, d/2), \text{Ball}^{\text{out}}_{\text{G}}(u, d/2)$;
if $V \backslash (A_{in} \cup A_{out}) \not\subseteq \text{Ball}^{\text{in}}_{\text{G}}(u, d/2) \bigcap \text{Ball}^{\text{out}}_{\text{G}}(u, d/2)$ *or* $|A_{in} \cup A_{out}| \geq .5|V|$ **then**
    return E
// Case 2: both A_{in}, A_{out} are small.
$E^{rem}_1 \leftarrow LowDiameterDecomposition(G[A_{in}], d)$;
$E^{rem}_2 \leftarrow LowDiameterDecomposition(G[A_{out} \backslash A_{in}], d)$;
return $E^{rem}_{in} \bigcup E^{rem}_{out} \bigcup E^{rem}_1 \bigcup E^{rem}_2$;

Property *balanced or contains V_{in}*. According to the definition of d_i, one can show that $d_i < d/4$ w.h.p., which implies $|\text{Ball}^{\text{in}}_{\text{G}}(v_i, d_i)| \leq .7|V|$ (because v_i is light.). Since k is the smallest integer such that $\left|\cup_{j \leq k} \text{Ball}^{\text{in}}_{\text{G}}(v_j, d_j)\right| > 0.1|V|$, it must be the case $\left|\cup_{j \leq k} \text{Ball}^{\text{in}}_{\text{G}}(v_j, d_j)\right| < 0.8|V|$. Moreover, if $\left|\cup_{j \leq k} \text{Ball}^{\text{in}}_{\text{G}}(v_j, d_j)\right| > 0.1|V|$ is not true, then $k = \ell$ and $V_{in} \subseteq A_{in}$.

Property *light boundary*. This is the most technical part and the rest of this subsection is devoted to sketching the proof idea.

Notice that the only randomness of *FindBalancedSet* comes from $d_1, d_2, ..., d_\ell$. For convenience, write $\mathbf{d} = (d_1, d_2, ..., d_\ell)$. Since $\delta^-(A)$ only depends on $\mathbf{d}$, we may define $\delta^-(A)_{\mathbf{d}}$ as the edge set $\delta^-(A)$ generated by the algorithm with $\mathbf{d}$ as the randomness.

Algorithm 2 $A \leftarrow FindBalancedSet(G, V', d, *)$.

Input: Non-negative weighted directed graph $G = (V, E, w)$, a vertex set $V' \subseteq V$ and an integer d satisfying $|\operatorname{Ball}^*_G(v, d/4)| \leq .7|V|$ for any $v \in V'$.

Output: A set of vertices $A \subseteq V$ which have a Light Boundary and is either Balanced or contains V'.

1 Suppose $V' = \{v_1, v_2, ..., v_\ell\}$. Each vertex v_i samples $d_i \sim GE[\min((c \log n)/d, 1)]_{\leq \lfloor d/4 \rfloor}$ (A geometric random variable truncated to $\lfloor d/4 \rfloor$) ;

2 Find the smallest $i \in [\ell]$ such that $\left|\cup_{j \leq i} \operatorname{Ball}^*_G(v_j, d_j)\right| > 0.1|V|$, denoted as k. If k does not exist, i.e. $\left|\cup_{j \in [\ell]} \operatorname{Ball}^*_G(v_j, d_j)\right| \leq 0.1|V|$, set $k = \ell$. (Implementation in full paper);

3 **return** $\cup_{j \leq k} \operatorname{Ball}^*_G(v_j, d_j)$, ;

To analyze the light boundary property, we will describe another algorithm that, given $\mathbf{d} = (d_1, d_2, ..., d_\ell)$, outputs an edge set $E_{\mathbf{d}}$, such that

1. $\delta^-(A)_{\mathbf{d}} \subseteq E_{\mathbf{d}}$ always holds for any $\mathbf{d}$, and
2. an edge e is included in $E_{\mathbf{d}}$ with probability $O(w(e) \log(n)/d)$.

(The algorithm to generate $E_{\mathbf{d}}$) Initially set $E_{\mathbf{d}} = \emptyset$. For iterations $i = 1, 2, ..., \ell$, do

- Mark all edges (u, v) with $u, v \in \operatorname{Ball}^{\text{in}}_G(v_i, d_i)$ as "invulnerable", and add all edges in $\delta^-(\operatorname{Ball}^{\text{in}}_G(v_i, d_i))$ that are not invulnerable to $E_{\mathbf{d}}$.

Note that $E_{\mathbf{d}}$ is produced by a sequential algorithm, which is easier to analyze, but the algorithm never actually computes $E_{\mathbf{d}}$. We can show that $\delta^-(A) \subseteq E_{\mathbf{d}}$ is always true: Recall that $A = \cup_{j \leq k} \operatorname{Ball}^{\text{in}}_G(v_j, d_j)$. Any edge in $\delta^-(A)$ is not invulnerable before the end of the k-th iteration; any edge in $\delta^-(A)$ is also on the boundary of some B_j for $j \leq k$, which means it has already been added to $E_{\mathbf{d}}$ before the end of the k-th iteration.

The last thing is to show that an edge e is included in $E_{\mathbf{d}}$ with probability $O(w(e) \log(n)/d)$. To this end, consider the following alternative explanation of the procedure when we do the i-th iteration: v_i gradually grows the radius of the ball centered on v_i, each round increases the radius by 1, and stops with probability $\Theta(\log(n)/d)$. This is exactly how d_i is defined. Observe that each edge (u, v) will be included in $E_{\mathbf{d}}$ if and only if the first v_i that grows its ball to reach v failed to reach u (if it reached u, then this edge is marked as invulnerable and will never be added to $E_{\mathbf{d}}$). By the memoryless property of the geometric distribution, this happens with probability $\Theta(w(e) \cdot \log(n)/d)$.

References

1 B. Awerbuch, M. Luby, A.V. Goldberg, and S.A. Plotkin. Network decomposition and locality in distributed computation. In *30th Annual Symposium on Foundations of Computer Science*, pages 364–369, 1989. doi:10.1109/SFCS.1989.63504.

2 Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 93–104. IEEE, 2020. doi:10.1109/FOCS46700.2020.00018.

3 Yair Bartal. Probabilistic approximation of metric spaces and its algorithmic applications. In *Proceedings of 37th Conference on Foundations of Computer Science*, pages 184–193. IEEE, 1996.

4 Ruben Becker, Yuval Emek, and Christoph Lenzen. Low diameter graph decompositions by approximate distance computation. In *ITCS*, volume 151 of *LIPIcs*, pages 50:1–50:29. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020.

5 Aaron Bernstein, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Near-optimal decremental sssp in dense weighted digraphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1112–1122. IEEE, 2020.

6 Aaron Bernstein and Danupon Nanongkai. Distributed exact weighted all-pairs shortest paths in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 334–342, 2019.

7 Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *2022 IEEE 63rd annual symposium on foundations of computer science (FOCS)*, pages 600–611. IEEE, 2022.

8 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proceedings of the 51st Annual ACM SIGACT Symposium on theory of computing*, pages 365–376, 2019.

9 Aija Berzina, Andrej Dubrovsky, Rusins Freivalds, Lelde Lace, and Oksana Scegulnaja. Quantum query complexity for some graph problems. In *SOFSEM*, volume 2932 of *Lecture Notes in Computer Science*, pages 140–150. Springer, 2004.

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

11 Nairen Cao and Jeremy Fineman. Parallel exact shortest paths in almost linear work and square root depth. In *SODA*. SIAM, 2023.

12 Nairen Cao, Jeremy T. Fineman, and Katina Russell. Brief announcement: An improved distributed approximate single source shortest paths algorithm. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, pages 493–496, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3465084.3467945.

13 Nairen Cao, Jeremy T. Fineman, and Katina Russell. Parallel shortest paths with negative edge weights. In *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '22, pages 177–190, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3490148.3538583.

14 Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F Italiano, Jakub Łącki, and Nikos Parotsidis. Decremental single-source reachability and strongly connected components in $O(m\sqrt{n})$ total update time. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 315–324. IEEE, 2016.

15 Li Chen, Rasmus Kyng, Yang P. Liu, Richard Peng, Maximilian Probst Gutenberg, and Sushant Sachdeva. Maximum flow and minimum-cost flow in almost-linear time. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 612–623, 2022. doi:10.1109/FOCS54457.2022.00064.

16 Michael B. Cohen, Aleksander Mądry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $\tilde{O}(m^{10/7} \log w)$ time: (extended abstract). In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '17, pages 752–771, USA, 2017. Society for Industrial and Applied Mathematics.

17 Christoph Dürr, Mark Heiligman, Peter Høyer, and Mehdi Mhalla. Quantum query complexity of some graph problems. *SIAM J. Comput.*, 35(6):1310–1328, 2006.

18 M. Elkin and O. Neiman. Hopsets with constant hopbound, and applications to approximate shortest paths. In *57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 128–137, Los Alamitos, CA, USA, October 2016. IEEE Computer Society. doi:10.1109/FOCS.2016.22.

19 Sebastian Forster, Gramoz Goranci, Yang P. Liu, Richard Peng, Xiaorui Sun, and Mingquan Ye. Minor sparsifiers and the distributed laplacian paradigm. In *2021 IEEE 62nd Annual*

Symposium on Foundations of Computer Science (FOCS), pages 989–999, 2022. doi:10.1109/FOCS52979.2021.00099.

20 Mohsen Ghaffari, Christoph Grunau, Bernhard Haeupler, Saeed Ilchi, and Václav Rozhoň. Improved distributed network decomposition, hitting sets, and spanners, via derandomization. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2532–2566. SIAM, 2023. doi:10.1137/1.9781611977554.ch97.

21 Mohsen Ghaffari, Fabian Kuhn, and Yannic Maus. On the complexity of local distributed graph problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 784–797, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3055399.3055471.

22 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995. doi:10.1137/S0097539792231179.

23 Arun Jambulapati, Yang P Liu, and Aaron Sidford. Parallel reachability in almost linear work and square root depth. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1664–1686. IEEE, 2019.

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

25 Gary L. Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *SPAA*, pages 196–203. ACM, 2013.

26 David Peleg and Vitaly Rubinovich. A near-tight lower bound on the time complexity of distributed mst construction. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 253–261. IEEE, 1999.

27 Václav Rozhon, Michael Elkin, Christoph Grunau, and Bernhard Haeupler. Deterministic low-diameter decompositions for weighted graphs and distributed and parallel applications. In *FOCS*, pages 1114–1121. IEEE, 2022.

28 Václav Rozhoň and Mohsen Ghaffari. Polylogarithmic-time deterministic network decomposition and distributed derandomization. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 350–363, 2020.

29 Václav Rozhoň, Bernhard Haeupler, Anders Martinsson, Christoph Grunau, and Goran Zuzic. Parallel breadth-first search and exact shortest paths and stronger notions for approximate distances. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 321–334, 2023.

30 Václav Rozhoň, Christoph Grunau, Bernhard Haeupler, Goran Zuzic, and Jason Li. Undirected $(1+\epsilon)$-shortest paths via minor-aggregates: Near-optimal deterministic parallel and distributed algorithms. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 478–487, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520074.

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

32 Warren Schudy. Finding strongly connected components in parallel using $o(\log^2 n)$ reachability queries. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 146–151, 2008.

33 Jan van den Brand, Yin Tat Lee, Danupon Nanongkai, Richard Peng, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Bipartite matching in nearly-linear time on moderately dense graphs. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 919–930. IEEE, 2020. doi:10.1109/FOCS46700.2020.00090.