From Hop Reduction to Sparsification for Negative Length Shortest Paths

Kent Quanrud* Navid Tajkhorshid†

December 1, 2025

Abstract

The textbook algorithm for real-weighted single-source shortest paths takes $O(mn)$ time on a graph with m edges and n vertices. A recent breakthrough algorithm by Fineman [Fin24] takes $\tilde{O}(mn^{8/9})$ randomized time. The running time was subsequently improved to $\tilde{O}(mn^{4/5})$ [HJQ25] and then $\tilde{O}(mn^{3/4} + m^{4/5}n)$ [HJQ26].

We build on the algorithms of [Fin24; HJQ25; HJQ26] to obtain faster strongly-polynomial randomized-time algorithms for negative-length shortest paths. An important new technique in this algorithm repurposes previous "hop-reducers" from [Fin24; HJQ26] into "negative edge sparsifiers", reducing the number of negative edges by essentially the same factor by which the "hops" were previously reduced. A simple recursive algorithm based on sparsifying the layered hop reducers of [Fin24] already gives an $\tilde{O}(mn^{\sqrt{3}-1}) < O(mn^{.7321})$ randomized running time, improving [HJQ26] uniformly.

We also improve the construction of the bootstrapped hop reducers in [HJQ26] by proposing new sparse shortcut graphs replacing the dense shortcut graphs in [HJQ26]. Integrating all three of layered sparsification, recursion, and sparse bootstrapping into the algorithm of [HJQ26] gives new upper bounds of $O(mn^{.7193})$ randomized time for $m \geq n^{1.03456}$ and $O\Big((mn)^{.8620}\Big)$ randomized time for $m \leq n^{1.03456}$.

Lastly, concurrent work by [LLRZ25] obtained an $\tilde{O}(n^{2.5})$ randomized time algorithm for the same problem, and along the way improved the running time of the "betweenness reduction" step in Fineman's framework. Dropping in this subroutine as a black box improves the running time of the simple recursive sparsification algorithm to $\tilde{O}\Big(mn^{1/\sqrt{2}}\Big) \leq O(mn^{.70711})$, and a slightly modified recursive sparsification algorithm runs in $O(mn^{.69562})$ randomized time for $m \geq n^{1.0274}$ and $O((mn)^{0.850})$ for $m \leq n^{1.0274}$.

1 Introduction

Single-source shortest paths problem is a classical problem in graph algorithms and a standard topic of basic algorithms courses. The input consists of a directed graph $G = (V, E)$ with edge lengths $\ell : E \to \mathbb{R}$ and a source vertex $s \in V$. The goal is to output either the distance from s to every other vertex, or a negative cycle.

The textbook dynamic programming algorithm runs in $O(mn)$ time, where m is the number of edges, and n is the number of vertices. It was discovered in the 1950s independently by Shimbel [Shi55], Ford [For56], Bellman [Bel58], and Moore [Moo59] and remained the fastest algorithm

*krq@purdue.edu. Purdue University, West Lafayette, Indiana. Supported in part by NSF grant CCF-2129816.
†navidt@illinois.edu. University of Illinois at Urbana-Champaign, Urbana, Illinois.

until a recent breakthrough $\tilde{O}\left(mn^{8/9}\right)$ randomized time algorithm by Fineman [Fin24]. Fineman developed a new framework based on identifying "remote" sets of negative edges and neutralizing them efficiently by "hop reducers", which we discuss in greater detail below.

There are faster algorithms for important special cases including several notable recent developments. For nonnegative edge lengths, the widely-taught Dijkstra's algorithm takes $O(m + n \log n)$ time [Dij59; FT87], and an exciting new algorithm from [DMM+25] takes $O\left(m \log^{2/3} n\right)$ time. Weakly polynomial algorithms for integral edges weights have also recently attained significant milestones. [BNW22] gave the first nearly linear time algorithm (with respect to the bit complexity) for shortest paths. Concurrently, [CKL+25] gave the first almost-linear time algorithm (again, with respect to the bit complexity) for minimum cost flow, which generalizes shortest paths.

We are interested in the fully general setting of [Shi55; For56; Bel58; Moo59; Fin24] where the edge weights are real-valued. Since Fineman's surprising result, there have been two followup works building on Fineman's framework to further improve the running time. [HJQ25] first improved the running time to $\tilde{O}\left(mn^{4/5}\right)$ randomized time by introducing "proper hop distances" to more efficiently extract remote sets. [HJQ26] improved the running time further to $\tilde{O}\left(mn^{3/4} + m^{4/5}n\right)$ randomized time by developing a more efficient "bootstrapped" hop reducer replacing Fineman's layered hop reducer. This paper takes the next step in this line of research, with the following improved bounds.

Theorem 1.1. *Single-source shortest paths with real-valued edge lengths can be computed with high probability in $\tilde{O}\left(mn^{\frac{7-\sqrt{17}}{4}}\right)$ randomized time for $m \geq m_0$ and $\tilde{O}\left((mn)^{\frac{66-2\sqrt{17}}{67}}\right)$ randomized time for $m \leq m_0$, where $m_0 = O\left(n^{\frac{33-7\sqrt{17}}{4}}\right)$. (Here we have $\frac{33-7\sqrt{17}}{4} \approx 1.03456$, $\frac{7-\sqrt{17}}{4} \leq 0.719224$, and $\frac{66-2\sqrt{17}}{67} \leq 0.861997$.)*

Perhaps more importantly than the exact running time, this work introduces a few conceptual ideas on top of [Fin24; HJQ25; HJQ26] and to our understanding of real-weighted shortest paths more generally. There are really four main new techniques. The first idea is a randomized sparsification technique that transmutes hop reduction directly into a proportional decrease in the number of negative edges. The second idea is that of recursion, particularly to the sparsified graphs that now have substantially fewer negative edges. These first two techniques give a relatively simple, recursive algorithm that runs in $\tilde{O}\left(mn^{\sqrt{3}-1}\right)$ randomized time, improving on [HJQ26] without any of the bootstrapping machinery.

The remaining two techniques enhance the boostrapping hop reducer construction from [HJQ26]. First, we develop a sparse auxiliary "shortcut" graph that substitutes for a dense one in [HJQ26], and helps address the larger running time in sparse graphs in [HJQ26] (i.e., the $\tilde{O}\left(m^{4/5}n\right)$ term). The second technique "boosts" the bootstrap contruction by initiating the bootstrapping from a larger "base" subgraph, where the larger base case is supplied by the aforementioned method of sparsified recursion. Altogether we obtain the final bound of Theorem 1.1.

Independent work and improved running times. The results above were prepared and submitted for review in early November [QT25]. Concurrently, independent work by Li, Li, Rao, and Zhang [LLRZ25] obtained an $\tilde{O}(n^{2.5})$ randomized time algorithm for real-weighted shortest paths. Among other ideas, [LLRZ25] developed a faster "betweenness reduction" subroutine leveraging recursion. Substituting this subroutine as a black box improves the simple recursive algorithm from $\tilde{O}\left(mn^{\sqrt{3}-1}\right)$ to $\tilde{O}\left(mn^{1/\sqrt{2}}\right) \leq O(mn^{\cdot 70711})$ randomized time, and the bootstrap algorithm to $O(mn^{\cdot 7044})$ above some density threshold. A third algorithm integrating a subset of the techniques

from the bootstrapping algorithm into the recursive sparsification algorithm obtained the following running times.

Theorem 1.2. *Single-source shortest paths with real-valued edge lengths can be computed with high probability in $O(mn^{.69562})$ randomized time for $m \geq n^{1.0274}$, and $O\Big((mn)^{0.85}\Big)$ for $m \leq n^{1.0274}$.*

We describe the new subroutine from [LLRZ25], update the bounds for the two existing algorithms, and describe the third algorithm attaining Theorem 1.2, in Appendix A.

2 High-level overview and techniques

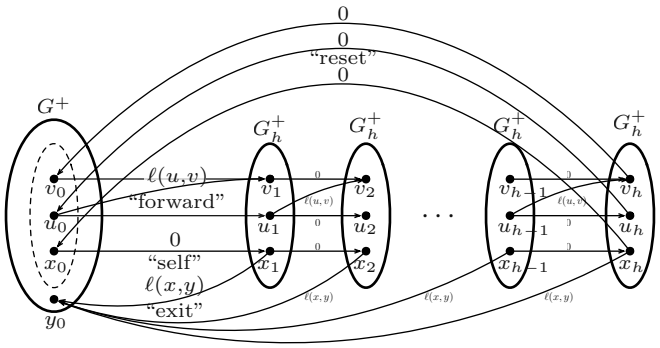
As the state-of-the-art algorithms have grown more complex and technical over time, it may help to give a high-level overview explaining the motivation and intuition behind the new techniques. We start by informally introducing a few notions and definitions key to the framework established by [Fin24]; these and additional definitions are described more formally in Section 3. To start, a "hop" refers to a negative edge, and an "h-hop walk" is a walk with at most h hops. The "h-hop distance" is the minimum length among all h-hop walks from u to v, and denoted $d^h(u, v)$. A "proper walk" walk is one where all the negative edges are distinct. A set of vertex potentials $\varphi : V \to \mathbb{R}$ induces edge lengths $\ell_\varphi(u, v) = \varphi(u) + \ell(u, v) - \varphi(v)$ that preserve the shortest path structure. We let $d_\varphi(u, v)$ denote distances with respect to ℓ_φ. A potential is *valid* if it introduces no new negative arcs, i.e. $\ell_\varphi(e) \geq 0$ if $\ell(e) \geq 0$. We say φ neutralizes a negative edge (u, v) if $\ell_\varphi(u, v) \geq 0$. Johnson [Joh77] observed that the potentials $\varphi(v) = d(V, v) = \min_{s \in V} d(s, v)$ are valid and neutralize the entire graph. In general, any potential of the form $\varphi(v) = d^h(V, v)$ is valid.

The goal is to iteratively compute valid potentials φ that neutralize negative arcs at a rate faster than $\tilde{O}(m)$ time per neutralized arc. Let k denote the number of negative vertices; by standard preprocessing, $k \leq n$. [Fin24] introduced the notion of *remote* sets of negative vertices that can be neutralized very efficiently. For a parameter $h \in \mathbb{N}$, a set of negative vertices U is "h-remote" if they can reach at most n/h vertices and m/h edges via negative-length h-hop walks. [Fin24] observed that given an h-remote set of vertices, one can construct an "h-hop reducer" H for the subgraph G_U with $O(m)$ edges. This means that H contains the vertices of G as a subset of V_H, and for any two vertices $u, v \in G$, and hop parameter $\eta \in \mathbb{N}$,

$$d_G(u, v) \leq d_H^{\lceil \eta/h \rceil}(u, v) \leq d_G^\eta(u, v)$$

In particular, one can compute Johnson's potentials for G, as a single-source, $(|U|/h)$-hop distance computation in H – a factor h speedup. Meanwhile, [HJQ25] showed that an h-remote set of size $\Omega\Big(\sqrt{kh}\Big)$ can be obtained in $\tilde{O}(mh^2)$ time. The $\tilde{O}\Big(mn^{4/5}\Big)$ running time in [HJQ25] is obtained by repeatedly extracting h-remote sets and neutralizing them via this h-hop reducer from [Fin24], for $h = \tilde{\Theta}\Big(k^{1/5}\Big)$.

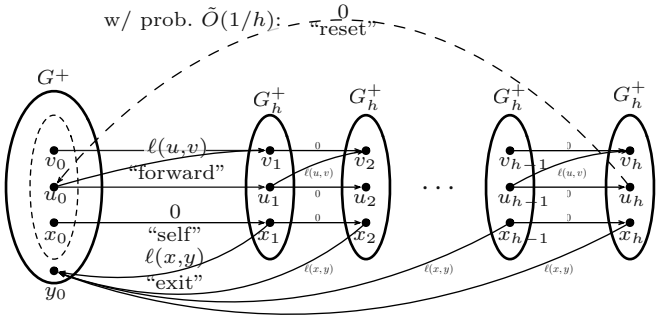
The hop reducer from [Fin24] is a simple and intuitive layered graph with $h + 1$ layers. Let $G_h = (V_h, E_h)$ be the subgraph induced by the h-hop negative reach of G^+. The first layer L_0 is a copy of the nonnegative part of the graph, G^+. The remaining h layers $L_1, \ldots, L_h$ are each a copy of G_h^+. For $v \in V$ and $i \in \{0, \ldots, h\}$, we let v_i denote the copy of v in ith layer L_i. Each negative arc (u, v) in U is added as a forward arc $(u_i, v_{i+1 \bmod h})$ between consecutive layers L_i and L_{i+1} for each layer index i. We also add length-0 "self" arcs $(u_i, u_{i+1 \bmod h})$ for each $u \in V_h$ and each index i. Finally, for every arc $(x, y) \in \partial^+(V_h)$ leaving the h-hop negative reach V_h, and each layer L_i for $i > 1$, we add an "exit" arc (x_i, y_0) with the same edge length.



Call the layered graph above H'. It is easy to see that H' preserves distances in the sense that for $u, v \in V$, $d_{G_U}(u, v) = d_{H'}(u_0, v_0)$. Now consider the potentials $\varphi : V_{H'} \to \mathbb{R}$ defined by $\varphi(v_i) = d^i(V, u)$. In the reweighted graph $H'' = H'_\varphi$, the only negative edges are the arcs (u_{h+1}, v_0) wrapping around from the final layer back to the first. Crucially, $\ell_\varphi(u_i, v_0) \geq 0$ for every exit arc $(u, v) \in \partial^+(V_h)$ since v is not in the h-hop negative reach of U. Now, for any h-hop walk $W : s \rightsquigarrow t$ in G, we have a 1-hop walk $W' : s_0 \rightsquigarrow t_0$ in H'' where each hop in w is mapped to a forward arc from one layer to the next, before wrapping back to get to t_0. It follows (with care) that H'' is an h-hop reducer, preserving distances from G, while reducing the number of hops in any walk by a factor of h.

[HJQ26] proposes a more involved hop reducer construction based on *bootstrapping*, which we will describe in a moment. Ultimately, the algorithms in [Fin24; HJQ25; HJQ26] all use h-hop reducers to reduce the number of hops needed to compute Johnson potentials for G_U. The first technique we introduce, recursive sparsification, is the first time an algorithm does *not* (directly) use hop reduction to speed up shortest path computations.

Consider again the layered h-hop reducer H'' described above. All the negative arcs "reset" from the last layer to the first. Long, many-hop walks through V_h are mapped through the layers of H'', so that h-hop subpaths become 1-hop "round trips" using all the layers of H''. Keeping the picture of a long walk W through G_h with at least h hops in mind, suppose we randomly sampled a subset $U' \subseteq U$ where each negative vertex/arc is sampled independently with probability $\Omega(\log(n)/h)$. The random sample U' hits the walk once every $\Omega(h/\log(n))$ hops on average, and once every h hops with high probability.



Assume this high probability event. Let H be the subgraph of H'' obtained by restricting the negative reset arcs to those in U'. We can route W through the layers of the H, taking at most h non-sampled hops through the layers before taking one of the sampled "reset" arcs. This observation, that walks through the negative reach can still be mapped through H with high probability, leads to the fact that H (essentially) preserves all distances in G_U, with the following salient property: in contrast to G_U, H has only $O(|U| \log(n)/h)$ negative edges!

Now, if we can neutralize H, then we can compute the Johnson's potentials for G_U in a single Dijkstra computation over the neutralized H. How do we neutralize H?

Recurse! This simple algorithm leads to a running time recursion of the form

$$T(m,k) = \tilde{O}\Big(\sqrt{k/h}\Big(mh^2 + T\Big(m, \sqrt{k/h}\Big)\Big)\Big),$$

This recursion is solved by $T(m,k) = \tilde{O}\Big(mk^{\sqrt{3}-1}\Big)$, where $\sqrt{3}-1 \approx 0.732051$. It is surprising that such a relatively simple algorithm already improves [HJQ26] for all graph densities. It is also curious to note that this algorithm never explicitly applies hop reduction to accelerate the computation of Johnson's potentials. Evidently, it is more beneficial to exchange the hop reduction for a proportional decrease in the number of negative arcs.

2.1 A leaner bootstrapped hop reducer

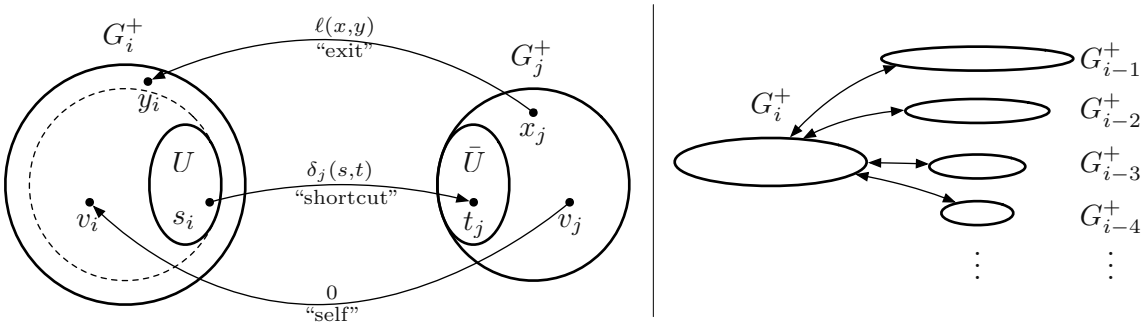
The improvement from [HJQ25] to [HJQ26] comes primarily from a stronger hop reducer. Let us say that a set of negative edges is (h_1, h_2)-remote if it can h_1-hop negatively reach at most n/h_2 vertices. Thus U is h-remote if it is (h,h)-remote. [HJQ26] observed that in essentially the same $\tilde{O}(mh^2)$ running time to previously compute an h-remote set U of size $\tilde{\Omega}\Big(\sqrt{kh}\Big)$, one can compute a set of negative edges U of size $\tilde{\Omega}\Big(\sqrt{k}\Big)$ that is $(h_1, h_1/h^2)$-remote for all $h_1 \geq \Omega(\log n)$. In particular, the $\tilde{O}(1)$-hop negative reach of U has n/h^2 vertices, and grows smoothly until the h^2-hop negative reach becomes the entire graph.

The goal is to construct an h^2-hop reducer for G_U in $\tilde{O}(mh^2 + m|U|/h^2)$ time. [HJQ26] achieves this by the following *bootstrapping construction*. Assume for simplicity that G is sufficiently dense. Consider the subgraphs G_i of G_U induced by the 2^i-hop negative reach for $i = 1, ..., 2\log h$. The incremental goal is to build a $\Omega(2^i)$-hop reducer H_i, with $O(m2^i/h^2)$ edges, for each G_i; for $i = 2\log h$ this gives the desired $O(m)$-edge $\Omega(h^2)$-hop reducer for $G_{2\log h} = G_U$. Now, it is easy to efficiently construct a 2-hop reducer from G_1 because G_1, with $O(m/h^2)$ edges, is so small. The challenge is trickier for larger i. For example, for $i > \log h$, the 2^i-layered hop reducer described above has $\Omega(m2^{2i}/h^2) > m$ edges.

The key idea is to use the previous hop reducers $H_1, \ldots, H_{i-1}$ to construct the next hop reducer H_i. An initial inspiration might be to try to use H_{i-1} to compute distances in G_i between vertices in U much faster; then those distances can be used to "shortcut" the layered graph construction. The "shortcut hop reducer" would only be one additional layer, with auxiliary forward arcs between the tail of each negative edge to the head of each negative arc (from U), with arc lengths based on the distances computed via H_{i-1}. The problem here is that it takes too long to compute hop distances from every $u \in U$, even when it is only a constant number of hops in H_i.

The actual construction in [HJQ26] is more subtle. For each $j < i$, [HJQ26] uses the hop reducer H_j to compute "distance estimates" $\delta_j(u,v)$ between every pair $u, v \in U$ that are at least as good as any proper $\Theta(2^j)$ hop walk from u to v. In particular, it does not try to compete with walks with significantly less than 2^j hops. This restriction allows for the following speedup: rather than compute single source hop distances from every $u \in U$ in $\tilde{O}(m|U|2^j/h^2)$ time, we can randomly sample $\tilde{O}(|U|\log(n)/2^j)$ vertices $X_j \subseteq U$ uniformly at random, and compute the $O(2^j)$-hop distances in G_j (via H_j) to and from every $x \in X_j$, in $\tilde{O}(m|U|/h^2)$ time. For every $s, t \in U$, and sampled $u \in X_j$, the computed distances from s to u and then to t gives a valid distance from s to t. With high probability, X_j "hits" all the shortest walks with at least $\Omega(2^j)$ hops. As such, the distance estimates via X_j "capture" all the proper $\Theta(2^j)$-hop distances in G_j.

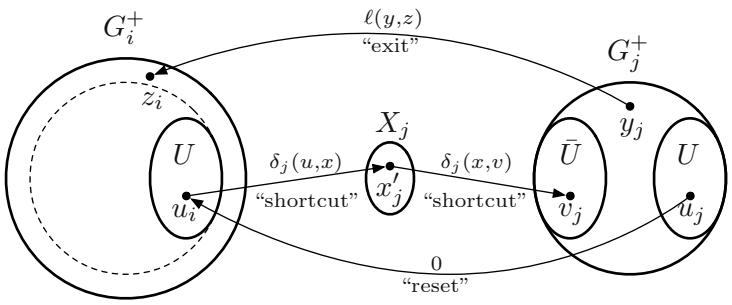
At this point we have a distance estimate $\delta_j(u,v)$ that is bounded above by the length of any $\Omega(2^j)$-hop (u,v)-walk in G_j, for all $j < i$. To construct the $\Omega(2^i)$-hop reducer H_i for G_i, [HJQ26] starts with disjoint copies of G_j^+ for each index $j \le i$. We let v_j denote the copy of v in G_j^+, for $v \in V$ and $j \le i$. For every negative arc (u,v), and each index j, we add the "shortcut" arc (u_i, v_j) with an edge length based on $\delta_j(u,v)$. (Omitting details, some adjustment is made to $\delta_j(u,v)$ when the distance estimate is observed to be too good to be true.) There are also "exit" arcs (y_j, z_i) for each arc (y,z) leaving the 2^j-hop negative reach. Lastly, there are also length-0 "reset" arcs (u_{i-1}, u_i) for each $u \in U$.



The overall hop reducer is arranged as a star, with a "base" copy of G_i^+ in the center, connected to each G_j^+ with $j < i$ via the gadget described above. Each G_j^+-gadget can be interpreted as a compressed version of the layered graph construction for G_j, with each "roundtrip" through G_j^+ reducing 2^j hops, with the significant difference this gadget only "captures" walks with $\Theta(2^j)$-hop walks, instead of any walk with $O(2^j)$ hops. With an appropriate potential function, the only negative arcs are the reset arcs from G_{i-1}^+ back to G_i^+. With a more delicate argument than for the layered graph construction described previously, one can argue that any walk in G_i can be mapped through our construction with 2^{i-1}-factor less hops, giving us a 2^{i-1}-hop reducer H_i for G_i. Bootstrapping in this fashion until $i = 2\log h$ leads to a desired h^2-hop reducer for G.

But consider the number of edges in the ith hop reducer H_i. The disjoint copies of G_j contribute at most $O(m2^i/h^2)$ edges, as desired. However, there are $|U|^2 = \tilde{\Omega}(n)$ shortcut edges, which for small i can be a bottleneck in sufficiently sparse graphs. The net effect is that [HJQ26] obtains a $\tilde{O}\left(mn^{3/4} + m^{4/5}n\right)$ running time; in particular, $\tilde{O}\left(m^{4/5}n\right)$ rather than $\tilde{O}\left(mn^{3/4}\right)$ for $m \le \tilde{O}\left(n^{5/4}\right)$.

[HJQ26] is hamstrung by a mismatch between the sparsified shortest path computation obtaining the distance estimates, and the dense sets of shortcut edges based on the distance estimates. Recall that the distance estimates δ_j for G_j^+ are based on computing distances to and from a randomly sampled subset $X_j \subseteq U$, while we add shortcut edges for every pair $s, t \in U$. It is natural and more efficient to have a construction that mirrors the computation, introducing an auxiliary copy of X_j between G_i^+ and G_j^+, with edges from the copy of U in G_i^+ to the auxiliary copy of X_j based on the distances to X_j, and edges from the copy of X_j to the copy of U in G_j^+ based on the distances from X_j. Altogether we would have a "bowtie" connecting G_i^+ to G_j^+ via X_j, with $2|U||X_j|$ vertices.



Our next contribution is precisely such a construction. Consider the diagram above. For a vertex u and index j, we let u_j denote the copy of u in G_j^+ and u'_j the copy of u in X_j (when these auxiliary copies exist.) For negative vertex $u \in U$ and $x \in X_j$, we have a shortcut arc (u_i, x'_j) with a length $\delta_j(u,x)$. For each head of a negative edge, $v \in \overline{U}$, and sampled vertex $x \in X_j$, we have another shortcut arc with a have length $\delta_j(x,v)$. Observe that we only have $O(|U||X_j|) = \tilde{O}\Big(|U|^2/2^j\Big)$ shortcut edges. The subtle challenge comes in setting the right edge lengths for $\delta_j(u,x)$ and $\delta_j(x,v)$.

The values $\delta_j(u,x)$ and $\delta_j(x,v)$ need to be "sound", in the sense that they do not underestimate actual distances in G_j. They should also be complete over all proper $\Theta(2^h)$-hop walks from U to $\overline{U}$ in the following sense: for every pair $u \in U$ and $v \in \overline{U}$, there is some $x \in X$ such that $\delta_j(u,x) + \delta_j(x,v)$ is at most the length of any (u,v) walk through G_j with $\Theta(2^j)$ hops. The natural candidate is to use the 2^j-hop reducer from H_j to compute $O(2^j)$ hops starting and ending at each $x \in X_j$, and set $\delta_j(u,x)$ and $\delta_j(x,v)$ to these computed distances.

The second step of the hop reducer construction is to define valid vertex potentials over X_j and G_j^+ that neutralize all edges in this gadget except for the reset arcs. Our initial values for $\delta_j(u,x)$ and $\delta_j(x,v)$ are not quite appropriate for this task and require the following adjustments. First, for $u \in U$ and $x \in X_j$, if the initially computed value $\delta_j(u,x)$ is less than $d_j^{2^{j-1}}(V_j,x)$, we increase it to $d_j^{2^{j-1}}(V_j,x)$. Second, for $x \in X_j$ and $v \in \overline{U}$, if the initially computed value of $\delta_j(x,v)$ is less than $d^{2^j}(V_j,v) - d^{2^{j-1}}(V_j,x)$, we increase it $d^{2^j}(V_j,v) - d^{2^{j-1}}(V_j,x)$. Increasing edge lengths does not risk soundness, and one can argue that these increased edge length are still complete, in the sense described above.

Now, with these adjusted edge lengths, we define a potential over X_j and G_j^+ by setting $\varphi\Big(x'_j\Big) = d_j^{2^{j-1}}(V_j,x_j)$ for $x \in X_j$ and $\varphi(v_j) = d_j^{2^j}(V_j,v)$ for $v \in V_j$, where $d_j(\cdot,\cdot)$ denotes distances in G_j. One can show that these vertex potentials are valid over G_j^+ and the exit arcs from G_j^+ to G_i^+, and also neutralize the shortcut arcs to and from X_j. The only negative arcs remaining are the reset arcs.

Starting with G_i^+, and adding this "sparse shortcut graph" gadget for every G_j^+, $j < i$, gives a $\Theta(2^i)$-hop H_i reducer for G_i^+. The jth shortcut graph now has $O\Big(|U|^2/2^j\Big)$ shortcut edges, instead of $O\Big(|U|^2\Big)$.

Boosting the bootstrapped hop reducer. Alas, the sparser shortcut-gadget construction does not reduce the number of edges in the first shortcut gadget for G_1^+, and overall the number of edges in the hop reducer has only decreased by a logarithmic factor. The bottleneck is the density of the gadgets for the smallest subgraphs $G_1^+, G_2^+, ...$

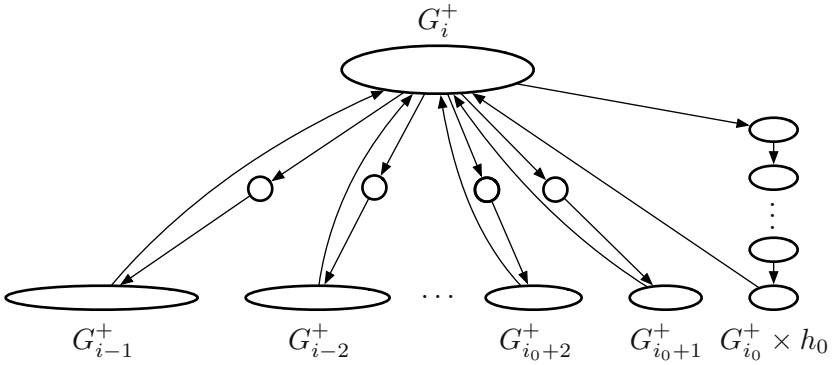
Here we return to the ideas of layered sparsification and recursion to boost the bootstrapping construction. The bootstrapping is bottlenecked at the smallest negative-reach subgraphs. Instead, we directly neutralize the subgraph induced by the h_1-hop negative reach with layered sparsification and recursion, for a moderate value of h_1, and start the bootstrapping process from there. Starting

at h_1-hop negative reach, instead of constant negative reach, also allows us to increase the size of our remote set by a poly(h_1)-factor.

More precisely, let $i_0 \in \mathbb{N}$ be a parameter to be determined, and let $i_1 = 2i_0$. Let $h_0 = 2^{i_0}$ and $h_1 = 2^{i_1} = h_0^2$. In $\tilde{O}(mh)$ time, we sample a set of $\tilde{\Omega}(\sqrt{kh_0})$ negative vertices U that is $(\eta, h/\eta)$-remote for all $\eta \geq h_0$. Let V_j be the 2^j-hop negative reach, and G_j the subgraph of G_U induced by V_j, as before.

We first apply the layered sparsification and recursive technique to G_{i_1}, using G_{i_0} as the layers. G_{i_1} has mh_1/h edges, nh_1/h vertices, and $|U|$ negative vertices. G_{i_0} has mh_0/h edges and nh_0/h vertices. By making h_0 layers of G_{i_0} over a base graph of G_{i_1} and sparsifying the negative edges, we obtain a graph H'_{i_1} with $O(mh_1/h)$ edges, $\tilde{O}(|U|/h_0)$ negative edges, and preserving distances from G_{i_1}. We recursively neutralize H'_{i_1}, and use the neutralized H'_{i_1} to compute Johnson potentials and neutralize G_{i_1}.

In neutralizing G_{i_1}, we've also neutralized G_j for all $j < i_1$. These neutralized graphs can be used to compute the lengths for the shortcut edges, and construct the sparse gadget described above, for all G_j with $i_0 < j \leq i_1$. For each such j, the sparse gadget for G_j captures all proper walks with $\Theta(2^j)$ hops through G_j. Since we do not create any gadgets for $j < i_0$, G_{i_0} needs a different gadget to capture all walks with at most h_0 hops in G_{i_0}. To this end we add the familiar layered auxiliary graph, with h_0 copies of $G^+_{i_0}$.



We now have a $\Theta(2^{i_1})$-hop reducer H_{i_1} for G_{i_1}, for a remote set U of $\tilde{\Omega}(\sqrt{kh_0})$ negative vertices, with the sparse gadgets starting from G_{i_0+1}, instead of G_1. Continuing the bootstrap process from H_{i_1} leads to an $\Theta(h)$-hop reducer H for all of G_U. Since the shortcut gadgets only go down to G_{i_0+1}, each reducer has $\tilde{O}\left(|U|^2/h_0\right)$ added shortcut edges, instead of $\tilde{O}\left(|U|^2\right)$. Finally, we compute Johnson's potentials neutralizing G_U via H, in $\tilde{O}(m|U|/h)$ time, as a $|U|/h$-hop distance computation. Choosing i_0 appropriately to balance parameters leads to the $O(mn^{.7193})$ randomized running time for $m \geq n^{1.03456}$. An additional preprocessing step (using layered sparsification again!) leads to the $O((mn)^{.8620})$ randomized running time in sparser graphs.

Why didn't we sparsify and recurse on H, and on each intermediate hop reducer H_i along the way? Recursion wouldn't hurt, but it still takes $\tilde{O}(m|U|/h)$ time to compute the distance estimates $\delta_j(u, v)$ needed to label the shortcut edges in each sparse gadget. Consequently the $\tilde{O}(m|U|/h)$ time spent directly computing Johnson's potentials in H, instead of recursing, is not a bottleneck. Of course, one could have recursed on these hop reducers anyway, resulting in an algorithm with the same running time that never explicitly uses hop reduction to speed up a distance computation.

In conclusion, this work introduces a new perspective recasting hop reducers as negative-edge sparsifiers. This step makes recursion more viable and leads quickly to a relatively simple algorithm improving the state of the art. These techniques were then incorporated into bootstrapping (which this work also improves) to improve the running time further. Naturally, we don't believe these

running times are the best possible. At this stage, we are still developing basic techniques and making elementary observations on a relatively new problem. We are hopeful that the new ideas and techniques, especially the idea of sparsification via hop reduction, will be helpful for future algorithms.

3 Preliminaries

Let $\mu = m + n \log n$ and let k denote the number of negative edges.

Distances. The distance from s to t, denoted by $d(s,t)$, is defined as the infimum length over all walks from s to t. For vertex sets S and T, we let $d(S,T)$ denote the minimum distance $d(s,t)$ over all $s \in S$ and $t \in T$.

Preprocessing and negative vertices. By standard preprocessing, we assume that the maximum in-degree and out-degree are both $O(m/n)$, that there are $k \leq n/2$ negative edges, and that each negative edge (u,v) is the unique incoming edge of its head v and the unique outgoing edge of its tail u [Fin24; HJQ25]. We identify each negative edge (u,v) with its tail u, and call u a *negative vertex*. When there is no risk of confusion, we may reference a negative edge by its negative vertex and vice-versa. We let N denote the set of negative vertices. For a set of negative vertices $U \subseteq N$, we let $\overline{U} = \{v : u \in U, (u,v) \in E\}$ be the set of heads of the negative edges associated with U.

For a set of negative vertices U, we let G_U denote the subgraph obtained by restricting the set of negative edges to those corresponding to U. We let $d_U(\cdot,\cdot) = d_{G_U}(\cdot,\cdot)$ denote distances in G_U. We let $G^+ = G_\emptyset$ denote the subgraph of nonnegative edges.

Hop distances and proper hop distances. The number of *hops* in a walk is the number of negative edges along the walk, counted with repetition. An *h-hop walk* is a walk with at most h negative edges. The *h-hop distance* from s to t is the minimum length over all h-hop (s,t)-walks. Hop distances satisfy

$$d^{h+1}(s,t) = \min\Big\{d^h(s,t),\, \min_{u \in N} d^h(s,u) + \ell(u,v) + d^0(v,t)\Big\}. \tag{1}$$

Given $d^h(s,u)$ for all u, one can compute $\min_{u \in N} d^h(s,u) + \ell(u,v) + d^0(v,t)$ for all t with a single call to Dijkstra's algorithm over an appropriate auxiliary graph. Thus (1) also describes a dynamic programming algorithm computing single-source hop distances in $O(\mu)$ time per hop [DI17; BNW22]. With parent pointers, one can also recover the underlying h-hop walks in the same running time.

A *proper h-hop walk* is defined as a walk with exactly h negative edges and where all negative vertices are distinct. For vertices s,t, let $\hat{d}^h(s,t)$ denote the infimum length over all proper h-hop walks from s to t. We call $\hat{d}^h(s,t)$ the *proper h-hop distance* from s to t.

Proper hop distances are generally intractable. For large h, proper hop distances capture NP-Hard problems such as Hamiltonian path. They were introduced in [HJQ25] to extract larger "negative sandwiches", a precursor to remote sets, with an argument that bypasses this intractability. In [HJQ26], proper hop distances played a second role in the construction of the bootstrapped hop reducer. Here, a random sampling technique used to compute the distance estimates only works for walks with many distinct negative edges.

Vertex potentials. Given vertex potentials $\varphi : V \to \mathbb{R}$, let $\ell_\varphi(e) \stackrel{\text{def}}{=} \ell(e) + \varphi(u) - \varphi(v)$ for an edge $e = (u,v)$. We say that φ *neutralizes* a negative edge e if $\ell_\varphi(e) \geq 0$, and neutralizes a negative vertex if it neutralizes the corresponding negative edge. Johnson [Joh77] observed that the potential $\varphi(v) = d(V,v)$ neutralizes all edges (assuming there are no negative-length cycles). We let G_φ denote the graph with edges reweighted by φ and d_φ denote distances with respect to ℓ_φ.

We say that potentials φ are *valid* if $\ell_\varphi(e) \geq 0$ for all e with $\ell(e) \geq 0$; i.e., φ does not introduce any new negative edges. For any hop parameter $h \in \mathbb{N}$ and any set of vertices S, the potentials $\varphi_1(v) = d^h(S, v)$ and $\varphi_2(v) = -d^h(v, S)$ are valid. Given two valid potentials φ_1 and φ_2, $\max\{\varphi_1, \varphi_2\}$ and $\min\{\varphi_1, \varphi_2\}$ are also valid potentials. If φ_1 is valid for G, and φ_2 is valid for G_{φ_1}, then $\varphi_1 + \varphi_2$ is valid for G.

All algorithms discussed here follow an incremental reweighting approach, dating back to [Gol95], where the graph is repeatedly reweighted along valid potentials that neutralize a subset of negative edges, until (almost) all edges are nonnegative. (A small number of remaining negative edges can always be neutralized by Johnson's technique in nearly linear time per negative edge.) Then the distances can be computed by Dijkstra's algorithm in the reweighted graph. Henceforth we focus exclusively on the effort to neutralize all or almost all of the negative edges.

When algorithms reweight the graph along multiple potentials in one iteration, we treat any initially negative edge as negative throughout the iteration, even if incidentally neutralized. This is needed to preserve hop-counting invariants. k remains the number of negative edges at the beginning of the iteration.

Remote edges. We say that a vertex s can *h-hop negatively reach* another vertex t if $s = t$ or there is an h-hop (s, t)-walk with negative length. For parameter $h, r \in \mathbb{N}$, and a set of negative vertices $U \subseteq V$, U is *(h, r)-remote* if the collective h-hop negative reach of U has at most n/r vertices. We say U is *h-remote* if it is (h, h)-remote.

The algorithms here and in [Fin24; HJQ25; HJQ26] alternate between extracting polynomially sized remote sets, and neutralizing them more efficiently. The first step of extracting a remote set is quite involved, and since the original method was introduced in [Fin24], has been improved in both subsequent works [HJQ25; HJQ26].

This work focuses exclusively on the second step of neutralizing remote sets, and uses previous techniques to extract the remote set in a black box fashion. We will use the following lemma.

Lemma 3.1. *Let $h_0 = \Omega(\log n)$ and $h \geq h_0$. One can compute, with high probability in $O\Big(h\mu \log^2 n + h^2\sqrt{k/h_0^3}\log^2 n\Big)$ randomized time, either:*

(i) A negative cycle.

(ii) Valid potentials φ neutralizing a set of $\Omega\Big(\sqrt{kh_0}\Big)$ negative vertices.

(iii) Valid potentials φ and a set U of negative vertices such that:

(a) $|U| \geq \Omega\Big(\sqrt{kh_0}\Big)$.

(b) U has η-hop negative reach of size $n\eta/h$ for all $\eta \geq h_0$.

Lemma 3.1 combines Lemmas 3.4 and 6.1 from [HJQ26]. The version in [HJQ26] is stated for $h_0 = O(\log n)$; the same proof extends to a general parameter h_0 as stated above.

Hop reducers. h-remote sets were introduced by [Fin24] to construct linear-size hop reducers. Formally, we say that a graph $H = (V_H, E_H)$ is an *h-hop reducer* for a graph $G = (V, E)$ if $V \subseteq V_H$ and $d_G(s, t) \leq d_H^{\lceil \eta/h \rceil}(s, t) \leq d_G^\eta(s, t)$ for all $s, t \in V$ and $\eta \in \mathbb{N}$.

The layered graph from [Fin24], described in Section 2, was an h-hop reducer of size $O(m)$. The bootstrapped construction from [HJQ26] gave an h^2-hop reducer of size $\tilde{O}(m)$.

4 Layered sparsification and recursion

We first describe and analyze a relatively simple algorithm running in $\tilde{O}\Big(mn^{\sqrt{3}-1}\Big)$ randomized time. This running time improves the previous $\tilde{O}\Big(mn^{3/4} + m^{4/5}n\Big)$ running time while cleanly

demonstrating two of the four main new techniques, layered sparsification and recursion. Subsequent sections build on these ideas to obtain even faster running times.

Layered sparsification recasts the layered auxiliary graph from [Fin24] from a hop-reducer to a "negative edge sparsifier", reducing the number of negative edges by essentially the same hop reduction factor.

Lemma 4.1. *Let U be a set of negative vertices with h-hop negative reach n/r. Let $m' = (2 + h/r)m + (h/r)n$ and $n' = (2 + h/r)n$. In $O(m')$ randomized time and with high probability, one can compute a graph H, potentials $\varphi : V_H \to \mathbb{R}$, and mappings $\pi_0, \pi_1 : V_G \to V_H$, such that:*

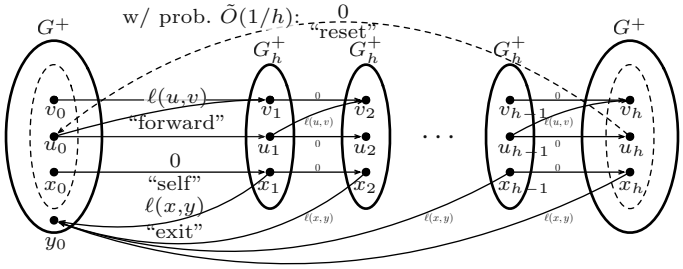
(a) H has m' edges and n' vertices.

(b) H_φ has $O(|U| \log(n)/h)$ negative edges.

(c) For any two vertices $u, v \in V_G$, $d_{G_U}(u, v) = d_H(\pi_0(u), \pi_1(v))$.

Proof. For ease of notation we let $G = G_U$. We assume $h \geq \Omega(\log n)$, since otherwise we can take $H = G$.

The auxiliary graph H is an $(h+1)$-layer graph based on the layered h-hop reducer introduced in [Fin24]. Let G_h be the h-hop negative reach of U. We start with two copies of G^+ (layers 0 and h) and $h-1$ copies of G_h^+ (layers 1 through $h-1$). For a vertex v and index $i \in \{0, \ldots, h\}$, let v_i denote the copy of v in layer i. We connect these $h+1$ nonnegative subgraphs with four types of arcs. First, for each layer $i \in \{0, \ldots, h-1\}$ and negative arc (u, v), we add a "forward arc" (u_i, v_{i+1}) from the ith layer to the next with length $\ell(u, v)$. Second, for each such layer i and vertex $x \in G_h$ we also add a "self-arc" (x_i, x_{i+1}) with length 0. Third, for each index $i \in \{1, \ldots, h\}$ and each arc $(x, y) \in \partial^+(G_h)$ leaving G_h, we add an "exit" arc (x_i, y_0) of length $\ell(x, y)$. Lastly, let $U_0 \subseteq U$ sample $O(|U| \log(n)/h)$ negative vertices uniformly at random from U. For each sampled negative vertex $u \in U_0$, we add a "reset arc" (u_h, u_0) with length 0. Lastly, we define the maps $\pi_0, \pi_1 : V_G \to V_H$ by $\pi_0(v) = v_0$ and $\pi_1(v) = v_h$.



We define potentials φ over V_H by $\varphi(v_i) = d_U^i(V, v)$. It is easy to see that φ is valid within each of the nonnegative subgraphs, and over the self-arcs. It is also straightforward to verify that φ neutralizes all forward arcs. As observed in [Fin24], φ is valid over the exit arcs (u_i, v_0), where $u \in G_h$ and $v \notin G_h$, precisely because v is not in the h-hop negative reach of U. Thus the only negative arcs in H_φ are the reset arcs (u_h, u_0) for $U_0 \subseteq U$, of which there are $O(|U| \log(n)/h)$.

It remains to prove property (c). The edges in H are either length 0 self or reset arcs (u_i, u_j) between auxiliary copies of the same vertex u, or copies (u_i, v_j) of an edge (u, v) in G. By dropping the length-0 self and reset arcs, replacing auxiliary arcs (u_i, v_j) with the underlying arcs (u, v), every walk from s_i to t_j in H maps to a walk from s to t in G with the same distance. Thus $d_H(s_0, t_h) \geq d_G(s, t)$ for all $s, t \in V$.

Towards proving the reverse inequality, we first claim that

$$d_H(s_0, t_h) \le d_G^h(s,t) \text{ for all } s,t \in V. \tag{2}$$

To this end, we prove that

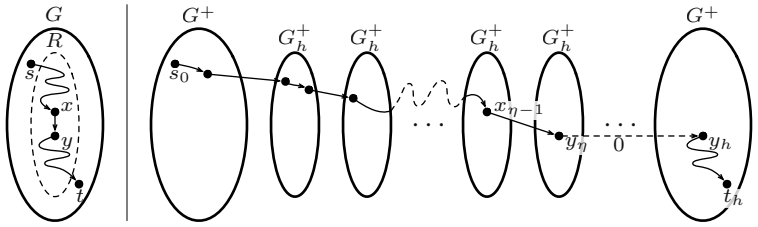
$$d_H(s_0, t_h) \le \hat{d}_G^\eta(s,t) \text{ for all } s,t \in V \text{ and } \eta \le h,$$

by induction on η.

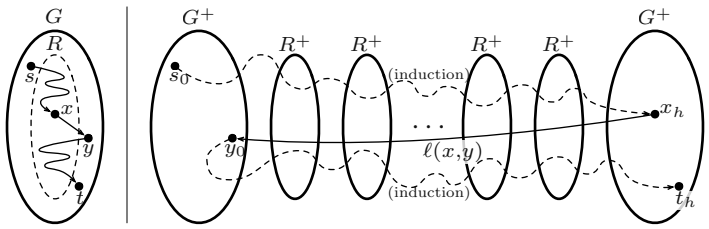
In the base case, suppose we have a 0-hop walk $W : s \rightsquigarrow t$. Let $W' : s_0 \rightsquigarrow t_h$ where we first map W to the identical walk from s_0 to t_0 in the 0th layer, and then take self arcs (t_i, t_{i+1}) from t_0 to t_h. W' has the same length as W.

In the general case, let $W : s \rightsquigarrow t$ be a proper η-hop walk of length $\hat{d}_G^\eta(s,t)$ for $1 \le \eta \le h$. We have two cases depending on whether W stays in G_h all throughout between its first and last hops.

In the first case, suppose W stays in G_h between its first and last hops. Let (x,y) be the last hop in G_h. Let $W_1 : s \rightsquigarrow x$ be the $(\eta-1)$-hop prefix of W up to x and let $W_2 : y \rightsquigarrow t$ denote the suffix beginning at y. Consider the walk $W_1' : s_0 \rightsquigarrow x_{\eta-1}$ in H, where we route W_1 in the natural way: starting from layer 0, we map the nonnegative arcs between the ith and $i+1$th hop to the corresponding nonnegative arcs in the ith layer, and we map the ith hop (u,v) to the corresponding forward arc (u_{i-1}, v_i) from the $(i-1)$th layer to the ith. Let $W_2' : y_h \rightsquigarrow t_h$ be the shortest 0-hop walk from y_h to t_h in the hth layer, which we recall is a copy of G^+. To connect $W_1' : s_0 \rightsquigarrow x_{\eta-1}$ to $W_2' : y_h \rightsquigarrow t_h$, from x_ζ, we take the forward arc $(x_{\eta-1}, y_\eta)$, and then self-arcs (y_i, y_{i+1}) until reaching y_h. All together, we have an (s_0, t_h)-walk W' whose length is identical to W.



In the second case, W leaves G_h between its first and last hop. Let $(x,y) \in \partial^+(G_h)$ be the first arc where W leaves G_h. Let $W_1 : s \rightsquigarrow x$ be the prefix of W up to x and let $W_2 : y \rightsquigarrow t$ denote the suffix beginning at y. Both W_1 and W_2 have less than η hops. By induction, there exist walks $W_1' : s_0 \rightsquigarrow x_h$ and $W_2' : y_0 \rightsquigarrow t_h$ no longer than W_1 and W_2, respectively. We connect W_1' and W_2' by inserting the exit arc (x_h, y_0) between them. Together, W_1', (x_h, y_0), and W_2' gives a 0-hop (s_0, t_h)-walk no longer than W.



This establishes inequality (2). Returning to property (c), fix any pair $s,t \in V$, and let $W : s \rightsquigarrow t$ be the shortest (s,t)-walk. Of the many hops in W, some are from a vertex sampled in U_0 and some are not. Let $u_1, u_2, \ldots, u_\ell \in U_0$ be the sampled vertices visited by W, and divide W into subwalks $W_1 : s \to u_1, W_2 : u_1 \to u_2, \ldots, W_{\ell+1} : u_\ell \to t$ at these points.

Since U_0 samples each negative vertex with probability $O(\log(n)/h)$, with high probability, there are no subsequences of h consecutive hops in W that do not include a vertex in U_0. In particular, each subwalk W_i has at most h-hops.

For each subwalk $W_i : u_i \to u_{i+1}$ (letting $u_0 = s$ and $u_{\ell+1} = t$), invoking the claim for h-hop distances above, let $W_i' : u_{i,0} \rightsquigarrow u_{i+1,h}$ be the walk in H with length at most $\ell(W_i)$. Let $W' : s_0 \rightsquigarrow t_h$ be the walk obtained by connecting the subwalks $W_i' : u_i \to u_{i+1}$ with reset arcs $u_{i+1,h} \to u_{0,h}$ in between. We have $\ell_H(W') = \sum_i \ell_H(W_i') \le \sum_i \ell(W_i) = \ell(W)$, as desired. □

The advantage of layered sparsification over hop reduction is that, by reducing the number of negative edges, we have a materially smaller subproblem amenable to recursion. Applying recursion in this straightforward manner leads to the following.

Theorem 4.2. *The SSSP problem for real-weighted graphs can be solved with high probability in $\tilde{O}\Big(mn^{\sqrt{3}-1}\Big)$ randomized time.*

Proof. More precisely, we prove that a graph with k negative edges and preprocessed as described in Section 3 can be neutralized in $\tilde{O}\Big(mk^{\sqrt{3}-1}\Big)$ randomized time.

The high-level idea combines Lemma 4.1 with recursion. Let $h = \tilde{O}(k^{\frac{2\sqrt{3}-3}{3}})$. The algorithm iteratively neutralizes subsets of negative edges until there are no negative edges. Each iteration, we invoke Lemma 3.1 with parameters $h_0 = h$. With high probability, we either find and return a negative cycle, directly neutralize $\sqrt{kh}$ negative vertices and repeat, or obtain a valid potential φ and a negative sandwich U of $\Omega\Big(\sqrt{kh}\Big)$ negative vertices with h-hop negative reach n/h in G_φ.

In the latter scenario, we apply Lemma 4.1 to U, obtaining the auxiliary graph H, potentials φ, and mappings π_0, π_1 as described in Lemma 4.1 with high probability. H has $O(m)$ edges and $O(n)$ vertices. We recurse on H_φ to compute potentials $\psi : V_H \to \mathbb{R}$ neutralizing H_φ. We then compute the Johnson potentials $\chi(v) = d_U(V, v)$ for G_U via the neutralized graph $H_{\varphi,\psi}$, in $O(m)$ time, neutralizing U.

Each randomized step succeeds with high probability, and there are polynomially many steps, so the entire algorithm succeeds with high probability.

Let $T(m, k)$ bound the running time on a graph with m edges and k negative edges. $T(m, k)$ is bounded recursively by

$$T(m,k) \le \tilde{O}\Big(\sqrt{k/h}\Big(mh^2 + T\Big(m, \sqrt{k/h}\Big)\Big)\Big),$$

which is satisfied by $T(m, k) = \tilde{O}\Big(mk^{\sqrt{3}-1}\Big)$. For $k = n$ we obtain the desired running time. □

5 Bootstrapping sparse hop reducers

This section revisits and improves the bootstrapped hop reducer from [HJQ26]. The bootstrapping construction from [HJQ26] was described previously in Section 2.1. The high-level idea, given a remote set of negative edges U, is to gradually build out a hop reducer for U working from the η-hop negative reach for small η, and gradually extending η until we have a hop reducer for all of G_U. In principle, for the smallest values η, the subgraph induced by the negative reach is small enough for direct approaches, and then the hop reducer for one value of η can be used to help construct the hop reducer for the next value of η. There are a number of details involved in this high-level idea, and this section will make two important changes to the bootstrapping construction described in [HJQ26].

Let $h, h_0 \in \mathbb{N}$ be parameters, to be determined, with $h > h_0 \geq \Omega(\log n)$. Let U be a set of negative vertices that is $(\eta, \eta/h)$-remote for all $\eta \geq h_0$. The goal of this section is to construct an h-hop reducer for G_U. Dropping any negative vertex outside of U, we assume that $G = G_U$.

The bootstrapping construction builds out hop reducers over the η-hop negative reach of U incrementally from small η all the way out to h. To index these steps, let $L = \lceil \log_2 h \rceil + 1$, $i_0 = \lceil \log_2 h_0 \rceil$, and $i_1 = 2i_0$. For each $i \in \{i_0, \ldots, L-1\}$, let V_i be the 2^i-hop negative reach of U, and let G_i be the subgraph induced by V_i. G_i contains $O(2^i n/h)$ vertices and $O(2^i m/h)$ edges. Let $V_L = V$ and $G_L = G$. For each i, let $d_i(\cdot,\cdot)$ denote distances in G_i. Let $\partial^+(V_i)$ be the directed cut of arcs from V_i to $V \setminus V_i$.

Following the bootstrapping framework of [HJQ26], for each index i successively, we create a 2^{i-2}-hop reducer H_i over the subgraph G_i. We construct the reducers incrementally, starting from H_{i_1+1}, and use distances computed in previous hop reducers to build the next hop reducer. An important calculation motivating the design is that each H_i will have $O(m2^i/h)$ edges, and will be used for $\tilde{O}(|U|/2^i)$ shortest path computations (as explained in Section 5.2). These two factors cancel out nicely so that we end up spending $\tilde{O}(m|U|/h)$ time in each hop reducer H_i, independent of i.

Within this framework we make several changes. First, we do not start the bootstrapping from the $O(1)$-hop negative reach as [HJQ26]. The first hop reducer we construct will be for the larger subgraph G_{i_1}. Ultimately, starting from a larger negative reach will allow us to neutralize larger remote sets U in each iteration, as explained later in Section 6. To account for the missing hop-reducers over the smaller negative-reach subgraphs, we will assume we are given as an additional input "distance estimates" (defined momentarily) for all subgraphs from G_{i_0+1} to G_{i_1}, that suffice to initialize the bootstrap process. Second, we create hop reducers for each subgraph G_i that are substantially sparser than those of [HJQ26], introducing $\tilde{O}\left(|U|^2/2^i\right)$ additional edges for graph G_i instead of $O\left(|U|^2\right)$ additional edges as in [HJQ26]. This construction interacts synergistically with the decision to start the bootstrapping at G_{i_1} instead of G_1, as the 2^i-factor is much larger. The net effect is to improve the running time in sparse graphs.

As alluded to above, the bootstrap construction uses shortest path distances computed in the hop reducers for smaller subgraphs to produce the next hop reducer for a larger subgraph. The distance estimates computed via the smaller reducers will satisfy certain conditions that are sufficient to construct the next hop reducer, which we formalize via the notion of "sparse distance estimates" defined below. Sparse distance estimates replace the (dense) distance estimates from [HJQ26], and for a particular level i, will collectively be a $\tilde{O}(1/2^i)$-factor smaller than the set of dense distance estimates at level i in [HJQ26].

Definition 5.1. *We define* sparse distance estimates *at level j as a set of negative vertices $X_j \subseteq U$, values $\delta_j(u,x)$ and $\delta_j(x,v)$ for $u \in U$, $x \in X_j$, and $v \in \overline{U}$, such that:*

(a) $\delta_j(u,x) \geq \max\left\{d(u,x), d_j^{2^{j-1}}(V_j, x)\right\}$ *for $u \in U$ and $x \in X_j$.*

(b) $\delta_j(x,v) \geq \max\left\{d(x,v), d_j^{2^j}(V_j, v) - d_j^{2^{j-1}}(V_j, x)\right\}$ *for all $x \in X_j$ and $v \in \overline{U}$.*

(c) $\min_{x \in X} \delta_j(u,x) + \delta_j(x,v) \leq \hat{d}_j^{\eta}(u,v)$ *for all $u \in U$, $v \in \overline{U}$, and $\eta \in [2^{j-2}, 2^{j-1}]$.*

The key difference from [HJQ26] is we only have estimates to and from a subset of vertices X_j, rather than estimates for every pair (we will have $|X_j| = \tilde{O}(|U|/2^j)$ in the final algorithm). Essentially, sparse distance estimates break every shortest proper $\Theta(2^{j-1})$-hop (u,v) walk into two 2^{j-1}-hop walks, from u to some x and from x to v, for some $x \in X_j$. This is reflected in property (c).

Properties (a) and (b) are carefully designed to work with a certain (natural) potential function in the construction of the hop reducer, as we explain momentarily.

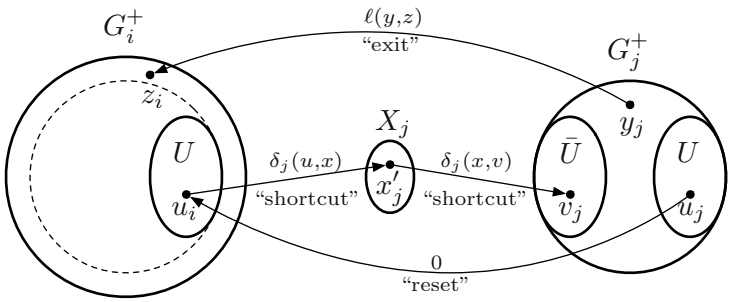
5.1 Constructing hop reducers from sparse distance estimates

Suppose we are at a level $i \in \{i_1 + 1, \ldots, L\}$, and we have sparse distance estimates at all preceding levels $j \in \{i_0, \ldots, i-1\}$ (either given or previously computed). The goal of this section is to use these distance estimates to create a $\Omega(2^i)$-hop reducer H_i for G_i. (The next section will then use H_i to compute the next set of sparse distance estimates for level i.)

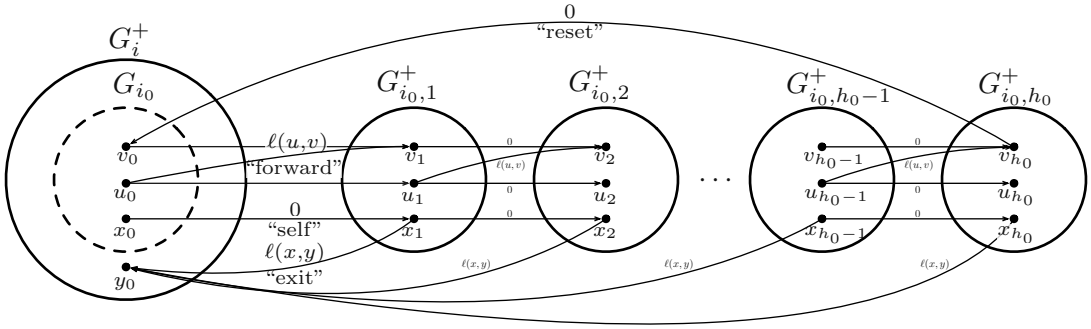
Lemma 5.2. *Let $i \in \{i_1 + 1, \ldots, L\}$, and suppose we have sparse distance estimates (X_j, δ_j) for all $j \in \{i_0, \ldots, i-1\}$. Then one can construct a 2^{i-2}-hop reducer for G_i with $O(2^i n/h)$ vertices and $O\Big(2^i m/h + \sum_{j=i_0+1}^{i-1} |U||X_j|\Big)$ edges.*

Proof. Initially, let $H_i' = G_i^+$. We call this the "base subgraph". For a vertex v, we let v_i denote its copy in the base subgraph.

For each index $j \in \{i_0 + 1, \ldots, i-1\}$, we augment H_i' with the following vertices and arcs that we collectively called the "jth shortcut graph". It was described previously in Section 2.1 and we describe it again here. We first create a disjoint copy of G_j^+. For each vertex $v \in V_j$, let v_j denote its copy in G_j^+. We also add a disjoint copy of X_j, which we denote X_j'. For each $x \in X_j$, we let x_j' denote its copy in X_j'. For $u \in U$ and $x \in X_j$, we add the "shortcut" arc (u, x_j') with length $\ell_{H_i}\big(u, x_j'\big) = \delta_j(u, x)$. For $x \in X_j$ and $v \in \overline{U}$, we add the "shortcut" arc (x_j', v_j) with length $\delta_j(x, v)$. For all $(y, z) \in \partial^+(V_j)$ with $z \in V_i$, we add the "exit" arc (y_j, z_i) with length $\ell(y, z)$. Lastly, for each $u \in U$, we add the "reset" arc (u_j, u_i) with length 0.



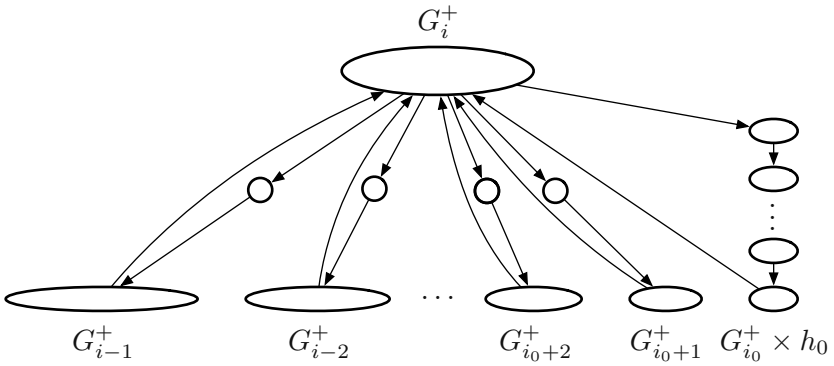
For i_0 we create a different subgraph, because this subgraph must also account for all the missing distance estimates from below level i_0. We essentially create the "layered (auxiliary) graph" from [Fin24], with some adjustments. We create $h_0 = 2^{i_0}$ copies of $G_{i_0}^+$. For $v \in V_{i_0}$, and $\eta \in [h_0]$, let $v_{i_0,\eta}$ denote the copy of v in the ηth copy of $G_{i_0}^+$. For each negative arc (u, v), and $\eta \in [h_0]$, we add the "forward" arc $(u_{i_0,\eta-1}, v_{i_0,\eta})$ with length $\ell_G(u, v)$ (letting $v_{i_0,0} = v_i$). For each arc (x, y) from V_{i_0} to $V_i \setminus V_{i_0}$, we add "exit" arcs $(x_{i_0,\eta}, y_i)$. Lastly, for each $v \in V_{i_0}$, we add "reset" arcs $(v_{i_0,\eta}, v_i)$ of length 0 for all η.



Altogether, H_i' consists of a copy of G_i^+ for the base graph, a shortcut graph for G_j for each $j \in \{i_0+1, \ldots, i-1\}$, and a layered graph for G_{i_0}. H_i' has

$$O\left(m(h_0^2+2^i)/h + \sum_{j=i_0+1}^{i-1} |U||X_j|\right) = O\left(2^i m/h + \sum_{j=i_0+1}^{i-1} |U||X_j|\right)$$

edges, since $2^i \geq 2^{2i_0} \geq h_0^2$. The only negative arcs are the shortcut arcs in the shortcut graphs, the forward arcs in the layered graph, and the reset arcs in the shortcut and layered graphs.



Every edge length in H_i' either matches the length of the underlying edge in G_i, is a length-0 arc between two copies of the same vertex, or (in the case of shortcut arcs) overestimates the distance between the underlying endpoints in G_i. More explicitly, for any two vertices $u, v \in V_i$, and any arc in H_i' from an auxiliary copy of u to an auxiliary copy of v, the length of that arc in H_i' dominates the distances from u to v in G_i. It follows that for all $u, v \in V_i$, the distance from u_i to v_i in H_i' dominates the distance from u to v in G_i.

We define potentials φ over the vertices of H_i' to neutralize the forward and shortcut arcs as follows. First, in the base subgraph, we set $\varphi(v_i) = 0$ for all $v \in V_i$. Next, consider the jth shortcut graph for $j \in \{i_0+1, ..., i-1\}$. For $x \in X_j$, we set $\varphi\left(x_j'\right) = d_j^{2^{j-1}}(V_j, x)$. For $v \in V_j$, we set $\varphi(v_j) = d_j^{2^j}(V_j, v)$. Lastly, in the layered graph for G_{i_0}, we reweight the vertices per the the construction in [Fin24]; namely, $\varphi(v_{i_0,\eta}) = d_{i_0}^{\eta}(V_{i_0}, v)$ for $\eta \in [h_0]$. We claim that φ is valid and neutralizes all the negative edges except for the reset arcs. It is easy to see that φ is valid over each disjoint copy of G_j^+ (for any j).

Consider an exit arc (y_j, z_i) of a jth shortcut graph. Since z is not in the 2^j-hop negative reach of U, we have

$$\ell_{H_i',\varphi}(y_j, z_i) = \varphi(y_j) + \ell(y,z) - \varphi(z_i) = d_j^{2^j}(V_j, y) + \ell(y,z) \geq 0.$$

Similarly, φ maintains the nonnegativity of each exit arc $(x_{i_0,\eta}, y_i)$ in the layered graph because the endpoint y is not in the 2^{i_0}-hop negative reach.

Consider the forward arcs in the jth shortcut graph, for $j \in \{i_0+1, \dots, i-1\}$. For the first type of shortcut arc, (u_i, x'_j) where $u \in U$ and $x \in X_j$, we have

$$\ell_{H'_i,\varphi}\Big(u_i, x'_j\Big) = 0 + \delta_j(u,x) - d_j^{2^{j-1}}(V_j, x) \geq 0$$

by property (a) of sparse distance estimates. For the second type of shortcut arc, (x'_j, v_j) where $x \in X_j$ and $v \in \overline{U}$, we have

$$\ell_{H'_i,\varphi}\Big(x'_j, v_j\Big) = d_j^{2^{j-1}}(V_j, x) + \delta_j(x,v) - d_j^{2^j}(V_j, x) \geq 0$$

by property (b) of sparse distance estimates.

Lastly, consider the forward arcs $(u_{i_0,\eta-1}, v_{i_0,\eta})$ in the layered graph, where $\eta \in [h_0]$. As observed in [Fin24], we have

$$\ell_{H'_i,\varphi}(u_{i_0,\eta-1}, v_{i_0,\eta}) = d_{i_0}^{\eta-1}(V_{i_0}, u) + \ell(u,v) - d_{i_0}^{\eta}(V_{i_0}, v) \geq 0$$

since $d_{i_0}^{\eta-1}(V_{i_0}, u) + \ell(u,v)$ is the length of some η-hop walk in G_{i_0} from V_{i_0} to v.

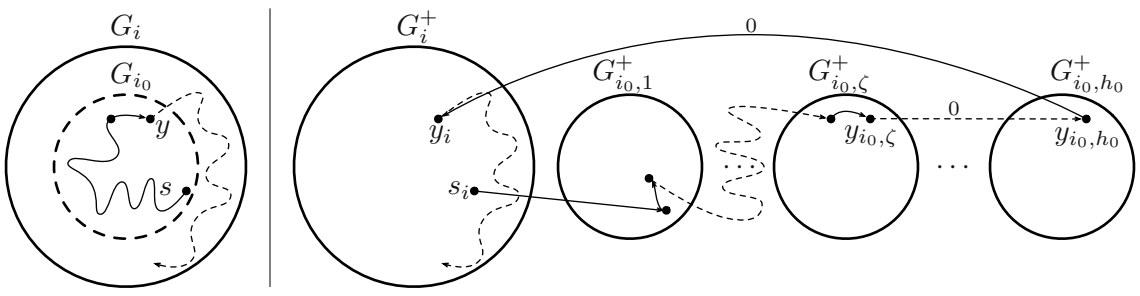
This addresses all of the arcs of H'_i except for the reset arcs in the shortcut and layered graphs. Letting the $H_i = H'_{i,\varphi}$, the reset arcs are the only negative arcs in H_i.

It remains to prove that H_i is a 2^{i-2}-hop reducer for G_i. Let $W : s \rightsquigarrow t$ be a proper η-hop walk in G_i. We claim there is a $\lceil \eta/2^{i-2} \rceil$-hop walk $W' : s_i \rightsquigarrow t_i$ in H_i with length at most the length of W. It suffices to prove the claim for $s \in U$. We prove the claim by induction on the number of hops in W, where $\eta = 0$ hops is immediate.

For each index j, let W_j be the maximal 2^{j-1}-hop prefix of W, and let $\overline{W}_j$ be the remaining suffix. We have two cases.

Case 1. Suppose W_j is contained in G_j for all $j \in \{i_0, \dots, i-1\}$. Let $j = i_0$ if $\eta \leq 2^{i_0-1}$, $j = i-1$ if $\eta \geq 2^{i-1}$, and $j = \lceil \log_2 \eta \rceil$ otherwise. Let $y \in V_j$ be the last vertex on W_j. We will map W_j to a 1-hop walk $W'_j : s_i \rightsquigarrow y_i$ in H_i via the G_j^+-gadget, using the reset edge to return to the base graph. We have two subcases depending on whether $j = i_0$.

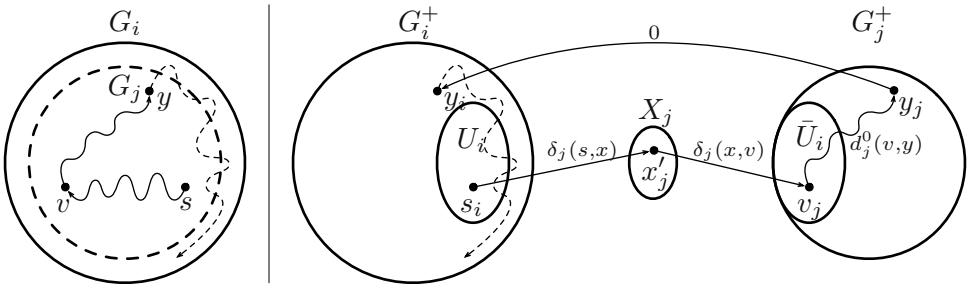
Case 1.a. Suppose $j = i_0$ and all of W_j is contained in G_{i_0}. We will use the layered graph to embed W_j. Let ζ be the number of hops in W_j. Let $W'_j : s_i \rightsquigarrow y_i$ be the walk in H_i that first follows the corresponding auxiliary edges from s_i to y_ζ in the layered graph, then takes self-arcs from y_ζ to y_{h_0} before taking the reset arc to y_0. More explicitly, the γth hop (u,v) in W_j is mapped to the forward arc $(u_{i_0,\gamma-1}, v_{i_0,\gamma})$ between the $(\gamma-1)$th and γth layers (or $(u_i, v_{i_0,\gamma})$ for $\gamma = 1$), and the subwalk of nonnegative arcs between the $(\gamma-1)$th and γth hop are mapped to the corresponding walk in the γth layer $G^+_{i_0,\gamma}$ (or G_i for $\gamma = 1$). W'_j has 1 hop, from the last reset arc (y_{h_0}, y_0), and the same length as W_j.



Case 1.b. Now suppose $j > i_0$. We embed W_j using the jth shortcut graph instead. Let $v \in \overline{U}$ be the head of the last hop in W_j and split W_j at v into $W_{j,1} : s \rightsquigarrow v$ and $W_{j,2} : v \rightsquigarrow y$. Since $W_{j,1}$ has between 2^{j-2} and 2^{j-1} hops, by property (c), there is a vertex $x \in X_j$ such that $\delta_j(s,x) + \delta_j(x,v) \le \ell(W_{j,1})$. The walk W'_j from s_i, to x'_j, to v_j, followed by the shortest walk in G_j^+ from v_j to y_j, and ending with the reset arc to y_i, has total length

$$\ell_{H_i}\left(W'_j\right) = \ell_{H'_i}\left(W'_j\right) = \delta_j(s,x) + \delta_j(x,v) + d^0(v,y) \le \ell(W_{j,1}) + \ell(W_{j,2}) \le \ell(W_j),$$

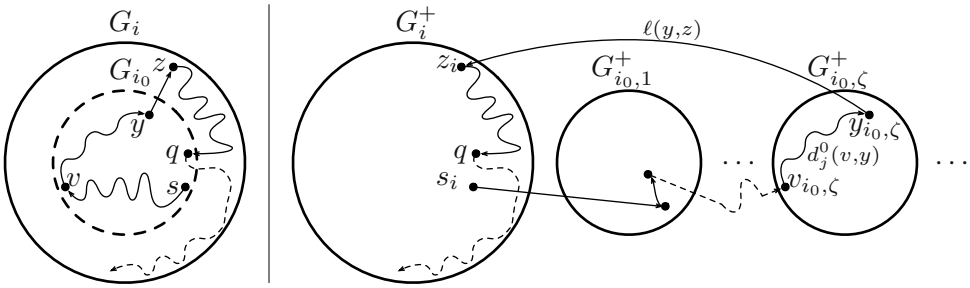
and one hop.



In either case, we have a 1-hop (s_i, y_i)-walk W'_j with $\ell_{H_i}\left(W'_j\right) \le \ell_{G_i}(W_j)$. Meanwhile, either $W_j = W$; or $j = i-1$, $y \in U$, and $\overline{W}_j : y \rightsquigarrow t$ is a $(\eta - 2^{i-2})$-hop walk in G_i. In the former, W'_j is a 1-hop (s_i, t_i)-walk in H_i with length at most the length of $W_j = W$, as desired. In the latter, by induction on the number of hops, there is a $(\lceil \eta/2^{i-2} \rceil - 1)$-hop walk $\overline{W}'_j$ from y_i to t_i with length $\ell_{H_i}\left(\overline{W}'_j\right) \le \ell_{G_i}\left(\overline{W}_j\right)$. Concatenated together, W'_j and $\overline{W}'_j$ gives the desired $\lceil \eta/2^{i-2} \rceil$-hop (s_i, t_i)-walk.

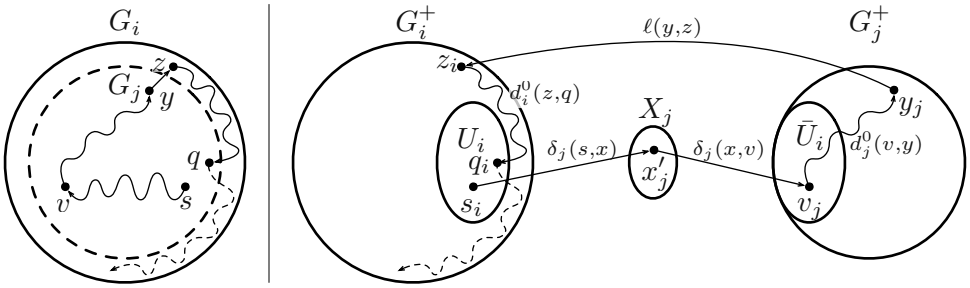
Case 2. Suppose W_j is not contained in G_j for some $j \in \{i_0, \dots, i-1\}$. Let j be the first such index. Let $(y,z) \in \partial^+(V_j)$ be the arc where W_j first steps out of V_j. Let q be the first negative vertex after (y,z) if one exists; otherwise let $q = t$. Let $W_q : s \rightsquigarrow q$ be the prefix of W up to q and let $\overline{W}_q : q \rightsquigarrow t$ be the remaining suffix. Let ζ be the number of hops in W_q. Again we have two subcases depending on whether $j = i_0$.

Case 2.a. Suppose $j = i_0$. We first embed the prefix of W_q up to y through the layered graph similarly to case 1: beginning from s_i, the hops in W_j map to forward arcs from one layer to the next, and the subwalks of nonnegative arcs between arcs are retraced within the appropriate layer, until reaching $y_{i_0,\zeta}$ in the ζth layer. We then take the exit arc $(y_{i_0,\zeta}, z_i)$ and retrace the remaining suffix of W_q through the base subgraph G_i^+ until ending at q_i. Letting $W'_q : s_i \rightsquigarrow q_i$ denote this walk, W'_q has 0 hops, and the same length as W_q.



Case 2.b. Suppose instead that $j > i_0$. Let v be the head of the final hop in W_q. Let $W_{q,1}$ be the prefix of W_q from s to v, $W_{q,2}$ be the subpath from v to y, and $W_{q,3}$ be the suffix of W_q from z to q. Since W_{j-1} is contained in G_{j-1}, $\zeta \in [2^{j-2}, 2^{j-1}]$. By property (c), $\delta_j(s,x) + \delta_j(x,v) \le \ell(W_{q_1})$ for some $x \in X_j$. Let $W'_{q,1}$ be the two-edge walk from s_i to v_j via x'_j. Let W'_{q_2} be the shortest walk from v_j to y_j through G_j^+, and let $W_{q,3}$ be the shortest walk from z_i to q_i in the base subgraph G_i^+. Concatenating $W'_{q,1}$, $W'_{q,2}$, the exit arc (y_j, z_i), and $W'_{q,3}$ gives a 0-hop (s_i, q_i)-walk W'_q with length

$$\begin{aligned}\ell\Big(W'_q\Big) &= \delta_j(s,x) + \delta_j(x,v) + d^0_j(v,y) + \ell(y,z) + d^0(z,q)\\ &\le \ell(W_{q_1}) + \ell(W_{q,2}) + \ell(y,z) + \ell(W_{q,3}) = \ell(W_q).\end{aligned}$$



Thus, whether or not $j = i_0$, we have a 0-hop (s_i, q_i)-walk W'_q with length equal to W_q. Meanwhile, either $\overline{W}_q$ is empty, or $q \in U$ and W_q is a $(\eta - \zeta)$-hop walk. Either trivially in the former case or by induction in the latter, there is a $\lceil(\eta-\zeta)/2^{i-2}\rceil$-hop walk $\overline{W}'_q : q_i \rightsquigarrow t_i$ in H_i with $\ell_{H_i}\Big(\overline{W}'_q\Big) \le \ell_{G_i}\Big(\overline{W}_q\Big)$. Together, W'_q and $\overline{W}'_q$ give the desired $\lceil\eta/2^{i-2}\rceil$-hop walk in H_i.

Between the two cases, we conclude that H_i is a 2^{i-2}-hop reducer for G_i. □

5.2 Computing sparse distance estimates from hop reducers

We now turn to the other half of the construction, where the hop reducer H_i is used to construct sparse distance estimates at level i. We first describe the subroutine at a high-level.

The initial idea comes from [HJQ26]. Recall that the sparse distance estimates at level i only need to compete with proper walks with $\Theta(2^i)$-hops (property (c)). Consider the shortest such walks. Since each such walk has at least $\Omega(2^j)$ negative edges, a uniformly random sample of negative edges of size $O(|U|\log(n)/2^j)$ will hit all of these walks with high probability. If we compute $\Omega(2^i)$-hop distances in H_i to and from all the sampled vertices, then the distances induced by going through the sampled vertices will be at least as good as any of these shortest walks.

The only catch is that the distances we compute in H_i may be much better than the actual proper $\Theta(2^i)$-hop distances in G_i because η hops in H_i can capture arbitrarily more than $2^{i-2}\eta$ hops in G_i. (Too small of a distance estimate ultimately prevents us from neutralizing the shortcut graphs constructed in Section 5.1 while preserving the nonnegativity of the exit arcs.) Thus in a second step we manually detect and increase overly optimistic distances to satisfy properties (a) and (b).

Lemma 5.3. *Given a 2^{i-2}-hop reducer H_i for G_i with m_i edges and n_i vertices, one can compute level i sparse distance estimates (X_i, δ_i) with $|X_i| \le O(|U|\log(n)/2^i)$ with high probability in $O(|U|\mu_i \log(n)/2^i + 2^{2i}\mu/h)$ time, where $\mu_i = m_i + n_i \log n_i$.*

Proof. Let $X_i \subseteq U$ sample $O(|U|\log(n)/2^i)$ negative vertices uniformly at random. Let

$$\delta_i(u,x) = \max\Big\{d^2_{H_i}(u,x), d_i^{2^{i-1}}(V_i,x)\Big\}$$

for $u \in U$ and $x \in X_i$ and let

$$\delta_i(x,v) = \max\Big\{d^2_{H_i}(x,v), d_i^{2^i}(V_i,v) - d_i^{2^{i-1}}(V_i,x)\Big\}$$

for $x \in X_i$ and $v \in \overline{U}$. Because H_i is a 2^{i-2}-hop reducer, we have

$$d_i(u,x) \le d^2_{H_i}(u,x) \le d_i^{2^{i-1}}(u,x) \text{ and } d_i(x,v) \le d^2_{H_i}(x,v) \le d_i^{2^{i-1}}(x,v).$$

To verify property (a), let $u \in U$ and $x \in X_j$. We have $\delta_i(u,x) \ge d(u,x)$ because distances in H_i always overestimate true distances in G_i, as desired. Similarly, property (b) follows from H_i always overestimating true distances in G_i.

Now consider property (c). Let $u \in U, v \in \overline{U}$, and let W be a shortest proper η-hop walk from u to v with $2^{i-2} \le \eta \le 2^{i-1}$ hops. X_i samples at least vertex x visited by W with high probability. For such a vertex x, we have

$$\delta_i(u,x) = \max\Big\{d^2_{H_i}(u,x), d_i^{2^{i-1}}(V_i,x)\Big\} \le d^{2^i-1}(u,x)$$

because H_i is a 2^{i-2}-hop reducer. Next we bound $\delta_i(x,v)$. We first have

$$d^2_{H_i}(x,v) \le d_i^{2^{i-1}}(x,v)$$

because H_i is a 2^{i-2}-hop reducer. We also have

$$d_i^{2^i}(V_i,v) \le d_i^{2^{i-1}}(V_i,x) + d_i^{2^{i-1}}(x,v)$$

by the triangle inequality. Thus

$$\delta_i(x,v) = \max\Big\{d^2_{H_i}(x,v), d_i^{2^i}(V_i,v) - d^{2^{i-1}}(V_i,x)\Big\} \le d^{2^{i-1}}(u,x).$$

Altogether we have

$$\delta_i(u,x) + \delta_i(x,v) \le d_i^{2^{i-1}}(u,x) + d_i^{2^{i-1}}(x,v) \le \ell(W) = d_i^{2^{i-1}}(u,v),$$

as desired. Taking the union bound over all u, v, and η, we conclude that (X_i, δ_i) are level i sparse distance estimates satisfying properties (a)–(c) with high probability.

As for the running time, computing $d_i^{2^{i-1}}(V_i,\cdot)$ and $d_i^{2^i}(V_i,\cdot)$ in G_i takes $O(2^{2i}\mu/h)$ time. Computing shortest 2-hop paths to and from each $x \in X_i$ in H_i takes $O(|X_i|\mu_i) = O(|U|\mu_i \log(n)/2^i)$ time. Taking these distance computations and setting the estimates $\delta_i(\cdot,\cdot)$ takes $O(|U||X_i|) = O(|U|^2 \log(n)/2^i)$ time. The bottleneck is the $O(|U|\mu_i \log(n)/2^i)$ time computing $O(1)$-hop distances in H_i for each vertex in X_i. □

5.3 Bootstrapping it all together

We conclude this section by constructing an $O(h)$-hop reducer for G. As in [HJQ26], we alternate between constructing hop reducers from distance estimates by Lemma 5.2 and computing distance estimates from Lemma 5.3 to build hop reducers, with the negative reach expanding iteration by iteration. The main difference here is we that we take as input distance estimates for levels $i_0 + 1$ to i_1 to initialize the bootstrapping process to begin from G_{i_1}, rather than from G_1.

Lemma 5.4. *Let U be a set of negative vertices that can negatively reach $O(n\eta/h)$ vertices for all $\eta \geq h_0$. Suppose we are given as input sparse distance estimates (δ_j, X_j) at level j for all $j \in \{i_0+1, \ldots, i_1\}$. Let $\Delta = \sum_{j=i_0+1}^{i_1} |X_j|$. Then one can compute an $(h/2)$-hop reducer for G_U with $O(m + |U|^2 \log(n)/h_0)$ edges and $O(n)$ vertices with high probability in $O\Big(h\mu + |U|\mu \log^2(n)/h + |U|^2 \Delta \log(n)/h_0^2 + |U|^3 \log^2(n)/h_0^4\Big)$ randomized time.*

Proof. We first apply Lemma 5.2 to construct a (2^{i_1-1})-hop reducer H_{i_1+1} for G_{i_1+1}. We then follow the bootstrapping approach in [HJQ26]: for all i from i_1+1 to $L-1$, we apply Lemma 5.3 with H_i to calculate sparse distance estimates (X_i, δ_i) at level i, and then apply Lemma 5.2 to construct a (2^{i-1})-hop reducer for G_{i+1}. At the end of the process, we have the desired (2^{L-2})-hop reducer for $G_L = G$.

We now calculate the runtime. The time spent constructing each reducer, by Lemma 5.2, is

$$O\left(\sum_{i=i_1+1}^{L} |H_i| + \frac{2^{2i}\mu}{h}\right) = O\left(h\mu + \sum_{i=i_1+1}^{L} |H_i|\right).$$

Note that the second term is negligible as it will be dominated by any single shortest path computation done in H_i. Let m_i and n_i denote the number of edges and vertices in H_i, and let $\mu_i = m_i + n_i \log n_i$. By Lemma 5.3, for $j \geq i_1 + 1$, we have $|X_j| \leq O(|U| \log n / 2^j)$. Therefore,

$$\begin{aligned}
\mu_i &= O\left(\frac{2^i\mu}{h} + \sum_{j=i_0+1}^{i-1} |U||X_j|\right) = O\left(\frac{2^i\mu}{h} + \sum_{j=i_0+1}^{i_1} |U||X_j| + \sum_{j=i_1+1}^{i-1} |U||X_j|\right) \\
&= O\left(\frac{2^i\mu}{h} + |U|\Delta + \sum_{j=i_1+1}^{i-1} \frac{|U|^2 \log n}{2^i}\right) = O\left(\frac{2^i\mu}{h} + |U|\Delta + \frac{|U|^2 \log n}{h_0^2}\right).
\end{aligned}$$

The time to compute the required distance estimates is

$$\begin{aligned}
O\left(\sum_{i=i_1+1}^{L-1} \frac{|U|\mu_i \log(n)}{2^i}\right) &= O\left(\sum_{i=i_1+1}^{L=1} \frac{2^i |U|\mu \log n}{2^i h} + \frac{|U|^2 \Delta \log n}{2^i} + \frac{|U|^3 \log^2 n}{2^i h_0^2}\right) \\
&= O\left(\frac{|U|\mu \log^2 n}{h} + \frac{|U|^2 \Delta \log n}{h_0^2} + \frac{|U|^3 \log^2 n}{h_0^4}\right),
\end{aligned}$$

as desired. □

6 Putting it all together

This section presents the final algorithm and proves Theorem 1.1. Let us take stock of the components developed so far. Lemma 3.1 allows us to extract a remote set U of negative vertices, and Lemma 5.4 constructs an $O(h)$-hop reducer for G_U. However, Lemma 5.4 also takes as input distance estimates at levels i_0+1 through i_1 to initialize the bootstrap process. Here we bring in the ideas of Section 4 to complete the construction. We use layered sparsification and recursion to neutralize G_{i_1}, and then use the neutralized G_{i_1} to generate the missing distance estimates.

Our algorithm is actually divided into two regimes based on the graph density. Let $m_0 = \tilde{\Theta}\Big(n^{(33-7\sqrt{17})/4}\Big)$. When $m \geq m_0$, then the algorithm will basically follow the description above. When $m \leq m_0$, we add a preprocessing step that improves the running time slightly.

We start with the algorithm and analysis in the non-sparse regime where $m \geq m_0$.

Theorem 6.1. *For $m \geq m_0$, single-source shortest paths with real-valued weights can be computed with high probability in $\tilde{O}\left(mn^{(7-\sqrt{17})/4}\right)$ randomized time.*

Proof. More precisely, we prove that a graph with m edges and k negative vertices can be neutralized with high probability in $\tilde{O}\left(\mu k^{\frac{7-\sqrt{17}}{4}} + k^{10-2\sqrt{17}}\right)$ randomized time (including when $m \leq m_0$). The claimed running time follows from taking $k = n$, running Dijkstra's algorithm in the neutralized graph and observing that the first term dominates the second when $m \geq m_0$.

Recall that the algorithm iteratively neutralizes negative vertices until no negative edges remain. Consider an iteration starting with k negative edges. Let h and h_0 be parameters to be determined with $\sqrt{k} \geq h \geq h_0 \geq \Omega(\log n)$, and h_0 a power of 2. Our goal in this iteration is to either neutralize $\Omega(\sqrt{kh_0})$ negative edges or return a negative cycle.

We first apply Lemma 3.1, and either (a) directly neutralize $\Omega(\sqrt{kh_0})$ (and end the iteration), (b) find a negative cycle (and return), or (c) obtain a set U of $\Omega(\sqrt{kh_0})$ negative vertices and a valid potential φ_1, such that for all $\eta \geq h_0$, U can η-hop negatively reach at most $n\eta/h$ vertices in the reweighted graph G_{φ_1}. Suppose the iteration continues in case (c). The goal for this iteration shifts to neutralizing U. For ease of notation, let G denote G_{U,φ_1} for the rest of the iteration.

Let $i_0 = \log_2(h_0)$ and $i_1 = 2i_0$. We will neutralize U by building a $O(h)$-hop reducer via Lemma 5.4, and then compute Johnson's potentials neutralizing U in this hop reducer. To this end we must first compute distance estimates at level i for all $i \in \{i_0 + 1, \ldots, i_1\}$.

For $i \in \{i_0, \ldots, i_1\}$, let $G_i = (V_i, E_i)$ be the subgraph induced by the 2^i-hop negative reach of U. G_{i_0} has $O(mh_0/n)$ edges and $O(nh_0/h)$ vertices, and G_{i_1} has $O(mh_0^2/h)$ edges and $O(nh_0^2/h)$ vertices.

We neutralize G_{i_1} with layered sparsification and recursion. We apply Lemma 4.1 to G_{i_1}, using the h_0-hop negative reach G_{i_0} as layers, which returns a graph $H = (V_H, E_H)$, potentials $\varphi : V_H \to \mathbb{R}$, and mappings $\pi_1, \pi_2 : V_{i_1} \to V_H$ as described in Lemma 4.1. We then recurse on H_φ to compute potentials ψ neutralizing H_φ (with high probability), and then use $H_{\varphi,\psi}$ to compute Johnson potentials $\chi : V_{i_1} \to \mathbb{R}$ neutralizing G_{i_1}.

The potentials χ also neutralize G_i for all $i \leq i_1$. For each $i \in \{i_0 + 1, ..., i_1\}$, we use the neutralized graph $G_{i,\chi}$ to produce distance estimates at level i by essentially the same techniques as in Section 5.2. There are multiple ways to go about this. One way is to repeat the steps of the proof of Lemma 5.3, using $G_{i,\chi}$ instead of a hop reducer to compute the distance estimates.

Alternatively, to apply Lemma 5.3 directly, we can create a hop reducer H_i for G_i where we initially start with disjoint copies of G_i^+ and $G_{i,\chi}$. For each vertex v, let v' denotes its copy in G_i^+ and v'' its copy in $G_{i,\chi}$. For each vertex v, we add an arc (v', v'') of weight $-\chi(v)$ and an arc (v'', v') of weight $\chi(v)$. By translating χ to be nonnegative, we can assume that the edges from $G_{i,\chi}$ to G_i^+ are nonnegative, leaving only the edges from G_i^+ to $G_{i,\chi}$ as possibly negative.

It is easy to see that H_i is an n-hop reducer. A 0-hop (s,t)-walk W maps directly to the same 0-hop (s',t')-walk in G_i^+. An (s,t)-walk W with at least one hop in G_i can be embedded as a one-hop (s',t')-walk in H_i of the same length by taking one hop from s' to s'', retracing W with 0 hops in $G_{i,\chi}$, and taking the nonnegative edge back from t'' to t'. Now, applying Lemma 5.3 to H_i gives us the desired distance estimates (X_i, δ_i) at level i.

We submit the distance estimates $\{(X_i, \delta_i) : i = i_0 + 1, \ldots, i_1\}$ to Lemma 5.4, returning a $O(h)$-hop reducer H. We use H to compute Johnson potentials for G, neutralizing U.

This describes a single iteration of the algorithm. Each randomized subroutine succeeds with high probability, hence each iteration also succeeds with high probability. There are at most $\tilde{O}\left(\sqrt{k/h_0}\right)$ iterations, so all the iterations also succeed with high probability. Altogether the algorithm successfully either neutralizes the graph, or returns a negative cycle, with high probability.

Running time analysis. It remains to bound the running time. Let $T(m, n, k)$ denote the running time on a graph with m edges, n vertices, and k negative edges. Consider a single iteration. The first step, applying Lemma 3.1 to extract the remote set U, takes

$$O\left(h\mu \log^2 n + h^2\sqrt{k/h_0^3}\log^2 n\right)$$

time. Next, sparsifying G_{i_0} with Lemma 4.1 and recursing on the sparsified graph H_i, takes

$$T(O(m_i), O(n_i), O(|U|\log(n)/h_0)) = O\left(T\left(mh_0^2/h, nh_0^2/h, \sqrt{k/h_0}\log(n)\right)\right)$$

time. Let $\mu_i = m_i + n_i \log n_i$ for $i \in \{i_0 + 1, \ldots, i_1\}$. It takes $O(\mu_{i_1})$ time to use H_{i_1} to compute the Johnson potentials χ neutralizing G_{i_1}. Next we compute distance estimates for each i with Lemma 5.3. For each i, this takes

$$O\left(\frac{|U|\mu_i \log(n)}{2^i} + \frac{2^{2i}\mu}{h}\right) = O\left(\frac{|U|\mu\log(n)}{h} + \frac{2^{2i}\mu}{h}\right)$$

time for each i. Thus we spend

$$O\left(\sum_{i=i_0+1}^{i_1} \frac{|U|\mu\log(n)}{h} + \frac{2^{2i}\mu}{h}\right) = O\left(\frac{|U|\mu\log^2(n)}{h} + h\mu\right)$$

computing distance estimates for levels $i \in \{i_0 + 1, \ldots, i_1\}$.

The next step uses the distance estimates to construct a $\Omega(h)$-hop reducer with Lemma 5.4. By Lemma 5.4, noting that $\Delta = \sum_{i=i_0+1}^{i_1} |X_i| = O(|U|\log(n)/h_0)$, it takes

$$O\left(h\mu + \frac{|U|\mu\log^2 n}{h} + \frac{|U|^3\log^2(n)}{h_0^3}\right)$$

time to construct the $\Omega(h)$ hop reducer H.

Lastly we calculate the Johnson potentials with a $(|U|/h)$-hop distance computation in H. Since H has $O\left(m + |U|^2\log(n)/h_0\right)$ edges and $O(n)$, this takes

$$O\left(\frac{|U|^3\log(n)}{hh_0} + \frac{|U|\mu}{h}\right)$$

time. Putting everything together, a single iteration with $|U| = O\left(\sqrt{kh}\right)$ remote vertices takes

$$\begin{aligned}
&\tilde{O}\left(hm + \frac{|U|m}{h} + \frac{|U|^3}{h_0^3} + T\left(\frac{mh_0^2}{h}, \frac{nh_0^2}{h}, \frac{|U|}{h_0}\right)\right) \\
&= \tilde{O}\left(hm + \frac{\sqrt{kh_0}m}{h} + \frac{k^{1.5}}{h_0^{1.5}} + T\left(\frac{mh_0^2}{h}, \frac{nh_0^2}{h}, \sqrt{\frac{k}{h_0}}\right)\right)
\end{aligned}$$

time. Altogether, $T(m, n, k)$ satisfies the following recursion.

$$T(m, n, k) = \tilde{O}\left(\sqrt{\frac{k}{h_0}}\left(hm + \frac{\sqrt{kh_0}m}{h} + \frac{k^{1.5}}{h_0^{1.5}} + T\left(\frac{mh_0^2}{h}, \frac{nh_0^2}{h}, \sqrt{\frac{k}{h_0}}\right)\right)\right).$$

For $h = \Theta\left(k^{\frac{\sqrt{17}-3}{4}}\right)$ and $h_0 = \Theta\left(k^{\sqrt{17}-4}\right)$, this recursion is satisfied by

$$T(m,n,k) = \tilde{O}\left(mk^{\frac{7-\sqrt{17}}{4}} + k^{10-2\sqrt{17}}\right),$$

as desired. □

In the very sparse regime where $m \leq m_0$, the algorithm above would give an $\tilde{O}\left(n^{10-2\sqrt{17}}\right)$ randomized running time, where $10 - 2\sqrt{17} \leq 1.75379$. The following algorithm, which adds a preprocessing step to the previous algorithm, improves on this bound for the very sparse regime.

Theorem 6.2. *For $m \leq m_0$, single-source shortest paths with real-valued edge lengths can be computed with high probability in $\tilde{O}\left((mn)^{(66-2\sqrt{17})/67}\right)$ randomized time.*

Proof. Let k denote the number of negative vertices in G. The proof of Theorem 6.1 actually gives an algorithm that runs in $\tilde{O}\left(mk^{\frac{7-\sqrt{17}}{4}} + k^{10-2\sqrt{17}}\right)$ time, for all m. When $m < k^{\frac{33-7\sqrt{17}}{5}}$, the second term becomes the bottleneck. In this regime, a slight increase in m in exchange for a slight decrease in k lowers the overall running time. We will construct a new graph H preserving distances in G, that decreases the number of negative edges k at the cost of increasing the total number of edges m. The ratio of edges to negative edges in H will be exactly the threshold delineating the sparse regime.

We construct such an H with layered sparsification. For a parameter h to be determined, we apply layered sparsification to G with h layers of G itself (Lemma 4.1 with $r = 1$) to get H, φ, π_0, and π_1, as described in Lemma 4.1. H has $m_H = \tilde{O}(hm)$ edges and $k_H = \tilde{O}(k/h)$ negative vertices. We then neutralize H with Theorem 6.1 and subsequently neutralize G by computing Johnson's potentials via the neutralized H.

We choose $h = \left(\frac{k^{33-7\sqrt{17}}}{m^4}\right)^{\frac{1}{37-7\sqrt{17}}}$, decreasing k and increasing m just enough to return to the non-sparse regime of Theorem 6.1. This gives a final runtime of

$$\tilde{O}\left(m_H k_H^{\frac{7-\sqrt{17}}{4}}\right) = \tilde{O}\left((mk)^{\frac{66-2\sqrt{17}}{67}}\right),$$

as desired. □

Conclusion. Together, the running times from Theorem 6.1 for $m \geq m_0$ and Theorem 6.2 for $m \leq m_0$ give Theorem 1.1.

We conclude with a few remarks reflecting on the improvement in running time from [HJQ26] to Theorem 1.1. Both algorithms ultimately use the bootstrapping construction and both algorithms are bottlenecked by the $\tilde{O}(m|U|/h)$ time computing distance estimates at each level. The subtle difference is that here we start the bootstrapping from the h_0-hop negative reach, for some h_0 polynomial in k, instead of at the smallest, $O(\log n)$-hop negative reach. In turn, we do not need U to be $O(\eta, \eta/h)$-remote for $\eta < h_0$, and we can increase the size of the remote set returned by Lemma 3.1 substantially from $\sqrt{k \log n}$ to $\sqrt{kh_0}$. The layered sparsification technique introduced in Section 4 gives us a faster recursive algorithm to neutralize the h_0-hop negative reach, allowing us to extend the value of h_0 without this step becoming a bottleneck.

In summary, the algorithm here accelerates [HJQ26] by neutralizing more negative edges by a polynomial, $\sqrt{h_0/\log n}$ factor in each iteration. Starting the bootstrap construction at a moderate h_0-hop negative reach allows the sandwich size to increase by a $\sqrt{h_0}$ factor. The efficiency gained

by layered sparsification allows us to increase h_0. These are the two driving factors in the faster running time.

At the moment, $\tilde{O}(m|U|/h)$ time spent computing distance estimates feels like a natural cost in the bootstrapping approach, and it remains an interesting challenge to overcome this bottleneck.

We also remark that the algorithm here goes beyond [HJQ26] for very sparse graphs. For sparse graphs, the $\tilde{O}\left(m^{4/5}n\right)$ from [HJQ26] is obtained from the $\tilde{O}\left(mn^{4/5}\right)$ algorithm for sufficiently dense graphs by rebalancing internal parameters, but the algorithmic steps are all the same. We go beyond simple reparametrization in Theorem 6.2 by using layered sparsification — increasing the total edges while decreasing the negative edges — before applying the algorithm from the non-sparse regime.

Remark 6.1. We did not try to optimize logarithmic factors in favor of a simpler exposition. If one chooses the parameters m_0, h, and h_0 to also account for logarithmic factors, then letting $m_0 = n^{(33-\sqrt{17})/4}/\log^7 n$, one gets

$$O\left(\mu n^{\frac{7-\sqrt{17}}{4}} \log^5 n\right)$$

randomized time for $m \geq m_0$, and

$$O\left((\mu n)^{\frac{66-2\sqrt{17}}{67}} \log^{\frac{394-16\sqrt{17}}{67}} n\right)$$

randomized time for $m \leq m_0$.

References

[Bel58] Richard Bellman. "On A Routing Problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90.

[BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. "Negative-Weight Single-Source Shortest Paths in Near-linear Time". In: *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*. IEEE, 2022, pp. 600–611.

[CKL+25] 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: *J. ACM* 72.3 (2025), 19:1–19:103. Preliminary version in FOCS, 2022.

[DI17] Yefim Dinitz and Rotem Itzhak. "Hybrid Bellman-Ford-Dijkstra algorithm". In: *J. Discrete Algorithms* 42 (2017), pp. 35–44.

[Dij59] Edsger W. Dijkstra. "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1 (1959), pp. 269–271.

[DMM+25] Ran Duan, Jiayi Mao, Xiao Mao, Xinkai Shu, and Longhui Yin. "Breaking the Sorting Barrier for Directed Single-Source Shortest Paths". In: *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*. ACM, 2025, pp. 36–44.

[Fin24] Jeremy T. Fineman. "Single-Source Shortest Paths with Negative Real Weights in $\tilde{O}(mn^{8/9})$ Time". In: *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*. ACM, 2024, pp. 3–14.

[For56] Lester R. Ford. *Network Flow Theory.* Santa Monica, CA: RAND Corporation, 1956.

[FT87] Michael L. Fredman and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms". In: *J. ACM* 34.3 (1987), pp. 596–615.

[Gol95] Andrew V. Goldberg. "Scaling Algorithms for the Shortest Paths Problem". In: *SIAM J. Comput.* 24.3 (1995), pp. 494–504.

[HJQ25] Yufan Huang, Peter Jin, and Kent Quanrud. "Faster single-source shortest paths with negative real weights via proper hop distance". In: *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*. SIAM, 2025, pp. 5239–5244.

[HJQ26] Yufan Huang, Peter Jin, and Kent Quanrud. "Faster negative length shortest paths by bootstrapping hop reducers". In: *Proceedings of the 2026 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2026, Vancouver, Canada, January 11-14, 2026*. 2026. Preprint available at `https://arxiv.org/abs/2506.00428`.

[Joh77] Donald B. Johnson. "Efficient Algorithms for Shortest Paths in Sparse Networks". In: *J. ACM* 24.1 (1977), pp. 1–13.

[LLRZ25] George Z. Li, Jason Li, Satish Rao, and Junkai Zhang. *Shortcutting for Negative-Weight Shortest Path*. 2025. arXiv: `2511.12714 [cs.DS]`.

[Moo59] Edward F. Moore. "The shortest path through a maze". In: *Proceedings of an International Symposium on the Theory of Switching 1957, Part II*. Cambridge, Massachusetts: Harvard University Press, 1959.

[QT25] Kent Quanrud and Navid Tajkhorshid. *From Hop Reduction to Sparsification for Negative Length Shortest Paths*. Nov. 2025. arXiv: `2511.18253v1 [cs.DS]`.

[Shi55] Alfonso Shimbel. "Structure in communication nets". In: *Proceedings of the Symposium on Information Networks*. New York, New York: Polytechnic Press of the Polytechnic Institute of Brooklyn, 1955, pp. 199–203.

A Faster betweenness reduction and updated running times

This section integrates the improved betweenness reduction subroutine of [LLRZ25], calculates the implied running times for the two algorithms described above, and describes a third algorithm that better leverages the new subroutine.

A.1 Betweenness reduction

"Betweenness reduction" is a preprocessing step introduced by Fineman [Fin24]. For three vertices $s, t, v \in V$ and $h \in \mathbb{Z}_{\geq 0}$, we say that v is *(weakly) h-hop negatively between* s and t if

$$d^h(s, v) + d^h(v, t) < 0.$$

For parameters $b, h \in \mathbb{N}$, [Fin24] gave a subroutine computing a valid potential φ such that with high probability, for any two vertices s, t, at most n/h vertices are negatively between s and t in the reweighted graph G_φ. [Fin24] algorithm ran in $\tilde{O}(mbh + b^2 n)$ randomized time and the $b^2 n$ term was essentially removed in follow up work [HJQ26]. Betweenness reduction is a critical step for extracting a remote set. This article, which is primarily focused on how to neutralize a remote set of edges, and not on how to obtain a remote set of edges, hides the betweenness reduction step in

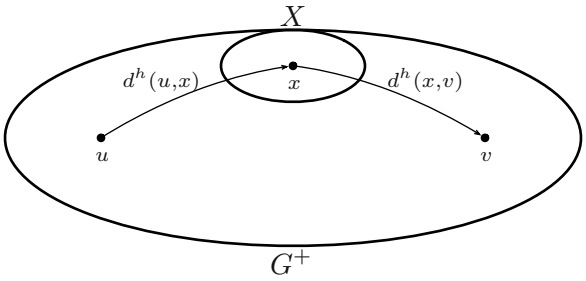
Lemma 3.1. The running time in Lemma 3.1 directly reflects the running time of the betweenness reduction step, and betweenness reduction is a bottleneck for all the algorithms in this article.

Li, Li, Rao, and Zhang [LLRZ25] obtained the following improved bounds for betweenness reduction.

Lemma A.1 ([LLRZ25, Lemma 11])**.** *Let $b, h \in \mathbb{N}$. One can compute a valid potential φ so that any two vertices have at most n/b vertices h-hop negatively between them in G_φ, in running time bounded by $O((m + n \log n)h)$ plus the time to neutralize a real-weighted graph with $O(mh)$ edges and $O(b \log n)$ negative edges.*

Proof sketch. We give a sketch of the proof that highlights the main ideas, as it is relatively simple. We refer to [LLRZ25] for full details and analysis. The new subroutine can be interpreted as simulating the approach in [Fin24] with two key ideas that improve the efficiency.

We first sketch the $\tilde{O}(mbh + nb^2)$-time betweenness reduction algorithm from [Fin24]. Let X be a set of $\tilde{O}(b)$ vertices uniformly at random. We construct an auxiliary graph H, beginning with G^+. For all $x \in X$ and $v \in V$, we add "shortcut" arcs (x, v) and (v, x) of length $d^h(x, v)$ and $d^h(v, x)$, respectively. (These edge lengths can be computed via an h-hop distance computation to and from x in G, for each $x \in X$, in $\tilde{O}(mbh)$ time.) The only negative arcs in H are the shortcut arcs.



Suppose we compute Johnson's potentials φ neutralizing H. (Only $O(|X|)$ hops are needed to compute these potentials, because every negative edge is incident to a vertex in X.) Since H contains G^+, the restriction of φ to the vertices in G^+ is also a valid potential for G. Moreover, because φ neutralizes the shortcut arcs in H, for all $s, t \in V$ and $x \in X$, we have

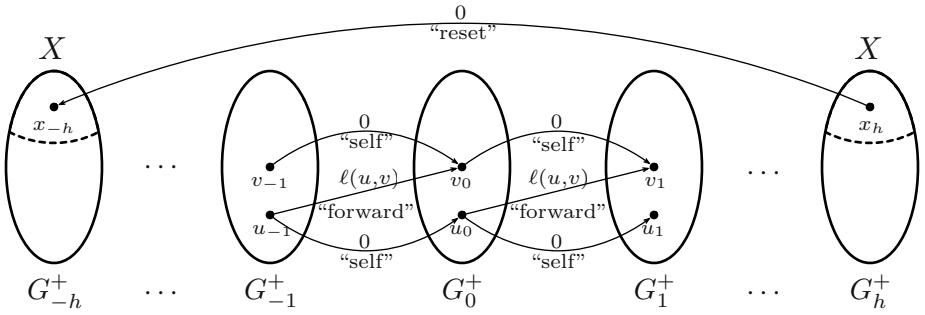
$$d^h_{G,\varphi}(s, x) + d^h_{G,\varphi}(x, t) = \ell_{H,\varphi}(s, x) + \ell_{H,\varphi}(x, t) \geq 0.$$

As argued in [Fin24], the inequality above implies that all but a $(1/b)$-fraction of the vertices are not negatively between s and t with high probability. The overall running time was $\tilde{O}(mbh + b^2 n)$, where the $b^2 n$ term accounts for the bn shortcut edges added to H. The primary bottleneck is the $\tilde{O}(mbh)$ time computing the lengths of the shortcut edges.

[LLRZ25] accelerates Fineman's subroutine with two key ideas. First, the new subroutine uses h-layer auxiliary graphs to effectively simulate h-hop distance computations without having to compute the distances explicitly to and from every sampled vertex. This comes at a cost of increasing the total number of edges from m to mh. Second, the auxiliary graph is arranged so that, after adding potentials that neutralize the negative "forward" arcs between the layers, the remaining auxiliary graph only has $|X|$ negative arcs. The key point is that the relatively few remaining negative arcs can be neutralized recursively. Thus the overall running time is that of neutralizing $|X| = \tilde{O}(b)$ negative arcs in a graph of mh edges.

The construction is essentially as follows. We construct a graph that initially consists of $2h + 1$ copies of G^+. We call these copies $G^+_{-h}, G^+_{1-h}, \ldots, G^+_{-1}, G^+_0, G^+_1, ..., G^+_{h-1}, G^+_h$. For each vertex v and index i, let v_i denote its copy in G^+_i. For every negative arc (u, v) and index $i < h$, we add a

"forward" arc (u_i, v_{i+1}) of the same length. For each vertex v and index $i < h$, we add a length-0 "self" arc (v_i, v_{i+1}). Lastly, for every sampled negative vertex $x \in X$, we add an auxiliary "reset" arc (x_h, x_{-h}) of length 0.



Relative to the construction in [Fin24], we can interpret the "top" layers $G^+_1, ..., G^+_h$ as simulating the shortcut arcs $d^h(u, x)$ for $u \in V$ and $x \in X$, and the "bottom" layers $G^+_{-h}, ..., G^+_{-1}$ as simulating the shortcut arcs $d^h(x, v)$ for $x \in X$ and $v \in V$. The "reset" arcs then allow us to concatenate shortcut arcs going through X.

Consider the potentials φ over V defined by

$$\varphi(v_i) = \begin{cases} d^i(V, v_i) & \text{if } i \geq 0 \\ -d^{-i}(v_i, V) & \text{if } i \leq 0. \end{cases}$$

It is easy to see that φ is valid over H and neutralizes all of the forward arcs. The only negative arcs in H_φ are the reset arcs (x_h, x_{-h}) for $x \in S$.

One recursively neutralizes H_φ to produce another potential ψ. The combined potentials $\varphi + \psi$, restricted to G^+_0, give valid potentials for G. For all $s, t \in V$ and $x \in X$, we have

$$d^h_{G,\varphi+\psi}(s, x) + d^h_{G,\varphi+\psi}(x, t) = d^0_{H,\varphi+\psi}(s_0, x_h) + \ell_{H,\varphi+\psi}(x_h, x_{-h}) + d^0_{H,\varphi+\psi}(x_{-h}, t_0) \geq 0.$$

It follows from [Fin24] that with high probability, all pairs of vertices have at most n/b vertices h-hop negatively between them in $G_{\varphi+\psi}$. □

With Lemma A.1, the running time from Lemma 3.1 improves to the following. If we only need betweenness reduction for a single hop parameter (as in the recursive sparsification algorithm), then the following bounds follow directly from Lemma A.1, Lemma 4.1 in [HJQ25], and Lemma 3.4 from [HJQ26].

Lemma A.2. *Let $h \geq \Omega(\log n)$. One can compute, with high probability and in the time to neutralize $O(b \log n)$ negative edges in a graph of $O(mh)$ edges, either:*

(i) A negative cycle.

(ii) Valid potentials φ neutralizing a set of $\Omega\left(\sqrt{kh}\right)$ negative vertices.

(iii) Valid potentials φ and a set U of negative vertices such that:

(a) $|U| \geq \Omega\left(\sqrt{kh}\right)$.

(b) U has h-hop negative reach of at most n/b vertices.

To obtain remoteness over a range of hop parameters, as in the bootstrapping algorithm, one can extend Lemma A.1 and obtain the following. (We refer the reader to [LLRZ25, Lemma 29] for the extension of Lemma A.1.)

Lemma A.3. *Let $h_0 = \Omega(\log n)$ and $h \geq h_0$. One can compute, with high probability, and in $\tilde{O}(mh)$ time plus the time to neutralize $\tilde{O}(b)$ negative edges in a graph of $\tilde{O}(mh_0)$ edges, either:*

(i) A negative cycle.

(ii) Valid potentials φ neutralizing a set of $\Omega\Big(\sqrt{kh_0}\Big)$ negative vertices.

(iii) Valid potentials φ and a set U of negative vertices such that:

(a) $|U| \geq \Omega\Big(\sqrt{kh_0}\Big)$.

(b) U has h_0-hop negative reach of size n/b.

(c) U has η-hop negative reach of size $n\eta/h$ for all $\eta \geq h_0$.

A.2 Plugging into the recursive algorithm

We now update the running time of the simple recursive algorithm based on the improved running time of Lemma A.2.

Recall the recursive algorithm from Section 4. For a parameter h, the algorithm repeatedly extracts an h-remote set U of size $\sqrt{kh}$, creates an h-layered graph of $O(m)$ edges, sparsifies the number of negative edges to $O\Big(\sqrt{k/h}\log(n)\Big)$, recursively neutralizes the sparsified layered graph, and uses the neutralized graph to compute Johnson's potentials for G_U. Letting $T(m,k)$ denote the running time to neutralize k negative edges in a graph of size m, Lemma A.2 improves the running time of the first step to $T(mh,h)$. Overall we have a recurrence of the form

$$T(m,k) = \tilde{O}\Big(\sqrt{k/h}\Big(T(mh,h) + T\Big(m,\sqrt{k/h}\Big)\Big)\Big).$$

This recurrence minimized by $h = k^{\frac{\sqrt{2}-1}{\sqrt{2}+1}}$, giving

$$T(m,k) = \tilde{O}\Big(mk^{1/\sqrt{2}}\Big).$$

A.3 Updating the bootstrapping running time

One can similarly revise the running time of the bootstrapping algorithm that obtained the bounds in Theorem 1.1. Here balancing parameters is not as straightforward. Some (natural) parameterizations lead to the same $\tilde{O}\Big(mn^{1/\sqrt{2}}\Big)$ running time (for sufficiently dense graphs) as above. One can do slightly better by reparameterizing the bootstrapping algorithm as follows. Let $\alpha \approx 0.7044$ and consider an iteration with k negative vertices. We set $h = k^{1-\alpha}$ and $h_0 = k^{2/\alpha-2-\alpha}$, and use Lemma A.3 to extract a set of negative vertices U with $\sqrt{kh^{1-\alpha}h_0^{\alpha}}$ negative vertices that is $(\eta, h/\eta)$-remote for all $\eta \in [h_0, h]$. The rest of the algorithm is identical. All together, this achieves a running time of $\tilde{O}(mk^{\alpha}) \leq O(mn^{.7044})$ when m is above some density threshold. We omit additional details because this algorithm appears to be slower than the following algorithm, which is also simpler. We also believe that a tweak of the bootstrapping algorithm can bring the running time very slightly, but would still be still slower than the ensuing algorithm.

A.4 The twice- (or thrice-) recursive algorithm

We now describe a third approach, that lies somewhere between the recursive sparsification algorithm and the bootstrapping algorithm in complexity. It uses the layered sparsification technique from the simple recursive sparsification algorithm, and the shortcut edges from the bootstrapping algorithm, but does not iteratively bootstrap hop reducers from a small negative reach to the entire graph as in the actual boostrapping algorithm. We originally called it the "twice-recursive algorithm" because

it recurses on two graphs to neutralize a remote set, although perhaps it ought to be called the "thrice-recursive algorithm" now that Lemma A.2 adds a third recursive call via the betweenness reduction step.

Let $\alpha \approx 0.69562077$ be the unique real root of the polynomial $p(x) = x^3 + 2x^2 + x - 2$. Let $\gamma = 1 + \frac{\alpha^2(1-\alpha)}{2(2+\alpha)} \approx 1.0273194$. Let $m_0 = n^\gamma$.

Lemma A.4. *For $m \geq m_0$, single-source shortest paths with real-weighted edge weights can be solved with high probability in $\tilde{O}(mn^\alpha)$ randomized time.*

Proof. More precisely, we prove that a graph with m edges and k negative edges, and $m \geq k^\gamma$, can be neutralized with high probability in $\tilde{O}(mk^\alpha)$ randomized time. The overall claim follows then follows from substituting $k = n$ and running Dijkstra's algorithm in the neutralized graph.

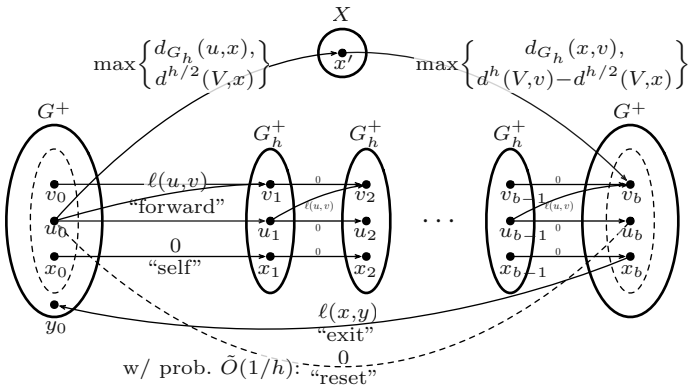
As with all other algorithms we've discussed, we neutralize the negative arcs iteratively. Let k be the number of negative arcs/vertices at the beginning of an iteration. Let $h = \tilde{\Theta}\left(k^{\alpha^2/(2+\alpha)}\right)$ and $b = \tilde{\Theta}\left(k^{(1-\alpha)}\right)$. ($h$ and b are balanced in hindsight.) We first invoke Lemma A.2 to either directly neutralize $\tilde{\Theta}\left(\sqrt{kh}\right)$ vertices (and repeat), identify a negative cycle (and exit), or obtain an (h, b)-remote set U of size $|U| = \tilde{\Theta}\left(\sqrt{kh}\right)$.

We continue in the third case. For ease of notation let $G = G_U$ for the rest of the iteration. Similar to the recursive sparsification algorithm, we construct a weighted graph $H = (V_H, E_H)$, potentials $\varphi : V_H \to \mathbb{R}$, and maps $\pi_0, \pi_1 : V \to V_H$ such that with high probability, (a) H_φ has $\tilde{O}(m)$ edges and $\tilde{O}\left(\sqrt{k/h}\right)$ negative edges, and (b) $d_H(\pi_0(s), \pi_1(t)) = d(s, t)$ for all $s, t \in V$. We neutralize H_φ recursively, use the neutralized graph to compute Johnson potentials neutralizing G, and conclude the iteration.

Next we describe and analyze the construction of the auxiliary graph H. The main point of H is that it effectively preserves distances in G while sparsifying the number of negative edges. One can interpret the following construction as extending the layered graph in Section 4 with "shortcut arcs" from the "shortcut gadgets" in Section 5.

Let V_h be the h-hop negative reach of U, and let G_h be the subgraph induced by V_h. Let X sample $O(|U| \log(n)/b)$ negative vertices uniformly at random. To construct H, we start with two disjoint copies of G^+, denoted G_0 and G_b; $b - 1$ copies of G_h^+, denoted $G_1, G_2, \ldots, G_{b-1}$; and a disjoint copy of X, denoted X'. For a vertex v and index i, we let v_i denote the copy of v in G_i (where applicable). For a sampled vertex $x \in X$, we let x' denote the auxiliary vertex in X'.

At a high level, H is a layered graph from G_0 to G_b, with additional shortcuts from G_0 to G_b via X, and sparsified reset arcs from G_b back to G_0. Specifically, for each negative arc (u, v) and index $i \in [b]$, we have a "forward" arc (u_{i-1}, v_i) of the same length. For each vertex $v \in V_h$ and index $i \in [b]$, we have length-0 "self" arcs (v_{i-1}, v_i). For each arc $(x, y) \in \partial^+(V_h)$ leaving the h-hop negative reach, we add an "exit" arc (x_b, y_0) of length $\ell(x, y)$. Next we add the "shortcut" arcs. For $u \in U$ and $x \in X$, we add an arc (u, x) of length $\max\left\{d_{G_h}(u, x), d^{h/2}(V, x)\right\}$. For $x \in X$ and $v \in \overline{U}$, we add an arc (x, v) of length $\max\left\{d_{G_h}(x, v), d^h(V, v) - d^{h/2}(V, x)\right\}$. Lastly, let $U_0 \subseteq U$ sample $O(|U| \log(n)/h)$ negative vertices uniformly at random. For each $u \in U_0$, we add a length-0 "reset" arc (u_b, u_0).



H has $O(m + kh\log(n)/b) \leq O(m)$ edges, noting that the second term is at most m_0 for our choices of h, b, and m_0. Every arc (u_i, v_j) in H is either a copy of an arc (u, v) in G, a self-arc (with $u = v$), or a shortcut arc. Obviously, a shortcut arc (u_0, x') or (x', v_b) has length at least $d_{G_h}(u, x) \geq d(u, x)$ and $d_{G_h}(x, v) \geq d(x, v)$. Meanwhile, exactly as argued in Lemma 5.3, we have $\ell_H(u_0, x') \leq d^{h/2}_{G_h}(u, x)$ and $\ell_H(x', v_b) \leq d^{h/2}_{G_h}(x, v)$ for all $u \in U$, $v \in \overline{U}$, and $x \in X$.

Define potentials φ over H by $\varphi(v_0) = 0$ for $v \in V$, $\varphi(v_i) = d^i(V, v)$ for $v \in V_h$ and $i \in [b-1]$, $\varphi(v_b) = d^h(V, v)$ for $v \in V$, and $\varphi(x') = d^{h/2}(V, x')$. It is straightforward to verify that φ is valid over H and neutralizes all the negative arcs except the $|U_0|$ reset arcs. As in Lemma 5.2, the shortcut arcs are neutralized because $\ell_H(u_0, x') \geq d^{h/2}(V, x)$ and $\ell_H(x', v_b) \geq d^h(V, v) - d^{h/2}(V, x)$ for $u \in U$, $v \in \overline{U}$, and $x \in X$.

We first analyze the distance-preserving correctness of H, and discuss the running time to construct H after. We claim that with high probability, $d_H(s_0, t_b) = d_G(s, t)$ for all $s, t \in V$. We have $d_H(s_0, t_b) \geq d(s, t)$ for all $s, t \in V$ because every arc from an auxiliary copy of a vertex u to an auxiliary copy a vertex v has length at least $d(u, v)$.

The reverse inequality follows from essentially the same reasons as Lemmas 4.1 and 5.2. First, observe that H contains the layered sparsification graph from Lemma 4.1, with $b - 1$ intermediate layers of the h-hop negative reach, as a subgraph. The same argument as in the proof of Lemma 4.1 proves that

$$d_H(s_0, t_b) \leq d^b(s, t)$$

for all $s, t \in V$.

Next we extend the inequality above to $d_H(s_0, t_b) \leq d^{h/2}(s, t)$ for all $s, t \in V$. To this end, we prove that $d_H(s_0, t_b) \leq d^{h/2}(s, t)$ for all $s, t \in V$ and $\eta \leq h/2$, by induction on η.

Let $W : s \rightsquigarrow t$ be a proper η-hop walk of length $d^\eta(s, t)$ for $b < \eta \leq h/2$. Here we have two cases depending on whether or not W stays in G_h between its first and last hop.

In the first case, suppose W stays in G_h between the first and last hop. Here the argument is the same as case 2.b of the proof of Lemma 5.2, observing that X hits at least one hop in W with high probability, using the shortcut arcs to route the subwalk between the first and last hops of W.

In the second case, suppose W leaves G_h between its first and last hop. Here the argument is the same as the argument from case 2 of the proof of Lemma 4.1, splitting W at the arc (x, y) leaving G_h, embedding the prefix and suffix by induction on η, and concatenating them with the exit arc (x_b, y_0).

Thus $d_H(s_0, t_b) \leq d^{h/2}(s, t)$ for all $s, t \in V$. The inequality extends to all hop lengths by the exact same argument as in the proof of Lemma 4.1: with high probability, U_0 hits any (s, t)-shortest walk once every $h/2$ hops with high probability. We embed each $h/2$-hop segment through H, and connect the embeddings with length-0 reset arcs.

It remains to analyze the running time of the algorithm. Consider the construction of H. It takes $O(h\mu)$ to compute the distances $d^i(V, v)$, $i \le h$, used to compute φ and to label the shortcut arcs. The other nontrivial step is computing the distances $d_{G_h}(u, x)$ and $d_{G_h}(x, v)$ for $x \in X$, $u \in U$, and $v \in \overline{U}$. To this end, we recursively neutralize G_h, and then compute shortest paths to and from each $x \in X$ in the neutralized graph. This takes one recursive call to a graph with m/b edges and $|U|$ negative edges, and $|X|$ nonnegative shortest path computations in a graph of m/b edges.

Let $T(m, k)$ denote the running time to neutralize a graph with m edges and k negative edges. $T(m, k)$ is modeled recursively by

$$T(m,k) = \tilde{O}\left(\sqrt{\frac{k}{h}}\left(T(mh, b) + T\Big(m/b, \sqrt{kh}\Big) + T\left(m, \sqrt{\frac{k}{h}}\right) + \frac{m\sqrt{kh}}{b^2}\right)\right),$$

where we note that the recursive calls will also be to subproblems in the dense regime. The recursion is solved by $T(m, k) \le \tilde{O}(mk^\alpha)$. □

For very sparse graphs with $m < n^{1+\gamma}$, similar to Theorem 6.2 for the bootstrapping algorithm, we used layered sparsification to reduce to the dense case and then apply Lemma A.4.

Lemma A.5. *For $m \le n^\gamma$, single-source shortest paths with real-valued edge lengths can be computed with $\tilde{O}\Big((mn)^{\frac{\alpha+\gamma}{1+\gamma}}\Big)$ randomized time, where $\frac{\alpha+\gamma}{1+\gamma} \approx 0.849861$.*

Proof. Suppose there are $k \le n$ negative edges; as with Lemma A.4, we directly analyze the time to neutralize a graph with m edges and k negative edges. Let $h = \tilde{O}\Big((k^\gamma/m)^{1/1+\gamma}\Big)$. As described in the proof of Theorem 6.2, let H be the graph obtained by layering G h times and sparsifying the negative edges by a factor of $O(h/\log n)$. H has $m_H \stackrel{\text{def}}{=} O(mh) = \tilde{O}\Big((mk)^{\gamma/1+\gamma}\Big)$ edges and $k_H = \tilde{O}(k/h) = O\Big((mk)^{1/(1+\gamma)}\Big)$ negative edges. In particular, $m_H \ge k_H^\gamma$. By Lemma A.4, we neutralize H with high probability in $\tilde{O}(m_H k_H^\alpha) = \tilde{O}\Big((mn)^{(\alpha+\gamma)/(1+\gamma)}\Big)$ randomized time, and use the neutralized H to compute distances for G. □

Together, Lemma A.4 and Lemma A.5 give the running times in Theorem 1.2.

Remark A.1. We were aware of the "twice recursive algorithm" when preparing the initial version of this paper, but it was almost completely superseded by the bootstrapping algorithm. With the previous running time of Lemma 3.1 for extracting a remote set, it only eked out a better running time for $m < n^{1.0235}$, by a factor of at most $n^{0.000107}$ — a comically small margin of improvement. We felt at the time that it wasn't worth the added complexity to the exposition. Evidently, the approach becomes more compelling under the new bounds of Lemma A.2. We found it interesting to see the twice-recursive approach become relatively stronger, while the bootstrapping approach apparently loses its edge, with the new betweenness-reduction subroutine.