Deterministic Negative-Weight Shortest Paths in Nearly Linear Time via Path Covers

Bernhard Haeupler* Yonggang Jiang† Thatchaphol Saranurak‡

Abstract

We present the first *deterministic* nearly-linear time algorithm for single-source shortest paths with negative edge weights on directed graphs: given a directed graph G with n vertices, m edges whose weights are integers in $\{-W, \ldots, W\}$, our algorithm either computes all distances from a source s or reports a negative cycle in $\tilde{O}(m) \cdot \log(nW)$ time.

All known near-linear time algorithms for this problem have been inherently randomized [BNW25, BCF23, FHL+25, LM24], as they crucially rely on *low-diameter decompositions*.

To overcome this barrier, we introduce a new structural primitive for directed graphs called the *path cover*. This plays a role analogous to *neighborhood covers* in undirected graphs [ABCP98], which have long been central to derandomizing algorithms that use low-diameter decomposition in the undirected setting. We believe that path covers will serve as a fundamental tool for the design of future deterministic algorithms on directed graphs.

*INSAIT, Sofia University "St. Kliment Ohridski" and ETH Zürich, bernhard.haeupler@insait.ai. Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure and through the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (ERC grant agreement 949272).

†MPI-INF and Saarland University, Germany, yjiang@mpi-inf.mpg.de. Part of this work was done while visiting INSAIT. Supported by Google PhD Fellowship.

‡University of Michigan, thsa@umich.edu. Supported by NSF Grant CCF-2238138 and a Sloan Fellowship. Part of this work was done at INSAIT. Partially funded by the Ministry of Education and Science of Bulgaria's support for INSAIT as part of the Bulgarian National Roadmap for Research Infrastructure.

Contents

1 Introduction

In the negative weight single-source shortest paths (SSSP) problem, given an m-edge n-vertex directed weighted graph $G = (V, E, w)$ with *possibly negative* edge weight $w(e) \in \{-W, \ldots, W-1, W\}$ for each $e \in E$ and a source vertex $s \in V$, the goal is to either compute the distance from s to every other vertex or find a negative-weight cycle.

When weights are non-negative, the textbook Dijkstra algorithm takes near-linear $O(m + n \log n)$ time. Another textbook Bellman-Ford algorithm can handle negative weights, but requires much slower $O(mn)$ time. This contrast motivates the search for near-linear time algorithms since 1980s [Gar85, GT89, Gol95, CMSV17, vdBLN+20, AMV20, VDBLL+21].

In 2022, two algorithmic breakthroughs happened. Chen et al. [CKL+22] gave a randomized algorithm with $m^{1+o(1)} \log W$ time that solves an even more general minimum cost flow problem. Their approach is based on convex optimization and dynamic data structures and was later improved to be deterministic [VDBCK+23, CKL+24, VDBCK+24]. However, the dynamic data structures required in this approach are complicated and the $m^{o(1)}$ factor in time seems hard to avoid.

Independently, in the same year, Bernstein, Nanongkai, and Wulff-Nilsen [BNW25] showed an algorithm with $O(m \log^8(n) \log(W))$ time, which is faster, combinatorial, and bypasses complicated dynamic data structures. The follow-up work [BCF23, ABC+23, LM24, FHL+25] subsequently sped up the time to $O((m + n \log \log n) \log^2(n) \log(nW))$. Unfortunately, algorithms in this line of work are inherently randomized.

Until now, there is no deterministic negative weight SSSP algorithm with near-linear $O(m \cdot \text{polylog}(nW))$ time.

Our result. We give the first deterministic near-linear algorithm, improving upon both lines of work above.

Theorem 1.1 (Deterministic SSSP)**.** *There is a* deterministic *algorithm solving the negative weight single-source shortest path problem in $O(m \log^8 n \log(nW))$ time.*

To prove Theorem 1.1, we introduce a novel graph structure called *path cover*, which we believe will find further applications beyond Theorem 1.1 itself. We do not optimize the polylogarithmic factor in order to present the clearest version of this new concept. A more relaxed version of path covers can speed up our algorithm to $O(m \log^5 n \log(nW))$ time. Below, we first define the path covering problem to motivate path covers.

The Path Covering Problem. We say a path p is a *d-path* if its length is at most d, and a subgraph $H \subseteq G$ *covers* p if $p \subseteq E(H)$. A graph H is *d-clustered* if every strongly-connected component of H has diameter at most d. The negative-weight SSSP problem can be reduced to the following problems.

Problem 1.1 (Covering d-Paths with Clustered Subgraphs)**.** *Fix a graph G with* non-negative *weights and $d \geq 0$. An adversary chooses an* unknown target *set P of paths of length at most d in G where $|P| = O(n)$. We must choose $\tilde{O}(d)$-clustered subgraphs $G'_1, \ldots, G'_z \subseteq G$ where $z = O(\log n)$ such that each path $p \in P$ is covered by some G'_i.*

The frameworks of [ABC+23] and [FHL+25] showed that a deterministic near-linear time algorithm for Problem 1.1 implies another one for negative-weight SSSP with $m^{1+o(1)} \log(nW)$ time and $O(m \cdot \text{polylog}(nW))$ time, respectively.

Previous Solutions: Either Randomized or Undirected. Problem 1.1 admits a randomized algorithm with near-linear time via the probabilistic *low-diameter decomposition* (LDD), a powerful primitive [Bar96, MPX13] extended to directed graphs by [BNW25]. For any d, an LDD algorithm samples a $\tilde{O}(d)$-clustered subgraph $G' \subseteq G$ that covers each d-path with constnat probability. Thus, $z = O(\log n)$ samples $G'_1, \ldots, G'_z$ will cover $|P| = \text{poly}(n)$ unknown target paths with high probability, solving Problem 1.1. All prior negative-weight SSSP algorithms with near-linear time [BNW25, BCF23, ABC+23, LM24, FHL+25] employ probabilistic LDDs, and, hence, are inherently randomized because of it.

For deterministic algorithms, Problem 1.1 is much more difficult. Observe that we must cover *all* exponentially many d-paths, as any uncovered path can be picked by the adversary into the target set P. Thus, even the *existence* of the solution is highly unclear.

Yet, in undirected graphs, a deterministic near-linear time algorithm follows directly from efficient constructions of *neighborhood covers* [ABCP98]. This very strong property of neighborhood covers has been used for developing many deterministic algorithms in undirected graphs [ABCP98, GP19, Chu21, CZ23, HLS24, CP25]. For completeness, we define LDD and neighborhood covers, and explain how they give the solution of Problem 1.1 in either the randomized or undirected settings in Section B.

This raises a natural question: is there a *deterministic* algorithm on *directed* graphs for Problem 1.1?

A Barrier for Subgraphs: No Deterministic Algorithm on Directed Graphs. Unfortunately, unlike undirected graphs, we show that Problem 1.1 *cannot* be solved deterministically on directed graphs. In fact, even the *total size* of the $\tilde{O}(d)$-clustered subgraphs must be $\tilde{\Omega}(m\sqrt{m})$ in the worst case. Thus, it takes $\tilde{\Omega}(m\sqrt{m})$ time to even write down these subgraphs.

Theorem 1.2 (Barrier for Subgraphs)**.** *For every $m, \lambda \geq 1$, there exists an $O(m)$-edge directed graph G and d such that every collection of $d\lambda$-clustered graphs $G'_1, \ldots, G'_z \subseteq G$ covering all d-paths in G must have total size $\sum_i |E(G'_i)| = \Omega(m\sqrt{m/\lambda})$.*

See Section C for the proof. This barrier seems bad because the only two main tools mentioned above, LDDs and neighborhood covers, naturally only give a collection of $\tilde{O}(d)$-clustered *subgraphs* of G and, hence, must suffer from the barrier of Theorem 1.2.

Bypassing the Barrier via Projection. We bypass the barrier in Theorem 1.2 using a *projection*, instead of subgraphs. A *projection* to G is a graph G' whose vertices of G' are copies of vertices in G, and edges of G' may connect the copy u' to v' only if the original vertex u is connected to v in G. More formally, there is a mapping $\pi : V(G') \to V(G)$ where, if $(u', v') \in E(G')$, then $(u = \pi(u'), v = \pi(v')) \in E(G)$.[1] Intuitively, a projection is formed by "merging" subgraphs together.

[1] That is, π is a graph homomorphism.

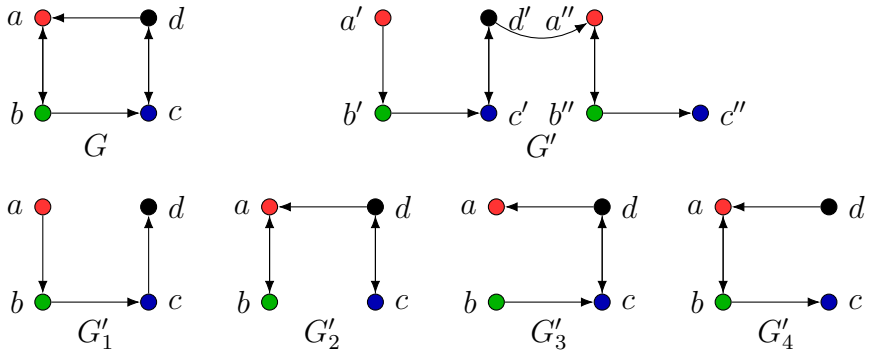

Figure 1: At the top right, G' is a 1-clustered *projection* to G that covers all 3-paths in G. At the second row, G'_1, G'_2, G'_3, G'_4 are 1-clustered *subgraphs* of G that cover all 3-paths in G.

Observe that every path $p' = (v'_1, \dots, v'_t)$ in G' is mapped to a path $p = (v_1, \dots v_t)$ in G where $v_i = \pi(v'_i)$. Here, we say that p is *covered* by G' and p' is a *lift* of p. If a projection G' covers all d-paths in G, then G' is a *d-path cover* of G.

As a toy example to illustrate the power of projection, consider a directed unweighted cycle G on $[n]$. Observe that a path $G' = (1, \dots, n, 1, \dots n)$ is an n-path cover of G, which is a 0-clustered graph, i.e., a DAG. In contrast, any collection of 0-clustered subgraphs $G'_1, \dots, G'_z \subseteq G$ cannot cover any n-path of G. Otherwise, it would introduce a cycle in G'_i. See Figure 1 for another example where subgraphs require larger total size than a projection.

As our main technical contribution, we show how projection can bypass the the barrier from Theorem 1.2 for subgraphs: while some graph requires $\tilde{O}(d)$-clustered subgraphs of total size $\tilde{\Omega}(m\sqrt{m})$ to cover all d-paths, we show that every graph G admits a $\tilde{O}(d)$-clustered projection G' to G of linear size that covers all d-paths.

Theorem 1.3 (Covering d-Paths with Clustered Projection)**.** *Given a graph G with* non-negative *weights and $d \geq 0$, there exists a projection G' to G where G' is a $O(d \log^6 n)$-clustered graph of size $|E(G')| \leq (1 + \frac{1}{\log n})|E(G)|$ that covers all d-paths of G. Moreover, G' can be deterministically computed in $O(m \log^4 n)$ time.*

In other words, G' is a d-path cover which is a $\tilde{O}(d)$-clustered graph of size $(1+\frac{1}{\log n})|E(G)|$. Importantly, this size bound is even stronger than that of the known randomized LDD-based approach, which requires the total size of $O(|E(G)| \log n)$.

Finally, we obtain our SSSP algorithm Theorem 1.1 by showing that the reduction of [ABC+23] from negative-weight SSSP to Problem 1.1 extends from subgraphs to projections. By replacing the randomized LDD with the deterministic projection from Theorem 1.3, we simultaneously derandomize their algorithm and speed up the time from $m^{1+o(1)}$ to near-linear.

This speed-up crucially exploits our size bound of $(1 + \frac{1}{\log n})|E(G)|$. Thus, when the algorithm makes $O(\log n)$-depth recursion, the total size blow-up is at most a constant factor as $(1+\frac{1}{\log n})^{O(\log n)}|E(G)| = O(|E(G)|)$. In contrast, the LDD-based approach gives subgraphs of total size $O(|E(G)| \log n)$. The $O(\log n)$ blow-up per level causes their $m^{1+o(1)}$ running time in [ABC+23].[2]

[2]The projection from Theorem 1.3 can derandomize [FHL+25], too. Here, a more relaxed size bound of $|E(G')| = O(|E(G)| \log n)$ is sufficient but their framework additionally requires that every SCC of G'

Path covers can be viewed as a deterministic counterpart of LDD in directed graphs, similar to how neighborhood cover is a deterministic counterpart of LDD in undirected graphs. Since neighborhood covers have been a powerful tool in undirected graphs, we believe Theorem 1.1 is only the first of many applications of path covers.

Related works. In the setting of *undirected* graphs, the idea of simplifying a graph G via another graph G', that makes copies of the vertices of the original graph G and connects them together, appeared in the *clan embedding* of [FL21] and the *copy tree embedding* of [HHZ22].

Independent work. In an independent work, Li [Li25] also obtains a near-linear time deterministic algorithm for negative-weight shortest paths. He defines a new variant of *padded decomposition* which can be of independent interest. In constructing our path cover, we perform a similar graph decomposition.

2 Overview

We start this section by sketching the *existence* of the d-path cover G' of G in Theorem 1.3.

We start with some standard notation: For every vertex $v \in V$ let $\deg_G^{\text{out}}(v)$ be the out-degree of v, i.e., the number of edges leaving v. For a vertex set $S \subseteq V$ let $\deg_G^{\text{out}}(S) = \sum_{v \in S} \deg_G^{\text{out}}(v)$ be the number of edges starting in S. Lastly, for any vertex $u \in V$, define

$$\text{Ball}_G^{\text{out}}(u, r) = \{v \in V \mid \text{dist}_G(u, v) \leq r\}$$

to be the ball centered at u with radius r.

Construction. Our algorithm is recursive.

If the diameter of G is one strongly connected component with diameter $O(d \log^3 n)$, then one can trivially return the whole graph and terminate the recursion.

Otherwise, the algorithm finds a good partition by picking an arbitrary node u and growing a ball from it by computing $\text{Ball}_G^{\text{out}}(u, i \cdot d)$ for $i = 1, 2, 3, \ldots$ until we find a "thin layer" i^* such that

$$\deg_G^{\text{out}}\big(\text{Ball}_G^{\text{out}}(u, i^* \cdot d)\big) \leq \left(1 + \frac{1}{\log^2 n}\right) \cdot \deg_G^{\text{out}}\big(\text{Ball}_G^{\text{out}}(u, (i^* - 1) \cdot d)\big) \,. \tag{1}$$

Since the ball cannot grow multiplicatively too often it is clear that $i^* = O(\log^3 n)$. The algorithm works separately and recursively on both

$$B^{\text{out}} := \text{Ball}_G^{\text{out}}(u, i^* \cdot d) \qquad \text{and} \qquad \bar{B}^{\text{out}} := V \setminus \text{Ball}_G^{\text{out}}(u, (i^* - 1) \cdot d) \,.$$

Note that B^{out} and $\bar{B}^{\text{out}}$ only *overlap* at the "ring" around u from radius $(i^* - 1) \cdot d$ to $i^* \cdot d$.

contains no duplicated copies of vertices. Although our projection G' *does* satisfy this property, the overall argument is more complicated, and we do not want to impose such restrictions on the definition of path cover. So, we choose to present a derandomization of [ABC+23] in this paper.

We focus here on the interesting case that the ball grown from u is not more than a constant fraction of the graph, i.e., $\deg_G^{\text{out}}(B^{\text{out}}) \leq 0.9m$, our proof shows that otherwise there is a way to carve out a large ball and still make progress. Note that by running Dijkstra's algorithm locally from u the time needed to grow the ball B^{out} is $\widetilde{O}\left(\deg_G^{\text{out}}(B^{\text{out}})\right)$ and therefore essentially proportional to its size and independent of the size of $\bar{B}^{\text{out}}$. This makes the recursion efficient. Let H^{out} and $\bar{H}^{\text{out}}$ be d-path covers of the induced subgraphs $G[B^{\text{out}}]$ and $G[\bar{B}^{\text{out}}]$, respectively, which are constructed *recursively*.

For technical reasons, we strengthen the property of H^{out} (and likewise for $\bar{H}^{\text{out}}$): not only do we require each d-path in $G[B^{\text{out}}]$ to have a lift p' in H^{out}, we require a *one-to-one* mapping between the start vertices of p and p'. In other words, each vertex v in $G[B^{\text{out}}]$ has a *representative* vertex v' in H^{out} such that if a d-path in G starts at v, then its lift in G' starts at v'.

We build the d-path cover of G (denoted G') by concatenating $\bar{H}^{\text{out}}$ and H^{out} as follows: for every $(u, v) \in E$ with $u \in \bar{B}^{\text{out}}$ and $v \in V \setminus \bar{B}^{\text{out}}$, we add edges from all copies of the vertex u in $\bar{H}^{\text{out}}$ to the *representative* of the vertex v in H^{out}. Having a representative for each vertex avoids having to add prohibitively many edges from all copies of u to all copies of v. Defining the representatives of vertices in G' throughout the recursion is straightforward once the path-covering property in the next paragraph is clear.

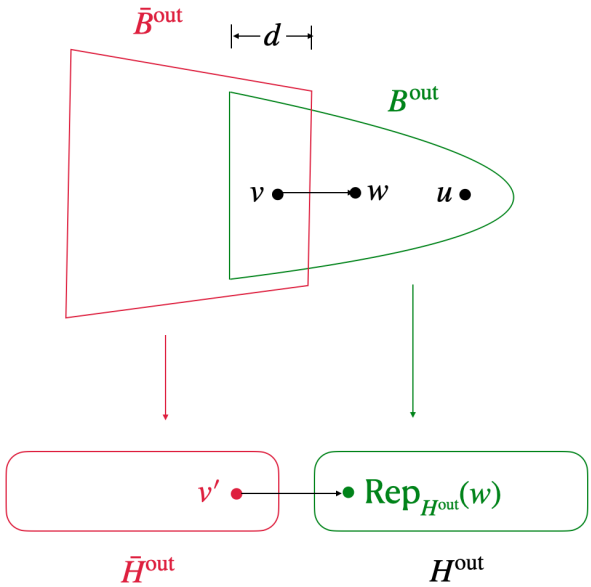

Figure 2: An illustration of the construction. A path entering $V - \bar{B}^{\text{out}}$ (for example a path starting at w) will not leave B^{out} again because of the length d layer in the middle, thus covered by H^{out}.

$\widetilde{O}(d)$-clustered graph. Notice that if $\bar{H}^{\text{out}}, H^{\text{out}}$ are $\widetilde{O}(d)$-clustered graphs, then G' is a $\widetilde{O}(d)$-clustered graph because edges can only be from $\bar{H}^{\text{out}}$ to H^{out}.

Path Covering. We show that G' indeed covers all d-paths in G.

We consider the case where a path p starts at $v_p \in \bar{B}^{\text{out}}$ and leaves $\bar{B}^{\text{out}}$ at some point

(i.e., enters $V \setminus \bar{B}^{\text{out}}$). Other situations (completely inside $\bar{B}^{\text{out}}$ or starting in $V \setminus \bar{B}^{\text{out}}$) can be handled similarly. Let v'_p be the first vertex on p that is not in $\bar{B}^{\text{out}}$. Let p_1 be the subpath of p before v'_p (excluding v'_p) and let p_2 be the remainder of p (starting at v'_p).

Notice that

$$v'_p \in V \setminus \bar{B}^{\text{out}} = V \setminus \big(V \setminus \text{Ball}^{\text{out}}_G(u, (i^* - 1) \cdot d)\big) = \text{Ball}^{\text{out}}_G(u, (i^* - 1) \cdot d)\,.$$

This means that every path starting at v'_p with length at most d (in particular, p_2) is contained in

$$\text{Ball}^{\text{out}}_G(u, (i^* - 1) \cdot d + d) = \text{Ball}^{\text{out}}_G(u, i^* \cdot d) = B^{\text{out}}.$$

Therefore p_1 lies completely inside $G[\bar{B}^{\text{out}}]$ and p_2 lies completely inside $G[B^{\text{out}}]$, so $\bar{H}^{\text{out}}$ covers p_1 with a lift p'_1 and H^{out} covers p_2 with a lift p'_2.

We then concatenate p'_1 and p'_2 using the edge from the last vertex of p'_1 to the first vertex of p'_2 (which is the representative of v'_p in H^{out}). The concatenated path maps to p.

Size of the path cover. We first show that the size of G' satisfies

$$|E(G')| \le \sum_{v' \in V(G')} \deg^{\text{out}}_G\big(\pi(v')\big).$$

Let (u', v') be an edge concatenating $\bar{H}^{\text{out}}$ and H^{out}, where $u' \in V(\bar{H}^{\text{out}})$ and $v' \in V(H^{\text{out}})$ are copies of $u \in \bar{B}^{\text{out}}$ and $v \in V \setminus \bar{B}^{\text{out}}$, respectively, so that $(u, v) \in E$ and v' is the representative of v in H^{out}. In this case, we say that (u, v) *contributes to* u'. Notice that (i) (u, v) can contribute to u' at most once among all edges concatenating $\bar{H}^{\text{out}}$ and H^{out}, because we add only one edge from u' to the representative of v in H^{out} (which is why we define representatives); and (ii) (u, v) cannot contribute to u' again in the recursive construction of $\bar{H}^{\text{out}}$, because (u, v) is not an edge of $G[\bar{B}^{\text{out}}]$. Combining these two facts, every vertex u' is contributed to by at most $\deg^{\text{out}}_G\big(\pi(u')\big)$ edges, which establishes the claimed inequality.

It remains to bound $\sum_{v' \in V(G')} \deg^{\text{out}}_G\big(\pi(v')\big)$. Intuitively, Equation (1) guarantees that each recursion layer increases the total degree by at most a factor of $\big(1 + 1/\log^2 n\big)$ times the smaller side which is sufficient to yields an overall bound of $(1 + 1/\log n) \cdot m$ on the final total degree (we defer the calculation to the technical section). Since we have $|E(G')| \le \sum_{v' \in V(G')} \deg^{\text{out}}_G\big(\pi(v')\big)$, we get the desired size bound

$$|E(G')| \le (1 + 1/\log n) \cdot m$$

3 Preliminaries

We use the following notations, $[a] = \{1, 2, \ldots, a\}$, $\widetilde{O}(f) = f \cdot \text{polylog}(n)$, where we always use n, m to denote the number of nodes and edges of the input graph. For a function $\pi : V \to V'$, we write $\pi^{-1}(v') = \{v \in V \mid \pi(v) = v'\}$. All algorithms in this paper are deterministic.

Graph. All graphs of this paper, unless specified, are directed graphs with edge weights, denoted by $G = (V, E, w)$. We assume edge weights are *integers* within the range $[-W, W]$. We use standard definitions of graph terminologies for directed graphs which can be found

in textbooks, including paths, distances, strongly connected components (SCCs), negative cycles, shortest-path trees, etc. For a cycle or path H, we use $w(H)$ to denote the sum of weights of edges on the cycle or path, and $|H|$ to denote the number of edges on the cycle or path (note that a path can have repeated edges; in this case, both the weight and the number of such an edge are counted multiple times). We use $w_{\geq 0}$ to denote the weight truncated at 0, i.e., $w_{\geq 0}(e) = \max\{0, w(e)\}$ for every $e \in E$. Similarly, we write $G_{\geq 0} = (V, E, w_{\geq 0})$.

In many cases of this paper, a path is not a *simple path*, so a path can have repeated vertices (but usually a shortest path is a simple path). We use $\text{Start}(p)$ and $\text{End}(p)$ to denote the starting vertex and ending vertex of a directed path p. We use $p_1 \oplus p_2$ to denote the concatenation of two paths: if $p_1 = (v_1, \ldots, v_z)$ and $p_2 = (u_1, \ldots, u_{z'})$, then $p_1 \oplus p_2 = (v_1, \ldots, v_z, u_1, \ldots, u_{z'})$. Notice that $p_1 \oplus p_2$ is a path only if (v_z, u_1) is an edge.

We use $V(G)$ and $E(G)$ to denote the vertex set and edge set of a graph G. We use w_G to stress that w_G is the weight function for G. We write $\deg_G^{\text{out}}(u)$ to denote the number of edges (u, v) for $v \in V$, and $\deg_G^{\text{in}}(v)$ to denote the number of edges (u, v) for $u \in V$. We write $\deg_G(v) = \deg_G^{\text{out}}(v) + \deg_G^{\text{in}}(v)$. We write $\deg_G^{\text{out}}(A) = \sum_{v \in A} \deg_G^{\text{out}}(v)$, $\deg_G^{\text{in}}(A) = \sum_{v \in A} \deg_G^{\text{in}}(v)$, and $\deg_G(A) = \sum_{v \in A} \deg_G(v)$.

Graph distance. We use $\text{dist}_G(s, t)$ to denote the distance from s to t. When G is clear from the context, we simply write $\text{dist}(s, t)$. We write

$$\text{Ball}_G^{\text{out}}(s, d) = \{v \in V(G) \mid \text{dist}_G(s, v) \leq d\} \quad \text{and} \quad \text{Ball}_G^{\text{in}}(s, d) = \{v \in V(G) \mid \text{dist}_G(v, s) \leq d\}.$$

We allow distance to be $+\infty$ (in the case of not connected) or $-\infty$ (in the case of reaching a negative cycle) so that every distance is well-defined.

Given a directed graph $G = (V, E, w)$ and a vertex set $A \subseteq V$, the *diameter* of A is defined as $\max_{s,t \in A} \text{dist}_{G[A]}(s, t)$ where $G[A]$ is the induced subgraph of G on A. The diameter of A is also the diameter of $G[A]$.

The single-source shortest path problem (SSSP). Given a directed graph $G = (V, E)$ with integer (possibly negative) edge weights and a source $s \in V$, either output $\text{dist}_G(s, v)$ for every $v \in V$, or output a negative cycle of G. It is folklore that we can get a shortest-path tree from a distance function, and vice versa.

Potential adjustment. Given a *potential* $\phi : V \to \mathbb{Z}$ and a graph $G = (V, E, w)$, we define an adjusted weight $w_\phi(u, v) = w(u, v) + \phi(u) - \phi(v)$. We define the adjusted graph $G_\phi = (V, E, w_\phi)$.

The following lemma shows the power of potential adjustment. The lemma is folklore and was already used in [Joh77].

Lemma 3.1 ([Joh77])**.** *Let G be a graph and $\phi : V \to \mathbb{Z}$ be a potential function. If G has no negative cycle, a subgraph is a shortest-path tree of G iff it is a shortest-path tree of G_ϕ. More precisely, let $d_G^s(u)$ be the distance in G from s to u, then*

$$d_{G_\phi}^s(u) = d_G^s(u) + \phi(s) - \phi(u)\,.$$

4 Path Cover via a Clustered Graph

In this section, we present a method for covering every bounded-length path in a directed graph using a clustered graph. We begin with the necessary definitions. All graphs in this section have non-negative edge weights.

4.1 Definitions and the Main Theorem

We first formally define graph projections introduced in the introduction.

Definition 4.1 (Graph Projections)**.** Let $G = (V, E, w)$ be a graph. A *projection onto* G is a graph $G' = (V', E', w')$ together with a projection map $\pi = \pi_{G'}$ where π is a *weight-preserving graph homomorphism*: for every $(x, y) \in E(G')$, we have $(\pi(x), \pi(y)) \in E(G)$ and $w'(x, y) = w(\pi(x), \pi(y))$.

When we refer to a projection G', we implicitly assume its map $\pi_{G'}$.

Definition 4.2 (Path Covering)**.** Given a path $p = (v_1, \ldots, v_z)$ in G, we say that a projection (G', π) onto G *covers* p if there exists a path $p' = (v'_1, \ldots, v'_z)$ in G' such that $\pi(v'_i) = v_i$ for every $i \in [z]$. We call p' the *lift* of p. The projection is *d-path-covering* if every simple path in G of length at most d is covered.

Our goal is to cover every simple path of length at most d by a projection whose SCCs have small diameter. For this purpose, we define clustered graphs as follows.

Definition 4.3 (Clustered Graphs)**.** We say that a directed graph G' is a *d-clustered graph* if every strongly connected component (SCC) of G' has diameter at most d.

Definition 4.4 (Path Cover)**.** A *d-path cover* of a directed graph $G = (V, E, w)$ with *diameter slack* λ is a d-path-covering projection G' where G' is a $\lambda \cdot d$-clustered graph.

We will prove the following theorem in this section.

Theorem 4.5 (Path Cover via Clustered DAGs)**.** *There exists a deterministic algorithm that, given a directed graph $G = (V, E, w)$ with non-negative edge weights, a diameter parameter d, and a slack parameter $\lambda \geq 10000 \log^4 n$, constructs a d-path cover G' of G with diameter slack λ and size at most*

$$\left(1 + \tfrac{100 \log^2 n}{\sqrt{\lambda}}\right) \cdot m,$$

in $O(\sqrt{\lambda} \cdot m \log n)$ time.

Moreover, we have

$$\sum_{v \in V(G')} \deg_G(\pi(v)) \leq \left(1 + \tfrac{100 \log^2 n}{\sqrt{\lambda}}\right) \cdot m.$$

To get Theorem 1.3, we only need to set $\lambda = 10000 \log^6 n$.

Remark 4.6. The running time can be improved to $O(m \log^2 n)$, independent of λ, if we relax the structure of the path cover so that we only guarantee *weak diameter w.r.t. G*. More precisely, for each strongly connected component U in G', $\max_{u,v \in U} \mathrm{dist}_G(\pi(u), \pi(v)) \leq d\lambda$. This weaker guarantee is sufficient for our negative-weight shortest path algorithms, and we can improve the running time from $O(m \log^8 n \log(nW))$ to $O(m \log^5 n \log(nW))$ in this way. However, we choose to construct the path cover with a strong diameter guarantee to ease the understanding of this new concept.

4.2 The Algorithm

In this section, we describe the algorithm for Theorem 4.5. Let $G = (V, E, w)$ be the input directed graph and d be the input diameter parameter.

The following definition is useful when describing algorithms. Let G' be a projection onto G. We say a vertex $v \in V(G)$ is *present* (in the projection) if $\pi^{-1}(v) \neq \emptyset$. We say the projection has *representatives* if every present vertex $v \in V(G)$ has a unique representative $\text{rep}_{G'}(v) \in V(G')$ with $\pi_{G'}(\text{rep}_{G'}(v)) = v$. In this case, we say $\text{rep}_{G'}(v)$ is a *representative*.[3]

Definition 4.7 (Layered Projection)**.** Let G_i be projections onto G with representatives for $i = 1, \ldots, z$. A *layered projection* $G' = \mathsf{Layer}((G_1, \ldots, G_z) \to G)$ induced from $(G_1, \ldots, G_z)$ onto G with representatives is constructed as follows. First, initialize G' as a disjoint union of $G_1, \ldots, G_z$ with the same projection map on vertices[4]. For every $v \in V(G)$, suppose v is present in G_i where i is the smallest such index, select the representative $\text{rep}_{G'}(v)$ to be $\text{rep}_{G_i}(v)$. Then, for every two vertices $u' \in V(G_i)$ and $v' \in V(G_j)$, if $i < j$, $(\pi_{G_i}(u'), \pi_{G_j}(v')) \in E$ and v' is a representative in G', we add the edge (u', v') to G' with weight $w(\pi_{G_i}(u'), \pi_{G_j}(v'))$.

The algorithm is recursive. For convenience, we use $\textsc{PathCover}(A)$ to denote the subroutine that takes a vertex set $A \subseteq V$ as input and returns a d-path cover of $G[A]$ with representatives. It suffices to call $\textsc{PathCover}(V)$ to finish Theorem 4.5.

We define the following parameters:

$$\epsilon = \frac{1}{\sqrt{\lambda}}, \qquad \epsilon' = \frac{9 \log n}{\lambda}.$$

Base case. When $|A| = 1$, we simply return $G[A]$ with the trivial projection map and representative.

Recursive step. Pick an arbitrary vertex u in A. We execute the following two procedures in an interleaved fashion: we perform one memory access of each procedure in turn, alternating between them, until one of them terminates. As soon as one procedure stops, the entire process halts.

1. **(Forward growing)** Run Dijkstra's algorithm to find the smallest $i^{\text{out}} \in \mathbb{N}_{>0}$ such that

$$\deg_G(\text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d)) \leq (1 + \epsilon') \cdot \deg_G(\text{Ball}^{\text{out}}_{G[A]}(u, (i^{\text{out}} - 1) \cdot d)).$$

   Such an i^{out} must exist because for sufficiently large i, we have $\text{Ball}^{\text{out}}_{G[A]}(u, i \cdot d) = G[A]$.

   To be precise, Dijkstra's algorithm fixes the distances from u to other nodes in increasing order. We run it only up to the point where the distance exceeds $i \cdot d$, in order to compute $\text{Ball}^{\text{out}}_{G[A]}(u, i \cdot d)$. This can take sublinear time because the algorithm stops once the desired index i^{out} is found.

[3]The representative in our case is basically the first appearance of the copy of v in G'. First is defined on the SCC order of G' (we will have the guarantee that each SCC is a subgraph of G, which does not have repeated vertices).

[4]When G_i and G_j share the same vertex v, we make different copies of v in G'.

For technical reasons, we assume that when Dijkstra's algorithm fixes the distance of a node v, it performs $\deg_G(v)$ memory accesses, i.e., it looks up all adjacent edges of v in G (but only performs $\deg^{\text{out}}_{G[A]}(v)$ relaxations). This setting is purely for the convenience of analysis.

2. **(Backward growing)** Identical to the forward growing procedure, but with all occurrences of 'out' replaced by 'in'.

For simplicity, we use the following notation. Notice that the algorithm only knows $B^{\text{out}}, \bar{B}^{\text{out}}$ but does not know $B^{\text{in}}, \bar{B}^{\text{in}}$ if forward growing stops earlier than backward growing, and vice versa:

$$B^{\text{out}} = \text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d), \qquad \bar{B}^{\text{out}} = A - \text{Ball}^{\text{out}}_{G[A]}(u, (i^{\text{out}} - 1) \cdot d),$$
$$B^{\text{in}} = \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d), \qquad \bar{B}^{\text{in}} = A - \text{Ball}^{\text{in}}_{G[A]}(u, (i^{\text{in}} - 1) \cdot d).$$

Without loss of generality, assume forward growing stops earlier than backward growing (the other case is handled symmetrically). Consider two cases.

1. **Case 1:** $\deg_G(B^{\text{out}}) < (1-\epsilon)\deg_G(A)$**.** In this case, let

$$H^{\text{out}} \leftarrow \textsc{PathCover}\big(B^{\text{out}}\big), \qquad \bar{H}^{\text{out}} \leftarrow \textsc{PathCover}\big(\bar{B}^{\text{out}}\big).$$

Let G' be the layered projection induced from $(\bar{H}^{\text{out}}, H^{\text{out}})$ (Definition 4.7). Return G' as the output of $\textsc{PathCover}(A)$.

2. **Case 2:** $\deg_G(B^{\text{out}}) \geq (1-\epsilon)\deg_G(A)$**.** Continue the backward growing until it stops. Now the algorithm also knows $B^{\text{in}}, \bar{B}^{\text{in}}$. Let T^{out} be the shortest-path tree in $G[B^{\text{out}}]$ with root u, and let T^{in} be the reversed shortest-path tree in $G[B^{\text{in}}]$ with root u (i.e., T^{in} contains the shortest paths from every node in B^{in} to u).

   Let
$$M \leftarrow B^{\text{out}} \cap B^{\text{in}}.$$
   Let $\tilde{T}^{\text{out}}$ be the subtree of T^{out} that contains all nodes in $V(T^{\text{out}})$ that can reach M in T^{out}. Similarly, define $\tilde{T}^{\text{in}}$ to be the subtree of T^{in} that contains all nodes in $V(T^{\text{in}})$ that can be reached from M in T^{in}.

   Define
$$H^{\text{mid}} \leftarrow G[V(\tilde{T}^{\text{out}}) \cup V(\tilde{T}^{\text{in}})].$$
   Notice that H^{mid} is a subgraph of G, so a projection map and representatives can be defined trivially.

   Make the following recursive calls:

$$\bar{H}^{\text{in}} \leftarrow \textsc{PathCover}\big(\bar{B}^{\text{in}}\big), \qquad \tilde{H}^{\text{out}} \leftarrow \textsc{PathCover}\big(\bar{B}^{\text{out}} \cap B^{\text{in}}\big).$$

   Let G' be the layered projection induced from $(\tilde{H}^{\text{out}}, H^{\text{mid}}, \bar{H}^{\text{in}})$. Return G' as the output of $\textsc{PathCover}(A)$.

For a graph illustration, see Figure 2 which shows Case 1. Case 2 can be viewed as after applying Case 1 to get $(H^{\text{in}}, \bar{H}^{\text{in}})$, we recursively build H^{in} by $(\tilde{H}^{\text{out}}, H^{\text{mid}})$.

4.3 Correctness

We prove the correctness of the algorithm described in Section 4.2. Specifically, PathCover(A) returns a d-path cover of $G[A]$. We proceed by induction on $|A|$.

Base case. Suppose $|A| = 1$. The only SCC has diameter 0. Thus, $G[A]$ is a d-clustered graph. Moreover, the trivial projection map certifies that all paths are covered.

Induction step. Assume PathCover is correct for every input size smaller than $|A|$. We first show the easy part: G' is a $\lambda \cdot d$-clustered graph. We need the following lemma.

Lemma 4.8. *Both i^{out} and i^{in} are at most $\lambda/4$.*

Proof. For every $i < i^{\text{out}}$, by the definition of i^{out},

$$\deg_G(\text{Ball}^{\text{out}}_{G[A]}(u, i \cdot d)) > (1+\epsilon') \cdot \deg_G(\text{Ball}^{\text{out}}_{G[A]}(u, (i-1) \cdot d)).$$

Combining these inequalities for every $i < i^{\text{out}}$ gives

$$m \geq \deg_G(\text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d)) > (1+\epsilon')^{i^{\text{out}}-1} = \left(1 + \frac{9 \log n}{\lambda}\right)^{i^{\text{out}}-1}.$$

Since $\lambda \geq 10000 \log^4 n$, we obtain $i^{\text{out}} \leq \lambda/4$. The same proof holds symmetrically for i^{in}. □

Bounding diameter. To show G' is a $\lambda \cdot d$-clustered graph, let C be an SCC of G'; we must show $\text{diam}(C) \leq \lambda \cdot d$. If the algorithm goes to Case 1, then G' is the layered projection induced from $(\bar{H}^{\text{out}}, H^{\text{out}})$. By Definition 4.7, there are only edges from $\bar{H}^{\text{out}}$ to H^{out} but not backwards, so C is either completely inside $\bar{H}^{\text{out}}$ or inside H^{out}; in either case, C has diameter at most $\lambda \cdot d$ by the induction hypothesis. If the algorithm goes to Case 2, then G' is the layered projection induced from $(\tilde{H}^{\text{out}}, H^{\text{mid}}, \bar{H}^{\text{in}})$. Similarly, if C is completely inside $\tilde{H}^{\text{out}}$ or $\bar{H}^{\text{in}}$, then C has diameter at most $\lambda \cdot d$ by the induction hypothesis. If C is inside H^{mid}, we show that H^{mid} is strongly connected and has diameter at most $\lambda \cdot d$. Recall

$$H^{\text{mid}} \leftarrow G[V(\tilde{T}^{\text{out}}) \cup V(\tilde{T}^{\text{in}})],$$

where $\tilde{T}^{\text{out}}$ is the subtree of T^{out} whose every node can reach $W = B^{\text{out}} \cap B^{\text{in}}$, and $\tilde{T}^{\text{in}}$ is the subtree of T^{in} whose every node can be reached from W. Recall also

$$B^{\text{out}} = \text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d), \qquad B^{\text{in}} = \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d),$$

and $T^{\text{out}}, T^{\text{in}}$ are shortest-path trees in $G[B^{\text{out}}]$ and $G[B^{\text{in}}]$. By Lemma 4.8, each shortest-path tree has depth at most $(\lambda/4) \cdot d$. Thus, every node in $\tilde{T}^{\text{out}}$ can reach W with distance at most $(\lambda/4) \cdot d$ in $\tilde{T}^{\text{out}}$; every node in $\tilde{T}^{\text{in}}$ can be reached from W with distance at most $(\lambda/4) \cdot d$ in $\tilde{T}^{\text{in}}$. Similarly, u can reach every node in $\tilde{T}^{\text{out}}$ within distance at most $(\lambda/4) \cdot d$ in $\tilde{T}^{\text{out}}$, and u can be reached from every node in $\tilde{T}^{\text{in}}$ within distance at most $(\lambda/4) \cdot d$ in $\tilde{T}^{\text{in}}$. We conclude that the diameter of H^{mid} is at most $\lambda \cdot d$.

Covering all length-d paths. Next we show that G' is a d-path cover of $G[A]$. Let p be an arbitrary path of $G[A]$ with length at most d. We prove that G' covers p (see Definition 4.2). Moreover, we prove a stronger statement: there exists a lift p' of p such that $\text{Start}(p') = \text{rep}(\text{Start}(p))$.

Since G' depends on whether the algorithm goes to Case 1 or Case 2, we prove the two cases separately.

Induction step (Case 1). Here G' is the layered projection induced from $(\bar{H}^{\text{out}}, H^{\text{out}})$ where

- $\bar{H}^{\text{out}}$ is a path cover of $G[\bar{B}^{\text{out}}]$, and
- H^{out} is a path cover of $G[B^{\text{out}}]$.

If p is completely contained in $G[\bar{B}^{\text{out}}]$, then by the induction hypothesis, p is covered by G'. Moreover, the lift p' of p has $\text{Start}(p') = \text{rep}_{\bar{H}^{\text{out}}}(\text{Start}(p))$. By Definition 4.7, $\text{rep}_{\bar{H}^{\text{out}}}(\text{Start}(p)) = \text{rep}_{G'}(\text{Start}(p))$ since $\bar{H}^{\text{out}}$ precedes H^{out}. Thus, $\text{Start}(p') = \text{rep}_{G'}(\text{Start}(p))$.

Now assume p is not completely contained in $G[\bar{B}^{\text{out}}]$, so there exists a vertex on p that lies in $A - \bar{B}^{\text{out}}$. Let a be the first such vertex on p.

Let p_1 be the subpath of p before vertex a (excluding a; possibly empty), and let p_2 be the subpath of p after vertex a (including a). By definition, p_1 is contained in $G[\bar{B}^{\text{out}}]$, so by induction, p_1 is covered by $\bar{H}^{\text{out}}$.

We now show that p_2 is contained in $G[B^{\text{out}}]$. Recall

$$A - \bar{B}^{\text{out}} = \text{Ball}^{\text{out}}_{G[A]}(u, (i^{\text{out}} - 1) \cdot d), \qquad B^{\text{out}} = \text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d).$$

Thus, if $a \in A - \bar{B}^{\text{out}}$, any path of length at most d starting at a is contained in B^{out}.

Hence p_1 is covered by $\bar{H}^{\text{out}}$ and p_2 is covered by H^{out}, with lifts p'_1, p'_2 respectively. We have $\pi(\text{End}(p'_1)) = \text{End}(p_1)$ and $\text{Start}(p'_2) = \text{rep}_{H^{\text{out}}}(\text{Start}(p_2))$. Since $\text{Start}(p_2) = a \in A - \bar{B}^{\text{out}}$, a is not present in $\bar{H}^{\text{out}}$, so $\text{rep}_{G'}(a) = \text{rep}_{H^{\text{out}}}(a)$ and $\text{Start}(p'_2) = \text{rep}_{G'}(a)$. By Definition 4.7, there is an edge $(\text{End}(p'_1), \text{Start}(p'_2))$ in G', so $p'_1 \oplus p'_2$ is a path in G' and is a lift of $p = p_1 \oplus p_2$.

If p'_1 is non-empty, then $\text{Start}(p'_1) = \text{rep}_{\bar{H}^{\text{out}}}(\text{Start}(p_1))$ by induction, hence $\text{Start}(p'_1 \oplus p'_2) = \text{rep}_{G'}(\text{Start}(p))$ since $\bar{H}^{\text{out}}$ precedes H^{out} in G'. Otherwise, p'_1 is empty, so p starts at $a \in V - \bar{B}^{\text{out}}$. In this case, since a is not present in $\bar{H}^{\text{out}}$, we must have $\text{rep}_{G'}(a) = \text{rep}_{H^{\text{out}}}(a) = \text{Start}(p'_2)$, where the second equality is by induction.

Induction step (Case 2). Here G' is the layered projection induced from $(\tilde{H}^{\text{out}}, H^{\text{mid}}, \bar{H}^{\text{in}})$, where

- $\tilde{H}^{\text{out}}$ is a path cover of $G[\bar{B}^{\text{out}} \cap B^{\text{in}}]$,
- $H^{\text{mid}} \supseteq G[B^{\text{out}} \cap B^{\text{in}}]$,
- $\bar{H}^{\text{in}}$ is a path cover of $G[\bar{B}^{\text{in}}]$.

If p is completely contained in $G[\bar{B}^{\text{out}} \cap B^{\text{in}}]$, then p is covered by G'. Moreover, since $\tilde{H}^{\text{out}}$ precedes $H^{\text{mid}}, \bar{H}^{\text{in}}$, we have $\text{rep}_{G'}(\text{Start}(p)) = \text{rep}_{\tilde{H}^{\text{out}}}(\text{Start}(p))$, which equals the starting vertex of the lift of p in G'.

Otherwise, there is a vertex of p in $A - (\bar{B}^{\text{out}} \cap B^{\text{in}})$; let a be the first such vertex. Let p_1 be the subpath of p before a (excluding a; possibly empty), and let p_2 be the subpath of p starting at a. Then $p_1 \subseteq G[\bar{B}^{\text{out}} \cap B^{\text{in}}]$, so p_1 is covered by $\tilde{H}^{\text{out}}$ with lift p'_1. If p_1 is non-empty, then $\text{rep}_{G'}(\text{Start}(p)) = \text{rep}_{\tilde{H}^{\text{out}}}(\text{Start}(p)) = \text{Start}(p'_1)$ by induction.

We first show that p_2 is completely contained in $V(H^{\text{mid}}) \cup \bar{B}^{\text{in}}$. Suppose Recall

$$A - (\bar{B}^{\text{out}} \cap B^{\text{in}}) \subseteq (A - \bar{B}^{\text{out}}) \cup (A - B^{\text{in}}) = \text{Ball}^{\text{out}}_{G[A]}(u, (i^{\text{out}} - 1) \cdot d) \cup (A - \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d)),$$
$$V(H^{\text{mid}}) \cup \bar{B}^{\text{in}} \supseteq (B^{\text{out}} \cap B^{\text{in}}) \cup \bar{B}^{\text{in}} \supseteq \text{Ball}^{\text{out}}_{G[A]}(u, i^{\text{out}} \cdot d) \cup (A - \text{Ball}^{\text{in}}_{G[A]}(u, (i^{\text{in}} - 1) \cdot d)).$$

Hence, if $a \in A - (\bar{B}^{\text{out}} \cap B^{\text{in}})$ and p_2 has length at most d, it is contained in $V(H^{\text{mid}}) \cup \bar{B}^{\text{in}}$. We now argue within $V(H^{\text{mid}}) \cup \bar{B}^{\text{in}}$.

If p_2 is completely contained in $V(H^{\text{mid}})$, then p_2 is covered by H^{mid} trivially (as H^{mid} is an induced subgraph of G) with a lift p'_2. Moreover, $\text{rep}_{G'}(a) = \text{rep}_{H^{\text{mid}}}(a) = a = \text{Start}(p'_2)$ since a is not present in $\tilde{H}^{\text{out}}$.

Otherwise, p_2 has a vertex in $A - (\bar{B}^{\text{out}} \cap B^{\text{in}}) - V(H^{\text{mid}})$; let b be the first such vertex. Let $p_{2,1}$ be the subpath of p_2 before b (excluding b; possibly empty) and let $p_{2,2}$ be the subpath of p_2 from b onward. Then $p_{2,1} \subseteq V(H^{\text{mid}})$ and hence is covered by H^{mid} with lift $p'_{2,1}$. If $p_{2,1}$ is non-empty, then as above $\text{rep}_{G'}(a) = \text{Start}(p'_{2,1})$.

We prove that $p_{2,2} \subseteq G[\bar{B}^{\text{in}}]$, so that it is covered by $\bar{H}^{\text{in}}$ with lift $p'_{2,2}$. Note

$$A - (\bar{B}^{\text{out}} \cap B^{\text{in}}) - V(H^{\text{mid}}) \subseteq A - (\bar{B}^{\text{out}} \cap B^{\text{in}}) - (B^{\text{out}} \cap B^{\text{in}}) = A - B^{\text{in}} = A - \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d),$$
$$\bar{B}^{\text{in}} = A - \text{Ball}^{\text{in}}_{G[A]}(u, (i^{\text{in}} - 1) \cdot d).$$

Since $b = \text{Start}(p_{2,2}) \in A - \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d)$ and $p_{2,2}$ has length at most d, $p_{2,2}$ is contained in $\bar{B}^{\text{in}}$ by definition (otherwise it would enter $\text{Ball}^{\text{in}}_{G[A]}(u, (i^{\text{in}} - 1) \cdot d)$, contradicting $b \in A - \text{Ball}^{\text{in}}_{G[A]}(u, i^{\text{in}} \cdot d)$).

Moreover, b is not present in $\tilde{H}^{\text{out}}$ or H^{mid}, so $\text{rep}_{G'}(b) = \text{rep}_{\bar{H}^{\text{in}}}(b) = \text{Start}(p'_{2,2})$ by induction.

Now p_1 is covered by $\tilde{H}^{\text{out}}$ with lift p'_1, $p_{2,1}$ is covered by H^{mid} with lift $p'_{2,1}$, and $p_{2,2}$ is covered by $\bar{H}^{\text{in}}$ with lift $p'_{2,2}$. Assuming all are non-empty (the empty cases are analogous), we have $\pi(\text{End}(p'_1)) = \text{End}(p_1)$, $\text{Start}(p'_{2,1}) = \text{rep}_{G'}(a)$ and $\pi(\text{End}(p'_{2,1})) = \text{End}(p_{2,1})$, and $\text{Start}(p'_{2,2}) = \text{rep}_{G'}(b)$. By Definition 4.7, edges $(\text{End}(p'_1), \text{Start}(p'_{2,1}))$ and $(\text{End}(p'_{2,1}), \text{Start}(p'_{2,2}))$ are in G', so $p'_1 \oplus p'_{2,1} \oplus p'_{2,2}$ is a path in G' and is a lift of $p = p_1 \oplus p_{2,1} \oplus p_{2,2}$.

4.4 Size and Time Complexity

The algorithm is recursive. We first prove the following lemma.

Lemma 4.9. *Suppose* $G' \leftarrow \textsc{PathCover}(A)$. *The size of* G' *is at most* $\sum_{v \in V(G')} \deg_{G[A]}(\pi(v))$.

Proof. By induction on $|A|$. When $|A| = 1$, there are no edges in G', so the lemma is trivial. Suppose $|A| > 1$. The algorithm returns different G' based on two cases.

Case 1. The size of G' contains two parts: (1) the sizes of $\bar{H}^{\text{out}}, H^{\text{out}}$, which, by induction, are at most

$$\sum_{v \in V(\bar{H}^{\text{out}})} \deg_{G[\bar{B}^{\text{out}}]}(\pi(v)) + \sum_{v \in V(H^{\text{out}})} \deg_{G[B^{\text{out}}]}(\pi(v)).$$

(2) The edges added by Definition 4.7. For every $v \in V(\bar{H}^{\text{out}})$, the number of edges added from v equals $\deg_{G[A]}(\pi(v)) - \deg_{G[\bar{B}^{\text{out}}]}(\pi(v))$, because only vertices in $A - \bar{B}^{\text{out}}$ can have representatives in H^{out}; all other vertices are present in $\bar{H}^{\text{out}}$. Summing over $\sum_{v \in V(\bar{H}^{\text{out}})}(\deg_{G[A]}(\pi(v)) - \deg_{G[\bar{B}^{\text{out}}]}(\pi(v)))$ and $\sum_{v \in V(\bar{H}^{\text{out}})} \deg_{G[\bar{B}^{\text{out}}]}(\pi(v)) + \sum_{v \in V(H^{\text{out}})} \deg_{G[B^{\text{out}}]}(\pi(v))$ gives the bound $\sum_{v \in V(G')} \deg_{G[A]}(\pi(v))$.

Case 2. *(i) The $\tilde{H}^{\text{out}}$-term.* All edges with both endpoints in $\tilde{H}^{\text{out}}$ lie inside the induced subgraph $G[\bar{B}^{\text{out}} \cap B^{\text{in}}]$. Thus

$$|E(\tilde{H}^{\text{out}})| \leq \sum_{v \in V(\tilde{H}^{\text{out}})} \deg_{G[\bar{B}^{\text{out}} \cap B^{\text{in}}]}(\pi(v)).$$

Any additional edge incident to $v \in V(\tilde{H}^{\text{out}})$ that goes to $H^{\text{mid}} \cup \bar{H}^{\text{in}}$ must use a representative outside $\bar{B}^{\text{out}} \cap B^{\text{in}}$, so the number of such outgoing edges is at most

$$\sum_{v \in V(\tilde{H}^{\text{out}})} \Big(\deg_{G[A]}(\pi(v)) - \deg_{G[\bar{B}^{\text{out}} \cap B^{\text{in}}]}(\pi(v)) \Big).$$

Consequently, the total number of edges charged to $\tilde{H}^{\text{out}}$ is at most

$$\sum_{v \in V(\tilde{H}^{\text{out}})} \deg_{G[A]}(\pi(v)).$$

(ii) The H^{mid}-term. Edges internal to H^{mid} are contained in the induced subgraph on $V(H^{\text{mid}})$, so

$$|E(H^{\text{mid}})| \leq \sum_{v \in V(H^{\text{mid}})} \deg_{G[V(H^{\text{mid}})]}(\pi(v)).$$

Edges from H^{mid} to $\bar{H}^{\text{in}}$ are added only from the H^{mid}-side, and for each $v \in V(H^{\text{mid}})$ there are at most

$$\deg_{G[A]}(\pi(v)) - \deg_{G[V(H^{\text{mid}})]}(\pi(v))$$

such edges. Hence the total number of edges charged to H^{mid} is at most

$$\sum_{v \in V(H^{\text{mid}})} \deg_{G[A]}(\pi(v)).$$

(iii) The $\bar{H}^{\text{in}}$-term. By the induction hypothesis applied within $\bar{H}^{\text{in}}$, the edges with both endpoints in $\bar{H}^{\text{in}}$ are at most

$$|E(\bar{H}^{\text{in}})| \leq \sum_{v \in V(\bar{H}^{\text{in}})} \deg_{G[A]}(\pi(v)).$$

Inter-piece edges incident to $\bar{H}^{\text{in}}$ are *not* charged here (they were already charged to their tails in $\tilde{H}^{\text{out}}$ or H^{mid}), so there is no double counting.

Summing the contributions from (i)–(iii) yields

$$|E(G')| \le \sum_{v \in V(\tilde{H}^{\text{out}})} \deg_{G[A]}(\pi(v)) + \sum_{v \in V(H^{\text{mid}})} \deg_{G[A]}(\pi(v)) + \sum_{v \in V(\bar{H}^{\text{in}})} \deg_{G[A]}(\pi(v))$$

$$= \sum_{v \in V(G')} \deg_{G[A]}(\pi(v)),$$

which completes the proof.

□

Thus, it remains to bound $\sum_{v \in V(G')} \deg_G(\pi(v))$, which we prove next.

Lemma 4.10. *Suppose* $G' \leftarrow \textsc{PathCover}(A)$. *Then*

$$\sum_{v \in V(G')} \deg_G(\pi(v)) \le \left(1 + 9(\log(\deg_G(A))) \cdot \frac{\epsilon'}{\epsilon}\right) \deg_G(A).$$

The running time of $\textsc{PathCover}(A)$ *is at most* $\log(\deg_G(A)) \cdot O(\deg_G(A)/\epsilon)$.

Proof. By induction on $|A|$. For $|A| = 1$, PathCover returns $G[A]$, and the claim is trivial.

The algorithm first runs two procedures (forward and backward growing) in lockstep, one edge access at a time, and stops as soon as one stops. The time between two edge accesses is at most $O(\log n)$ using a standard heap. Assume forward growing stops earlier.

The forward procedure runs Dijkstra from u until it finds i^{out}, costing $O(\log n \cdot \deg_G(B^{\text{out}}))$.

We analyze the two cases.

Case 1: $\deg_G(B^{\text{out}}) < (1-\epsilon)\deg_G(A)$**.** Here $V(G') = V(H^{\text{out}}) \cup V(\bar{H}^{\text{out}})$ with $H^{\text{out}} = \textsc{PathCover}(B^{\text{out}})$ and $\bar{H}^{\text{out}} = \textsc{PathCover}(\bar{B}^{\text{out}})$. By induction,

$$\begin{aligned}
\sum_{v \in V(G')} \deg_G(\pi(v)) &\le \left(1 + \frac{9\epsilon' \log\left|\deg_G(B^{\text{out}})\right|}{\epsilon}\right) \deg_G(B^{\text{out}}) + \left(1 + \frac{9\epsilon' \log\left|\deg_G(\bar{B}^{\text{out}})\right|}{\epsilon}\right) \deg_G(\bar{B}^{\text{out}}) \\
&\le \left(1 + \frac{9\epsilon'(\log(1-\epsilon) + \log\left|\deg_G(A)\right|)}{\epsilon}\right)(1+\epsilon')\deg_G(A - \bar{B}^{\text{out}}) \\
&\quad + \left(1 + \frac{9\epsilon' \log\left|\deg_G(\bar{B}^{\text{out}})\right|}{\epsilon}\right) \deg_G(\bar{B}^{\text{out}}) \\
&\le \left(1 + \frac{9\epsilon' \log\left|\deg_G(A)\right|}{\epsilon}\right) \deg_G(A).
\end{aligned}$$

The second inequality uses $\deg_G(B^{\text{out}}) < (1-\epsilon)\deg_G(A)$, $\lambda \ge 1000 \log^2 n$, and the definitions of $B^{\text{out}}, \bar{B}^{\text{out}}$. The third inequality uses

$$\left(1 + \frac{9\epsilon'(\log(1-\epsilon) + \log\left|\deg_G(A)\right|)}{\epsilon}\right) \le 2, \qquad \frac{\log(1-\epsilon)}{\epsilon} \le -0.4,$$

after plugging in ϵ', ϵ.

The running time consists of two recursive calls and building the layered projection. The latter is bounded by $\sum_{v \in V(G')} \deg_G(\pi(v))$, since every edge we add corresponds to some $v' \in V(G')$ and an edge $(\pi(v'), v)$ in G. The total complexity is

$$O(\log n \cdot \deg_G(B^{\text{out}})) + \tfrac{\log(\deg_G(A)) + \log(1-\epsilon)}{\epsilon} \cdot O(\log n \cdot (\deg_G(B^{\text{out}}) + \deg_G(\bar{B}^{\text{out}}))) + O(\deg_G(A))$$
$$\leq \log(\deg_G(A)) \cdot O(\log n \cdot \deg_G(A)/\epsilon),$$

where the first term is for finding i^{out} (or i^{in}), the second for recursion by induction, and the last for constructing G', which is $\leq 2\deg_G(A)$ since $\lambda > 10000 \log^4 n$.

Case 2: $\deg_G(B^{\text{out}}) \geq (1-\epsilon)\deg_G(A)$**.** Now $V(G') = V(\tilde{H}^{\text{out}}) \cup V(H^{\text{mid}}) \cup V(\bar{H}^{\text{in}})$ with $\tilde{H}^{\text{out}} = \textsc{PathCover}(\bar{B}^{\text{out}} \cap B^{\text{in}})$, $H^{\text{mid}} = G[B^{\text{out}} \cap B^{\text{in}}]$, and $\bar{H}^{\text{in}} = \textsc{PathCover}(\bar{B}^{\text{in}})$. We need:

Lemma 4.11. $\deg_G(\bar{B}^{\text{out}}), \deg_G(\bar{B}^{\text{in}}) \leq 2\epsilon \deg_G(A)$.

Proof. Since forward growing stops earlier than backward growing, the backward procedure accesses at least as many edges before stopping. As each accessed edge contributes to $\deg_G(B^{\text{in}})$, we have $\deg_G(B^{\text{out}}), \deg_G(B^{\text{in}}) \geq (1-\epsilon)\deg_G(A)$. By the definitions of $\bar{B}^{\text{out}}, \bar{B}^{\text{in}}$,

$$\deg_G(\bar{B}^{\text{out}}), \deg_G(\bar{B}^{\text{in}}) \leq \epsilon \deg_G(A) + \epsilon' \deg_G(A) \leq 2\epsilon \deg_G(A).$$

□

By induction,

$$\begin{aligned}
\sum_{v \in V(G')} \deg_G(\pi(v)) &\leq \left(1 + \frac{9\epsilon' \log\left|\deg_G(\bar{B}^{\text{out}} \cap B^{\text{in}})\right|}{\epsilon}\right) \deg_G(\bar{B}^{\text{out}} \cap B^{\text{in}}) \\
&\quad + \left(1 + \frac{9\epsilon' \log\left|\deg_G(\bar{B}^{\text{in}})\right|}{\epsilon}\right) \deg_G(\bar{B}^{\text{in}}) \\
&\quad + \deg_G(A) \\
&\leq \left(1 + \frac{9\epsilon' \left(\log\left|\deg_G(A)\right| + \log(2\epsilon)\right)}{\epsilon}\right) \cdot 2\epsilon \deg_G(A) \\
&\quad + \left(1 + \frac{9\epsilon' \left(\log\left|\deg_G(A)\right| + \log(2\epsilon)\right)}{\epsilon}\right) \cdot 2\epsilon \deg_G(A) + \deg_G(A) \\
&\leq \deg_G(A) + 9\epsilon \deg_G(A) \leq \left(1 + \frac{9\epsilon' \log\left|\deg_G(A)\right|}{\epsilon}\right) \deg_G(A),
\end{aligned}$$

using Lemma 4.11 and the definitions of $B^{\text{out}}, \bar{B}^{\text{out}}$, then plugging in ϵ, ϵ'.

The running time here includes two global SSSP computations costing $O(\deg_G(A) \log n)$, plus recursion time bounded by $(\log(1-\epsilon) + \log(\deg_G(A))) \cdot O(\log n \cdot \deg_G(A)/\epsilon)$. Thus, the total is $\log(\deg_G(A)) \cdot O(\log n \cdot \deg_G(A)/\epsilon)$. □

5 Deterministic Negative Weight SSSP

In Section 5.1, we show how to solve the *restricted SSSP* problem (defined below). Bringmann, Cassis, and Fischer [BCF23] showed a black-box reduction from solving SSSP on general graphs to restricted SSSP. Although the main statements in [BCF23] are randomized (see Sections 4 and 5 in [BCF23]), the reduction to a restricted graph is deterministic.

Definition 5.1 ([BCF23])**.** A *restricted graph* is a directed graph $G = (V, E, w)$ satisfying

- the edge weights lie in the set $\{-1, 0, 1, \ldots, n\}$,[5]
- for every cycle C, we have $w(C)/|C| \geq 1$,
- there is a vertex $s \in V$ with an edge of weight 0 from s to every other vertex.

The problem *restricted SSSP* is: given a restricted graph, find a shortest-path tree from s.

Sections 4 and 5 of [BCF23] prove the following reduction.

Lemma 5.2 (Sections 4 and 5 of [BCF23])**.** *If there is a deterministic algorithm for restricted SSSP with running time $T_{\text{RSSSP}}(m, n)$, then there is a deterministic algorithm for general SSSP with running time $O(T_{\text{RSSSP}}(m, n) \cdot \log(Wn))$.*

In Section 5.1, we prove the next lemma.

Lemma 5.3. *There is a deterministic algorithm solving restricted SSSP in $O(m \log^8 n)$ time.*

Combining Lemmas 5.2 and 5.3 yields Theorem 1.1.

5.1 An Algorithm for Restricted SSSP

We now describe the algorithm for restricted SSSP. The input is a directed graph $G = (V, E, w)$ and a source s satisfying Definition 5.1. The algorithm outputs a shortest-path tree of G from s.

Before stating the algorithm, we recall two standard subroutines: (1) computing distances when all negative edges must cross different SCCs, and (2) computing distances when the number of negative edges on every shortest path is small (by alternating Dijkstra and Bellman–Ford). We defer proofs to Section A.

Lemma 5.4. *Given a directed graph $G = (V, E, w)$ where $w(u, v) \geq 0$ for every edge (u, v) with u, v in the same SCC of G, there is a deterministic $O(m \log n)$-time algorithm for SSSP on G.*

Lemma 5.5 (SSSP with Few Negative Edges)**.** *There is a deterministic algorithm which, given a directed graph $G = (V, E, w)$ with no negative cycle, a source s, and an integer k such that every s-to-v shortest path uses at most k negative edges, computes $\text{dist}_G(s, \cdot)$ in $O(km \log n)$ time.*

[5] In the original definition of [BCF23], there is no upper bound n on the edge weights. We can assume an upper bound n because any edge of weight $> n$ cannot appear in the shortest-path tree from s: otherwise that path would have weight at least 1 (the most negative edge has weight at least -1 and a simple path has at most $n - 1$ edges), contradicting that s has a 0-weight edge to every vertex. This assumption also appears in [FHL+25].

Our algorithm is recursive. We describe an algorithm $\kappa\text{SSSP}(H, s, k)$ that solves a restricted SSSP instance (H, s) under the promise that every shortest s-to-v path in H uses at most k negative edges. It suffices to call $\kappa\text{SSSP}(H, s, n)$ to prove Lemma 5.3.

Base case. When $k = O(\log^6 n)$, apply Lemma 5.5 to (H, s, k) to obtain all distances from s in H. For the remainder, assume $k = \omega(\log^6 n)$.

Building a path cover. Compute a d_{cov}-path cover $G'_{\geq 0}$ of $H_{\geq 0}$ with at most $(1+1/\log n) \cdot |E(H)|$ edges and diameter slack λ such that

$$d_{\text{diam}} \leq \lambda \cdot d_{\text{cov}} \;=\; k/2.$$

By Theorem 4.5, it suffices to set

$$\lambda = 10000 \log^6 n \qquad \text{and} \qquad d_{\text{cov}} = \frac{k}{20000 \log^6 n},$$

and invoke Theorem 4.5 to obtain such a $G'_{\geq 0}$.

Recursive call. From $G'_{\geq 0}$, form G' by restoring each 0-weight edge back to its original weight in H: for every $(u', v') \in E(G'_{\geq 0})$ with $w(\pi(u'), \pi(v')) < 0$, set $w_{G'}(u', v') := w(\pi(u'), \pi(v'))$.

Define G'_{SCC} from G' by:

1. computing the SCC decomposition of G' and deleting all edges whose endpoints lie in different SCCs, and
2. adding a super-source s' and an edge of weight 0 from s' to every vertex of G'.

Call $\kappa\text{SSSP}(G'_{\text{SCC}}, s', k/2)$ to obtain $d_{s'}(\cdot) = \text{dist}_{G'_{\text{SCC}}}(s', \cdot)$.

Computing distances for H. Construct G'' as follows.

1. Make $x = 2\lambda$ copies of G', denoted $G'_1, \ldots, G'_x$. For $u' \in V(G')$, let $u'_i \in V(G'_i)$ be its copy.
2. For every $1 \leq i < x$, each $(u, v) \in E(H)$, and each copy $u'_i \in \pi^{-1}(u)$ in G'_i and the representative $v'_{i+1} \in \pi^{-1}(v)$ of v in G'_{i+1}, add the edge (u'_i, v'_{i+1}) with weight $w(u, v)$.
3. Add a source s'' and, for every vertex u'_i with $\pi(u') = s$, add the 0-weight edge (s'', u'_i).

Define a potential ϕ on G'' by

$$\phi(u'_i) = d_{s'}(u') \quad \text{for } u'_i \neq s'', \qquad \phi(s'') = 0.$$

We will show that G''_ϕ satisfies the premise of Lemma 5.4. Invoke Lemma 5.4 on G''_ϕ to compute $d_{s''} : V(G'') \to \mathbb{Z}$. For every $u \in V(H)$, output

$$d_s(u) = \min\{\, d_{s''}(u'_i) + \phi(u'_i) \;\mid\; u'_i \in V(G''),\ \pi(u') = u \,\}.$$

5.2 Correctness

Lemma 5.6. $\kappa\text{SSSP}(H, s, k)$ *correctly outputs* $\text{dist}_H(s, \cdot)$ *provided that*

- (H, s) *is a restricted SSSP instance, and*
- *every shortest path from* s *in* H *uses at most* k *negative edges.*

We proceed by induction on k. The base case $k = O(\log^6 n)$ follows directly from Lemma 5.5. For the induction step, assume $k = \omega(\log^6 n)$.

Validity of oracle calls. We first verify the two oracle calls: the recursive call $\kappa\text{SSSP}(G'_{\text{SCC}}, s', k/2)$ and the call to Lemma 5.4.

Lemma 5.7. (G'_{SCC}, s') *is a restricted SSSP instance.*

Proof. We verify the three bullets in Definition 5.1. Since $G'_{\geq 0}$ is a projection onto $H_{\geq 0}$ (Definition 4.1) and G' restores weights only by replacing some 0-weights with the original weights in H, G' is a projection onto H. As $G'_{\text{SCC}} - \{s'\}$ is a subgraph of G', it is also a projection onto H; hence all edge weights lie in $\{-1, 0, 1, \ldots, n\}$ (edges adjacent to s' have weight 0).

Let C be any cycle in G'_{SCC}. Since s' has no incoming edges, C lies in $G'_{\text{SCC}} - \{s'\}$, which projects to a (possibly non-simple) cycle $\pi(C)$ in H. By Definition 5.1, $w(\pi(C))/|\pi(C)| \geq 1$, hence $w(C)/|C| \geq 1$. The last bullet holds by construction of s'. □

Lemma 5.8. *Every shortest path from* s' *in* G'_{SCC} *uses at most* $k/2$ *negative edges.*

Proof. Suppose, for contradiction, that a shortest s'-to-v' path p in G'_{SCC} uses $> k/2$ negative edges. Let p' be p with its first vertex (the source s') removed; then p' still uses $> k/2$ negative edges and lies inside a single SCC of G' by construction of G'_{SCC}. Also $w_{G'_{\text{SCC}}}(p') \leq 0$, as otherwise the direct 0-edge from s' to $\text{End}(p')$ would give a strictly shorter path, contradicting minimality of p.

Write $\text{Start}(p') = u'$ and $\text{End}(p') = v'$. Since each SCC of $G'_{\geq 0}$ has (strong) diameter at most $d_{\text{diam}} \leq k/2$, there exists in $H_{\geq 0}$ a path $\tilde{p}$ from $\pi(v')$ to $\pi(u')$ of length at most $k/2$. The walk $\pi(p') \oplus \tilde{p}$ forms a (possibly non-simple) cycle in H, so by Definition 5.1,

$$w(\pi(p')) + w(\tilde{p}) \;\geq\; |\pi(p')| + |\tilde{p}|.$$

Using $w(\pi(p')) = w_{G'_{\text{SCC}}}(p') \leq 0$ and $w(\tilde{p}) \leq w_{\geq 0}(\tilde{p}) \leq k/2$, the number of *all* edges in p' is

$$|p'| = |\pi(p')| \;\leq\; k/2,$$

contradicting that p' has more than $k/2$ *negative* edges. □

Lemma 5.9. *For every edge* $(u'', v'') \in E(G''_\phi)$ *with* u'', v'' *in the same SCC of* G''_ϕ*, we have* $w_{G''_\phi}(u'', v'') \geq 0$.

Proof. The source s'' has no incoming edges, so assume $u'', v'' \neq s''$. By construction of G'', edges go only from G'_i to G'_j with $j \geq i$, hence if u'', v'' lie in the same SCC of G''_ϕ, they must

belong to the same copy G'_i and correspond to $u', v' \in V(G')$ lying in the same SCC of G'. It suffices to show

$$w_{G'}(u', v') + \phi(u') - \phi(v') \geq 0.$$

By definition, $\phi(z') = \mathrm{dist}_{G'_{\mathrm{SCC}}}(s', z')$ for $z' \in V(G')$. The triangle inequality in G'_{SCC} gives

$$w_{G'_{\mathrm{SCC}}}(u', v') + \mathrm{dist}_{G'_{\mathrm{SCC}}}(s', u') - \mathrm{dist}_{G'_{\mathrm{SCC}}}(s', v') \;\geq\; 0.$$

Since $w_{G'_{\mathrm{SCC}}}(u', v') = w_{G'}(u', v')$, we obtain $w_{G'}(u', v') + \phi(u') - \phi(v') \geq 0$, as desired. □

Correctness of the returned distances.

Lemma 5.10. *$d_s(u) \geq \mathrm{dist}_H(s, u)$ for all $u \in V(H)$.*

Proof. By Lemma 5.4 and Lemma 5.9, $d_{s''}(z)$ equals $\mathrm{dist}_{G''_\phi}(s'', z)$ for all $z \in V(G'')$. By Lemma 3.1,

$$d_{s''}(u'_i) + \phi(u'_i) = \mathrm{dist}_{G''}(s'', u'_i).$$

Any s''-to-u'_i path in G'' maps to an s-to-u path in H of the same weight by replacing each u'_i with $\pi(u')$; the initial edge (s'', u'_i) has weight 0 and ensures the path starts at s. Taking minima over copies yields $d_s(u) \geq \mathrm{dist}_H(s, u)$. □

Lemma 5.11. *$d_s(u) \leq \mathrm{dist}_H(s, u)$ for all $u \in V(H)$.*

Proof. Let p be a shortest s-to-u path in H. Since every edge weight is at least -1, we have the number of negative edges in p is at most k (by assumption) and hence $w_{\geq 0}(p) \leq k$; otherwise $w(p) \geq -k + w_{\geq 0}(p) > 0$, contradicting the 0-edges from s.

Recall $x = 2\lambda$ and $d_{\mathrm{cov}} = k/(2\lambda)$. Partition p into at most x subpaths $p = p_1 \oplus \cdots \oplus p_x$ with $w_{\geq 0}(p_i) \leq d_{\mathrm{cov}}$. Each p_i is covered by $G'_{\geq 0}$, hence also by G' (weights restored). For each i, let p'_i be a lift in G', and let p''_i be the copy of p'_i in G'_i. By construction of G'', there is an edge from $\mathrm{End}(p''_i)$ to $\mathrm{Start}(p''_{i+1})$ with weight equal to $w(\mathrm{End}(p_i), \mathrm{Start}(p_{i+1}))$. Thus $p'' = p''_1 \oplus \cdots \oplus p''_x$ is an s''-to-u'_x path in G'' with $w_{G''}(p'') = w(p)$. Therefore,

$$\min_{\pi(u')=u} \mathrm{dist}_{G''}(s'', u'_i) \leq \mathrm{dist}_H(s, u).$$

Using $d_{s''}(u'_i) + \phi(u'_i) = \mathrm{dist}_{G''}(s'', u'_i)$ finishes the proof. □

5.3 Time Complexity

Let $T(m, k)$ denote the running time of $\kappa\mathrm{SSSP}(H, s, k)$. We show $T(m, n) = O(m \log^8 n)$ by writing a recurrence for $T(m, k)$.

- Building the path cover for $H_{\geq 0}$ takes $O(m \log^4 n)$ time by Theorem 4.5 (since $\sqrt{\lambda} = 100 \log^3 n$).
- The recursive call requires an SCC decomposition in $O(m)$ time and costs

$$T\left(\left(1 + \tfrac{1}{\log n}\right) m,\ \tfrac{k}{2}\right).$$

- For computing distances, we build G''. Each vertex u'_i has out-degree $O(\deg_H(\pi(u')))$, and there are $x = O(\log^6 n)$ copies, hence

$$|E(G'')| \;\le\; O(\log^6 n) \cdot \sum_{u' \in V(G')} \deg_H(\pi(u')) \;=\; O(m \log^6 n)$$

  by Theorem 4.5. Therefore Lemma 5.4 on G'' costs $O(m \log^7 n)$ time.

We obtain the recurrence

$$T(m,k) \;\le\; T\left(\left(1 + \frac{1}{\log n}\right) m, \frac{k}{2}\right) \;+\; O\big(m \log^7 n\big)\,,$$

with base case $T(m, O(\log^6 n)) = O(m \log^7 n)$ by Lemma 5.5. Solving gives $T(m,n) = O(m \log^8 n)$.

References

[ABC+23] Vikrant Ashvinkumar, Aaron Bernstein, Nairen Cao, Christoph Grunau, Bernhard Haeupler, Yonggang Jiang, Danupon Nanongkai, and Hsin Hao Su. Parallel and distributed exact single-source shortest paths with negative edge weights. *arXiv preprint arXiv:2303.00811*, 2023. 1, 2, 3, 4

[ABCP98] Baruch Awerbuch, Bonnie Berger, Lenore Cowen, and David Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28(1):263–277, 1998. , 2, 25

[AMV20] Kyriakos Axiotis, Aleksander Madry, and Adrian Vladu. Circulation control for faster minimum cost flow in unit-capacity graphs. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 93–104. IEEE, 2020. 1

[Bar96] 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. 2

[BCF23] Karl Bringmann, Alejandro Cassis, and Nick Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In *FOCS*, pages 515–538. IEEE, 2023. , 1, 2, 17

[BNW25] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. *Commun. ACM*, 68(2):87–94, 2025. First announced at FOCS'22. , 1, 2, 25

[Chu21] Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 626–639, 2021. 2

[CKL+22] 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 *FOCS*, pages 612–623. IEEE, 2022. 1

[CKL+24] Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, and Maximilian Probst Gutenberg. Almost-linear time algorithms for incremental graphs: Cycle detection, sccs, st shortest path, and minimum-cost flow. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1165–1173, 2024. 1

[CMSV17] Michael B Cohen, Aleksander Madry, Piotr Sankowski, and Adrian Vladu. Negative-weight shortest paths and unit capacity minimum cost flow in $o(m^{10/7}logw)$ time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 752–771. SIAM, 2017. 1

[CP25] Julia Chuzhoy and Merav Parter. Fully dynamic algorithms for graph spanners via low-diameter router decomposition. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 785–823. SIAM, 2025. 2

[CZ23] Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*, pages 1159–1172, 2023. 2

[FHL+25] Nick Fischer, Bernhard Haeupler, Rustam Latypov, Antti Roeyskoe, and Aurelio L. Sulser. A simple parallel algorithm with near-linear work for negative-weight single-source shortest path. In *SOSA*, pages 216–225. SIAM, 2025. , 1, 2, 3, 17

[FL21] Arnold Filtser and Hung Le. Clan embeddings into trees, and low treewidth graphs. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 342–355, 2021. 4

[Gar85] Harold N Garbow. Scaling algorithms for network problems. *Journal of Computer and System Sciences*, 31(2):148–168, 1985. 1

[Gol95] Andrew V Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995. 1

[GP19] Mohsen Ghaffari and Julian Portmann. Improved network decompositions using small messages with applications on mis, neighborhood covers, and beyond. In *33rd International Symposium on Distributed Computing (DISC 2019)*, pages 18–1. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. 2

[GT89] Harold N Gabow and Robert E Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989. 1

[HHZ22] Bernhard Haepler, D Ellis Hershkowitz, and Goran Zuzic. Adaptive-adversary-robust algorithms via small copy tree embeddings. In *30th Annual European*

Symposium on Algorithms (ESA 2022), volume 244, page 63. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022. 4

[HLS24] Bernhard Haeupler, Yaowei Long, and Thatchaphol Saranurak. Dynamic deterministic constant-approximate distance oracles with n^ϵ worst-case update time. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2033–2044. IEEE, 2024. 2

[Joh77] Donald B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977. 7

[Li25] Jason Li. Deterministic padded decompositions and negative-weight shortest paths. 2025. 4

[LM24] Jason Li and Connor Mowry. A bottom-up algorithm for negative-weight sssp with integrated negative cycle finding. *arXiv preprint arXiv:2411.19449*, 2024. , 1, 2

[MPX13] Gary L Miller, Richard Peng, and Shen Chen Xu. Parallel graph decompositions using random shifts. In *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*, pages 196–203, 2013. 2

[VDBCK+23] Jan Van Den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Richard Peng, Maximilian Probst Gutenberg, Sushant Sachdeva, and Aaron Sidford. A deterministic almost-linear time algorithm for minimum-cost flow. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 503–514. IEEE, 2023. 1

[VDBCK+24] Jan Van Den Brand, Li Chen, Rasmus Kyng, Yang P Liu, Simon Meierhans, Maximilian Probst Gutenberg, and Sushant Sachdeva. Almost-linear time algorithms for decremental graphs: Min-cost flow and more via duality. In *2024 IEEE 65th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 2010–2032. IEEE, 2024. 1

[VDBLL+21] Jan Van Den Brand, Yin Tat Lee, Yang P Liu, Thatchaphol Saranurak, Aaron Sidford, Zhao Song, and Di Wang. Minimum cost flows, mdps, and ℓ_1-regression in nearly linear time for dense instances. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 859–869, 2021. 1

[vdBLN+20] 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 *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 919–930. IEEE, 2020. 1

A Basic Subroutines for SSSP

Proof of Lemma 5.4. Let the SCCs of G be $(V_1, V_2, \ldots, V_z)$ in a topological order of the condensation DAG. Define a potential function ϕ by

$$\forall\, i \in [z],\ \forall\, v \in V_i : \quad \phi(v) = -W \cdot i.$$

If $(u, v) \in E$ has u, v in the same SCC, then $w_\phi(u, v) = w(u, v) \geq 0$. Otherwise, if $u \in V_i$ and $v \in V_j$ with $i < j$, then

$$w_\phi(u, v) = w(u, v) + \phi(u) - \phi(v) = w(u, v) + W(j - i) \geq w(u, v) + W \geq 0.$$

Hence all edges of G_ϕ have nonnegative weight, so Dijkstra's algorithm computes SSSP on G_ϕ in $O(m \log n)$ time. By Lemma 3.1, this yields a shortest-path tree for G as well. □

Proof of Lemma 5.5. Build a $(k+1)$-layer graph G' as follows. For each $i \in [k+1]$, let G_i be a copy of $G_{\geq 0}$ (the subgraph of nonnegative-weight edges), and let u_i denote the copy of $u \in V$ in layer i. For every negative edge $(u, v) \in E$ with $w(u, v) < 0$ and every $i \in [k]$, add the interlayer edge (u_i, v_{i+1}) with weight $w(u, v)$. Run Dijkstra's algorithm from s_1 in G', and return

$$\min_{i \in [k+1]} \operatorname{dist}_{G'}(s_1, u_i) \qquad \text{as } \operatorname{dist}_G(s, u).$$

Running time. We have $|V(G')| = (k+1)|V|$ and

$$|E(G')| = (k+1)|E_{\geq 0}| + k|E_{<0}| \leq O(km).$$

Thus Dijkstra on G' takes $O(km \log |V(G')|) = O(km \log n)$ time since $|V(G')| \leq (k+1)n$.

Correctness. First, fix $i \in [k+1]$ and let P' be any $s_1 \to u_i$ path in G'. Projecting layers and replacing each interlayer edge by its original negative edge yields an $s \to u$ walk in G of weight at most $w(P')$. Hence $\operatorname{dist}_{G'}(s_1, u_i) \geq \operatorname{dist}_G(s, u)$.

Conversely, let p be a shortest $s \to u$ path in G. By assumption, p uses at most k negative edges. Deleting these negative edges decomposes p as

$$p \;=\; p_1 \oplus p_2 \oplus \cdots \oplus p_z,$$

with $z \leq k+1$ and each p_j consisting only of nonnegative edges. For each $j \in [z]$, let p'_j be the corresponding path in layer G_j (same vertices, same weight). For each negative edge between p_j and p_{j+1}, use the interlayer edge from layer j to $j+1$. Concatenating $p'_1, \ldots, p'_z$ with these interlayer edges yields an $s_1 \to u_z$ path in G' of weight exactly $w(p) = \operatorname{dist}_G(s, u)$. Therefore

$$\min_i \operatorname{dist}_{G'}(s_1, u_i) \leq \operatorname{dist}_G(s, u).$$

Combining both directions gives $\operatorname{dist}_G(s, u) = \min_i \operatorname{dist}_{G'}(s_1, u_i)$. □

B Randomized or Undirected Solutions for Problem 1.1

Low Diameter Decomposition: Randomized Solution. Let G be a directed graph G with non-negative weight and $d' \geq 0$. A *low-diameter decomposition* (LDD) is an edge set

$E' \subseteq E$ such that $G \setminus E'$ is a $\tilde{O}(d')$-clustered graph and, for each $e \in E$, $\Pr[e \in E'] \leq \frac{w(e)}{d'} \log n$. Bernstein et al. [BNW25] showed how to sample E' in near-linear time.

Observe that, by sampling $z = O(\log n)$ copies of LDD E'_i of diameter parameter $d' = 2d \log n$ and set $G'_i = G \setminus E'_i$, each length-d path p is covered by some G'_i with probability at least $1 - 1/n^{10}$. This is because, for each i,

$$\Pr[p \cap E'_i \neq \emptyset] \leq \sum_{e \in p} \Pr[e \in E'_i] \leq \sum_{e \in p} \frac{w(e)}{d'} \log n \leq \frac{w(p)}{d'} \log n \leq 1/2.$$

So, there is i where $p \subseteq G \setminus E'_i$ with probability $1 - 1/2^z \geq 1 - 1/n^{10}$. Thus, every path in the unknown target set is covered with probability at least $1 - 1/n^8$ by union bound.

Neighborhood Cover: Undirected Solution. A d-clustering $\mathcal{C}$ consists of vertex-disjoint subsets called *clusters*, each of which has diameter at most d. For any vertex v, let $B_v(v, d) = \{w \mid \text{dist}_G(v, w) \leq d\}$.

A d-neighborhood cover $\mathcal{N}_d$ of G is a collection of $z = O(\log n)$ many $O(d \log n)$-clustering $\mathcal{C}_1, \ldots, \mathcal{C}_z$ such that, for every vertex v, $B_v(v, d') \subseteq S$ for some cluster S in some clustering $\mathcal{C}_i$. For any d, Awerbuch et al. [ABCP98] showed how to construct $\mathcal{N}_d$ deterministically in near-linear time.

Observe that $\mathcal{N}_d$ gives the solution to Problem 1.1. Indeed, for each $i \leq z$, let $G'_i = \bigcup_{S \in \mathcal{C}_i} G[S]$. Consider any length-$d$ path p in G starting from vertex u. We have that $p \subseteq B_v(v, d) \subseteq G'_i$ for some i.

C A Barrier for Covering Paths via Subgraphs

In this section, we prove Theorem 1.2.

Theorem 1.2 (Barrier for Subgraphs)**.** *For every $m, \lambda \geq 1$, there exists an $O(m)$-edge directed graph G and d such that every collection of $d\lambda$-clustered graphs $G'_1, \ldots, G'_z \subseteq G$ covering all d-paths in G must have total size $\sum_i |E(G'_i)| = \Omega(m\sqrt{m/\lambda})$.*

Fix $m, \lambda \geq 1$. We will build a directed graph G with $O(m)$ edges and a distance parameter d.

Step 1: choose parameters. Let

$$L := \lfloor \sqrt{m/\lambda} \rfloor \quad \text{and} \quad d := 2 + 3L.$$

So $L = \Theta(\sqrt{m/\lambda})$ and $d = \Theta(\sqrt{m/\lambda})$.

We will also use one more parameter

$$R := 2d\lambda.$$

Note that $R > d\lambda$ by construction.

Step 2: ladder gadget. Create vertices $a_1, a_2, \ldots, a_{L+1}$. For each layer $j = 1, \ldots, L$ do:

- create R new vertices $w_{j,1}, \ldots, w_{j,R}$,

- add the directed cycle

$$w_{j,1} \to w_{j,2} \to \cdots \to w_{j,R} \to w_{j,1},$$

- add edges $a_j \to w_{j,r}$ for every $r \in [R]$,
- add edges $w_{j,r} \to a_{j+1}$ for every $r \in [R]$.

Thus, to "go through" layer j you can do

$$a_j \to w_{j,r} \to w_{j,r+1} \to a_{j+1},$$

which is 3 edges.

The number of edges in one layer is

$$R \text{ (cycle)} + R \text{ (entries)} + R \text{ (exits)} = 3R = 3 \cdot 2d\lambda = 6d\lambda.$$

Since there are L layers, the total number of edges in the ladder is

$$6d\lambda \cdot L = 6(2 + 3L)\lambda L = O(\lambda L^2) = O(\lambda \cdot (m/\lambda)) = O(m).$$

Key property of the ladder. Let H be any subgraph of G whose SCCs all have diameter at most $d\lambda$. Consider a fixed layer j. If H contained all R edges of the directed cycle of layer j, then that SCC would have directed diameter $R - 1 \geq d\lambda$, contradicting that H is $d\lambda$-clustered. Hence: For every $d\lambda$-clustered subgraph H and every layer $j \in [L]$, H omits at least one edge of that layer's cycle.

Step 3: star in front of the ladder. Add a special vertex r (the center of the star) and add the edge $r \to a_1$. Then add

$$M := \lfloor m/20 \rfloor$$

new vertices $s_1, \ldots, s_M$ and add star edges

$$s_t \to r \quad \text{for } t = 1, \ldots, M.$$

This adds $M + 1 = O(m)$ edges. Since the ladder already cost $O(m)$ edges, the whole graph G has $O(m)$ edges.

Step 4: the d-paths we care about. Consider a path that:

1. starts with some star edge $s_t \to r$,
2. then takes $r \to a_1$,
3. then for each layer $j = 1, \ldots, L$ takes a 3-edge traversal

$$a_j \to w_{j,r_j} \to w_{j,r_j+1} \to a_{j+1}.$$

This path has length

$$1 \text{ (star)} \ + \ 1 \text{ (to } a_1) \ + \ 3L \ = \ 2 + 3L \ = \ d.$$

So these are exactly d-paths in G.

Step 5: diagonalization for one star edge. Let $\{G'_1, \ldots, G'_z\}$ be any collection of $d\lambda$-clustered subgraphs of G that covers all d-paths.

Fix one star edge $e_t := s_t \to r$. Define

$$I_t := \{\, i \in [z] : e_t \in E(G'_i) \,\}.$$

We claim that

$$|I_t| \ \geq \ L. \tag{2}$$

Suppose toward a contradiction that $|I_t| = q < L$. For each $i \in I_t$, for every layer j it omits at least one cycle edge in that layer. Since $q < L$, we can pick for every $i \in I_t$ a *distinct* layer $\ell(i) \in [L]$ and let f_i be one cycle edge in layer $\ell(i)$ that G'_i omits.

Now we build a d-path P as in Step 4, but with the following extra choice: in layer $\ell(i)$ we force the traversal to *use* exactly the edge f_i, and in all other layers we traverse arbitrarily. This is a valid d-path because every layer allows us to pick any two consecutive cycle vertices.

By construction:

- If $i \notin I_t$, then G'_i does not contain the first edge e_t, so $P \not\subseteq G'_i$.
- If $i \in I_t$, then G'_i does not contain f_i (by choice), and P uses f_i, so again $P \not\subseteq G'_i$.

Thus P is a d-path that is not contained in any G'_i, contradicting the assumption that the family covers all d-paths. Therefore (2) holds.

Step 6: counting. For each of the $M = \Theta(m)$ star edges e_t we have found at least L subgraphs that contain it. So the total number of *edge–subgraph incidences* is at least

$$\sum_{t=1}^{M} |I_t| \ \geq \ M \cdot L.$$

But

$$\sum_{i=1}^{z} |E(G'_i)|$$

counts all edge–subgraph incidences (each time an edge appears in some G'_i it is counted once). Hence

$$\sum_{i=1}^{z} |E(G'_i)| \ \geq \ M \cdot L.$$

Finally, $M = \Theta(m)$ and $L = \Theta\big(\sqrt{m/\lambda}\big)$, so

$$\sum_{i=1}^{z} |E(G'_i)| \ \geq \ \Omega(m) \cdot \Omega\big(\sqrt{m/\lambda}\big) \ = \ \Omega\big(m\sqrt{m/\lambda}\big),$$

which is exactly the desired lower bound.

This completes the proof.