Negative-Weight Single-Source Shortest Paths in Near-linear Time

Aaron Bernstein
Department of Computer Science
Rutgers University
New Brunswick, NJ, USA
bernstei@gmail.com

Danupon Nanongkai
MPI for Informatics,
University of Copenhagen,
and KTH
danupon@gmail.com

Christian Wulff-Nilsen
BARC, Department of Computer Science
University of Copenhagen
Copenhagen, Denmark
koolooz@di.ku.dk

***Abstract*—We present a randomized algorithm that computes single-source shortest paths (SSSP) in $O(m \log^8(n) \log W)$ time when edge weights are integral and can be negative.[1] This essentially resolves the classic negative-weight SSSP problem. The previous bounds are $\tilde{O}((m + n^{1.5}) \log W)$ [BLNPSSSW FOCS'20] and $m^{4/3+o(1)} \log W$ [AMV FOCS'20]. Near-linear time algorithms were known previously only for the special case of planar directed graphs [Fakcharoenphol and Rao FOCS'01].**

In contrast to all recent developments that rely on sophisticated continuous optimization methods and dynamic algorithms, our algorithm is simple: it requires only a simple graph decomposition and elementary combinatorial tools. In fact, ours is the first combinatorial algorithm for negative-weight SSSP to break through the classic $\tilde{O}(m\sqrt{n} \log W)$ bound from over three decades ago [Gabow and Tarjan SICOMP'89].

***Index Terms*—graphs and networks, path and circuit problems, graph algorithms, analysis of algorithms**

I. Introduction

We consider the single-source shortest paths (SSSP) problem with (possibly negative) integer weights. Given an m-edge n-vertex directed weighted graph $G = (V, E, w)$ with integral edge weight $w(e)$ for every edge $e \in E$ and a source vertex $s \in V$, we want to compute the distance from s to v, denoted by $\text{dist}_G(s, v)$, for every vertex in v.

Two textbook algorithms for SSSP are Bellman-Ford and Dijkstra's algorithm. Dijkstra's algorithm is near-linear time ($O(m + n \log n)$ time), but restricted to *nonnegative* edge weights.[2] With negative weights, we can use the Bellman-Ford algorithm, which only requires that there is no *negative-weight cycle* reachable from s in G; in particular, the algorithm either returns $\text{dist}_G(s, v) \neq -\infty$ for every vertex v or reports that there is a cycle reachable from s whose total weight is negative. Unfortunately, the runtime of Bellman-Ford is $O(mn)$.

Designing faster algorithms for SSSP with negative edge weights (denoted negative-weight SSSP) is one of the most fundamental and long-standing problems in graph algorithms, and has witnessed waves of exciting improvements every few decades since the 50s. Early works in the 50s, due to Shimbel [3], Ford [4], Bellman [5], and Moore [6] resulted in the $O(mn)$ runtime. In the 80s and 90s, the scaling technique led to a wave of improvements (Gabow [7], Gabow and Tarjan [8], and Goldberg [9]), resulting in runtime $O(m\sqrt{n} \log W)$, where $W \geq 2$ is the minimum integer such that $w(e) \geq -W$ for all $e \in E$.[3] In the last few years, advances in continuous optimization and dynamic algorithms have led to a new wave of improvements, which achieve faster algorithms for the more general problems of transshipment and min-cost flow, and thus imply the same bounds for negative-weight SSSP (Cohen, Madry, Sankowski, Vladu [12]; Axiotis, Madry, Vladu [13]; BLNPSSSW [14]–[16]). This line of work resulted in an near-linear runtime ($\tilde{O}((m+n^{1.5}) \log W)$ time) on moderately dense graphs [14] and $m^{4/3+o(1)} \log W$ runtime on sparse graphs [13].[4] For the special case of planar directed graphs [17]–[21], near-linear time complexities were known since the 2001 breakthrough of Fakcharoenphol and Rao [19] where the best current bound is $O(n \log^2(n)/ \log \log n)$ [21]. No near-linear time algorithm is known even for a somewhat larger class of graphs such as bounded-genus and minor-free graphs (which still requires $\tilde{O}(n^{4/3} \log W)$ time [22]). This state of the art motivates two natural questions:

1) *Can we get near-linear runtime for all graphs?*
2) *Can we achieve efficient algorithms without complex machinery?*

For the second question, note that currently all state-of-the-art results for negative-weight SSSP are based on min-cost

Full version is available at https://arxiv.org/abs/2203.03456.

This work was done while Nanongkai was at the University of Copenhagen, Denmark. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672. Nanongkai was also partially supported by the Swedish Research Council (Reg. No. 2019-05622).

Wulff-Nilsen was supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme.

Bernstein is supported by NSF CAREER grant 1942010.

[1]Throughout, n and m denote the number of vertices and edges, respectively, and $W \geq 2$ is such that every edge weight is at least $-W$. $\tilde{O}$ hides polylogarithmic factors.

[2]In the word RAM model, Thorup improved the runtime to $O(m + n \log \log(C))$ when C is the maximal edge weight [1] and to linear time for *undirected* graphs [2].

[3]The case when n is big and W is small can be improved by the $O(n^\omega W)$-time algorithms of Sankowski [10], and Yuster and Zwick [11].

[4]$\tilde{O}$-notation hides polylogarithmic factors. The dependencies on W stated in [13], [14] are slightly higher than what we state here. These dependencies can be reduced by standard techniques (weight scaling, adding dummy source, and eliminating high-weight edges).

2575-8454/22/$31.00 ©2022 IEEE
DOI 10.1109/FOCS54457.2022.00063

flow algorithms, and hence rely on sophisticated continuous optimization methods and a number of complex dynamic algebraic and graph algorithms (e.g. [23]–[28]). It would be useful to develop simple efficient algorithms that are specifically tailored to negative-weight SSSP, and thus circumvent the complexity currently inherent in flow algorithms; the best known bound of this kind is still the classic $O(m\sqrt{n}\log(W))$ from over three decades ago [8], [9]. A related question is whether it is possible to achieve efficient algorithms for the problem using combinatorial tools, or whether there are fundamental barriers that make continuous optimization necessary.

A. Our Result

In this paper we resolve both of the above questions for negative-weight SSSP: we present a simple combinatorial algorithm that reduces the running time all the way down to near-linear.

Theorem I.1. *There exists a randomized (Las Vegas) algorithm that takes $O(m\log^8(n)\log(W))$ time with high probability (and in expectation) for an m-edge input graph G_{in} and source s_{in}. It either returns a shortest path tree from s_{in} or returns a negative-weight cycle.*

Our algorithm relies only on basic combinatorial tools; the presentation is self-contained and only uses standard black-boxes such as Dijkstra's and Bellman-Ford algorithms. In particular, it is a scaling algorithm enhanced by a simple graph decomposition algorithm called *Low Diameter Decomposition* which has been studied since the 80s; our decomposition is obtained in a manner similar to some known algorithms (see Section I-B for a more detailed discussion). Our main technical contribution is showing how low-diameter decomposition—which works only on graphs with non-negative weights—can be used to develop a recursive scaling algorithm for SSSP with negative weights. As far as we know, all previous applications of this decomposition were used for parallel/distributed/dynamic settings for problems that do not involve negative weights, and our algorithm is also the first to take advantage of it in the classical sequential setting; we also show that in this setting, there is a simple and efficient algorithm to compute it.

a) Perspective on Other Problems:: While our result is specific to negative-weight SSSP, we note that question *(2)* above in fact applies to a much wider range of problems. The current landscape of graph algorithms is that for many of the most fundamental problems, including ones taught in undergraduate courses and used regularly in practice, the state-of-the-art solution is a complex algorithm for the more general min-cost flow problem: some examples include negative-weight SSSP, bipartite matching, the assignment problem, edge/vertex-disjoint paths, s-t cut, densest subgraph, max flow, transshipment, and vertex connectivity. This suggests a research agenda of designing simple algorithms for these fundamental problems, and perhaps eventually their generalizations such as min-cost flow. We view our result on negative-weight SSSP as a first step in this direction.

b) Independent Result [29].: Independently from our result, the recent major breakthrough by Chen, Kyng, Liu, Peng, Probst Gutenberg, and Sachdeva [29] culminates the line of works based on continuous optimization (e.g. [12]–[16], [30]–[36]) and achieves an almost-linear time bound[5] for min-cost flow. The authors thus almost match our bounds for negative-weight SSSP as a special case of their result: their runtime is $m^{1+o(1)}\log(W)$ versus our $O(m\cdot\text{polylog}(n)\log(W))$ bound. The two results are entirely different, and as far as we know there is no overlap in techniques.

The above landmark result essentially resolves the running-time complexity for a wide range of fundamental graph problems, modulo the extra $m^{o(1)}$ factor. We believe that this makes it a natural time to pursue question *(2)* for these problems, outlined above.

B. Techniques

Our main contribution is a new recursive scaling algorithm called ScaleDown: see Section IV, including an overview in Section IV-A. In this subsection, we highlight other techniques that may be of independent interest.

a) Low-Diameter Decomposition.: One of our key subroutines is an algorithm that decomposes any directed graph with *non-negative* edge weights into strongly-connected components (SCCs) of small diameter. In particular, the algorithm computes a small set of edges E^{rem} such that all SCCs in the graph $G\setminus E^{rem}$ have small weak diameter. Although the lemma below only applies to non-negative weights, we will show that it is in fact extremely useful for our problem.

Lemma I.2. *There is an algorithm* $\text{LowDiamDecomposition}(G,D)$ *with the following guarantees:*

- *INPUT: an m-edge, n-vertex graph $G=(V,E,w)$ with non-negative integer edge weight function w and a positive integer D.*
- *OUTPUT: A set of edges E^{rem} with the following guarantees:*
  - *each SCC of $G\setminus E^{rem}$ has weak diameter at most D; that is, if u,v are in the same SCC, then $\text{dist}_G(u,v)\le D$ and $\text{dist}_G(v,u)\le D$.*
  - *For every $e\in E$, $\Pr[e\in E^{rem}] = O\left(\frac{w(e)\cdot(\log n)^2}{D}+n^{-10}\right)$. These probabilities are not guaranteed to be independent.*[6]
- *RUNNING TIME: The algorithm has running time $O((m+n)\log^2 n)$*

The decomposition above is similar to other low-diameter decompositions used in both undirected and directed graphs, though the precise guarantees vary a lot between papers [37]–[49]. The closest similarity is to the algorithm PARTITION of Bernstein, Probst-Gutenberg, and Wulff-Nilsen [48]. The main

[5] $\tilde{O}(m^{1+o(1)}\log^2 U)$ time when vertex demands, edge costs, and upper/lower edge capacities are all integral and bounded by U in absolute value.

[6] The 10 in the exponent suffices for our application but can be replaced by an arbitrarily large constant.

difference is that the algorithm of [48] needed to work in a dynamic setting, and as a result their algorithm is too slow for our purposes. Our decomposition algorithm follows the general framework of [48], but with several key differences to ensure faster running time; our algorithm is also simpler, since it only applies to the static setting. In the full version of the paper on arXiv [50], we present the entire algorithm from scratch.

b) No Negative-Weight Cycle Assumption via a Black-Box Reduction.: Although it is possible to prove Theorem I.1 directly, the need to return the actual cycle somewhat complicates the details. For this reason, we focus most of our paper on designing an algorithm that returns correct distances when the input graph contains no negative-weight cycle and guarantees nothing otherwise. See the description of subroutine SPmain in Theorem III.4 as an example. We can focus on the above case because we have a black-box reduction from Theorem I.1 to the above case that incurs an extra $O(\log^2(n))$ factor in the runtime. See [50] for details.

c) Log Factors.: We focused this paper on ease of presentation, and have thus not optimized the log factors in Theorem I.1. For example, we could likely shave two log factors by finding the cycle directly, rather than using the black-box reduction above.

II. Preliminaries

Throughout, we only consider graphs with integer weights. For any weighted graph $G = (V, E, w)$, define $V(G) = V, E(G) = E$, and

$$E^{neg}(G) := \{e \in E \mid w(e) < 0\}.$$

Define $W_G := \max\{2, -\min_{e\in E}\{w(e)\}\}$; that is, W_G is the most negative edge weight in the graph[7]. Given any set of edges $S \subseteq E$ we define $w(S) = \sum_{e\in S} w(e)$. We say that a cycle C in G is a negative-weight cycle if $w(C) < 0$. We define $\mathrm{dist}_G(u, v)$ to be the shortest distance from u to v; if there is a negative-weight cycle on some uv-path then we define $\mathrm{dist}_G(u, v) = -\infty$.

Consider graph $G = (V, E, w)$ and consider subsets $V' \subseteq G$ and $E' \subseteq E$. We define $G[V']$ to be the subgraph of G induced by V'. We slightly abuse notation and write $H = (V', E', w)$ to denote the subgraph where the weight function w is restricted to edges in E'. We define $G \setminus V' = G[V \setminus V']$ and $G \setminus E' = (V, E \setminus E', w)$; i.e. they are graphs where we remove vertices and edges in V' and E' respectively. We sometimes write $G\setminus v$ and $E\setminus e$ instead of $G\setminus\{v\}$ and $G\setminus\{e\}$, respectively, for any $v \in V$ and $e \in E$. We say that a subgraph H of G has *weak* diameter D if for any $u, v \in V(H)$ we have that $\mathrm{dist}_G(u, v) \leq D$. We always let G_{in} and s_{in} refer to the main input graph/source of Theorem I.1.

Assumption II.1 (Properties of input graph G_{in}; justified by Lemma II.2)**.** We assume throughout the paper that the main input graph $G_{in} = (V, E, w_{in})$ satisfies the following properties:

1) $w_{in}(e) \geq -1$ for all $e \in E$ (thus, $W_{G_{in}} = 2$).
2) Every vertex in G_{in} has constant out-degree.

Lemma II.2. *Say that there is an algorithm as in Theorem I.1 for the special case when the graph G_{in} satisfies the properties of Assumption II.1, with running time $T(m, n)$. Then there is algorithm as in Theorem I.1 for any input graph G_{in} with integral weights that has running time $O(T(m, m)\log(W_{G_{in}}))$.*

Proof. Let us first consider the first assumption, i.e that $w_{in}(e) \geq -1$. The scaling framework of Goldberg [9] shows that an algorithm for this case implies an algorithm for any integer-weighted G at the expense of an extra $\log(W_G)$ factor.[8]

For the assumption that every vertex in G_{in} has constant out-degree, we use a by-now standard technique of creating Θ(out-degree(v)) copies of each vertex v, so that each copy has constant out-degree; the resulting graph was $O(E)$ vertices and $O(E)$ edges.[9] □

a) Dummy Source and Negative Edges.: The definitions below capture a common transformation we apply to negative weights and also allow us to formalize the number of negative edges on a shortest path. Note that most of our algorithms/definitions will not refer to the input source s_{in}, but instead to a dummy source that has edges of weight 0 to every vertex.

Definition II.3 (G_s,w_s,G^B,w^B,G^B_s,w^B_s)**.** Given any graph $G = (V, E, w)$, we let $G_s = (V \cup \{s\}, E \cup \{(s, v)\}_{v\in V}, w_s)$ refer to the graph G with a dummy source s added, where there is an edge of weight 0 from s to v for every $v \in V$ and no edges into s. Note that G_s has a negative-weight cycle if and only if G does and that $\mathrm{dist}_{G_s}(s, v) = \min_{u\in V} \mathrm{dist}_G(u, v)$.

For any integer B, let $G^B = (V, E, w^B)$ denote the graph obtained by adding B to all negative edge weights in G, i.e. $w^B(e) = w(e) + B$ for all $e \in E^{neg}(G)$ and $w^B(e) = w(e)$ for $e \in E \setminus E^{neg}(G)$. Note that $(G^B)_s = (G_s)^B$ so we can simply write $G^B_s = (V \cup \{s\}, E \cup \{(s, v)\}_{v\in V}, w^B_s)$.

Definition II.4 ($\eta_G(v)$, $P_G(v)$)**.** For any graph $G = (V, E, w)$, let G_s and s be as in Definition II.3. Define $\eta_G(v) := \min\{|E^{neg}(G) \bigcap P| : P \text{ is a shortest } sv- \text{ path in } G_s\}$ unless $\mathrm{dist}_{G,s}(s, v) = -\infty$ in which case we define $\eta_G(v) := \infty$. Let $\eta(G) = \max_{v\in V} \eta_G(v)$. When $\mathrm{dist}_G(s, v) \neq -\infty$, let $P_G(v)$ be a shortest sv-path on G_s such that

$$|E^{neg}(G) \bigcap P_G(v)| = \eta_G(v). \tag{1}$$

When the context is clear, we drop the subscripts.

A. Price Functions and Equivalence

Our algorithm heavily relies on price functions, originally introduced by Johnson [51]

[7] We set $W_G \geq 2$ so that we can write $\log(W_G)$ in our runtime

[8] Quoting [9]: "Note that the basic problem solved at each iteration of the bit scaling method is a special version of the shortest paths problem where the arc lengths are integers greater or equal to -1."

[9] One way to do this is to replace every vertex with a directed zero-weight cycle whose size is the in-degree plus out-degree of the vertex and then attach the adjacent edges to this cycle.

Definition II.5 (Price Function)**.** Consider a graph $G = (V, E, w)$ and let ϕ be any function: $V \to \mathbb{Z}$, $\mathbb{Z}$ is the set of integers. 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 $(G_\phi)_\psi = G_{\phi+\psi}$.

Definition II.6 (Graph Equivalence)**.** We say that two graphs $G = (V, E, w)$ and $G' = (V, E, w')$ are *equivalent* if **(1)** any shortest path in G is also a shortest path in G' and vice-versa and **(2)** G contains a negative-weight cycle if and only if G' does.

Lemma II.7 ([51])**.** *Consider any graph $G = (V, E, w)$ and price function ϕ. For any pair $u, v \in V$ we have $\mathrm{dist}_{G_\phi}(u, v) = \mathrm{dist}_G(u, v) + \phi(u) - \phi(v)$, and for any cycle C we have $w(C) = w_\phi(C)$. As a result, G and G_ϕ are equivalent. Finally, if $G' = (V, E, w)$ and $G' = (V, E, w')$ and $w' = c \cdot w(e)$ for some positive c, then G and G' are equivalent.*

The overall goal of our algorithm will be to compute a price function ϕ such that all edge weights in G_ϕ are non-negative (assuming no negative-weight cycle); we can then run Dijkstra on G_ϕ. The lemma below, originally used by Johnson, will be one of the tools we use.

Lemma II.8 ([51])**.** *Let $G = (V, E)$ be a directed graph with no negative-weight cycle and let s be the dummy source in G_s. Let $\phi(v) = \mathrm{dist}_{G_s}(s, v)$ for all $v \in V$. Then, all edge weights in G_ϕ are non-negative. (The lemma follows trivially from the fact that $\mathrm{dist}(s, v) \le \mathrm{dist}(s, u) + w(u, v)$.)*

III. The Framework

In this section we describe the input/output guarantees of all the subroutines used in our algorithm, as well as some of the algorithms themselves.

A. Basic Subroutines

Lemma III.1 ($\mathrm{Dijkstra}$)**.** *There exists an algorithm $\mathrm{Dijkstra}(G, s)$ that takes as input a graph G with non-negative edge weights and a vertex $s \in V$ and outputs a shortest path tree from s in G. The running time is $O(m+n\log(n))$.*

It is easy to see that if G is a DAG (Directed Acyclic Graph), computing a price function ϕ such that G_ϕ has non-negative edge weights is straightforward: simply loop over the topological order $v_1, ..., v_n$ and set $\phi(v_i)$ so that all incoming edges have non-negative weight. The lemma below generalizes this approach to graphs where only the "DAG part" has negative edges.

Lemma III.2 ($\mathrm{FixDAGEdges}$)**.** *There exists an algorithm $\mathrm{FixDAGEdges}(G, \mathcal{P})$ that takes as input a graph G and a partition $\mathcal{P} := \{V_1, V_2, \ldots\}$ of vertices of G such that*

1) *for every i, the induced subgraph $G[V_i]$ contains no negative-weight edges, and*
2) *when we contract every V_i into a node, the resulting graph is a DAG (i.e. contains no cycle).*

The algorithm outputs a price function $\phi : V \to \mathbb{Z}$ such that $w_\phi(u, v) \ge 0$ for all $(u, v) \in E$. The running time is $O(m+n)$.

Proof sketch. The algorithm is extremely simple: it loops over the SCCs V_i in topological order, and when it reaches V_i it sets the same price $\phi(v)$ for every $v \in V_i$ that ensures there are no non-negative edges entering V_i; since all $\phi(v)$ are the same, this does not affect edge-weights inside V_i. The pseudocode and analysis can be found in the full version of the paper [50]. □

The next subroutine shows that computing shortest paths in a graph G can be done efficiently as long as $\eta(v)$ is small on average (see Definition II.4 for $\eta(v)$). Note that this subroutine is the reason we use the assumption that every vertex has constant out-degree (Assumption II.1).

Lemma III.3. ($\mathrm{ElimNeg}$) *There exists an algorithm $\mathrm{ElimNeg}(G)$ that takes as input a graph $G = (V, E, w)$ in which all vertices have constant out-degree. The algorithm outputs a price function ϕ such that $w_\phi(e) \ge 0$ for all $e \in E$ and has running time $O(\log(n) \cdot (n + \sum_{v \in V} \eta_G(v)))$ (Definition II.4); note that if G contains a negative-weight cycle then $\sum_{v \in V} \eta_G(v) = \infty$ so the algorithm will never terminate and hence not produce any output.*

Proof sketch. Creating the graph G_s as in Definition II.3, computing all $\mathrm{dist}_{G_s}(s, v)$, and then applying Lemma II.7 yields the desired price function. Thus, it suffices to describe how $\mathrm{ElimNeg}(G)$ computes $\mathrm{dist}_{G_s}(s, v)$ for all $v \in V$.

The algorithm is a straightforward combination of Dijkstra's and Bellman-Ford's algorithms. The algorithm maintains distance estimates $d(v)$ for each vertex v. It then proceeds in multiple iterations, where each iteration first runs a Dijkstra Phase that ensures that all non-negative edges are relaxed and then a Bellman-Ford Phase ensuring that all negative edges are relaxed. Consider a vertex v and let P be a shortest path from s to v in G_s with $\eta_G(v)$ edges of $E^{neg}(G)$. It is easy to see that $\eta_G(v) + 1$ iterations suffice to ensure that $d(v) = \mathrm{dist}_{G_s}(s, v)$. In each of these iterations, v is extracted from and added to the priority queue of the Dijkstra Phase only $O(1)$ times. Furthermore, after the $\eta_G(v) + 1$ iterations, v will not be involved in any priority queue operations. Since the bottleneck in the running time is the queue updates, we get the desired time bound of $O(\log(n) \cdot \sum_{v \in V} (\eta_G(v)+1)) = O(\log(n) \cdot (n + \sum_{v \in V} \eta_G(v)))$.

This completes the proof sketch. The full proof can be found in [50]. □

B. The Interface of the Two Main Algorithms

Our two main algorithms are called $\mathrm{ScaleDown}$ and SPmain. The latter is a relatively simple outer shell. The main technical complexity lies in $\mathrm{ScaleDown}$, which calls itself recursively.

Theorem III.4 (SPmain)**.** *There exists an algorithm $\mathrm{SPmain}(G_{in}, s_{in})$ that takes as input a graph G_{in} and a source s_{in} satisfying the properties of Assumption II.1. If the*

algorithm terminates, it outputs a shortest path tree T from s_{in}. The running time guarantees are as follows:

- *If the graph G_{in} contains a negative-weight cycle then the algorithm never terminates.*
- *If the graph G_{in} does not contain a negative-weight cycle then the algorithm has expected running time $\mathcal{T}_{spmain} = O(m\log^5(n))$.*

Theorem III.5 (ScaleDown)**.** *There exists the following algorithm* $\mathrm{ScaleDown}(G=(V,E,w),\Delta,B)$.

1) *INPUT REQUIREMENTS:*
   a) *B is positive integer, w is integral, and $w(e) \geq -2B$ for all $e \in E$*
   b) *If the graph G does not contain a negative-weight cycle then the input must satisfy $\eta(G^B) \leq \Delta$; that is, for every $v \in V$ there is a shortest sv-path in G_s^B with at most Δ negative edges (Definitions II.3 and II.4)*
   c) *All vertices in G have constant out-degree*
2) *OUTPUT: If it terminates, the algorithm returns an integral price function ϕ such that $w_\phi(e) \geq -B$ for all $e \in E$*
3) *RUNNING TIME: If G does not contain a negative-weight cycle, then the algorithm has expected runtime $O\left(m\log^3(n)\log(\Delta)\right)$. Remark: If G contains a negative-weight cycle, there is no guarantee on the runtime, and the algorithm might not even terminate; but if the algorithm does terminate, it always produces a correct output.*

a) Remark: Termination and Negative-Weight Cycles.: Note that for both algorithms above, if G_{in} contains a negative-weight cycle then the algorithm might simply run forever, i.e. not terminate and not produce any output. In fact the algorithm SPmain *never* terminates if G_{in} contains a negative-weight cycle. The algorithm $\mathrm{ScaleDown}$ may or may not terminate in this case: our guarantee is only that if it does terminate, it always produces a correct output.

In short, neither algorithm is required to produce an output in the case where G_{in} contains a negative-weight cycle, so we recommend the reader to focus on the case where G_{in} does not contain a negative-weight cycle.

b) Proof sketch of Theorem I.1.: Algorithm SPmain leads to our main result in Theorem I.1. First, it is easy to see that SPmain leads to a Monte Carlo algorithm with the following guarantees: if G_{in} does not contain a negative-weight cycle then the algorithm outputs correct distances with probability $\geq 1/2$ and an error message otherwise; if G_{in} contains a negative-weight cycle the algorithm always outputs an error message. This Monte Carlo algorithm is obtained by simply running $\mathrm{SPmain}(G_{in}, s_{in})$ for $2\mathcal{T}_{spmain}$ time steps, and returning an error message if SPmain fails to terminate within that time; by Markov's inequality, the probability of success is $\geq 1/2$.

We then show that a Monte Carlo algorithm with the above guarantees can be converted in a black-box manner into the Las Vegas result of Theorem I.1; See [50] for details.

IV. Algorithm ScaleDown (Theorem III.5)

We start by describing the algorithm $\mathrm{ScaleDown}$, as this contains our main conceptual contributions; the much simpler algorithm SPmain is described in the following section. Full pseudocode of $\mathrm{ScaleDown}$ is given in Algorithm 1. The algorithm mostly works with graph $G^B = (V, E, w^B)$. For the analysis, the readers may want to familiarize themselves with, e.g., G_s^B, w_s^B, $P_{G^B}(v)$ and $\eta(G^B)$ from Definitions II.3 and II.4. In particular, throughout this section, source s *always* refers to a dummy source with edges of weight 0 to every vertex.

Note that $m = \Theta(n)$ since the input condition requires constant out-degree for every vertex. So, we use m and n interchangeably in this section. We briefly describe the $\mathrm{ScaleDown}$ algorithm and sketch the main ideas of the analysis in Section IV-A, before showing the full analysis in Section IV-B.

A. Overview

The algorithm runs in phases, where in the last phase it calls $\mathrm{ElimNeg}(G_{\phi_2}^B)$ for some price function ϕ_2. Recall (Lemma III.3) that if $\mathrm{ElimNeg}$ terminates, it returns price function ψ' that makes all edges in $G_{\phi_2}^B$ non-negative; in other words, $G_{\phi_3}^B$ contains no negative weights for $\phi_3 = \phi_2 + \psi'$. This means that $w_{\phi_3}(e) \geq -B$ for all $e \in E$ as desired (because $w^B(e) \leq w(e) + B$). This already proves the output correctness of $\mathrm{ScaleDown}$ (Item 2 of Theorem III.5). (See Theorem IV.1 for the detailed proof.) Thus it remains to bound the runtime when G contains no negative-weight cycle (Item 3). *In the rest of this subsection we assume that G contains no negative-weight cycle.*

Bounding the runtime when $\Delta \leq 2$ is easy: The algorithm simply jumps to Phase 3 with $\phi_2 = 0$ (Line 1 in Algorithm 1). Since $\eta(G^B) \leq \Delta \leq 2$ (the input requirement; Item 1b), the runtime of $\mathrm{ElimNeg}(G^B)$ is $O((m+\sum_{v\in V}\eta_{G^B}(v))\log m) = O(m\Delta\log m) = O(m\log m)$.

For $\Delta > 2$, we require some properties from Phases 0-2 in order to bound the runtime of Phase 3. In Phase 0, we partition vertices into strongly-connected components (SCCs)[10] $V_1, V_2, \ldots$ such that each V_i has weak diameter $dB = B\Delta/2$ in G. We do this by calling $E^{rem} \leftarrow \mathrm{LowDiamDecomposition}(G_{\geq 0}^B, dB)$, where $G_{\geq 0}^B$ is obtained by rounding all negative weights in G^B up to 0; we then let $V_1, V_2, \ldots$ be the SCCs of $G^B \setminus E^{rem}$. (We need $G_{\geq 0}^B$ since $\mathrm{LowDiamDecomposition}$ can not handle negative weights.[11]) See Lemma IV.3 for the formal statement and proof.

The algorithm now proceeds in three phases. In Phase 1 it computes a price function ϕ_1 that makes the edges inside each SCC V_i non-negative; in Phase 2 it computes ϕ_2 such that the edges between SCCs in $G^B \setminus E^{rem}$ are also non-negative; finally in phase 3 it makes non-negative the edges in E^{rem} by calling $\mathrm{ElimNeg}$.

[10] Recall that a SCC is a *maximal* set $C \subseteq V$ such that for every $u, v \in V$, there are paths from u to v and from v to u. See, e.g., Chapter 22.5 in [52].

[11] One can also use $G_{\geq 0}$ instead of $G_{\geq 0}^B$. We choose $G_{\geq 0}^B$ since some proofs become slightly simpler.

Algorithm 1: Algorithm for ScaleDown($G = (V, E, w), \Delta, B$)

if $\Delta \leq 2$ **then**
  Let $\phi_2 = 0$ and jump to Phase 3 (Line 10)
Let $d = \Delta/2$. Let $G^B_{\geq 0} := (V, E, w^B_{\geq 0})$ where $w^B_{\geq 0}(e) := \max\{0, w^B(e)\}$ for all $e \in E$
// Phase 0: Decompose V to SCCs $V_1, V_2, \ldots$ with weak diameter dB in G
$E^{rem} \leftarrow$ LowDiamDecomposition($G^B_{\geq 0}, dB$) (Lemma I.2)
Compute Strongly Connected Components (SCCs) of $G^B \setminus E^{rem}$, denoted by $V_1, V_2, \ldots$
// Properties: (Lemma IV.3) For each $u, v \in V_i$, $\text{dist}_G(u, v) \leq dB$.
// (Lemma IV.4) If $\eta(G^B) \leq \Delta$, then for every $v \in V_i$, $E[P_{G^B}(v) \cap E^{rem}] = O(\log^2 n)$
// Phase 1: Make edges inside the SCCs $G^B[V_i]$ non-negative
Let $H = \bigcup_i G[V_i]$, i.e. H only contains edges inside the SCCs.
// (Lemma IV.5) If G has no negative-weight cycle, then $\eta(H^B) \leq d = \Delta/2$.
$\phi_1 \leftarrow$ ScaleDown($H, \Delta/2, B$)
// (Corollary IV.6) $w_{H^B_{\phi_1}}(e) \geq 0$ for all $e \in H$
// Phase 2: Make all edges in $G^B \setminus E^{rem}$ non-negative
$\psi \leftarrow$ FixDAGEdges($G^B_{\phi_1} \setminus E^{rem}, \{V_1, V_2, \ldots\}$) (Lemma III.2)
$\phi_2 \leftarrow \phi_1 + \psi$ // (Lemma IV.7) All edges in $(G^B \setminus E^{rem})_{\phi_2}$ are non-negative
// Phase 3: Make all edges in G^B non-negative
$\psi' \leftarrow$ ElimNeg($G^B_{\phi_2}$) (Lemma III.3)
// (Theorem IV.2) expected time $O(m \log^3 m)$
$\phi_3 = \phi_2 + \psi'$ // (Theorem IV.1) All edges in $G^B_{\phi_3}$ are non-negative.
return ϕ_3 ; // Since $w^B_{\phi_3}(e) \geq 0$, we have $w_{\phi_3}(e) \geq -B$

(Phase 1) Our goal in Phase 1 is to compute ϕ_1 such that $w^B_{\phi_1}(e) \geq 0$ for every edge e in $G^B[V_i]$ for all i. To do this, we recursively call ScaleDown($H, \Delta/2, B$), where H is a union of all the SCCs $G[V_i]$. The main reason that we can recursively call ScaleDown with parameter $\Delta/2$ is because we can argue that, when G does not contain a negative-weight cycle,

$$\eta(H^B) \leq d = \Delta/2.$$

As a rough sketch, the above bound holds because if any shortest path P from dummy source s in some $(G^B[V_i])_s$ contains more than d negative-weight edges, then it can be shown that $w(P) < -dB$; this is the step where we crucially rely on the difference between $w^B(P)$ and $w(P)$. Combining $w(P) < -dB$ with the fact that $G^B[V_i]$ has weak diameter at most dB implies that G contains a negative-weight cycle. See Lemma IV.5 for the detailed proof.

(Phase 2) Now that all edges in $G^B_{\phi_1}[V_i]$ are non-negative, we turn to the remaining edges in $G^B \setminus E^{rem}$. Since these remaining edges (i.e. those not in the SCCs) form a directed acyclic graph (DAG), we can simply call FixDAGEdges($G^B_{\phi_1} \setminus E^{rem}, \{V_1, V_2, \ldots\}$) (Lemma III.2) to get a price function ψ such that all edges in $(G^B_{\phi_1} \setminus E^{rem})_\psi = G^B_{\phi_2} \setminus E^{rem}$ are non-negative. (See Lemma IV.7.)

(Phase 3) By the time we reach this phase, the only negative edges remaining are the ones in E^{rem}; that is, $E^{neg}(G^B_{\phi_2}) \subseteq E^{rem}$. We are now ready to show that the runtime of Phase 3, which is $O((m + \sum_{v \in V} \eta_{G^B_{\phi_2}}(v)) \log m)$ (Lemma III.3), is $O(m \log^3 m)$ in expectation.

We do so by proving that for any $v \in V$,

$$E\left[\eta_{G^B_{\phi_2}}(v)\right] = O(\log^2 m).$$

(See Equation (5) near the end of the next subsection.) A competitive reader might want to try to prove the above via a series of inequalities: $\eta_{G^B_{\phi_2}}(v) \leq |P_{G^B}(v) \cap E^{neg}(G^B_{\phi_2})| \leq |P_{G^B}(v) \cap E^{rem}|$, and also, the guarantees of LowDiamDecomposition (Lemma I.2) imply that after Phase 0, $E\left[|P_{G^B}(v) \cap E^{rem}|\right] = O(\log^2 m)$. (Proved in Lemma IV.4.)

Finally, observe that there are $O(\log \Delta)$ recursive calls, and the runtime of each call is dominated by the $O(m \log^3 m)$ time of Phase 3. So, the total expected runtime is $O(m \log^3(m) \log \Delta)$

a) Remark.: Our sequence of phases 0-3 is reminiscent of the sequencing used by Bernstein, Probst-Gutenberg, and Saranurak in their result on dynamic reachability [53], although the actual work within each phase is entirely different, and the decompositions have different guarantees. The authors of [53] decompose the graph into a DAG of *expanders* plus some separator edges (analogous to our phase 0); they then handle reachability inside expanders (phase 1), followed by reachability using the DAG edges (phase 2), and finally incorporate the separator edges (phase 3).

B. Full Analysis

Theorem III.5 follows from Theorems IV.1 and IV.2 below. We start with Theorem IV.1 which is quite trivial to prove.

Theorem IV.1. ScaleDown($G = (V, E, w), \Delta, B$) *either does not terminate or returns* $\phi = \phi_3$ *such that* $w_\phi(e) \geq -B$ *for all* $e \in E$.

Proof. Consider when we call ElimNeg($G^B_{\phi_2}$) (Lemma III.3) in Phase 3 for some integral price function ϕ_2. Either this step does not terminate or returns an integral price function ψ' such that $(G^B_{\phi_2})_{\psi'} = G^B_{\phi_2 + \psi'} = G^B_{\phi_3}$ contains no negative-weight

edges; i.e. $w^B_{\phi_3}(e) \geq 0$ for all $e \in E$. Since $w^B(e) \leq w(e)+B$, we have $w_{\phi_3}(e) \geq w^B_{\phi_3}(e) - B \geq -B$ for all $e \in E$. □

Theorem IV.1 implies that the output condition of ScaleDown (item 2 in Theorem III.5) is always satisfied, regardless of whether G contains a negative-weight cycle or not. It remains to show that if G does not contain a negative-weight cycle, then $\mathrm{ScaleDown}(G = (V, E, w), \Delta, B)$ has expected runtime of $O(m \log^3(m) \log(\Delta))$. It suffices to show the following.

Theorem IV.2. *If G does not contain a negative-weight cycle, then the expected time complexity of Phase 3 is $O(m \log^3 m)$.*

This suffices because, first of all, it is easy to see that Phase 0 requires $O(m \log^2(m))$ time (by Lemma I.2) and other phases (except the recursion on Line 7) requires $O(m + n)$ time. Moreover, observe that if G contains no negative-weight cycle, then the same holds for H in the recursion call $\mathrm{ScaleDown}(H, \Delta/2, B)$ (Line 7 of Algorithm 1); thus, if G contains no negative-weight cycle, then all recursive calls also get an input with no negative-weight cycle. So, by Theorem IV.2 the time to execute a single call in the recursion tree is $O(m \log^3 m)$ in expectation. Since there are $O(\log \Delta)$ recursive calls, the total running time is $O(m \log^3(m) \log(\Delta))$ by linearity of expectation.

a) Proof of Theorem IV.2.: The rest of this subsection is devoted to proving Theorem IV.2. From now on, we consider any graph G that does not contain a negative-weight cycle. (We often continue to state this assumption in lemma statements so that they are self-contained.)

b) Base case: $\Delta \leq 2$.: This means that for every vertex v, $\eta_{G^B}(v) \leq \eta(G^B) \leq \Delta \leq 2$ (see the input requirement of ScaleDown in Item 1b of Theorem III.5). So, the runtime of Phase 3 is

$$O\left(\left(m + \sum_{v \in V} \eta_{G^B}(v)\right) \log m\right) = O\left(m\Delta \log m\right) = O\left(m \log m\right).$$

We now consider when $\Delta > 2$ and show properties achieved in each phase. We will use these properties from earlier phases in analyzing the runtime of Phase 3.

c) Phase 0: Low-diameter Decomposition.: It is straightforward that the SCCs $G[V_i]$ have weak diameter at most dB (this property will be used in Phase 1):

Lemma IV.3. *For every i and every $u, v \in V_i$, $\mathrm{dist}_G(u, v) \leq dB$.*

Proof. For every $u, v \in V_i$, we have $\mathrm{dist}_G(u, v) \leq \mathrm{dist}_{G^B_{\geq 0}}(u, v) \leq dB$ where the first inequality is because $w(e) \leq w^B_{\geq 0}(e)$ for every edge $e \in E$ and the second inequality is by the output guarantee of LowDiamDecomposition (Lemma I.2). □

Another crucial property from the decomposition is this: Recall from Definition II.4 that $P_{G^B}(v)$ is the shortest sv-path in G^B_s with $\eta_{G^B}(v)$ negative-weight edges. We show below that in expectation $P_{G^B}(v)$ contains only $O(\log^2 n)$ edges from E^{rem}. This will be used in Phase 3.

Lemma IV.4. *If $\eta(G^B) \leq \Delta$, then for every $v \in V$, $E\left[|P_{G^B}(v) \cap E^{rem}|\right] = O(\log^2 m)$.*

Proof. Consider any $v \in V$. The crux of the proof is the following bound on the weight of $P_{G^B}(v)$ in $G^B_{\geq 0}$:

$$w^B_{\geq 0}(P_{G^B}(v)) \leq \eta_{G^B}(v) \cdot B \tag{2}$$

where we define $w^B_{\geq 0}(s, u) = 0$ for every $u \in V$. Recall the definition of of w^B_s from Definition II.3 and note that $w^B_s(P_{G^B}(v)) \leq 0$ because there is an edge of weight 0 from s to every $v \in V$. We thus have Equation (2) because

$$\begin{aligned} w^B_{\geq 0}(P_{G^B}(v)) &\leq w^B_s(P_{G^B}(v)) + |P_{G^B}(v) \cap E^{neg}(G^B)| \cdot B \\ &\leq |P_{G^B}(v) \cap E^{neg}(G^B)| \cdot B \\ &= \eta_{G^B}(v) \cdot B \end{aligned}$$

where the first inequality follows since $w^B(e) \geq -B$ for all $e \in E$, the second inequality follows since $w^B_s(P_{G^B}(v)) \leq 0$, and the equality follows by definition of $P_{G^B}(v)$. Recall from the output guarantee of LowDiamDecomposition (Lemma I.2) that $\Pr[e \in E^{rem}] = O(w^B_{\geq 0}(e) \cdot (\log n)^2/D + n^{-10})$, where in our case $D = dB = B\Delta/2$. This, the linearity of expectation, and (2) imply that

$$\begin{aligned} &E[P_{G^B}(v) \cap E^{rem}] \\ &= O\left(\frac{w^B_{\geq 0}(P_{G^B}(v)) \cdot (\log n)^2}{B\Delta/2} + |(P_{G^B}(v))| \cdot n^{-10}\right) \\ &\overset{(2)}{=} O\left(\frac{2\eta_{G^B}(v) \cdot (\log n)^2}{\Delta} + n^{-9}\right) \end{aligned}$$

which is $O(\log^2 n)$ when $\eta(G^B) \leq \Delta$. □

d) Phase 1: Make edges inside the SCCs $G^B[V_i]$ non-negative.: We argue that $\mathrm{ScaleDown}(H, \Delta/2, B)$ is called with an input that satisfies its input requirements (Theorem III.5). The most important requirement is $\eta(H^B) \leq \Delta/2$ (Item 1b) which we prove below (other requirements are trivially satisfied). Recall that we set $d := \Delta/2$ in Line 3.

Lemma IV.5. *If G has no negative-weight cycle, then $\eta(H^B) \leq d = \Delta/2$.*

Proof. Consider any vertex $v \in V$. Let $P := P_{H^B}(v) \setminus s$; i.e. P is obtained by removing s from a shortest sv-path in H^B_s that contains $\eta_{H^B}(v)$ negative weights in H^B_s. Let u be the first vertex in P. Note three easy facts:

(a) $w_{H^B}(e) = w_H(e) + B$ for all $e \in E^{neg}(H^B)$,
(b) $|E^{neg}(H^B) \cap P| = |E^{neg}(H^B) \cap P_{H^B}(v)| = \eta_{H^B}(v)$, and
(c) $w_{H^B}(P) = w_{H^B_s}(P_{H^B}(v)) \leq w_{H^B_s}(s, v) = 0$,

where (b) and (c) are because the edges from s to u and v in H^B_s have weight zero. Then,

$$\begin{aligned} \mathrm{dist}_G(u, v) \leq w_H(P) &\overset{(a)}{\leq} w_{H^B}(P) - |E^{neg}(H^B) \cap P| \cdot B \\ &\overset{(b)}{=} w_{H^B}(P) - \eta_{H^B}(v) \cdot B \overset{(c)}{\leq} -\eta_{H^B}(v) \cdot B. \end{aligned} \tag{3}$$

Note that u and v are in the same SCC V_i;[12] thus, by Lemma IV.3:

$$\mathrm{dist}_G(v,u) \le dB. \tag{4}$$

If G contains no negative-weight cycle, then $\mathrm{dist}_G(u,v) + \mathrm{dist}_G(v,u) \ge 0$ and thus $\eta_{H^B}(v) \le dB \cdot (1/B) = d$ by Equations (3) and (4). Since this holds for every $v \in V$, Lemma IV.5 follows. □

Consequently, ScaleDown (Theorem III.5) is guaranteed to output ϕ_1 as follows.

Corollary IV.6. *If G has no negative-weight cycle, then all edges in $G^B_{\phi_1}[V_i]$ are non-negative for every i.*

e) Phase 2: Make all edges in $G^B \setminus E^{rem}$ non-negative.: Now that all edges in $G^B_{\phi_1}[V_i]$ are non-negative, we turn to the remaining edges in $G^B \setminus E^{rem}$. Intuitively, since these remaining edges (i.e. those not in the SCCs) form a directed acyclic graph (DAG), calling $\mathrm{FixDAGEdges}(G^B_{\phi_1} \setminus E^{rem}, \{V_1, V_2, \ldots\})$ (Lemma III.2) in Phase 2 produces the following result.

Lemma IV.7. *If G has no negative-weight cycle, all weights in $G^B_{\phi_2} \setminus E^{rem}$ are non-negative.*

Proof. Clearly, $G^B_{\phi_1} \setminus E^{rem}$ and $\{V_1, V_2, \ldots\}$ satisfy the input conditions of Lemma III.2, i.e. (1) $(G^B \setminus E^{rem})_{\phi_1}[V_i]$ contains no negative-weight edges for every i (this is due to Corollary IV.6), and (2) when we contract every V_i into a node, the resulting graph is a DAG (this follows from the fact that the V_i are precisely the (maximal) SCCs of $G^B_{\phi_1} \setminus E^{rem}$).[13] Thus, $\mathrm{FixDAGEdges}((G^B \setminus E^{rem})_{\phi_1}, \{V_1, V_2, \ldots\})$ returns ψ such that $(G^B_{\phi_1} \setminus E^{rem})_\psi = G^B_{\phi_2} \setminus E^{rem}$ contains no negative-weight edges. □

f) Phase 3's runtime.: Now we are ready to prove Theorem IV.2, i.e. the runtime bound of $\mathrm{ElimNeg}(G^B_{\phi_2})$ in Phase 3 when G contains no negative-weight cycle. Recall (Lemma III.3 and definition II.4) that the runtime of $\mathrm{ElimNeg}(G^B_{\phi_2})$ is

$$O\left(\left(m + \sum_{v \in V} \eta_{G^B_{\phi_2}}(v)\right) \log m\right)$$

Fix any $v \in V$. Note that, regardless of the value of ϕ_2, $P_{G^B}(v)$ is a shortest sv-path in $(G^B_{\phi_2})_s$ (because $(G^B_{\phi_2})_s$ and G^B_s are equivalent and $P_{G^B}(v)$ is a shortest sv-path in G^B_s by definition). Thus,

$$\begin{aligned}\eta_{G^B_{\phi_2}}(v) &= \min\{|P \cap E^{neg}(G^B_{\phi_2})| : P \text{ is a shortest } sv\text{-path} \\ &\qquad \text{in } (G^B_{\phi_2})_s\} \\ &\le |P_{G^B}(v) \cap E^{neg}(G^B_{\phi_2})|\end{aligned}$$

where the equality follows from Definition II.4 and the inequality follows by the above.

[12] in fact all vertices in P are in the same SCC V_i, because we define $H = \bigcup_i G[V_i]$.

[13] See, e.g., Lemma 22.14 in [52].

By Lemma IV.7, all negative-weight edges in $G^B_{\phi_2}$ are in E^{rem}, i.e. $E^{neg}(G^B_{\phi_2}) \subseteq E^{rem}$; so,

$$\eta_{G^B_{\phi_2}}(v) \le |P_{G^B}(v) \cap E^{rem}|.$$

By Lemma IV.4 and the fact that $\eta(G^B) \le \Delta$ (input requirement Item 1b in Theorem III.5),[14]

$$E\left[\eta_{G^B_{\phi_2}}(v)\right] \le E\left[|P_{G^B}(v) \cap E^{rem}|\right] = O(\log^2 m). \tag{5}$$

Thus, the expected runtime of $\mathrm{ElimNeg}(G^B_{\phi_2})$ is

$$O\left(\left(m + E\left[\sum_{v \in V} \eta_{G^B_{\phi_2}}(v)\right]\right) \log m\right) = O\left(m \log^3 m\right).$$

V. Algorithm SPmain (Theorem III.4)

In this section we present algorithm $\mathrm{SPmain}(G_{in}, s_{in})$ (Theorem III.4), which always runs on the main input graph/source.

a) Description of Algorithm $\mathrm{SPmain}(G_{in}, s_{in})$*.:* See Algorithm 2 for pseudocode. Recall that if G_{in} contains a negative-weight cycle, then the algorithm is not supposed to terminate; for intuition, we recommend the reader focus on the case where G_{in} contains no negative-weight cycle.

The algorithm first creates an equivalent graph $\bar{G}$ by scaling up edge weights by $2n$ (Line 1), and also rounds B (Line 2), all to ensure that everything remains integral. It then repeatedly calls ScaleDown until we have a price function ϕ_t such that $w_{\phi_t}(e) \ge -1$ (See for loop in Line 4). The algorithm then defines a graph $G^* = (V, E, w^*)$ with $w^*(e) = w_{\phi_t}(e) + 1$ (Line 7). In the analysis, we will argue that because we are dealing with the scaled graph $\bar{G}$, the additive $+1$ is insignificant and does not affect the shortest path structure (Claim V.3), so running Dijkstra on G^* will return correct shortest paths in G (Lines 8 and 9).

b) Correctness: We focus on the case where the algorithm terminates, and hence every line is executed. First we argue that weights in G^* (Line 7) are non-negative.

Claim V.1. *If the algorithm terminates, then for all $e \in E$ and $i \in [0, t := \log_2(B)]$ we have that $\bar{w}_i$ is integral and that $\bar{w}_i(e) \ge -B/2^i$ for all $e \in E$. Note that this implies that $\bar{w}_t(e) \ge -1$ for all $e \in E$, and so the graph G^* has non-negative weights.*

Proof. We prove the claim by induction on i. For base case $i = 0$, the claim holds for $\bar{G}_{\phi_0} = \bar{G}$ because $w_{in}(e) \ge -1$ (see Assumption II.1), so $\bar{w}(e) \ge -2n \ge -B$ (see Lines 1 and 2).

Now assume by induction that the claim holds for $\bar{G}_{\phi_{i-1}}$. The call to $\mathrm{ScaleDown}(\bar{G}_{\phi_{i-1}}, \Delta := n, B/2^i)$ in Line 5 satisfies the necessary input properties (See Theorem III.5):

[14] The expectation in (5) is over the random outcomes of the low-diameter decomposition in Phase 0 and the recursion in Phase 1. Note that both $\eta_{G^B_{\phi_2}}(v)$ and $|P_{G^B}(v) \cap E^{rem}|$ are random variables. Since we always have $\eta_{G^B_{\phi_2}}(v) \le |P_{G^B}(v) \cap E^{rem}|$, we also have $E\left[\eta_{G^B_{\phi_2}}(v)\right] \le E\left[|P_{G^B}(v) \cap E^{rem}|\right]$.

Algorithm 2: Algorithm for SPmain($G_{in} = (V, E, w_{in}), s_{in}$)

1 $\bar{w}(e) \leftarrow w_{in}(e) \cdot 2n$ for all $e \in E$, $\bar{G} \leftarrow (V, E, \bar{w})$, $B \leftarrow 2n$. `// scale up edge weights`
2 Round B up to nearest power of 2 `// still have` $\bar{w}(e) \geq -B$ `for all` $e \in E$
3 $\phi_0(v) = 0$ for all $v \in V$ `// identity price function`
4 **for** $i = 1$ *to* $t := \log_2(B)$ **do**
5 $\quad \psi_i \leftarrow \text{ScaleDown}(\bar{G}_{\phi_{i-1}}, \Delta := n, B/2^i)$
6 $\quad \phi_i \leftarrow \phi_{i-1} + \psi_i$ `// (Claim V.1)` $w_{\phi_i}(e) \geq -B/2^i$ `for all` $e \in E$
7 $G^* \leftarrow (V, E, w^*)$ where $w^*(e) \leftarrow \bar{w}_{\phi_t}(e) + 1$ for all $e \in E$.
`// Observe:` G^* `in above line has non-negative weights`
8 Compute a shortest path tree T from s using Dijkstra(G^*, s) (Lemma III.1)
`// (Claim V.3) Will Show: any shortest path in` G^* `is also shortest in` G
9 **return** shortest path tree T .

property 1a holds by the induction hypotheses; property 1b holds because we have $\eta_G(v) \leq n$ for any graph G with no negative-weight cycle; property 1c holds because G_{in} has constant out-degree, and the algorithm never changes the graph topology. Thus, by the output guarantee of ScaleDown we have that $(\bar{w}_{\phi_{i-1}})_{\psi_i}(e) \geq (B/2^{i-1})/2 = B/2^i$. The claim follows because as noted in Definition II.5, $(\bar{w}_{\phi_{i-1}})_{\psi_i} = \bar{w}_{\phi_{i-1}+\psi_i} = \bar{w}_{\phi_i}$. □

Corollary V.2. *If G_{in} contains a negative-weight cycle then the algorithm does not terminate.*

Proof. Say, for contradiction, that the algorithm terminates; then by Claim V.1 we have that $\bar{w}_{\phi_t}(e) \geq -1$. Now, let C be any negative-weight cycle in G_{in}. Since all weights in $\bar{G}$ are multiples of $2n$ (Line 1), we know that $\bar{w}(C) \leq -2n$. But we also know that $\bar{w}_{\phi_t}(C) \geq -|C| \geq -n$. So $\bar{w}(C) \neq \bar{w}_{\phi_t}(C)$, which contradicts Lemma II.7. □

Now we show that the algorithm produces a correct output.

Claim V.3. *Say that G_{in} contains no negative-weight cycle. Then, the algorithm terminates, and if P is a shortest sv-path in G^* (Line 7) then it is also a shortest sv-path in G_{in}. Thus, the shortest path tree T of G^* computed in Line 9 is also a shortest path tree in G_{in}.*

Proof. The algorithm terminates because each call to $\text{ScaleDown}(\bar{G}_{\phi_{i-1}}, \Delta := n, B/2^i)$ terminates, because $\bar{G}_{\phi_{i-1}}$ is equivalent to G_{in} (Lemma II.7) and so does not contain a negative-weight cycle. Now we show that

$$P \text{ is also a shortest path in } \bar{G}_{\phi_t} \tag{6}$$

which implies the claim because $\bar{G}_{\phi_t}$ and G_{in} are equivalent. We assume that $s \neq v$ because otherwise the claim is trivial. Observe that since all weights in $\bar{G}$ are multiples of $2n$ (Line 1), all shortest distances are also multiples of $2n$, so for any two sv-paths P_{sv} and P'_{sv}, $|\bar{w}(P_{sv}) - \bar{w}(P'_{sv})|$ is either 0 or $> n$. It is easy to check that by Lemma II.7, we also have that

$$|\bar{w}_{\phi_t}(P_{sv}) - \bar{w}_{\phi_t}(P'_{sv})| \text{ is either 0 or } > n. \tag{7}$$

Moreover, by definition of G^* we have

$$\bar{w}_{\phi_t}(P_{sv}) < w^*(P_{sv}) = \bar{w}_{\phi_t}(P_{sv}) + |P_{sv}| < \bar{w}_{\phi_t}(P_{sv}) + n. \tag{8}$$

Now, we prove (6) . Assume for contradiction that there was a shorter path P' in $\bar{G}_{\phi_t}$. Then,

$$\bar{w}_{\phi_t}(P) - \bar{w}_{\phi_t}(P') \overset{(7)}{>} n. \tag{9}$$

So, $w^*(P') \overset{(8)}{<} \bar{w}_{\phi_t}(P') + n \overset{(9)}{<} \bar{w}_{\phi_t}(P) \overset{(8)}{<} w^*(P)$, contradicting P being shortest in G^*. □

c) Running Time Analysis: By Corollary V.2, if G_{in} does not contain a negative-weight cycle then the algorithm does not terminate, as desired. We now focus on the case where G_{in} does not contain a negative-weight cycle. The running time of the algorithm is dominated by the $\log(B) = O(\log(n))$ calls to $\text{ScaleDown}(\bar{G}_{\phi_{i-1}}, \Delta := n, B/2^i)$. Note that all the input graphs $\bar{G}_{\phi_{i-1}}$ are equivalent to G_{in}, so they do not contain a negative-weight cycle. By Theorem III.5, the expected runtime of each call to ScaleDown is $O(m \log^3(n) \log(\Delta)) = O(m \log^4(n))$. So, the expected runtime of SPmain is $O(m \log^5(n))$.

VI. Algorithm for Low-Diameter Decomposition

In this section, we prove Lemma I.2. We start with some basic definitions.

Definition VI.1 (balls and boundaries)**.** Given a directed graph $G = (V, E)$, a vertex $v \in V$, and a distance-parameter R, we define $\text{Ball}^{\text{out}}_{\text{G}}(v, R) = \{u \in V \mid \text{dist}(v, u) \leq R\}$. We define $\textsf{boundary}(\text{Ball}^{\text{out}}_{\text{G}}(v, R)) = \{(x, y) \in E \mid x \in \text{Ball}^{\text{out}}_{\text{G}}(v, R) \wedge y \notin \text{Ball}^{\text{out}}_{\text{G}}(v, R)\}$. Similarly, we define $\text{Ball}^{\text{in}}_{\text{G}}(v, R) = \{u \in V \mid \text{dist}(u, v) \leq R\}$ and we define $\textsf{boundary}(\text{Ball}^{\text{in}}_{\text{G}}(v, R)) = \{(x, y) \in E \mid y \in \text{Ball}^{\text{in}}_{\text{G}}(v, R) \wedge x \notin \text{Ball}^{\text{in}}_{\text{G}}(v, R)\}$. We often use $\text{Ball}^{*}_{\text{G}}$ to denote that a ball can be either an out-ball or an in-ball.

Definition VI.2 (Geometric Distribution)**.** Consider a coin whose probability of heads is $p \in (0, 1]$. The geometric distribution $\text{Geo}(p)$ is the probability distribution of the number X of independent coin tosses until obtaining the first heads. We have $\Pr[X = k] = p(1-p)^{k-1}$ for every $k \in \{1, 2, 3, ...\}$.

Remark VI.3. *In Lemma I.2 and throughout this section, n always refers to the number of vertices in the main graph G_{in}, i.e. graph in which we are trying to compute shortest paths. This is to ensure that "high probability" is defined in terms of the number of vertices in G_{in}, rather than in terms of potentially small auxiliary graphs. Note that whenever our*

shortest path algorithm executes $\mathrm{LDD}(G, D)$ *we always have* $|V(G)| \leq n$ *(we never add new vertices).*

a) The algorithm: The pseudocode for our low-diameter decomposition algorithm can be found in Algorithm 3. Roughly, in Phase 1 each vertex is marked as either *in-light*, *out-light*, or *heavy*. In Phase 2 we repeatedly "remove" balls centered at either in-light or out-light vertices: Let v be any in-light vertex (the process is similar for out-light vertices). Consider a ball $\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, R_v)$ with radius R_v selected randomly from the geometric distribution $\mathrm{Geo}(p)$ (Lines 12 and 13; using, e.g., [54]). We add edges pointing into this ball (i.e. edges in $\mathsf{boundary}(\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, R_v))$) to $E^{boundary}$, which will be later added to E^{rem}. We recurse the algorithm on this ball (Line 16) which may add more edges to E^{rem} (via $E^{recurse}$). Finally, we remove this ball from the graph and repeat the process. Note that the algorithm may also terminate prematurely with $E^{rem} = E(G)$ in Lines 15 and 20. We will show that this happens with low probability.

Due to the space limit, we defer the analysis to the full version [55].

VII. Proof of Theorem I.1 via a Black-Box Reduction

In this section, we prove Theorem I.1 using the following Monte Carlo algorithm as a black box:

Theorem VII.1. *There exists a randomized algorithm that takes* $O(m \log^6(n) \log(W_{G_{in}}))$ *time for an* m*-edge input graph* G_{in} *and source* s_{in} *and behaves as follows:*

- *if* G_{in} *contains a negative-weight cycle, then the algorithm always returns an error message,*
- *if* G_{in} *contains* no *negative-weight cycle, then the algorithm returns a shortest path tree from* s_{in} *with high probability, and otherwise returns an error message.*

Note that the algorithm always outputs either an error message, or a (correct) shortest path tree.

In the following, let $\mathrm{SPMonteCarlo}(G_{in}, s_{in})$ refer to the algorithm of Theorem VII.1. The goal of this section is to give a Las Vegas algorithm, $\mathrm{SPLasVegas}(G_{in}, s_{in})$, whose running time is $O(m \log^8(n))$ w.h.p when G_{in} satisfies the properties of assumption II.1. Applying the black-box reduction Lemma II.2 extends this to general G_{in} and yields Theorem I.1.

The first step of SPLasVegas will be to find the smallest integer $B \geq 0$ such that no negative cycles exist in G^B_{in}. This is done using the algorithm $\mathrm{FindThresh}$ of the following lemma.

Lemma VII.2. *Let* H *be an* m*-edge* n*-vertex graph with integer weights and let* $s \in V(H)$*. Then there is an algorithm,* $\mathrm{FindThresh}(H, s)$ *which outputs a value* $B \geq 0$ *such that w.h.p.,*

- *If* H *has no negative cycles then* $B = 0$*, and*
- *If* H *has a negative cycle then* $B > 0$*,* H^{B-1} *contains a negative cycle, and* H^B *does not.*

The running time of $\mathrm{FindThresh}(H, s)$ *is* $O(m \log^6(n) \log^2(W_H))$.

Algorithm 3: Algorithm for $\mathrm{LDD}(G = (V, E), D)$

1 Let n be the global variable in Remark VI.3
2 $G_0 \leftarrow G, E^{rem} \leftarrow \emptyset$

// Phase 1: mark vertices as light or heavy
3 $k \leftarrow c \ln(n)$ for large enough constant c
4 $S \leftarrow \{s_1, \ldots, s_k\}$, where each s_i is a random node in V // possible: $s_i = s_j$ for $i \neq j$
5 For each $s_i \in S$ compute $\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(s_i, D/4)$ and $\mathrm{Ball}^{\mathrm{out}}_{\mathrm{G}}(s_i, D/4)$
6 For each $v \in V$ compute $\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, D/4) \bigcap S$ and $\mathrm{Ball}^{\mathrm{out}}_{\mathrm{G}}(v, D/4) \bigcap S$ using Line 5
7 **foreach** $v \in V$ **do**
8 If $|\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, D/4) \bigcap S| \leq .6k$, mark v *in-light* // whp $|\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, D/4)| \leq .7|V(G)|$
9 Else if $|\mathrm{Ball}^{\mathrm{out}}_{\mathrm{G}}(v, D/4) \bigcap S| \leq .6k$, mark v *out-light* // whp $|\mathrm{Ball}^{\mathrm{out}}_{\mathrm{G}}(v, D/4)| \leq .7|V(G)|$
10 Else mark v *heavy* // w.h.p $\mathrm{Ball}^{\mathrm{in}}_{\mathrm{G}}(v, D/4) > .5|V(G)|$ and $\mathrm{Ball}^{\mathrm{out}}_{\mathrm{G}}(v, D/4) > .5|V(G)|$

// Phase 2: carve out balls until no light vertices remain
11 **while** G *contains a node* v *marked* $*$*-light for* $* \in \{in, out\}$ **do**
12 Sample $R_v \sim \mathrm{Geo}(p)$ for $p = \min\{1, 80 \log(n)/D\}$.
13 Compute $\mathrm{Ball}^{*}_{\mathrm{G}}(v, R_v)$.
14 $E^{boundary} \leftarrow \mathsf{boundary}(\mathrm{Ball}^{*}_{\mathrm{G}}(v, R_v))$ // add boundary edges of ball to E^{rem}.
15 If $R_v > D/4$ or $|\mathrm{Ball}^{*}_{\mathrm{G}}(v, R_v)| > .7|V(G)|$ then return $E^{rem} \leftarrow E(G)$ and terminate // Pr[terminate] $\leq 1/n^{20}$
16 $E^{recurse} \leftarrow \mathrm{LDD}(G[\mathrm{Ball}^{*}_{\mathrm{G}}(v, R_v)], D)$ // recurse on ball
17 $E^{rem} \leftarrow E^{rem} \bigcup E^{boundary} \bigcup E^{recurse}$.
18 $G \leftarrow G \setminus \mathrm{Ball}^{*}_{\mathrm{G}}(v, R_v)$ // remove ball from G

// Clean Up: check that remaining vertices have small weak diameter in initial input graph G_0
19 Let v be an arbitrary vertex of G.
20 If $\mathrm{Ball}^{in}_{G_0}(v, D/2) \not\supseteq V(G)$ or $\mathrm{Ball}^{out}_{G_0}(v, D/2) \not\supseteq V(G)$ then then return $E^{rem} \leftarrow E(G)$ and terminate // Pr[terminate] $\leq 1/n^{20}$
// if above does not terminate, then all remaining vertices in $V(G)$ have weak diameter $\leq D$
21 Return E^{rem}

In the following, if the high probability event of Lemma VII.2 holds, we refer to B as a *correct* value.

A. The Las Vegas algorithm

Recall our assumption that G_{in} satisfies $w(e) \geq -1$ for each $e \in E_{in}$. Pseudo-code for $\text{SPLasVegas}(G_{in}, s_{in})$, can be found in Algorithm 4. For intuition when reading the pseudo-code, note that we will show that w.h.p probability none of the restart events occur.

Algorithm 4: Algorithm $\text{SPLasVegas}(G_{in}, s_{in})$

```
Let G' be G_in with every edge weight multiplied by n^3
B ← FindThresh(G', s_in)
if B = 0 then
    if SPMonteCarlo(G_in, s_in) returns error then
        restart SPLasVegas(G_in, s_in);
    Let T be the tree output by SPMonteCarlo(G_in, s_in)
    return T
if SPMonteCarlo((G')^B, s_in) returns error then
    restart SPLasVegas(G_in, s_in);
Let φ(v) = dist_{(G')^B}(s_in, v) for all v ∈ V be obtained from the tree output by SPMonteCarlo((G')^B, s_in)
G_+ ← ((G')^B)_φ                    // (Lemma II.7) edge-weights in G_+ are non-negative
Obtain the subgraph G_{≤n} of G_+ consisting of edges of weight at most n
if G_{≤n} is acyclic then restart SPLasVegas(G_in, s_in);
Let C be an arbitrary cycle of G_{≤n}
if C is not negative in G_in then restart SPLasVegas(G_in, s_in);
return C
```

Correctness of $\text{SPLasVegas}(G_{in}, s_{in})$ is trivial as the algorithm explicitly checks that its output is correct just prior to halting:

Lemma VII.3. *If* $\text{SPLasVegas}(G_{in}, s_{in})$ *outputs a cycle, that cycle is negative in* G_{in}*. If* $\text{SPLasVegas}(G_{in}, s_{in})$ *outputs a tree, that tree is a shortest path tree from* s_{in} *in* G_{in}*.*

Due to the space limit, we defer the running time analysis to the full version [55].

VIII. Open Problems

We already mentioned a research agenda of designing fast simple algorithms for fundamental problems in Section I. While the ultimate goal is such an algorithm for the min-cost flow problem, getting such an algorithm for maximum-cardinality bipartite matching problem would already be a breakthrough.

Another extremely important direction is developing algorithmic techniques and frameworks that can be used across different models of computation. It remains to check whether our techniques can be used to show that negative-weight SSSP can be solved as efficiently as solving SSSP without negative weights in various models of computation such as parallel (PRAM), distributed (CONGEST), dynamic and semi-streaming settings. More broadly, even a simple problem like *reachability* (finding whether a vertex s can reach another vertex t) is not yet well understood in these models of computation.

Finally, we note that for *arbitrary* edge weights, where we cannot use scaling, the fastest algorithm for shortest paths with negative weights remains Bellman-Ford with running time $O(mn)$. The same $O(mn)$ bound is the also state-of-the-art for max-weight bipartite matching. Getting running time $o(n^{3-\epsilon})$ for either of these problems remains a major open problem.

IX. Acknowledgement

We thank Sepehr Assadi, Joakim Blikstad, Parinya Chalermsook, Mohsen Ghaffari, Satish Rao, Thatchaphol Saranurak and anonymous FOCS reviewers for helpful comments.

References

[1] M. Thorup, "Integer priority queues with decrease key in constant time and the single source shortest paths problem," *J. Comput. Syst. Sci.*, vol. 69, no. 3, pp. 330–353, 2004, announced at STOC'03.

[2] ——, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM*, vol. 46, no. 3, pp. 362–394, 1999, announced at FOCS'97.

[3] A. Shimbel, "Structure in Communication Nets," in *In Proceedings of the Symposium on Information Networks*. Brooklyn: Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955, pp. 199–203.

[4] R. Ford, *Paper P-923*. Santa Moncia, California: The RAND Corperation, 1956.

[5] R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[6] E. F. Moore, "The Shortest Path Through a Maze," in Proceedings of the International Symposium on the Theory of Switching, 1959, pp. 285–292.

[7] H. N. Gabow, "Scaling algorithms for network problems," *J. Comput. Syst. Sci.*, vol. 31, no. 2, pp. 148–168, 1985, announced at FOCS'83.

[8] H. N. Gabow and R. E. Tarjan, "Faster scaling algorithms for network problems," *SIAM J. Comput.*, vol. 18, no. 5, pp. 1013–1036, 1989.

[9] A. V. Goldberg, "Scaling algorithms for the shortest paths problem," *SIAM J. Comput.*, vol. 24, no. 3, pp. 494–504, 1995, announced at SODA'93.

[10] P. Sankowski, *Algorithms – ESA 2005: 13th Annual European Symposium, Palma de Mallorca, Spain, October 3-6, 2005. Proceedings*. pages 770–778. Springer Berlin Heidelberg, Berlin, Heidelberg: chapter Shortest Paths in Matrix Multiplication Time, 2005.

[11] R. Yuster and U. Zwick, "Answering distance queries in directed graphs using fast matrix multiplication," pp. 389–396, 2005.

[12] M. B. Cohen, A. Madry, P. Sankowski, and A. Vladu, "Negative-weight shortest paths and unit capacity minimum cost flow in $O(m^{10/7} \log W)$ time," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 2017, pp. 752–771.

[13] K. Axiotis, A. Madry, and A. Vladu, "Circulation control for faster minimum cost flow in unit-capacity graphs," in *arXiv preprint*. https//arxiv.org/pdf/2003.04863.pdf, 2020.

[14] J. v. d. Brand, Y. T. Lee, D. Nanongkai, R. Peng, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Bipartite matching in nearly-linear time on moderately dense graphs," in *FOCS*. IEEE, 2020, pp. 919–930.

[15] J. v. d. Brand, Y. T. Lee, Y. P. Liu, T. Saranurak, A. Sidford, Z. Song, and D. Wang, "Minimum cost flows, mdps, and $_1$-regression in nearly linear time for dense instances," in *STOC*. ACM, 2021, pp. 859–869.

[16] J. v. d. Brand, Y. T. Lee, A. Sidford, and Z. Song, "Solving tall dense linear programs in nearly linear time," in *STOC*. https://arxiv.org/pdf/2002.02304.pdf, 2020.

[17] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM journal on numerical analysis*, vol. 16, no. 2, pp. 346–358, 1979.

[18] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 3–23, 1997, announced at STOC'94.

[19] J. Fakcharoenphol and S. Rao, "Planar graphs, negative weight edges, shortest paths, and near linear time," *J. Comput. Syst. Sci.*, vol. 72, no. 5, pp. 868–889, 2006, announced at FOCS'01.

[20] P. N. Klein, S. Mozes, and O. Weimann, "Shortest paths in directed planar graphs with negative lengths: A linear-space $O(n \log^2 n)$-time algorithm," *ACM Trans. Algorithms*, vol. 6, no. 2, pp. 30:1–30:18, 2010, announced at SODA'09. [Online]. Available: https://doi.org/10.1145/1721837.1721846

[21] S. Mozes and C. Wulff-Nilsen, "Shortest paths in planar graphs with real lengths in $O(n\log^2 n/\log\log n)$ time," in *ESA (2)*, ser. Lecture Notes in Computer Science, vol. 6347. Springer, 2010, pp. 206–217.

[22] C. Wulff-Nilsen, "Separator theorems for minor-free and shallow minor-free graphs with applications," in *FOCS*. IEEE Computer Society, 2011, pp. 37–46.

[23] T. Saranurak and D. Wang, "Expander decomposition and pruning: Faster, stronger, and simpler," in *SODA*. SIAM, 2019, pp. 2616–2635.

[24] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," in *FOCS*. IEEE Computer Society, 2017, pp. 950–961.

[25] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," in *FOCS*, 2020, https://arxiv.org/pdf/1910.08025.pdf.

[26] A. Bernstein, J. v. d. Brand, M. Probst Gutenberg, D. Nanongkai, T. Saranurak, A. Sidford, and H. Sun, "Fully-dynamic graph sparsifiers against an adaptive adversary," *CoRR*, vol. abs/2004.08432, 2020.

[27] D. Nanongkai and T. Saranurak, "Dynamic spanning forest with worst-case update time: adaptive, las vegas, and $O(n^{1/2-\epsilon})$-time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2017, pp. 1122–1129.

[28] C. Wulff-Nilsen, "Fully-dynamic minimum spanning forest with improved worst-case update time," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, 2017, pp. 1130–1143.

[29] L. Chen, R. Kyng, Y. P. Liu, R. Peng, M. P. Gutenberg, and S. Sachdeva, "Maximum flow and minimum-cost flow in almost-linear time," *FOCS*, 2022.

[30] S. I. Daitch and D. A. Spielman, "Faster approximate lossy generalized flow via interior point algorithms," in *STOC*. ACM, 2008, pp. 451–460.

[31] A. Madry, "Navigating central path with electrical flows: From flows to matchings, and back," in *FOCS*. IEEE Computer Society, 2013, pp. 253–262.

[32] Y. T. Lee and A. Sidford, "Path finding methods for linear programming: Solving linear programs in õ(sqrt(rank)) iterations and faster algorithms for maximum flow," in *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, 2014, pp. 424–433. [Online]. Available: https://doi.org/10.1109/FOCS.2014.52

[33] A. Madry, "Computing maximum flow with augmenting electrical flows," in *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2016, pp. 593–602.

[34] M. B. Cohen, Y. T. Lee, and Z. Song, "Solving linear programs in the current matrix multiplication time," in *STOC*, 2019, https://arxiv.org/pdf/1810.07896.

[35] J. v. d. Brand, "A deterministic linear program solver in current matrix multiplication time," in *SODA*. SIAM, 2020, pp. 259–278.

[36] Y. P. Liu and A. Sidford, "Faster energy maximization for faster maximum flow," in *STOC*. https://arxiv.org/pdf/1910.14276.pdf, 2020.

[37] B. Awerbuch, "Complexity of network synchronization," *J. ACM*, vol. 32, no. 4, pp. 804–823, 1985, announced at STOC'84.

[38] B. Awerbuch, A. V. Goldberg, M. Luby, and S. A. Plotkin, "Network decomposition and locality in distributed computation," in *FOCS*. IEEE Computer Society, 1989, pp. 364–369.

[39] B. Awerbuch and D. Peleg, "Routing with polynomial communication-space trade-off," *SIAM J. Discret. Math.*, vol. 5, no. 2, pp. 151–162, 1992.

[40] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, "Fast network decomposition (extended abstract)," in *PODC*. ACM, 1992, pp. 169–177.

[41] N. Linial and M. E. Saks, "Low diameter graph decompositions," *Comb.*, vol. 13, no. 4, pp. 441–454, 1993. [Online]. Available: https://doi.org/10.1007/BF01303516

[42] Y. Bartal, "Probabilistic approximations of metric spaces and its algorithmic applications," in *37th Annual Symposium on Foundations of Computer Science, FOCS '96, Burlington, Vermont, USA, 14-16 October, 1996*. IEEE Computer Society, 1996, pp. 184–193. [Online]. Available: https://doi.org/10.1109/SFCS.1996.548477

[43] G. E. Blelloch, A. Gupta, I. Koutis, G. L. Miller, R. Peng, and K. Tangwongsan, "Nearly-linear work parallel SDD solvers, low-diameter decomposition, and low-stretch subgraphs," *Theory Comput. Syst.*, vol. 55, no. 3, pp. 521–554, 2014. [Online]. Available: https://doi.org/10.1007/s00224-013-9444-5

[44] G. L. Miller, R. Peng, and S. C. Xu, "Parallel graph decompositions using random shifts," in *25th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '13, Montreal, QC, Canada - July 23 - 25, 2013*, G. E. Blelloch and B. Vöcking, Eds. ACM, 2013, pp. 196–203. [Online]. Available: https://doi.org/10.1145/2486159.2486180

[45] J. Pachocki, L. Roditty, A. Sidford, R. Tov, and V. V. Williams, "Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, A. Czumaj, Ed. SIAM, 2018, pp. 1374–1392. [Online]. Available: https://doi.org/10.1137/1.9781611975031.91

[46] S. Forster and G. Goranci, "Dynamic low-stretch trees via dynamic low-diameter decompositions," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, M. Charikar and E. Cohen, Eds. ACM, 2019, pp. 377–388. [Online]. Available: https://doi.org/10.1145/3313276.3316381

[47] S. Chechik and T. Zhang, "Dynamic low-stretch spanning trees in subpolynomial time," in *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, S. Chawla, Ed. SIAM, 2020, pp. 463–475. [Online]. Available: https://doi.org/10.1137/1.9781611975994.28

[48] A. Bernstein, M. Probst Gutenberg, and C. Wulff-Nilsen, "Near-optimal decremental SSSP in dense weighted digraphs," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, S. Irani, Ed. IEEE, 2020, pp. 1112–1122. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00107

[49] S. Forster, M. Grösbacher, and T. de Vos, "An improved random shift algorithm for spanners and low diameter decompositions," in *OPODIS*, ser. LIPIcs, vol. 217. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, pp. 16:1–16:17.

[50] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen, "Negative-weight single-source shortest paths in almost-linear time," *CoRR*, vol. abs/2203.03456, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2203.03456

[51] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, 1977. [Online]. Available: https://doi.org/10.1145/321992.321993

[52] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[53] A. Bernstein, M. P. Gutenberg, and T. Saranurak, "Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing," in *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, S. Irani, Ed. IEEE, 2020, pp. 1123–1134. [Online]. Available: https://doi.org/10.1109/FOCS46700.2020.00108

[54] K. Bringmann and T. Friedrich, "Exact and efficient generation of geometric random variates and random graphs," in *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, ser. Lecture Notes in Computer Science, F. V. Fomin, R. Freivalds, M. Z. Kwiatkowska, and D. Peleg, Eds., vol. 7965. Springer, 2013, pp. 267–278. [Online]. Available: https://doi.org/10.1007/978-3-642-39206-1_23

[55] A. Bernstein, D. Nanongkai, and C. Wulff-Nilsen, "Negative-weight single-source shortest paths in near-linear time," *CoRR*, vol. abs/2203.03456, 2022.