Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time

MONIKA HENZINGER, University of Vienna, Austria
SEBASTIAN KRINNINGER, University of Salzburg, Austria
DANUPON NANONGKAI, KTH Royal Institute of Technology, Sweden

In the decremental single-source shortest paths (SSSP) problem, we want to maintain the distances between a given source node s and every other node in an n-node m-edge graph G undergoing edge deletions. While its static counterpart can be solved in near-linear time, this decremental problem is much more challenging even in the *undirected unweighted* case. In this case, the classic $O(mn)$ total update time of Even and Shiloach [16] has been the fastest known algorithm for three decades. At the cost of a $(1+\epsilon)$-approximation factor, the running time was recently improved to $n^{2+o(1)}$ by Bernstein and Roditty [9]. In this article, we bring the running time down to near-linear: We give a $(1+\epsilon)$-approximation algorithm with $m^{1+o(1)}$ expected total update time, thus obtaining *near-linear time*. Moreover, we obtain $m^{1+o(1)} \log W$ time for the weighted case, where the edge weights are integers from 1 to W. The only prior work on weighted graphs in $o(mn)$ time is the $mn^{0.9+o(1)}$-time algorithm by Henzinger et al. [18, 19], which works for directed graphs with quasi-polynomial edge weights. The expected running time bound of our algorithm holds against an oblivious adversary.

In contrast to the previous results, which rely on maintaining a sparse emulator, our algorithm relies on maintaining a so-called *sparse* (h,ϵ)*-hop set* introduced by Cohen [12] in the PRAM literature. An (h,ϵ)-hop set of a graph $G=(V,E)$ is a set F of weighted edges such that the distance between any pair of nodes in G can be $(1+\epsilon)$-approximated by their h-hop distance (given by a path containing at most h edges) on $G'=(V,E\cup F)$. Our algorithm can maintain an $(n^{o(1)},\epsilon)$-hop set of near-linear size in near-linear time under edge deletions. It is the first of its kind to the best of our knowledge. To maintain approximate distances using this hop set, we extend the monotone Even-Shiloach tree of Henzinger et al. [20] and combine it with the bounded-hop SSSP technique of Bernstein [4, 5] and Mądry [27]. These two new tools might be of independent interest.

CCS Concepts: • **Theory of computation → Shortest paths**; **Dynamic graph algorithms**; *Sparsification and spanners*;

A preliminary version of this paper was presented at the 55th IEEE Symposium on Foundations of Computer Science (FOCS 2014). The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement No. 340506. S. Krinninger's work was done in large part while at University of Vienna, Austria, and while supported by IK I049-N. D. Nanongkai's work was partially done while at ICERM, Brown University, USA, and while supported by the following research grants: Nanyang Technological University Grant No. M58110000, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 Grant No. MOE2010-T2-2-082, and Singapore MOE AcRF Tier 1 Grant No. MOE2012-T1-001-094.
Authors' addresses: M. Henzinger, University of Vienna, Faculty of Computer Science, Währinger Straße 29, 1090, Wien, Austria; email: monika.henzinger@univie.ac.at; S. Krinninger, University of Salzburg, Department of Computer Sciences, Jakob-Haringer-Straße 2, 5020, Salzburg, Austria; email: sebastian.krinninger@sbg.ac.at; D. Nanongkai, KTH Royal Institute of Technology, School of Computer Science and Communication (CSC), Lindstedtsvägen 3, SE-100 44, Stockholm, Sweden; email: danupon@gmail.com.

0004-5411/2018/11-ART36 $15.00
https://doi.org/10.1145/3218657

Additional Key Words and Phrases: Approximate shortest paths, hop sets

ACM Reference format:
Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2018. Decremental Single-Source Shortest Paths on Undirected Graphs in Near-Linear Total Update Time. *J. ACM* 65, 6, Article 36 (November 2018), 40 pages.
https://doi.org/10.1145/3218657

1 INTRODUCTION

Dynamic graph algorithms refer to data structures on graphs that support update and query operations. They are classified according to the type of update operations they allow: *decremental* algorithms allow only edge deletions, *incremental* algorithms allow only edge insertions, and *fully dynamic* algorithms allow both insertions and deletions. In this article, we consider decremental algorithms for the *single-source shortest paths (SSSP)* problem on *undirected* graphs. The *unweighted* case of this problem allows the following operations.

- DELETE(u, v): delete the edge (u, v) from the graph, and
- DISTANCE(v): return the distance $\text{dist}_G(s, v)$ between node s and node v in the current graph G.

The *weighted* case allows an additional operation INCREASE(u, v, Δ), which increases the weight of the edge (u, v) by Δ. We allow positive integer edge weights in the range from 1 to W, for some parameter W. For any $\alpha \geq 1$, we say that an algorithm is an *α-approximation* algorithm if, for any distance query DISTANCE(x), it returns a distance estimate $\delta(s, x)$ such that $\text{dist}_G(s, x) \leq \delta(s, x) \leq \alpha\ \text{dist}_G(s, x)$. There are two time complexity measures associated with this problem: *query time* denoting the time needed to answer *each* distance query, and *total update time* denoting the time needed to process *all* edge deletions. The running time will be measured in terms of n, the number of nodes in the graph, and m, the number of edges *before* the first deletion. For the weighted case, we additionally consider the dependence on W, the maximum edge weight. We use $\tilde{O}$-notation to hide factors that are polylogarithmic in n. In this article, we focus on algorithms with small ($O(1)$ or polylog$n$) query time, and the main goal is to minimize the total update time, which will simply be referred to as *time* when the context is clear.

Related Work. The static version of SSSP can be easily solved in $\tilde{O}(m)$ time using, e.g., Dijkstra's algorithm. Moreover, due to the deep result of Thorup [34], it can even be solved in linear ($O(m)$) time in undirected graphs with positive integer edge weights. This implies that in our setting we can naively solve decremental SSSP in $O(m^2)$ total update time by running the static algorithm after every deletion. The first non-trivial decremental algorithm is due to Even and Shiloach [16] from 1981 and takes $O(mn)$ total update time in unweighted undirected graphs. This algorithm will be referred to as *ES-tree* throughout this article. It has many applications such as for decremental strongly connected components [29] and multicommodity flow problems [27]; yet, the ES-tree has resisted many attempts of improving it for decades. Roditty and Zwick [32] explained this phenomenon by providing evidence that the ES-tree is optimal for maintaining exact distances even on *unweighted undirected* graphs, unless there is a major breakthrough for Boolean matrix multiplication and many other long-standing problems [37]. After the preliminary version of our work appeared, Henzinger et al. [21] showed that, up to subpolynomial factors, $O(mn)$ is essentially the best possible total update time for maintaining exact distances under the assumption that there is no "truly subcubic" algorithm for a problem called online Boolean matrix-vector multiplication. Under the same assumption, they also showed that there is no fully dynamic α-approximate SSSP

algorithm such that $\alpha < 2$ with amortized time $O(m^{\gamma-\delta})$ *per update* and query time $O(m^{1-\gamma-\delta})$ for any $\gamma \in (0,1)$ and $\delta > 0$.[1] In incremental and decremental algorithms, respectively, the same type of trade-off holds between the *worst-case* update time and the query time. It is thus natural to shift the focus to *amortized decremental approximation algorithms*; the amortization is usually done implicitly by only considering the *total* update time over a sequence of up to m deletions.

The first improvement for unweighted undirected graphs was due to Bernstein and Roditty [9] who presented a randomized $(1+\epsilon)$-approximation algorithm with $n^{2+O(1/\sqrt{\log n})}$ total update time.[2] This time bound is only slightly larger than quadratic and beats the $O(mn)$ time of the ES-tree unless the input graph is very sparse. After the preliminary version of our work appeared, Bernstein and Chechik, presented deterministic $(1+\epsilon)$-approximation algorithms for unweighted undirected graphs with total update times $\tilde{O}(n^2)$ [7] and $\tilde{O}(n^{1.25}\sqrt{m}) = \tilde{O}(mn^{3/4})$ [8], respectively. In weighted undirected graphs, an extension of the technique gives a total update time of $\tilde{O}(n^2 \log W)$ [6].

For the case of directed graphs, Henzinger and King [17] observed that the ES-tree can be easily adapted to unweighted directed graphs. King [24] later extended the ES-tree to an $O(mnW)$-time algorithm for weighted directed graphs. A rounding technique used in recent algorithms of Bernstein [4, 5] and Mądry [27], as well as earlier papers on approximate shortest paths [11, 25, 38], gives a $(1+\epsilon)$-approximate $\tilde{O}(mn \log W)$-time algorithm for weighted directed graphs. Very recently, we obtained a randomized $(1+\epsilon)$-approximation algorithm with total update time $mn^{0.9+o(1)}$ for decremental approximate SSSP in weighted directed graphs [18, 19] if $W \leq 2^{\log^c n}$ for some constant c. This gives the first $o(mn)$-time algorithm for the directed case, as well as other important problems such as single-source reachability and strongly connected components [10, 26, 29, 31]. Also very recently, Abboud and Vassilevska Williams [2] showed that "deamortizing" our algorithms in Reference [18] might not be possible: a combinatorial algorithm with *worst case* update time and query time of $O(n^{2-\delta})$ (for any $\delta > 0$) per deletion implies a faster combinatorial algorithm for Boolean matrix multiplication and, for the more general problem of maintaining the number of reachable nodes from a source under deletions (which our algorithms in Reference [18] can do) a worst case update and query time of $O(m^{1-\delta})$ (for any $\delta > 0$) will falsify the strong exponential time hypothesis.

Our Results. Given the significance of the decremental SSSP problem, it is important to understand its time complexity.

In this article, we obtain a near-linear time algorithm for decremental $(1+\epsilon)$-approximate SSSP in weighted undirected graphs. Its total update time is $m^{1+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)} \log W$ and it maintains an estimate of the distance between the source node and every other node, guaranteeing constant worst-case query time. The algorithm is randomized and assumes an oblivious adversary who fixes the sequence of updates in advance, an assumption that so far was also made for all other results on approximate decremental SSSP utilizing randomization. The algorithm is always correct and the bound on its total update time holds in expectation, which makes it a Las Vegas algorithm. In both the weighted and the unweighted setting, our algorithm significantly improves upon previous algorithms, leaving room for running time improvements only with respect to subpolynomial factors, which so far has only been achieved in the very dense regime [6, 7].

As a consequence of our techniques, we also obtain an algorithm for the all-pairs shortest paths (APSP) problem. For every integer $k \geq 2$ and every $0 < \epsilon \leq 1$, we obtain a randomized

[1] This conditional lower bound then holds for graphs with $m \leq \min(n^{1/\gamma}, n^{1/(1-\gamma)})$ many edges.

[2] To enhance readability, we assume that ϵ is a constant when citing related work, thus omitting the dependence on ϵ in the running times.

decremental $((2+\epsilon)^k-1)$-approximate APSP algorithm with query time $O(k^k)$ and total update time $m^{1+1/k+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)}\log^2 W$ in expectation. We remark that for $k=2$ and $1/\epsilon = \text{polylog}n$ our result gives a $(3+\epsilon)$-approximation with constant query time and total update time $m^{1+1/2+o(1)}\log W$. For very sparse graphs with $m=\Theta(n)$, this is almost optimal in the sense that it almost matches the static running time [35] of $O(m\sqrt{n})$, providing stretch of $3+\epsilon$ instead of 3 as in the static setting. Our result on approximate APSP has to be compared with the following prior work. For weighted directed graphs Bernstein [5] gave a randomized decremental $(1+\epsilon)$-approximate APSP algorithm with constant query time and total update time $\tilde{O}(mn\log W)$. For unweighted undirected graphs there are two previous results that improve upon this update time at the cost of larger approximation error. First, for any integer $k\geq 2$, Bernstein and Roditty [9] gave a randomized decremental $(2k-1+\epsilon)$-approximate APSP algorithm with constant query time and total update time $mn^{1/k+O(1/\sqrt{\log n})}$. Second, for any integer $k\geq 2$, Abraham et al. [3] gave a randomized decremental $2^{O(\rho k)}$-approximate APSP algorithm for unweighted undirected graphs with query time $O(k\rho)$ and total update time $\tilde{O}(mn^{1/k})$, where $\rho = (1+\lceil(\log n^{1-1/k})/\log(m/n^{1-1/k})\rceil)$.

Outline. We give preliminaries on decremental approximate shortest path algorithms in Section 2 and provide a technical overview of our approach in Section 3. Our algorithm, presented in Section 4 to 6, uses the following hierarchical approach: Given a decremental approximate SSSP algorithm for distances up to D_i with total update time $m^{1+o(1)}$, we can maintain so-called approximate balls for distances up to D_i with total time $m^{1+o(1)}$ as well. And given a decremental algorithm for maintaining approximate balls for distances up to D_i with total update time $m^{1+o(1)}$, we can use the approximate balls to define a hop set that allows us to maintain approximate shortest paths for distances up to $D_{i+1}=n^{o(1)}D_i$ with total update time $m^{1+o(1)}$. This scheme is repeated until D_i is large enough to cover the full distance range. We have formulated the two parts of this scheme as reductions. In Section 4, we give a decremental algorithm for maintaining approximate balls that internally uses a decremental approximate SSSP algorithm. In Section 5, we give a decremental approximate SSSP algorithm that internally uses a decremental algorithm for maintaining approximate balls. In Section 6, we explain the hierarchical approach for putting these two parts together and obtain the decremental $(1+\epsilon)$-approximate SSSP algorithm with a total update time of $m^{1+o(1)}$ for the full distance range. In addition to this result, the algorithm for maintaining approximate balls, together with a suitable query algorithm, gives us a decremental approximate APSP algorithm. This algorithm is also given in Section 6. Finally, we conclude the article in Section 7.

2 PRELIMINARIES

In this article, we want to maintain approximate shortest paths in an undirected graph $G=(V,E)$ with positive integer edge weights in the range from 1 to W, for some parameter W. The graph undergoes a sequence of *updates*, which might be edge deletions or edge weight increases. This is called the *decremental setting*. We denote by V the set of nodes of G and by E the set of edges of G. We denote by n the number of nodes of G and by m the number of edges of G before the first edge deletion.

For every weighted undirected graph G, we denote the weight of an edge (u,v) in G by $w_G(u,v)$. The *distance* $\text{dist}_G(u,v)$ between a node u and a node v in G is the weight of the shortest path, i.e., the minimum-weight path, between u and v in G. If there is no path between u and v in G, then we set $\text{dist}_G(x,y)=\infty$. For every set of nodes $U\subseteq V$, we denote by $E[U]$ the set of edges incident to the nodes of U, i.e., $E[U]=\{(u,v)\in E \mid u\in U\}$.[3] Furthermore, for every set of nodes $U\subseteq V$, we denote by $G|U$ the subgraph of G induced by the nodes in U, i.e., $G|U$ contains all edges (u,v) such

[3]Since G is an undirected graph, this definition is equivalent to $E[U]=\{(u,v)\in E \mid u\in U \text{ or } v\in U\}$.

that (u, v) is contained in E and u and v are both contained in U, or in short: $G|U = (U, E \cap U^2)$. Similarly, for every set of edges $F \subseteq V^2$ and every set of nodes $U \subseteq V$, we denote by $F|U$ the subset of F induced by U.

We say that a distance estimate $\delta(u, v)$ is an (α, β)-approximation of the true distance $\text{dist}_G(u, v)$ if $\text{dist}_G(u, v) \leq \delta(u, v) \leq \alpha \text{dist}_G(u, v) + \beta$, i.e., $\delta(u, v)$ never underestimates the true distance and overestimates it with a multiplicative error of at most α and an additive error of at most β. If there is no additive error, then we simply say α-approximation instead of $(\alpha, 0)$-approximation.

In our algorithms, we will use graphs that do not only undergo edge deletions and edge weight increases but also edge insertions. For such a graph H, we denote by $\mathcal{E}(H)$ the number of edges ever contained in H, i.e., the number of edges contained in H before any deletion or insertion plus the number of inserted edges. We denote by $\mathcal{W}(H)$ the number of updates to the edges in H. Similarly, for a set of edges F, we denote by $\mathcal{E}(F)$ the number of edges ever contained in F and by $\mathcal{W}(F)$ the number of updates to the edges in F.

The central data structure in decremental algorithms for exact and approximate shortest paths is the Even-Shiloach tree (ES-tree). This data structure maintains a shortest paths tree from a root node up to a given depth D.

Lemma 2.1 ([16, 17, 24]). *There is a data structure called ES-tree that, given a weighted directed graph G undergoing deletions and edge weight increases, a root node s, and a depth parameter D, maintains, for every node v a value $\delta(v, s)$ such that $\delta(v, s) = \text{dist}_G(v, s)$ if $\text{dist}_G(v, s) \leq D$ and $\delta(v, s) = \infty$ if $\text{dist}_G(v, s) > D$. It has constant query time and a total update time of $O(mD + n)$.*

Note that Dinitz, as part of his max-flow algorithm [13], earlier developed an algorithm with similar guarantees for the decremental single-source single-sink shortest path problem [14].

Recent approaches for solving approximate decremental SSSP and APSP use special graphs called *emulators*. An (α, β)-emulator H of a graph G is a graph containing the nodes of G such that $\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq \alpha\ \text{dist}_G(u, v) + \beta$ for all nodes u and v.[4] Maintaining exact distances on H provides an (α, β)-approximation of distances in G. As good emulators are sparser than the original graph this is usually more efficient than maintaining exact distances on G. However, the edges of H also have to be maintained while G undergoes updates. For unweighted, undirected graphs undergoing edge deletions, the emulator of Thorup and Zwick (based on the second spanner construction in [36]), which provides a relatively good approximation, can be maintained quite efficiently [9]. However, the definition of this emulator requires the occasional insertion of edges into the emulator. Thus, it is not possible to run a purely decremental algorithm on top of it.

There have been approaches to design algorithms that mimic the behavior of the classic ES-tree when run on an emulator that undergoes insertions. The first approach by Bernstein and Roditty [9] extends the ES-tree to a fully dynamic algorithm and analyzes the additional work incurred by the insertions. The second approach was introduced by us in Reference [20] and is called *monotone ES-tree*. It basically ignores insertions of edges into H and never decreases the distance estimate it maintains. However, this algorithm does not provide an (α, β)-approximation on *any* arbitrary (α, β)-approximate emulator as it needs to exploit structural properties of the emulator to guarantee the approximation. In Reference [20], we gave an analysis of the monotone ES-tree when run on a specific $(1 + \epsilon, 2)$-emulator, and in the current article, we use a different analysis for our new algorithms. If we want to use the monotone ES-tree to maintain (α, β)-approximate distances up to depth D, then we will set the maximum level in the monotone ES-tree to $L = \alpha D + \beta$. The running time of the monotone ES-tree as analyzed in Reference [20] is as follows.

[4]For the related notion of a spanner, we additionally have to require that H is a subgraph of G.

Lemma 2.2. *For every $L \geq 1$, the total update time of a monotone ES-tree up to maximum level L on a graph H undergoing edge deletions, edge insertions, and edge weight increases is $O(\mathcal{E}(H) \cdot L + \mathcal{W}(H) + n)$.*

3 TECHNICAL OVERVIEW

In the following, we explain the main ideas of this article, which lead to an algorithm for maintaining a hop set of a graph undergoing edge deletions.

General Idea. With the well-known algorithm of Even and Shiloach, we can maintain a shortest paths tree from a source node up to a given depth D under edge deletions in time $O(mD)$. In unweighted graphs, all simple paths have at most $n-1$ edges and, therefore, we can set $D = n$ to maintain a full shortest paths tree. In weighted graphs with positive integer edge weights from 1 to W, all simple paths have weight at most $(n-1)W$ and, therefore, we can set $D = nW$ to maintain a full shortest paths tree. Using an established rounding technique [4, 5, 11, 25, 27, 28, 38], one can use this algorithm to maintain $(1+\epsilon)$-approximate single-source shortest paths up to h edges in time $O(mh \log(nW)/\epsilon)$. By setting $h = n$, we can use this algorithm to maintain a full approximate shortest paths tree, even in weighted graphs. This algorithm would be very efficient if the graph had a small hop diameter, i.e., if for any pair of nodes there were a shortest path with a small number of edges. Our idea is to artificially construct such a graph.

To this end, we will use a so-called *hop set*. An (h, ϵ)- *hop set* F of a graph $G = (V, E)$ is a set of weighted edges $F \subseteq V^2$, where the weight of each edge $(u, v) \in F$ is at least $\text{dist}_G(u, v)$, such that in the graph $H = (V, E \cup F)$ there exists, for every pair of nodes u and v, a path from u to v of weight at most $(1+\epsilon)\text{dist}_G(u, v)$ and with at most h hops. In this terminology, the number of hops of a path is its number of edges. If we run the approximate SSSP algorithm on H, then we obtain a running time of $O((m + |F|)h \log(nW)/\epsilon)$. In our algorithm, we will obtain an $(n^{o(1)}, \epsilon)$-hop set of size $m^{1+o(1)}$ and thus, given the hop set, the running time will be $m^{1+o(1)} \log(nW)/\epsilon$. It is, however, not enough to simply construct the hop set at the beginning. We also need a dynamic algorithm for maintaining the hop set under edge deletions in G. We will present an algorithm that performs this task also in almost linear time over all deletions.

Roughly speaking, we achieve the following. Given a graph $G = (V, E)$ undergoing edge deletions, we can maintain a restricted hop set F such that, for all pairs of nodes u and v, if the shortest path π from u to v in G has $h \geq n^{1/q}$ hops, then in the shortcut graph $H = (V, E \cup F)$ there is a path from u to v of weight at most $(1+\epsilon)\text{dist}_G(u, v)$ and with at most $\lceil h/n^{1/q} \rceil \log n$ hops. Our high-level idea for maintaining an (unrestricted) $(n^{o(1)}, \epsilon)$ hop set is the following hierarchical approach. We start with $H_0 = G$ to maintain a hop set F_1 of G, which reduces the number of hops by a factor of $\log n/n^{1/q}$ at the cost of a multiplicative error of $1+\epsilon$. Given F_1, we use the shortcut graph $H_1 = (V, E \cup F_1)$ to maintain a hop set F_2 of G that reduces the number of hops by another factor of $\log n/n^{1/q}$ introducing another error of $1+\epsilon$. By repeating this process q times, we arrive at a hop set that guarantees, for all pairs of nodes u and v, a path of weight at most $(1+\epsilon)^q \text{dist}_G(u, v)$ and with at most $(\log n)^q$ hops. Figure 1 visualizes this hierarchical approach.

The notion of hop set was first introduced by Cohen [12] in the PRAM literature and is conceptually related to the notion of emulator. It is also related to the notion of *shortest-path diameter* used in distributed computing (e.g., References [23, 28]). To the best of our knowledge, the only place that this hop set concept was used before in the dynamic algorithms literature (without the name being mentioned) is Bernstein's fully dynamic $(2+\epsilon)$-approximate APSP algorithm [4]. There, Bernstein shows that the clustering of Thorup and Zwick [35] yields an $(n^{o(1)}, \epsilon)$-hop set by connecting each node with all nodes in its cluster. In his fully dynamic algorithm, this clustering is recomputed *from scratch* after every edge update. Conceptually, our hop set is a decremental

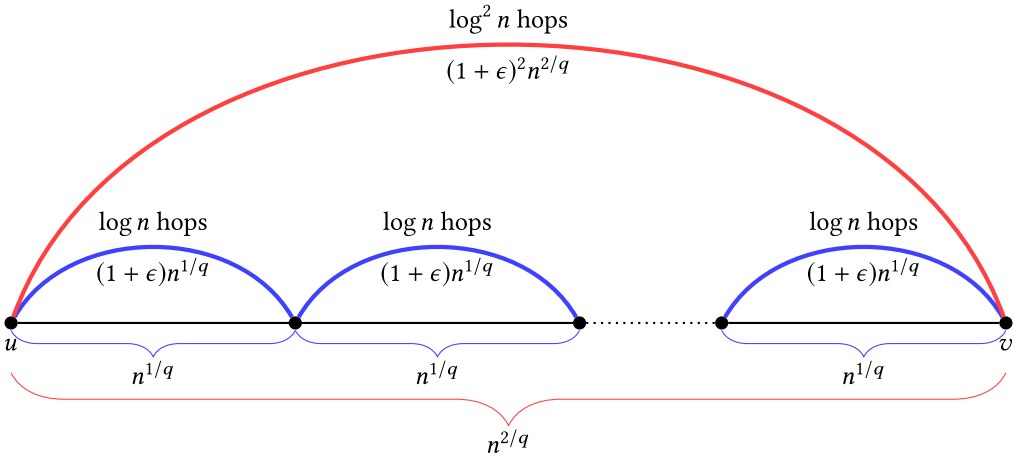

Fig. 1. Illustration of the hierarchical approach for maintaining the hop set reduction. Here, $q = \Theta(\sqrt{\log n})$ and u and v are nodes that are at distance $n^{2/q}$ from each other. First, we find a hop set that shortcuts all subpaths of weight $n^{1/q}$ by paths of weight at most $(1+\epsilon)n^{1/q}$ and with at most $\log n$ hops. Second, we use the shortcuts of the first hop set to find a hop set that shortcuts the path from u to v of weight $n^{2/q}$ by a path of weight at most $(1+\epsilon)^2 n^{2/q}$ and with at most $\log^2 n$ hops.

variant of Bernstein's hop set based, however, on a slightly simpler clustering. After the preliminary version of our work appeared, Elkin and Neiman [15] and Huang and Pettie [22] proved that the hop set based on the Thorup-Zwick clustering provides a close-to optimal trade-off between hop parameter and size [1] for $(1+\epsilon)$-approximate distances.

Static Hop Set. We first assume that $G = (V, E)$ is an unweighted undirected graph, and for simplicity, we also assume that ϵ is a constant. We explain how to obtain a hop set of G using a randomized construction of Thorup and Zwick [36] based on the notion of balls of nodes. We describe this construction and the hop-set analysis in the following.

Let $2 \leq p \leq \log n$ be a parameter and consider a sequence of sets of nodes $A_0, A_1, \ldots, A_p$ obtained as follows. We set $A_0 = V$ and $A_p = \emptyset$, and for $1 \leq i \leq p-1$, we obtain the set A_i by picking each node of V independently with probability $1/n^{i/p}$. The expected size of A_i is $n^{1-i/p}$. For every node u, we define the priority of u as the maximum i such that $u \in A_i$. For a node u of priority i, we define

$$Ball(u) = \{v \in V \mid \text{dist}_G(u, v) < \text{dist}_G(u, A_{i+1})\}, \tag{1}$$

where $\text{dist}_G(u, A_{i+1}) = \min_{v \in A_{i+1}} \text{dist}_G(u, v)$. Note that $\text{dist}_G(u, A_p) = \infty$, and thus if $u \in A_{p-1}$, then $Ball(u) = V$. For each node u of priority i the size of $Ball(u)$ is $n^{(i+1)/p}$ in expectation by the following argument: Order the nodes in non-decreasing distance from u. Each of these nodes belongs to A_{i+1} with probability $1/n^{(i+1)/p}$ and therefore, in expectation, we need to see $n^{(i+1)/p}$ nodes until one of them is contained in A_{i+1}. It follows that the expected size of all balls of priority i is at most $n^{1+1/p}$ (the expected size of A_i times the expected size of $Ball(u)$ for each node u of priority i) and the expected size of all balls, i.e., $\sum_{u \in V} |Ball(u)|$, is at most $pn^{1+1/p}$.

Let F be the set of edges $F = \{(u, v) \in V^2 \mid v \in Ball(u)\}$ and give each edge $(u, v) \in F$ the weight $w_F(u, v) = \text{dist}_G(u, v)$. By the argument above, the expected size of F is at most $pn^{1+1/p}$. An argument of Thorup and Zwick [36] shows that the weighted graph $H = (V, F)$ has the following

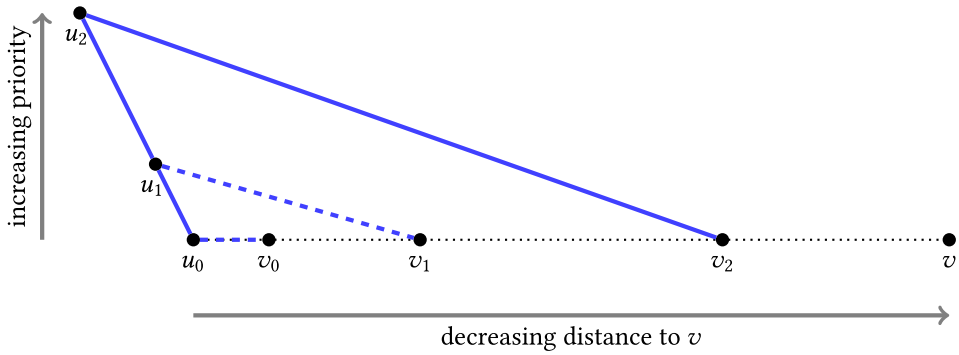

Fig. 2. Illustration of the approximation argument for $p = 3$ priorities. The dotted line is the shortest path π from u_0 to v in G. The thick, blue edges are the edges of F used to shorten the distance to v. The dashed, blue edges are not contained in F and imply the existence of edges to nearby nodes of increasing priority. Starting from u_0, a node of priority 0, we let v_0 be the node on π such that $\text{dist}_G(u_0, v_0) = r_0 := 1$, i.e., the neighbor of u_0 on π. If the edge (u_0, v_0) is not contained in F, then F contains an edge (u_0, u_1) to a node u_1 of priority at least 1 such that $\text{dist}_G(u_0, u_1) \leq r_0$. Let v_1 be the node on π such that $\text{dist}_G(u_1, v_1) = r_1 := 1 + 2/\epsilon$. If the edge (u_1, v_1) is not contained in F, then F contains an edge (u_1, u_2) to a node u_2 of priority at least 2 such that $\text{dist}_G(u_1, u_2) \leq r_1$. Let v_2 be the node on π such that $\text{dist}_G(u_2, v_2) = r_2 := (1 + 2/\epsilon)(2 + 2/\epsilon)$. Since 2 is the highest priority, u_2 contains the edge (u_2, v_2). Note that the weight of these three edges from F is at most $r_0 + r_1 + r_2$ and $\text{dist}_G(u_0, v_2) \geq r_2 - (r_0 + r_1)$. Since $r_2 = (1 + 2/\epsilon)(r_0 + r_1)$, the ratio between these two quantities is $(1 + \epsilon)$.

property for every pair of nodes u and v and any $0 < \epsilon \leq 1$ such that $1/\epsilon$ is integer[5]:

$$\text{dist}_G(u, v) \leq \text{dist}_H(u, v) \leq (1 + \epsilon)\text{dist}_G(u, v) + 2\left(2 + \frac{2}{\epsilon}\right)^{p-2}.$$

Note that the choice of ϵ gives a trade-off in the error between the multiplicative part $(1 + \epsilon)$ and the additive part $2(2 + 2/\epsilon)^{p-2}$. In the literature, such a graph H is known as an *emulator* of G with multiplicative error $(1 + \epsilon)$ and additive error $2(2 + 2/\epsilon)^{p-2}$.[6] Roughly speaking, the strategy in their proof is as follows. Let u' be the node following u on the shortest path π from u to v in G. If the edge (u, u') is also contained in H, then we can shorten the distance to v by 1 without introducing any approximation error (recall that we assume that G is unweighted). Otherwise, one can show that there is a path π' with at most p edges in H from u to a node v' closer to v than u such that the ratio between the weight of π' and the distance from u to v' is at most $(1 + \epsilon)$, and, if $v' = v$, then the weight of π' is at most $2(2 + 2/\epsilon)^{p-2}$. The proof needs the following property of the balls: for every node x of priority i and every node y, either $y \in \mathit{Ball}(x)$ or there is some node z of priority $j > i$ such that $\text{dist}_G(x, z) = \text{dist}_G(x, A_{i+1}) \leq \text{dist}_G(x, y)$. We illustrate the proof strategy in Figure 2.

Observe that the same strategy can be used for the following hop-reduction argument: Given any integer $\Delta \leq n$, let u' be the node that is at distance Δ from u on the shortest path from u to v in G. If the edge (u, u') is contained in H, then we can shorten the distance to v by Δ without introducing any approximation error. Otherwise, one can show that there is a path π' with at most

[5]The requirement that $1/\epsilon$ must be integer is not needed in the paper of Thorup and Zwick; we have added it here to simplify the exposition.

[6]In their paper, Thorup and Zwick [36] actually define a graph H' whose set of edges is the union of the shortest paths trees from every node u to all nodes in its ball. This graph has the same approximation error and the same size as H; since H' is a subgraph of G, it is called a *spanner* of G.

p edges in H from u to a node v' closer to v than u such that the ratio between the weight of π' and the distance from u to v' is at most $(1+\epsilon)$, and, if $v' = v$, then the weight of π' is at most $2(2+2/\epsilon)^{p-2} \cdot \Delta$. Every time we repeat this argument the distance to v is shortened by at least Δ. Therefore, there is a path from u to v in H with at most $p\lceil \text{dist}_G(u,v)/\Delta \rceil$ edges that has weight at most $(1+\epsilon)\text{dist}_G(u,v) + 2(2+2/\epsilon)^{p-2} \cdot \Delta$. Bernstein [4] observed that in this type of argument the latter statement would also be true if we had removed all edges from F of weight more than $(1+2/\epsilon)(2+2/\epsilon)^{p-2}$, which is the maximum weight of the edge to v' in the proof strategy above outlined in Figure 2. We will need this fact in the dynamic algorithm as it allows us to limit the depth of the balls.

By a suitable choice of $p = \Theta(\sqrt{\log n})$ (as a function of n and ϵ), we can guarantee that $2(2+2/\epsilon)^{p-2} \leq \epsilon n^{1/p}$ and $n^{1/p} = n^{o(1)}$. Now define $q = p$ and $\Delta_k = n^{k/q}$ for each $0 \leq k \leq q-2$. Then, we have, for every $0 \leq k \leq q-2$ and all pairs of nodes u and v,

$$\begin{aligned}\text{dist}_G(u,v) \leq \text{dist}_H(u,v) &\leq (1+\epsilon)\text{dist}_G(u,v) + 2\left(2+\frac{2}{\epsilon}\right)^{p-2} \cdot \Delta_k \\ &\leq (1+\epsilon)\text{dist}_G(u,v) + \epsilon n^{1/p} \cdot \Delta_k \\ &= (1+\epsilon)\text{dist}_G(u,v) + \epsilon\Delta_{k+1}.\end{aligned}$$

Thus, if $\Delta_{k+1} \leq \text{dist}_G(u,v) \leq \Delta_{k+2}$, then there is a path from u to v in H of weight at most

$$(1+\epsilon)\text{dist}_G(u,v) + \epsilon\Delta_{k+1} \leq (1+\epsilon)\text{dist}_G(u,v) + \epsilon\ \text{dist}_G(u,v) = (1+2\epsilon)\text{dist}_G(u,v)$$

and with at most $p\lceil \text{dist}_G(u,v)/\Delta_k \rceil \leq (p+1)\Delta_{k+2}/\Delta_k = (p+1)n^{2/q} = n^{o(1)}$ edges. However, if $\text{dist}_G(u,v) \leq \Delta_1 = n^{1/q}$, then, as each edge of G has weight at least 1, there is a path from u to v of weight $\text{dist}_G(u,v)$ with at most $n^{1/p} = n^{o(1)}$ edges. It follows that F is an $(n^{o(1)}, 2\epsilon)$-hop set of size $O(pn^{1+1/p}) = n^{1+o(1)}$. Using the parameter $\epsilon' = \epsilon/2$ instead of ϵ, we obtain an $(n^{o(1)}, \epsilon)$-hop set of size $n^{1+o(1)}$.

Efficient Construction. So far, we have ignored the running time for computing the balls and thus constructing F, even in the static setting. Thorup and Zwick [36] have remarked that a naive algorithm for computing the balls takes time $O(mn)$. We can reduce this running time by sampling edges instead of nodes.

We modify the process for obtaining the sequence of sets $A_0, A_1, \ldots, A_p$ as follows. We set $A_0 = V$ and $A_p = \emptyset$ and for $1 \leq i \leq p-1$, we obtain the set A_i by picking each edge of E independently with probability $1/m^{i/p}$ and adding both endpoints of each sampled edge to A_i. The priority of a node u is the maximum i such that $u \in A_i$. We define, for every node u of priority i, $Ball(u)$ just like in Equation (1), but using the new definition of A_i. Note that the expected size of A_i is $O(m^{1-i/p})$ for every $1 \leq i \leq p-1$.

The balls can now be computed as follows. First, following Thorup and Zwick [35], we compute, for each $1 \leq i \leq p-1$, $\text{dist}_G(u, A_i) = \min_{v \in A_i} \text{dist}_G(u,v)$ for every node u by adding an artificial source node s_i that is connected to every node in A_i by an edge of weight 0. Using Dijkstra's algorithm, this takes time $O(p(m + n\log n))$. Second, we compute for every node u of priority i a shortest paths tree *up to depth* $\text{dist}_G(u, A_{i+1}) - 1$ to obtain all nodes contained in $Ball(u)$. Using an implementation of Dijkstra's algorithm that only puts nodes into its queue upon their first visit this takes time $O(|E[Ball(u)]| \log n)$ where $E[Ball(u)] = \{(x,y) \in E \mid x \in Ball(u) \text{ or } y \in Ball(u)\}$ is the set of edges incident to $Ball(u)$. Using the same ordering argument as before, our random sampling process for the edges guarantees that the expected size of $E[Ball(u)]$ is $m^{(i+1)/p}$. For $0 \leq i \leq p-1$ the expected size of A_i is $O(m^{1-i/p})$, and thus these Dijkstra computations take time $O(m^{1+1/p}\log n)$ for all nodes of priority i. By choosing $p = \Theta(\sqrt{\log n})$, as described above, we have $m^{1/p} = m^{o(1)}$ and thus the balls can be computed in time $m^{1+o(1)}$.

We define F as the set of edges $F = \{(u, v) \in V^2 \mid v \in Ball(u)\}$ and give each edge $(u, v) \in F$ the weight $w_F(u, v) = \text{dist}_G(u, v)$. The distance-preserving and hop-reducing properties of F still hold as stated above and its expected size is $O(pm^{1+1/p})$. Note that F is not necessarily a sparsification of G anymore (as the bound on its size is even more than m). For our purposes the sparsification aspect is not relevant, we only need the hop reduction. Thus, in the static setting, we can compute an $(m^{o(1)}, \epsilon)$-hop set (which is also an $(\epsilon, n^{o(1)})$-hop set) of expected size $m^{1+o(1)}$ in expected time $m^{1+o(1)}$.

Maintaining Balls Under Edge Deletions. As the graph G undergoes deletions the hop set has to be updated as well. Unfortunately, we do not know how to maintain the balls efficiently. However, we can maintain for all nodes u the approximate ball,

$$Ball(u, D) = \{v \in V \mid \log \text{dist}_G(u, v) < \lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor \text{ and } \text{dist}_G(u, v) \leq D\},$$

(where i is the priority of u) in time $O(pm^{1+1/p} D \log D)$. Note that $Ball(u, D)$ differs from the definition of $Ball(u)$ in the following ways. First, we use the inequality $\log \text{dist}_G(u, v) < \lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor$ instead of the inequality $\text{dist}_G(u, v) < \text{dist}_G(u, A_{i+1})$. This relaxed inequality alone increases the additive error in the hop-reduction argument from $2(2 + 2/\epsilon)^{p-2}\Delta$ to $4(3 + 4/\epsilon)^{p-2}\Delta$, since before a node v of higher priority than a given node u could be found directly at the boundary of the ball of u, whereas now v could be twice as far away. The increase in the additive error can easily be compensated by reducing the number of priorities p by a constant factor. Second, we limit the balls to a certain depth D. By using a small value of D, we will only obtain a restricted hop set that provides sufficient hop reduction for nodes that are relatively close to each other. We will show later that this is enough for our purposes. Despite these modifications, we clearly have $Ball(u, D) \subseteq Ball(u)$, and therefore all size bounds still apply.

In the first part of the algorithm for maintaining the balls, we maintain $\text{dist}_G(u, A_i)$ up to threshold D for every $1 \leq i \leq p - 1$ and every node u. We do this by adding an artificial source node s_i that has an edge of weight 0 to every node in A_i and maintain an ES-tree up to depth D from s_i. This step takes time $O(pmD)$.

Now, for every node u of priority i, we maintain $Ball(u, D)$ as follows. We maintain an ES-tree up to depth

$$\min\left(2^{\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor} - 1, D\right),$$

and every time $2^{\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor}$ increases, we restart the ES-tree. Naively, we incur a cost of $O(mD)$ for each instance of the ES-tree. However, we can easily implement the ES-tree in such a way that it never processes edges that are not contained in $E[Ball(u, D)]$.[7] Thus, the cost of each instance of the ES-tree is $O(|E[Ball(u, D)]|D)$. Remember that $Ball(u, D) \subseteq Ball(u)$ and that $E[Ball(u)]$ is at most $m^{(i+1)/p}$ in expectation. As $2^{\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor}$ can increase at most $\log D$ times until it exceeds D, we initialize at most $\log D$ ES-trees for the node u. Therefore, the total time needed for maintaining $Ball(u, D)$ is $O(m^{(i+1)/p} D \log D)$ in expectation. As there are at most $O(m^{1-i/p})$ nodes of priority i in expectation, the total time needed for maintaining all approximate balls is $O(pm^{1+1/p} D \log D)$ in expectation.

[7] If we prefer to use the ES-tree as a "black box," then we can, in a preprocessing step, find the initial set $Ball(u, D)$ and only build an ES-tree for this ball. All other nodes will never be contained in $Ball(u, D)$ anymore as long as the value of $2^{\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor}$ remains unchanged and, therefore, we can remove them. This can be done in time $O(|E[Ball(u, D)]| \log n)$ by using an implementation of Dijkstra's algorithm that only puts nodes into its queue upon their first visit.

Decremental Approximate SSSP. Let us first sketch an algorithm for maintaining shortest paths from a source node s with a running time of $m^{1+1/2+o(1)}$, for which we use $p = \Theta(\sqrt{\log n})$ priorities. We set $\Delta = \lfloor \sqrt{n} \rfloor$, p such that $(2 + 4/\epsilon)(3 + 4/\epsilon)^{p-2} \leq \epsilon n^{1/p}$ and $n^{1/p} = n^{o(1)}$, and $D = \lceil \epsilon n^{1/p} \rceil$. We maintain single-source shortest paths up to depth D from s using the ES-tree, which takes time $O(mD) = mn^{1/2+o(1)}$. To maintain approximate shortest paths to nodes that are at distance more than D from s, we use the following approach. We maintain $Ball(u, D)$ for every node u, as sketched above, which takes time $O(pm^{1+1/p} D \log D) = m^{1+1/2+o(1)}$ in expectation. At any time, we set the hop set to be the set of edges $F = \{(u, v) \in V^2 \mid v \in Ball(u, D)\}$ and give each edge $(u, v) \in F$ the weight $w_F(u, v) = \text{dist}_G(u, v)$. By our arguments above, the weighted graph $H = (V, F)$ has the following property: for every pair of nodes u and v such that $\text{dist}_G(u, v) \geq D$ (where $D \geq n^{1/p}\Delta$) there is a path π' in H of weight at most $(1 + \epsilon)\text{dist}_G(u, v) + \epsilon n^{1/p}\Delta \leq (1 + 2\epsilon)\text{dist}_G(u, v)$ and with at most $p\lceil \text{dist}_G(u, v)/\Delta \rceil$ edges.

To maintain approximate shortest paths for nodes at distance more than D from s, we will now use the hop reduction in combination with the following rounding technique. We set $\varphi = \epsilon\Delta/(p + 1)$ and let H' be the graph resulting from rounding up every edge weight in H to the next multiple of φ. By using H' instead of H, we incur an error of φ for every edge on the approximate shortest path π'. Thus, in H', π' has weight at most

$$\begin{aligned}(1 + 2\epsilon)\text{dist}_G(u, v) + \lceil p\ \text{dist}_G(u, v)/\Delta \rceil \cdot \varphi &= (1 + 2\epsilon)\text{dist}_G(u, v) + \epsilon\ \text{dist}_G(u, v) \\ &\leq (1 + 3\epsilon)\text{dist}_G(u, v).\end{aligned}$$

The efficiency now comes from the observation that we can run the algorithm on the graph H'' in which every edge weight in H' is scaled down by a factor of $1/\varphi$. The graph H'' has integer weights and the weights of all paths in H' and H'' differ exactly by the factor $1/\varphi$. Thus, instead of maintaining a shortest paths tree up to depth n in H, we only need to maintain a shortest paths tree in H'' up to depth $n/\varphi = p\sqrt{n}/\epsilon$. In this way, we obtain a $(1 + 3\epsilon)$-approximation for all nodes such that $\text{dist}_G(u, v) \geq D$.

However, we cannot simply use the ES-tree on H'', because as edges are deleted from G, nodes might join the approximate balls and therefore edges might be inserted into F and thus into H''. This means that a dynamic shortest paths algorithm running on H'' would not be situated in a purely decremental setting. However, the insertions have a "nice" structure. We can deal with them by using a previously developed technique, called *monotone ES-tree* [20]. The main idea of the monotone ES-tree is to ignore the level decreases made possible by inserting edges. The hop-set proof still goes through, even though we are not arguing about the current distance in H'' anymore, but the level of a node u in the monotone ES-tree. Maintaining the monotone ES-tree for distances up to D in H'' takes time $O(\mathcal{E}(H'')D)$ where $\mathcal{E}(H'')$ is the number of edges ever contained in H'' (including edges that are inserted over time) and $D = O(n^{1/2+1/p})$ as explained above. Each insertion of an edge into F corresponds to a node joining $Ball(u, D)$ for some node u. For a fixed node u of priority i there are at most $\log D$ possibilities for nodes to join $Ball(u, D)$ (namely each time $\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor$ increases) and every time at most $m^{(i+1)/p}$ nodes will join in expectation. It follows that $\mathcal{E}(H)$ is $m^{1+o(1)}$ in expectation and the running time of this step is $m^{1+1/2+o(1)}$ in expectation.

The almost linear-time algorithm is just slightly more complicated. Here, we use $p = \Theta(\sqrt{\log n})$ priorities and a hierarchy of $q = \sqrt{p}$ hop reductions. We further set $\Delta_k = n^{k/q}$ for each $0 \leq k \leq q - 2$. In the algorithm, we will maintain, for each $0 \leq k \leq q - 2$, a hop set F_k such that for every pair of nodes u and v with $\Delta_{k+1} \leq \text{dist}_G(u, v) \leq \Delta_{k+2}$ there is a path from u to v in $H_k = (V, F_k)$ of weight at most $(1 + 2\epsilon)\text{dist}_G(u, v)$ and with at most $p\ \text{dist}_G(u, v)/\Delta_k \leq pn^{2/q}$ hops. To achieve this, we use the following hierarchical approach. Given the hop set F_k, we can maintain approximate shortest

paths up to depth Δ_{k+2} in time $m^{1+o(1)}$ and given a data structure for maintaining approximate shortest paths up to depth Δ_k, we can maintain approximate balls and thus the hop set F_{k+1} in time $m^{1+o(1)}$. The hierarchy "starts" with using the ES-tree as an algorithm for maintaining an (exact) shortest paths tree up to depth $n^{2/q}$. Thus, running efficient monotone ES-trees on top of the hop sets and maintaining the hop sets (using efficient monotone ES-trees) go hand in hand.

There are two obstacles in implementing this hierarchical approach when we want to maintain the approximate balls in each of the q layers of the hierarchy. First, in our algorithm for maintaining the approximate balls sketched above, we have used the ES-tree as an exact decremental SSSP algorithm. In the hierarchical approach, we have to replace the ES-tree with the monotone ES-tree, which only provides approximate distance estimates. This will lead to approximation errors that increase with the number of layers. Second, by the arguments above the number of edges in F_k is $O(m^{1+1/p})$ for each $0 \leq k \leq q-2$. In the algorithm for maintaining the approximate balls for the next layer, this bound, however, is not good enough, because we run a separate instance of the monotone ES-tree for each node u. We deal with this issue by running the monotone ES-tree in the subgraph of G induced by the nodes initially contained in $Ball(u)$. For a node u of priority i this subgraph contains $m_i = m^{(i+1)/p}$ edges in expectation, and we can recursively run our algorithm on this smaller graph. By this process, we incur a factor of $m^{1/p}$ in the running time each time we increase the depth of the recursion. This results in a total update time of $m^{1+O(q/p)}$, which is $m^{1+O(1/q)} = m^{1+o(1)}$, since $q = \sqrt{p}$.

Extension to Weighted Graphs. The hop set construction described above only works for unweighted graphs. However, the main property that we needed was $\text{dist}_G(u, v) \leq n$ for any pair of nodes u and v. Using the rounding technique mentioned above, we can construct for each $0 \leq i \leq \lfloor \log nW \rfloor$ a graph G_i such that for all pairs of nodes u and v with $2^i \leq \text{dist}_G(u, v) \leq 2^{i+1}$ we have $\text{dist}_{G_i}(u, v) \leq 4n/\epsilon$ and the shortest path in G_i can be turned into a $(1+\epsilon)$-approximate shortest path in G by scaling up the edge weights. We now run $O(\log(nW))$ instances of our algorithm, one for each graph G_i, and maintain the hop set and approximate SSSP for each of them.

We only need to refine the analysis of the hop-set property in the following way. Remember that in the analysis we considered the shortest path π from u to v and defined the node u' that is at distance Δ from u on π. If the hop set contained the edge (u, u'), then we could reduce the distance to v by Δ. In weighted graphs (even after the scaling), we cannot guarantee there is a node at distance exactly Δ from u on π. Therefore, we define u' as the furthest node that is at distance *at most* Δ from u on π. Furthermore, we define u'' as the neighbor of u' on π, i.e., u'' is at distance *at least* Δ from u. Now, if the hop set contains the edge (u, u'), then we first use the edge (u, u') from the hop set and then the edge (u', u'') from the original graph to reduce the distance to v by at least Δ with only 2 hops. Note that for unweighted graphs it was sufficient to only use the edges of the hop set. For weighted graphs, we really have to add the edges of the hop set to the original graph in our algorithm.

4 FROM APPROXIMATE SSSP TO APPROXIMATE BALLS

In the following, we show how to maintain the approximate balls of every node if we already have an algorithm for maintaining approximate shortest paths. In our reduction, we will use the algorithm for maintaining approximate shortest paths as a "black box," requiring only very few properties.

We can view the balls as a distance oracle with exponentially increasing stretch. Similar to other distance oracles, we assign integer values called priorities to the nodes and ensure that the balls have the following structural property: for every pair of nodes u and v, either v is in the ball of u, or there is some node v' close to u that has higher priority than u. In the first case, we have found

an estimate of the distance between u and v as the approximate shortest path algorithm is used to maintain an estimate of the distance between u and all nodes in its ball. In the second case, we repeat the process for the nodes v' and u, incurring the "detour" of going from u to v' first. As the number of priorities is limited, this strategy succeeds eventually. In our analysis, we explicitly bound the distance between u and v' (and thus the weight of the "detour") by a function $s(x, l)$, where x is the distance between u and v and l is the difference in priorities between u and v'. In Section 6 it will become clear why our bound on $s(x, l)$ is good enough for our purposes of using the balls as a hop set and as a distance oracle, respectively. In addition to this structural property, we need to bound the total size of the balls to obtain useful applications. This bound can be ensured by an appropriate randomized assignment of priorities, with some complications arising from the fact that the black box decremental SSSP algorithm does not provide exact distances. This also helps for bounding the total update time for dynamically maintaining the balls. Formally, we prove the following statement in this section.

Proposition 4.1. *Assume there is a decremental approximate SSSP algorithm ApproxSSSP with the following properties, using fixed values $\alpha \geq 1$, $\beta \geq 0$, and $D \geq 1$: Given a weighted graph $G = (V, E)$ undergoing edge deletions and edge weight increases with n nodes and initially m edges, and a fixed source node $s \in V$, ApproxSSSP maintains, in total update time $T(m, n)$, for every node $v \in V$ a distance estimate $\delta(s, v)$, such that:*

- ***A1*** *$\delta(s, v) \geq \operatorname{dist}_G(s, v)$.*
- ***A2*** *If $\operatorname{dist}_G(s, v) \leq D$, then $\delta(s, v) \leq \alpha \operatorname{dist}_G(s, v) + \beta$.*
- ***A3*** *After every update in G, ApproxSSSP returns, for every node v such that $\delta(s, v)$ has changed, v together with the new value of $\delta(s, v)$.*

Then there is a decremental algorithm ApproxBalls for maintaining approximate balls with the following properties: Given a weighted graph $G = (V, E)$ undergoing edge deletions and edge weight increases with n nodes and initially m edges, and parameters $k \geq 2$ and $0 < \epsilon \leq 1$, it assigns to every node $u \in V$ a number from 0 to $k - 1$, called the priority *of u, and maintains for every node $u \in V$ a set of nodes $B(u)$ and a distance estimate $\delta(u, v)$ for every node $v \in B(u)$, such that:*

- ***B1*** *For every node u and every node $v \in B(u)$, we have $\operatorname{dist}_G(u, v) \leq \delta(u, v) \leq \alpha \operatorname{dist}_G(u, v) + \beta$.*
- ***B2*** *For all $x \geq 0$, set $s(x, 0) = x$, and for all $x \geq 0$ and $l \geq 1$, set*

$$s(x, l) = a(a + 1)^{l-1}x + ((a + 1)^l - 1)b/a,$$

  where $a = (1 + \epsilon)\alpha$ and $b = (1 + \epsilon)\beta$. Then for every $0 \leq i \leq k - 1$, every node u of priority i, and every node v such that $s(\operatorname{dist}_G(u, v), k - 1 - i) \leq D$, either (1) $v \in B(u)$ or (2) there is some node v' of priority $j > i$ such that $u \in B(v')$ and $\operatorname{dist}_G(u, v') \leq s(\operatorname{dist}_G(u, v), j - i)$.
- ***B3*** *In expectation, $\sum_{u \in V} \mathcal{B}(u) = O(km^{1+1/k} \log D/\epsilon)$, where $\mathcal{B}(u)$ denotes the number of nodes ever contained in $B(u)$ over the sequence of updates to G.*
- ***B4*** *The total update time of ApproxBalls is*

$$t(m, n, k, \epsilon) = O\left(\left(km^{1+1/k} + \sum_{0 \leq i \leq k-1} \frac{m}{m^{i/k}} \cdot T(m_i, n_i)\right) \cdot \log n \frac{\log D}{\epsilon} + k \cdot T(m, n)\right)$$

  in expectation, where, for each $0 \leq i \leq k - 1$, $m_i = O(m^{(i+1)/k})$ and $n_i = O(m^{(i+1)/k})$.
- ***B5*** *After every update in G, ApproxBalls returns all pairs of nodes u and v such that v joins $B(u)$, v leaves $B(u)$, or $\hat{\delta}(u, v)$ changes.*

Note that by our definition of $s(x,l)$, we have $s(x,1) = ax + b$ for all $x \geq 0$ and $s(x,l+1) = (a+1)s(x,l) + b$ for all $x \geq 0$ and $l \geq 1$.

Our algorithm for maintaining the approximate balls $B(u)$ for every node $u \in V$ is as follows:

(1) At the initialization, we set $F_0 = E$ and $F_k = \emptyset$ and for $1 \leq i \leq k-1$, a set of edges F_i is obtained from sampling each edge of E independently with probability $1/m^{i/k}$. For every $0 \leq i \leq k-1$, we set $A_i = \{v \in V \mid \exists (v,w) \in F_i\}$ and for every node $v \in V$, we set the *priority of* u to be the maximum i such that $v \in A_i$.

(2) For each $1 \leq i \leq k-1$, we run an instance of ApproxSSSP from an artificial source node s_i that has an edge of weight 0 to every node in A_i. We denote by $\delta(u, A_i)$ the distance estimate provided by ApproxSSSP and set $\delta(u, A_k) = \infty$ for every node $u \in V$.

(3) For every $0 \leq i \leq k-1$ and every node $u \in V$ of priority i, we maintain the value

$$r(u) = \min\left(\frac{(1+\epsilon)^{\lfloor \log_{1+\epsilon} \delta(u,A_{i+1}) \rfloor} - \beta}{\alpha}, D+1\right)$$

and at the initialization and each time $r(u)$ increases we do the following:

(a) Compute the set of nodes $R(u) = \{v \in V \mid \text{dist}_G(u,v) < r(u)\}$.

(b) Run an instance of ApproxSSSP from u in $G|R(u)$, the subgraph of G induced by $R(u)$. Let $\delta(u,v)$ denote the estimate of the distance between u and v in $G|R(u)$ maintained by ApproxSSSP.

(c) Maintain $B(u) = \{v \in V \mid \delta(u,v) \leq \alpha D + \beta\}$: every time $\delta(u,v)$ changes for some node v, we check whether the inequality $\delta(u,v) \leq \alpha D + \beta$ still holds, and if not, then we remove v from $B(u)$.

Note that ApproxBalls has Property **B5**, i.e., it returns changes in the approximate balls and the distance estimates, which is possible, because ApproxSSSP has Property **A3**.

4.1 Relation to Exact Balls

In the following, we compare the approximate balls maintained by our algorithm to the exact balls, as used by Thorup and Zwick [36]. We show how the main properties of exact balls translate to approximate balls. We use the following definition of the (exact) ball of a node u of priority i:

$$Ball(u) = \{v \in V \mid \text{dist}_G(u,v) < \text{dist}_G(u, A_{i+1})\}.$$

The balls have the following simple property: If $v \notin Ball(u)$, then there is a node v' of priority $j > i$ such that $\text{dist}_G(u,v') \leq \text{dist}_G(u,v)$. We show that a relaxed version of this statement also holds for the approximate balls.

Lemma 4.2. *Let $0 \leq i \leq k-1$, let u be a node of priority i, and let v be a node such that $\text{dist}_G(u,v) \leq D$. If $v \notin B(u)$, then there is a node v' of priority $j > i$ such that $\text{dist}_G(u,v') \leq a\,\text{dist}_G(u,v) + b$, where $a = (1+\epsilon)\alpha$ and $b = (1+\epsilon)\beta$.*

Proof. We show the following: If $\text{dist}_G(u, A_{i+1}) > a\,\text{dist}_G(u,v) + b$, then $v \in B(u)$. The claim then follows from contraposition: If $v \notin B(u)$, then $\text{dist}_G(u, A_{i+1}) \leq a\,\text{dist}_G(u,v) + b$, and thus there exists some node $v' \in A_{i+1}$ (of priority $j \geq i+1$) such that $\text{dist}_G(u,v') \leq a\,\text{dist}_G(u,v) + b$.

Assume that $\text{dist}_G(u, A_{i+1}) \geq a\text{dist}_G(u,v) + b$. Since $\delta(u, A_{i+1}) \geq \text{dist}_G(u, A_{i+1})$, by Property **A1**, we have

$$\delta(u, A_{i+1}) \geq \text{dist}_G(u, A_{i+1}) > a\,\text{dist}_G(u,v) + b = (1+\epsilon)(\alpha\,\text{dist}_G(u,v) + \beta),$$

which is equivalent to

$$\text{dist}_G(u,v) < \frac{\frac{\delta(u,A_{i+1})}{1+\epsilon} - \beta}{\alpha}.$$

Since

$$(1+\epsilon)^{\lfloor \log_{1+\epsilon} \delta(u,A_{i+1}) \rfloor} \geq (1+\epsilon)^{\log_{1+\epsilon} \delta(u,A_{i+1})-1} = \frac{\delta(u,A_{i+1})}{1+\epsilon},$$

it follows that

$$\text{dist}_G(u,v) < \frac{(1+\epsilon)^{\lfloor \log_{1+\epsilon} \delta(u,A_{i+1}) \rfloor} - \beta}{\alpha}.$$

Since we have assumed that $\text{dist}_G(u,v) \leq D$, we get $\text{dist}_G(u,v) < r(u)$ by the definition of $r(u)$. The latter inequality also holds for all nodes on a shortest path π from u to v in G, and, as distances in G are non-decreasing, this in particular is true at the last point in time where $r(u)$ has changed. Therefore, all nodes of π are contained in $R(u)$, which implies that $\text{dist}_{G|R(u)}(u,v) = \text{dist}_G(u,v) \leq D$. Thus, by Property A2, it follows that $\delta(u,v) \leq \alpha \, \text{dist}_{G|R(u)}(u,v) + \beta \leq \alpha D + \beta$, i.e., $v \in B(u)$, as desired. □

We now show that the approximate balls are contained in the exact balls. The exact balls are useful in our analysis, because we can easily bound their size.

Lemma 4.3. *At any time $B(u) \subseteq Ball(u)$ for every node u.*

Proof. Let $R(u) = \{v \in V \mid \text{dist}_G(u,v) < r(u)\}$ denote the set of nodes at distance of at most $r(u)$ from u at the last time $r(u)$ has increased. Note that $B(u)$ is a set of nodes of the graph $G|R(u)$, and therefore $B(u) \subseteq R(u)$. It remains to show that $R(u) \subseteq Ball(u)$.

Let $v \in R(u)$ and let i be the priority of u. If $i = k-1$, then the claim is trivially true, because $Ball(u)$ contains all nodes that are connected to u in G. Consider, thus, the case $0 \leq i < k-1$. In the case $0 \leq i < k-1$, remember that if $\text{dist}_G(u,A_{i+1}) \geq r(u)$, then we trivially have $\text{dist}_G(u,v) < r(u) \leq \text{dist}_G(u,A_{i+1})$. If, however, $\text{dist}_G(u,A_{i+1}) < r(u)$, then in particular $\text{dist}_G(u,A_{i+1}) \leq D$ by the definition of $r(u)$ and by Property A2, we have $\delta(u,A_{i+1}) \leq \alpha \, \text{dist}_G(u,A_{i+1}) + \beta$. It follows that

$$\begin{aligned}\text{dist}_G(u,v) < r(u) &\leq \frac{(1+\epsilon)^{\lfloor \log_{1+\epsilon} \delta(u,A_{i+1}) \rfloor} - \beta}{\alpha} \\ &\leq \frac{(1+\epsilon)^{\log_{1+\epsilon} \delta(u,A_{i+1})} - \beta}{\alpha} = \frac{\delta(u,A_{i+1}) - \beta}{\alpha} \leq \text{dist}_G(u,A_{i+1}).\end{aligned}$$

In both cases, we get $\text{dist}_G(u,v) < \text{dist}_G(u,A_{i+1})$, and as this is the defining property of $Ball(u)$, we have $v \in Ball(u)$. □

Lemma 4.4. *At any time, for every $0 \leq i \leq k-1$ and every node u of priority i, we have $|Ball(u)| = O(m^{(i+1)/k})$ and $|E[Ball(u)]| = O(m^{(i+1)/k})$ in expectation over all random choices in sampling the set F_{i+1}.*

Proof. The claim is trivially true if $Ball(u) = \{u\}$, and we thus assume that $Ball(u) \supset \{u\}$ in the following. It further suffices to prove that $|E[Ball(u)]| = O(m^{(i+1)/k})$ as $|Ball(u)| \leq 2|E[Ball(u)]|$ and, as the claim immediately holds for $i = k-1$, we only need to consider the case $i < k-1$. For every edge $e = (v,w) \in E$, we define $\text{dist}_G(u,e) = \min(\text{dist}_G(u,v), \text{dist}_G(u,w))$. Order the edges of the graph according distance from u under to this definition of $\text{dist}_G(u,e)$ for each edge e, where ties are broken in an arbitrary but fixed order. Let $e = (u,v)$ be the first edge of F_{i+1} in this order and let $E' \subseteq F_i$ be the set of edges that are strictly smaller than e in this order. As each edge of the graph is contained in F_{i+1} with probability $1/m^{(i+1)/k}$ independently, we have $|E'| \leq m^{(i+1)/k}$ in expectation.

Assume without loss of generality that $\text{dist}_G(u,v) \leq \text{dist}_G(u,w)$, i.e., $\text{dist}_G(u,e) = \text{dist}_G(u,v)$. Let $v' \in Ball(u)$ and let $e' = (v',w')$ be some edge incident to v'; such an edge must exist, because

$Ball(u) \supset \{u\}$. Since the node v is contained in A_{i+1}, we have

$$\text{dist}_G(u, e') \leq \text{dist}_G(u, v') < \text{dist}_G(u, A_{i+1}) \leq \text{dist}_G(u, v) = \text{dist}_G(u, e),$$

which implies $e' \in E'$. It follows that $E[Ball(u)] \subseteq E'$ and thus $|E[Ball(u)]| \leq |E'| \leq m^{(i+1)/k}$ in expectation, as desired. □

4.2 Properties of Approximate Balls

We now show that the approximate balls and the corresponding distance estimates have Properties **B1**–**B4**. We first show that the distance estimates for nodes in the approximate balls have the desired approximation guarantee, although they have been computed in subgraphs of G.

Lemma 4.5 (Property **B1**). *For every pair of nodes u and v such that $v \in B(u)$, we have* $\text{dist}_G(u, v) \leq \delta(u, v) \leq \alpha\, \text{dist}_G(u, v) + \beta$.

Proof. By Property **A1**, we have $\delta(u, v) \geq \text{dist}_{G|R(u)}(u, v)$, and since $G|R(u)$ is a subgraph of G, we have $\text{dist}_{G|R(u)}(u, v) \geq \text{dist}_G(u, v)$. Therefore, the inequality $\delta(u, v) \geq \text{dist}_G(u, v)$ follows.

Since $v \in B(u)$, we have $\delta(u, v) \leq \alpha D + \beta$. If $\text{dist}_G(u, v) \geq D$, then trivially $\delta(u, v) \leq \alpha D + \beta \leq \alpha \text{dist}_G(u, v) + \beta$. If $\text{dist}_G(u, v) < D$, then there is a path π from u to v in G of weight at most D. This path was also contained in previous versions of G, possibly with smaller weight. In particular, π was also contained in the version of G at the last point in time for which the set $R(u)$ was recomputed. Since $v \in B(u) \subseteq R(u)$, we therefore also have $v' \in R(u)$ for every node v' on π. It follows that π is contained in $G|R(u)$ and thus $\text{dist}_{G|R(u)}(u, v) = \text{dist}_G(u, v) \leq D$. By Property **A2**, we then have $\delta(u, v) \leq \alpha \text{dist}_{G|R(u)}(u, v) + \beta = \alpha \text{dist}_G(u, v) + \beta$. □

We show now that the approximate balls have a certain structural property that either allows us to shortcut the path between two nodes or helps us in finding a nearby node of higher priority.

Lemma 4.6 (Property **B2**). *For all $x \geq 0$, set $s(x, 0) = x$, and for all $x \geq 0$ and $l \geq 1$, set*

$$s(x, l) = a(a + 1)^{l-1}x + ((a + 1)^l - 1)b/a,$$

where $a = (1 + \epsilon)\alpha$ and $b = (1 + \epsilon)\beta$. Then for every $0 \leq i \leq k - 1$, every node u of priority i, and every node v such that $s(\text{dist}_G(u, v), k - 1 - i) \leq D$, either (1) $v \in B(u)$ or (2) there is some node v' of priority $j > i$ such that $u \in B(v')$ and $\text{dist}_G(u, v') \leq s(\text{dist}_G(u, v), j - i)$.

Proof. As noted above, by our definition of $s(x, l)$, we have $s(x, 1) = ax + b$ for all $x \geq 0$ and $s(x, l + 1) = (a + 1)s(x, l) + b$ for all $x \geq 0$ and $l \geq 1$. Note that since $s(\cdot, \cdot)$ is non-decreasing in its second argument, we have, for all $0 \leq l \leq k - 1 - i$, $s(\text{dist}_G(u, v), l) \leq s(\text{dist}_G(u, v), k - 1 - i) \leq D$.

If $v \in B(u)$, then we are done. Otherwise, by Lemma 4.2, there is some node v_1 of priority $j_1 \geq i + 1$, such that

$$\text{dist}_G(v_1, u) \leq a\, \text{dist}_G(u, v) + b = s(\text{dist}_G(u, v), 1) \leq D.$$

Thus, if $u \in B(v_1)$, then we are done. Otherwise, by Lemma 4.2, there is some node v_2 of priority $j_2 \geq j_1 + 1 \geq i + 2$, such that

$$\text{dist}_G(v_2, v_1) \leq a\, \text{dist}_G(v_1, u) + b.$$

By the triangle inequality, we have

$$\begin{aligned}
\text{dist}_G(v_2, u) &\leq \text{dist}_G(v_2, v_1) + \text{dist}_G(v_1, u) \\
&\leq a\, \text{dist}_G(v_1, u) + b + \text{dist}_G(v_1, u) \\
&= (a + 1)\text{dist}_G(v_1, u) + b \\
&\leq (a + 1)s(\text{dist}_G(u, v), 1) + b \\
&= s(\text{dist}_G(u, v), 2) \leq D.
\end{aligned}$$

We now repeat this argument to obtain nodes $v_1, v_2, \ldots, v_l$ of priorities $j_1, j_2, \ldots, j_l$, such that $j_l \geq i + l$ and

$$\text{dist}_G(v_l, u) \leq s(\text{dist}_G(u, v), l) \leq D$$

until $u \in B(v_l)$. This happens eventually, since $A_k = \emptyset$, and thus for any node v_l of priority $k-1$ such that $\text{dist}_G(v_l, u) \leq s(\text{dist}_G(u, v), l) \leq D$, the following reasoning applies: Since $\delta(v_l, A_k) = \infty$, we always have $r(v_l) = D + 1$ and thus $\text{dist}_G(v_l, u) < r(v_l)$. The latter inequality also holds for all nodes on a shortest path π from u to v in G, and, as distances in G are non-decreasing, this in particular is true at the last point in time where $r(v_l)$ has changed. Therefore, all nodes of π are contained in $R(v_l)$, which implies that $\text{dist}_{G|R(v_l)}(v_l, u) = \text{dist}_G(v_l, u)$. Thus, by Property A2, it follows that $\delta(v_l, u) \leq \alpha\ \text{dist}_{G|R(v_l)}(v_l, u) + \beta \leq \alpha D + \beta$, i.e., $u \in B(v_l)$, as desired. □

Next, we bound the size of the system of approximate balls we maintain. Here, we use the fact that we can easily bound the size of the exact ball $Ball(u)$ for every node u and that by our definitions we ensure that the approximate balls are subsets of the exact balls.

Lemma 4.7 (Size of Approximate Balls (Property **B3**)). *In expectation, we have* $\sum_{u \in V} \mathcal{B}(u) = O(km^{1+1/k} \log D/\epsilon)$, *where* $\mathcal{B}(u)$ *denotes the number of nodes ever contained in* $B(u)$ *over the sequence of updates to* G.

Proof. We first bound $\mathcal{B}(u)$, the number of nodes ever contained in the approximate ball $B(u)$, of some node u. Let i denote the priority of u. Remember that nodes are joining $B(u)$ only when $r(u)$ increases and that

$$r(u) = \min\left(\frac{(1+\epsilon)^{\lfloor \log_{1+\epsilon} \delta(u, A_{i+1}) \rfloor} - \beta}{\alpha}, D + 1\right).$$

Thus, $r(u)$ can only increase if $\lfloor \log_{1+\epsilon} \delta(u, A_{i+1}) \rfloor$ increases and the left term in the minimum is at most $D + 1$. Since $1 + \epsilon \geq 1$ it follows that $r(u)$ increases only $O(\log_{1+\epsilon} D) = O(\log D/\epsilon)$ times, once it has non-negative value. As $B(u) \subseteq Ball(u)$ by Lemma 4.3, after every increase of $r(u)$ only nodes contained in $Ball(u)$ can join $B(u)$. By Lemma 4.4 the size of $Ball(u)$ is $O(m^{(i+1)/k})$ in expectation over all random choices in sampling the set F_{i+1}. Thus, the number of nodes ever contained in $B(u)$ is $\mathcal{B}(u) = O(m^{(i+1)/k} \log D/\epsilon)$ in expectation.

As the number of nodes of priority i is $O(m/m^{i/k})$ in expectation over all random choices in sampling the set F_i, the number of nodes ever contained in the approximate balls is $\sum_{u \in V} \mathcal{B}(u) = O(km^{1+1/k} \log D/\epsilon)$ in expectation. Here, we use that F_i and F_{i+1} are sampled independently. □

Finally, we analyze the running time of our algorithm for maintaining the approximate balls. Since we use the data structure ApproxSSSP as a black box, the running time of our algorithm depends on the running time of ApproxSSSP.

Lemma 4.8 (Running Time (Property **B4**)). *The total time needed for maintaining the sets* $B(u)$ *for all nodes* $u \in V$ *is*

$$O\left(\left(km^{1+1/k} + \sum_{0 \leq i \leq k-1} \frac{m}{m^{i/k}} \cdot T(m_i, n_i)\right) \cdot \log n \frac{\log D}{\epsilon} + k \cdot T(m, n)\right)$$

in expectation, where, for each $0 \leq i \leq k-1$, $m_i = O(m^{(i+1)/k})$ *and* $n_i = O(m^{(i+1)/k})$.

Proof. The initialization in Step 1 of the algorithm, where we determine the sets $A_0, \ldots, A_k$ takes time $O(km + n)$. In Step 2, we run for each $1 \leq i \leq k-1$ an instance of ApproxSSSP with depth D. This takes time $kT(m, n)$. Step 3, where we maintain for every $0 \leq i \leq k-1$ and every node u of priority i the approximate ball and corresponding distance estimates, can be analyzed as

follows. Remember that every time $r(u)$ increases, we first compute $R(u)$, the set of nodes that are at distance of at most $r(u)$ from u. Using an implementation of Dijkstra's algorithm that only puts nodes into its queue upon their first visit, this takes time $O(|E[R(u)]| \log n)$, where $E[R(u)]$ is the set of edges incident to $R(u)$. By Lemma 4.3, we have $R(u) \subseteq Ball(u)$, and by Lemma 4.4, we have $|Ball(u)| = O(m^{(i+1)/k})$ and $|E[Ball(u)]| = O(m^{(i+1)/k})$ in expectation over all random choices in sampling the set F_{i+1}. Thus, computing $R(u)$ takes time $O(m^{(i+1)/k} \log n)$ in expectation. We then maintain an instance of ApproxSSSP up to depth D on $G|R(u)$, the subgraph of G induced by $R(u)$. Note that $G|R(u)$ has at most $m_i = O(m^{(i+1)/k})$ edges and $n_i = O(m^{(i+1)/k})$ nodes in expectation and therefore this takes time $T(m_i, n_i)$ in expectation. As $r(u)$ increases $O(\log D/\epsilon)$ times and the number of nodes of priority i is at most $O(m/m^{i/k})$ in expectation over all random choices in sampling the set F_i, Step 3 takes time

$$O\left(\sum_{0 \le i \le k-1} \left(T(m_i, n_i) + m^{(i+1)/k} \log n\right) \cdot \frac{m}{m^{i/k}} \frac{\log D}{\epsilon}\right)$$

in expectation. Now the claimed total running time claimed for all three steps follows. □

5 FROM APPROXIMATE BALLS TO APPROXIMATE SSSP

In the following, we show how to maintain an approximate shortest paths tree if we already have an algorithm for maintaining approximate balls. Our main tool in this reduction is a hop set that we define from the approximate balls. We will add the "shortcut" edges of the hop set to the graph and scale down the edge weights, maintaining the approximate shortest paths with a monotone ES-tree.

The main challenge in this approach is bounding the approximation guarantee of the hop set. While this encompasses a proof of the approximation guarantee of the static variant of the hop set, such a "static" proof is not sufficient in the decremental setting; the monotone ES-tree needs to exploit the additional structure of the approximate balls that define the hop set. In particular, the following structural property is useful here: for every pair of nodes u and v, either v is in the ball of u, or there is some node v' close to u that has higher priority than u. If the distance between u and v' is measured by some function $s(x, l)$, where x is the distance between u and v' and l is the difference in priorities between v' and u, then we can give an upper bound on the value of $s(x, l)$ such that our approximation guarantee proof still goes through. Concerning the running time, there mainly are two factors that affect the running time of the monotone ES-tree. The first factor is the size of the hop set, more precisely the total number of edges ever contained in the hop set over all updates to the input graph. This number is bounded by the by the total size of the (approximate) balls, more precisely by the total number of nodes ever contained in the balls. The second factor is the the maximum depth considered in the tree. The maximum depth can be kept relatively small by rounding and scaling down all edge weights. This gives some additional approximation error to account for, but by the right choice of parameters, we can balance both costs at a moderate level. Formally, we prove the following statement in this section.

Proposition 5.1. *Assume there is a decremental algorithm ApproxBalls for maintaining approximate balls with the following properties, using fixed values $a \ge \alpha \ge 1$, $b \ge \beta \ge 0$, and $\hat{D} \ge 1$. Given a weighted graph $G = (V, E)$ undergoing edge deletions and edge weight increases with n nodes and initially m edges, and a parameter $k \ge 2$, it assigns to every node $u \in V$ a number from 0 to $k - 1$, called the* priority *of u, and maintains, in total update time $t(m, n, k)$, for every node $u \in V$ a set of nodes $B(u)$ and, for every node $v \in B(u)$, a distance estimate $\hat{\delta}(u, v)$, such that:*

B1 *For every node u and every node $v \in B(u)$, we have* $\text{dist}_G(u,v) \leq \hat{\delta}(u,v) \leq \alpha\, \text{dist}_G(u,v) + \beta$.

B2 *There is a non-decreasing function $s(\cdot,\cdot)$ such that, for all $x \geq 0$, $s(x,0) \leq x$ and $s(x,1) \leq ax + b$ for some $a \geq \alpha$ and $b \geq \beta$ and, for all $l \geq 1$,*

$$s(x, l+1) \leq (\alpha + 1 + \epsilon)(\alpha a s(x,l) + \alpha b + \beta) + \beta,$$

guaranteeing the following: For every $0 \leq i \leq k-1$, every node u of priority i and every node v such that $s(\text{dist}_G(u,v), k-1-i) \leq \hat{D}$, either (1) $v \in B(u)$ or (2) there exists some node $v' \in V$ of priority $j > i$ such that $u \in B(v')$ and $\text{dist}_G(u,v') \leq s(\text{dist}_G(u,v'), j-i)$.

B3 *After every update in G, ApproxBalls returns all pairs of nodes u and v such that v joins $B(u)$, v leaves $B(u)$, or $\hat{\delta}(u,v)$ changes.*

Then, there is an approximate SSSP data structure ApproxSSSP with the following properties: Given a weighted graph G undergoing edge deletions and edge weight increases with n nodes and initially m edges, a fixed source node s, and parameters p, Δ, D, and ϵ such that

$$2 \leq p \leq \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4a^3}{\epsilon}\right)}},$$

$\Delta \geq b, n^{1/p}\Delta \leq \hat{D}, D \geq \Delta$ and $0 < \epsilon \leq 1$, it maintains a distance estimate $\delta(s,v)$ for every node $v \in V$, such that:

A1 $\delta(s,v) \geq \text{dist}_G(s,v)$.

A2 *If* $\text{dist}_G(s,v) \leq D$, *then* $\delta(s,v) \leq (\alpha + 2\epsilon)\text{dist}_G(s,v) + \epsilon n^{1/p}\Delta$.

A3 *The total update time of ApproxSSSP is*

$$T(m,n,\Delta,D,\epsilon) = t(m,n,p) + O\left(p\left(\alpha D/\Delta + n^{1/p}\right)\left(m + \sum_{u \in V} \mathcal{B}(u)\right)/\epsilon + n\right),$$

where $\mathcal{B}(u)$ denotes the number of nodes ever contained in $B(u)$ over the sequence of updates to G.

A4 *After every update in G, ApproxSSSP returns each node v such that $\delta(s,v)$ has changed together with the new value of $\delta(s,v)$.*

We assume without loss of generality that the distance estimate maintained by ApproxBalls is non-decreasing. If ApproxBalls ever reports a decrease, then we can ignore it, because then Property B1 will still hold as distances in G are non-decreasing under edge deletions and edge weight increases.

5.1 Algorithm Description

The algorithm ApproxSSSP maintains the set of edges $F = \{(u,v) \in V^2 \mid v \in B(u)\}$ such that, for every node u and every node $v \in B(u)$, the edge (u,v) has weight $w_F(u,v) = \min(\hat{\delta}(u,v), \hat{\delta}(v,u))$ if also $u \in B(v)$ and $w_F(u,v) = \hat{\delta}(u,v)$, otherwise. We update F every time in the algorithm ApproxBalls a node joins or leaves an approximate ball or if the distance estimate $\hat{\delta}(u,v)$ increases for some pair of nodes u and v. By Property B3 this information is returned by ApproxBalls after every update in G. Thus, the set of edges F undergoes insertions, deletions, and weight increases.

In the following, we will define a shortcut graph H'' with scaled-down edge weights and our algorithm ApproxSSSP will simply run a monotone ES-tree [20] from s in H''. The monotone ES-tree has property **A1**, which is apparent from the pseudocode provided in Algorithm 1. We denote

the weight of an edge (u, v) in G by $w_G(u, v)$ and define H as a graph that has the same nodes as G and contains all edges of G and F that have weight at most $D + n^{1/p}\Delta$. We set the weight of every edge (u, v) in H to $w_H(u, v) = \min(w_G(u, v), w_F(u, v))$. We set

$$\varphi = \frac{\epsilon\Delta}{p+1}$$

and define H' as the graph that has the same nodes and edges as H and in which every edge (u, v) has weight

$$w_{H'}(u, v) = \left\lceil \frac{w_H(u, v)}{\varphi} \right\rceil \cdot \varphi,$$

i.e., we round every edge weight to the next multiple of φ. Furthermore, we define H'' as the graph that has the same nodes and edges as H' and in which every edge (u, v) has weight

$$w_{H''}(u, v) = \frac{w_{H'}(u, v)}{\varphi} = \left\lceil \frac{w_H(u, v)}{\varphi} \right\rceil,$$

i.e., we scale down every edge weight by a factor of $1/\varphi$. We maintain a monotone ES-tree with maximum level

$$L = (\alpha + 2\epsilon)D/\varphi + (p+1)n^{1/p}$$

from s and denote the level of a node v in this tree by $\ell(v)$. For every node v our algorithm returns the distance estimate $\delta(s, v) = \ell(v) \cdot \varphi$. Note that the graph H'' has integer edge weights and, as F might undergo insertions, deletions, and edge weight increases, the same type of updates might occur in H''. Furthermore, observe that the rounding guarantees that

$$w_H(u, v) \leq w_{H'}(u, v) \leq w_H(u, v) + \varphi$$

for every edge (u, v) of H'.

5.2 Running Time Analysis

We first provide the running time analysis. We run the algorithm in a graph in which we scale down the edge weights by a factor of φ and round up to the next integer. This makes the algorithm efficient.

Lemma 5.2 (Running Time (Property **A3**)). *The total update time of a monotone ES-tree with maximum level $L = (\alpha + 2\epsilon)D/\varphi + (p+1)n^{1/p}$ on H'' is*

$$O\left(p\left(\alpha D/\Delta + n^{1/p}\right)\left(m + \sum_{u \in V} \mathcal{B}(u)\right)/\epsilon + n\right),$$

where $\mathcal{B}(u)$ denotes the number of nodes ever contained in $B(u)$ over the sequence of updates to G.

Proof. By Lemma 2.2, the total time needed for maintaining the monotone ES-tree with maximum level L on H'' is

$$O(\mathcal{E}(H'') \cdot L + \mathcal{W}(H'') + n),$$

where $\mathcal{E}(H'')$ is the number of edges ever contained in H'' and $\mathcal{W}(H'')$ is the number of updates (i.e., edge deletions, edge weight increases, and edge insertions) to H''.

Remember that $\varphi = \epsilon\Delta/(p+1)$ and thus $L = O(p(\alpha D/(\epsilon\Delta) + n^{1/p}))$. We now bound $\mathcal{E}(H'')$ and $\mathcal{W}(H'')$. Note that at any time H'' has the same edges as H and each edge of H either is an edge from G, which contains m edges, or is an edge from F. As F is defined via the approximate balls (i.e., $(u, v) \in F$ if and only $v \in B(u)$), $\mathcal{E}(F)$, the number of edges ever contained in F, is at most $\sum_{u \in V} \mathcal{B}(u)$, the total number of nodes ever contained in the approximate balls. It follows that $\mathcal{E}(H'') \leq m + \mathcal{E}(F) \leq m + \sum_{u \in V} \mathcal{B}(u)$. For every edge counted by $\mathcal{E}(H'')$, we need to consider

at most one insertion and at most one deletion as well as at most $(D + n^{1/p}\Delta)/\varphi$ edge weight increases, since we have limited the maximum edge weight in H to $D + n^{1/p}\Delta$. Note that

$$(D + n^{1/p}\Delta)/\varphi = (D + n^{1/p}\Delta)(p+1)/(\epsilon\Delta) = O(p(D/\Delta + n^{1/p})/\epsilon).$$

Therefore, we have

$$\mathcal{W}(H'') \leq 2\mathcal{E}(H'') + \mathcal{E}(H'') \cdot (D + n^{1/p}\Delta)/\varphi = O\left(\left(m + \sum_{u \in V} \mathcal{B}(u)\right) \cdot p(D/\Delta + n^{1/p})/\epsilon\right).$$

We conclude that

$$\mathcal{E}(H'') \cdot L + \mathcal{W}(H'') = O\left(p(\alpha D/\Delta + n^{1/p})\left(m + \sum_{u \in V} \mathcal{B}(u)\right)/\epsilon\right),$$

and thus the claimed running time follows. □

5.3 Definitions of Values for Approximation Guarantee

Before we analyze the approximation guarantee, we define the most important values used in the analysis and provide bounds on their growth. We set

$$r_0 = \Delta,$$

and for every $0 \leq i \leq p-1$, we set

$$\begin{aligned} s_i &= ar_i + b, \\ w_i &= \alpha s_i + \beta, \text{ and} \\ r_i &= \frac{(\alpha + 1 + \epsilon)\sum_{0 \leq j \leq i-1} w_j + \beta}{\epsilon} (\text{if } i \geq 1). \end{aligned}$$

Intuitively, r_i is the distance by which we would like to shortcut the shortest path to the source node using a single hop-set edge for nodes of priority i. If this shortcut attempt fails, then s_i is the distance at which we would like to find a nearby node of priority $i+1$ and w_i is the weight of the hop-set edge to such a node.

We additionally set

$$\gamma_i = (\alpha + 1 + \epsilon) \sum_{i \leq j \leq p-2} w_j + \beta$$

for every $0 \leq i \leq p-1$, which equivalently can be obtained by setting $\gamma_{p-1} = \beta$ and $\gamma_i = \gamma_{i+1} + (\alpha + 1 + \epsilon)w_i$ for every $0 \leq i \leq p-2$. Finally, we set

$$\gamma = \gamma_0 + 2\epsilon\Delta.$$

Here γ_i is, intuitively speaking, the amount of additive error we will make on a hop-set path for a node of priority i and γ captures some additional rounding error for nodes of priority 0.

Lemma 5.3. *For all* $0 \leq i \leq p-1$, $\epsilon r_i = \gamma_0 - \gamma_i + \beta$.

Proof. Using the definition of γ_i, for all $0 \leq i \leq p-1$, we get

$$\gamma_0 - \gamma_i + \beta = (\alpha + 1 + \epsilon) \sum_{0 \leq j \leq p-2} w_j - (\alpha + 1 + \epsilon) \sum_{i \leq j \leq p-2} w_j + \beta = (\alpha + 1 + \epsilon) \sum_{0 \leq j \leq i-1} w_j + \beta = \epsilon r_i.$$

□

Lemma 5.4. $(4a^3/\epsilon)^p \le n^{1/p}$.

Proof. Remember that we have

$$p \le \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4a^3}{\epsilon}\right)}}.$$

We only need to rewrite both expressions as follows:

$$n^{1/p} = 2^{1/p \cdot \log n} \ge 2^{\frac{\sqrt{\log\left(\frac{4a^3}{\epsilon}\right)}}{\sqrt{\log n}} \cdot \log n} = 2^{\sqrt{\log\left(\frac{4a^3}{\epsilon}\right)} \cdot \sqrt{\log n}},$$

$$\left(\frac{4a^3}{\epsilon}\right)^p = 2^{p \cdot \log\left(\frac{4a^3}{\epsilon}\right)} \le 2^{\frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4a^3}{\epsilon}\right)}} \cdot \log\left(\frac{4a^3}{\epsilon}\right)} = 2^{\sqrt{\log n} \cdot \sqrt{\log\left(\frac{4a^3}{\epsilon}\right)}}.$$

□

Lemma 5.5. *For all* $0 \le i \le p-1$, *we have*

$$\sum_{0 \le j \le i} w_j \le \frac{(4^{i+1}-1)a^{3i+2}\Delta}{\epsilon^i}.$$

Proof. Remember that $\epsilon \le 1 \le \alpha \le a$ and $\beta \le b \le \Delta$. Now observe that for all $1 \le i \le p-1$, we have

$$r_i = \frac{(\alpha + 1 + \epsilon)\sum_{0 \le j \le i-1} w_j + \beta}{\epsilon} \le \frac{3a\sum_{0 \le j \le i-1} w_j + \Delta}{\epsilon},$$

and for all $0 \le i \le p-1$, we have

$$w_i = \alpha s_i + \beta \le a s_i + b = a(a r_i + b) + b = a^2 r_i + ab + b \le a^2 r_i + 2a\Delta.$$

We now prove the inequality by induction on i. We begin with the base case $i = 0$, where $r_0 = \Delta$ and

$$\sum_{0 \le j \le 0} w_j = w_0 \le a^2 r_0 + 2a\Delta = a^2\Delta + 2a\Delta \le 3a^2\Delta = \frac{(4-1)a^2\Delta}{\epsilon^0}.$$

In the induction step, we assume that $i \ge 1$:

$$\begin{aligned}
\sum_{0 \le j \le i} w_j &= \sum_{0 \le j \le i-1} w_j + w_i \\
&\le \sum_{0 \le j \le i-1} w_j + a^2 r_i + 2ab \\
&\le \sum_{0 \le j \le i-1} w_j + a^2 \cdot \frac{3a\sum_{0 \le j \le i-1} w_j + b}{\epsilon} + 2ab \\
&\le \frac{(3a^3+1)\sum_{0 \le j \le i-1} w_j + a^2 b + 2ab}{\epsilon} \\
&\le \frac{(3a^3+1)(4^i-1)a^{3(i-1)+2}\Delta + a^2 b + 2ab}{\epsilon^i} \\
&\le \frac{(3a^3+1)(4^i-1)a^{3(i-1)+2}\Delta + a^2\Delta + 2a\Delta}{\epsilon^i} \\
&\le \frac{((3+1)(4^i-1)+3)a^{3i+2}\Delta}{\epsilon^i} \\
&\le \frac{(4^{i+1}-1)a^{3i+2}\Delta}{\epsilon^i}.
\end{aligned}$$

□

Lemma 5.6. $a\gamma + b \leq \epsilon n^{1/p}\Delta$.

Proof. Remember that we have $\epsilon \leq 1 \leq \alpha \leq a$ and $\beta \leq b \leq \Delta$. By Lemma 5.5, we have

$$\sum_{0\leq j\leq p-2} w_j \leq \frac{(4^{p-1}-1)a^{3p-4}\Delta}{\epsilon^{p-2}}.$$

We now get

$$\begin{aligned}
\frac{a\gamma+b}{\epsilon} &= \frac{a\gamma_0 + 2\epsilon a\Delta + b}{\epsilon} \\
&= \frac{a(\alpha+1+\epsilon)\sum_{0\leq j\leq p-2} w_j + a\beta + 2\epsilon a\Delta + b}{\epsilon} \\
&\leq \frac{a(a+1+\epsilon)\sum_{0\leq j\leq p-2} w_j + a\Delta + 2a\Delta + \Delta}{\epsilon} \\
&\leq \frac{3a^2\sum_{0\leq j\leq p-2} w_j + 4a\Delta}{\epsilon} \\
&\leq \frac{3a^2(4^{p-1}-1)a^{3p-4}\Delta + 4a\Delta}{\epsilon^{p-1}} \\
&\leq \frac{4^p a^{3p}\Delta}{\epsilon^p} \\
&= (4a^3/\epsilon)^p\Delta \leq n^{1/p}\Delta.
\end{aligned}$$

The last inequality follows from Lemma 5.4. □

Lemma 5.7. $ar_{p-1} + b \leq n^{1/p}\Delta$.

Proof. By the definitions of r_{p-1} and γ_0, we have $r_{p-1} = \gamma_0/\epsilon$. Since $\gamma_0 \leq \gamma$ and $a\gamma + b \leq \epsilon n^{1/p}\Delta$ by Lemma 5.6, we have

$$ar_{p-1} + b = a\frac{\gamma_0}{\epsilon} + b \leq \frac{a\gamma_0 + b}{\epsilon} \leq \frac{a\gamma + b}{\epsilon} \leq n^{1/p}\Delta.$$ □

Lemma 5.8. *For all* $0 \leq i \leq j \leq p-1$, $s(r_i, j-i) \leq r_j$.

Proof. Fix some $0 \leq i \leq p-2$. The proof is by induction on j. In the first base case $j = i$, the claim is trivially true as $s(r_i, 0) \leq r_i$. Now, remember that for $j \geq 1$, we have

$$r_j = \frac{(\alpha+1+\epsilon)\sum_{0\leq j'\leq j-1} w_{j'} + \beta}{\epsilon} \geq (\alpha+1+\epsilon)w_{j-1} + \beta = (\alpha+1+\epsilon)(\alpha s_{j-1} + \beta) + \beta.$$

Thus, in the second base case $j = i+1$ the claim holds, because $s(r_i, j-i) = s(r_i, 1) \leq ar_i + b = s_i = s_{j-1} \leq r_j$. Finally, consider the induction step where we assume that the inequality holds for $j-1$ and have to show that it also holds for j, where $j \geq i+2$. By the induction hypothesis, we have $s(r_i, j-1-i) \leq r_{j-1}$, and since $j - i \geq 2$, we have

$$s(r_i, j-i) \leq (\alpha+1+\epsilon)(\alpha a s(r_i, j-i-1) + \alpha b + \beta) + \beta.$$

We now get

$$\begin{aligned}
r_j &\geq (\alpha+1+\epsilon)(\alpha s_{j-1} + \beta) + \beta \\
&= (\alpha+1+\epsilon)(\alpha a r_{j-1} + \alpha b + \beta) + \beta \\
&\geq (\alpha+1+\epsilon)(\alpha a s(r_i, j-i-1) + \alpha b + \beta) + \beta \\
&\geq s(r_i, j-i).
\end{aligned}$$ □

5.4 Analysis of Approximation Guarantee

We now analyze the approximation error of a monotone ES-tree maintained on H''. This approximation error consists of two parts. The first part is an approximation error that comes from the fact that the monotone ES-tree only considers paths from s with a relatively small number of edges and therefore has to use edges from the hop set F. The second part is the approximation error we get from rounding the edge weights. We first give a formula for the approximation error that depends on the priority of the nodes and their distance to the root of the monotone ES-tree.

Before we give the proof we review a few properties of the monotone ES-tree (see Reference [20] for the full algorithm). Similar to the classic ES-tree, the monotone ES-tree with root s maintains a level $\ell(v)$ for every node v. The monotone ES-tree is initialized by computing a shortest paths tree up to depth L from s in H'', and thus, initially, $\ell(v) = \text{dist}_{H''}(s, v)$. A single deletion or edge weight increase in G might result in a sequence of deletions, weight increases and insertions in F, and thus H''. The monotone ES-tree first processes the insertions and then the deletions and edge weight increases. It handles deletions and edge weights increases in the same way as the classic ES-tree. Once the level $\ell(u)$ of a node u exceeds the maximum level L, we set $\ell(u) = \infty$. The procedure for handling the insertion of an edge (u, v) is trivial: it only stores the new edge and in particular does *not* change $\ell(u)$ or $\ell(v)$. For completeness, we list the pseudocode of the monotone ES-tree in Algorithm 1.

For the analysis of the monotone ES-tree, we will use the following terminology. We say that an edge (u, v) is *stretched* if $\ell(u) > \ell(v) + w_{H''}(u, v)$. We say that a node u is *stretched* if it is incident to an edge (u, v) that is stretched. Note that for a node u that is not stretched, we have $\ell(u) \leq \ell(v) + w_{H''}(u, v)$ for every edge (u, v) contained in H''. In our proof, we will use the following properties of the monotone ES-tree.

Observation 5.9 ([20]). *The following holds for the monotone ES-tree:*

(1) The level of a node never decreases.
(2) An edge can only become stretched when it is inserted.
(3) As long as a node is stretched, its level does not change.
(4) For every tree edge (u, v) (where v is the parent of u), $\ell(u) \geq \ell(v) + w_{H''}(u, v)$.

Observe that property Equation (4) above implies property **A1**, i.e., that the returned distance estimate never underestimates the true distance.

A second prerequisite from Reference [20] tells us when we may apply a variant of the triangle inequality to argue about the levels of nodes.

Lemma 5.10 ([20]). *Let (u, v) be an edge of H'' such that $\ell(v) + w_{H''}(u, v) \leq L$. If (u, v) is not stretched and after the previous update in G the level of u was less than ∞, then for the current level of u, we have $\ell(u) \leq \ell(v) + w_{H''}(u, v)$.*

Note that the second precondition simply captures the property of the monotone ES-tree that once the level of a node exceeds L it is set to ∞ and will never be decreased anymore. At the initialization (i.e., before the first update in H''), the first precondition is fulfilled automatically as no edge is stretched yet.

To count the additive error from rounding the edge weights, we define, for every node u and every $0 \leq i \leq p - 1$, the function $h(u, i)$ as follows:

$$h(u, i) = \begin{cases} 0 & \text{if } u = s \\ (p+1)\left\lceil \frac{\max(\text{dist}_G(u,s) - r_i, 0)}{\Delta} \right\rceil + p + 1 - i & \text{otherwise} \end{cases}.$$

ALGORITHM 1: Monotone ES-tree

```
// Internal data structures:
// N(u): for every node u a heap N(u) whose intended use is to store for every neighbor
   v of u in the current graph the value of ℓ(v) + w_{H''}(u, v)
// Q: global heap whose intended use is to store nodes whose levels might need to be
   updated

Procedure Initialize()
    Compute shortest paths tree from s in H'' up to depth L
    foreach u ∈ V do
        Set ℓ(u) = dist_{H''}(s, u)
        for every edge (u, v) in H'' do insert v into heap N(u) of u with key ℓ(v) + w_{H''}(u, v)
    end

Procedure Delete(u, v)
    Increase(u, v, ∞)

Procedure Increase(u, v, w(u, v))
    // Increase weight of edge (u, v) to w(u, v)
    Insert u and v into heap Q with keys ℓ(u) and ℓ(v) respectively
    Update key of v in heap N(u) to ℓ(v) + w(u, v) and key of u in heap N(v) to ℓ(u) + w(u, v)
    UpdateLevels()

Procedure Insert(u, v, w(u, v))
    // Increase edge (u, v) of weight w(u, v)
    Insert v into heap N(u) with key ℓ(v) + w(u, v) and u into heap N(v) with key ℓ(u) + w_{H''}(u, v)

Procedure UpdateLevels()
    while heap Q is not empty do
        Take node u with minimum key ℓ(u) from heap Q and remove it from Q
        ℓ'(u) ← min_v(ℓ(v) + w_{H''}(u, v))
        // min_v(ℓ(v) + w_{H''}(u, v)) can be retrieved from the heap N(u).
           arg min_v(ℓ(v) + w_{H''}(u, v)) is u's parent in the ES-tree.
        if ℓ'(u) > ℓ(u) then
            ℓ(u) ← ℓ'(u)
            if ℓ'(u) > L then ℓ(u) ← ∞
            foreach neighbor v of u do
                update key of u in heap N(v) to ℓ(u) + w_{H''}(u, v)
                insert v into heap Q with key ℓ(v) if Q does not already contain v
            end
        end
    end
```

The intuition is that $h(u, i)$ bounds the number of hops from u to s, i.e., the number of edges required to go from u to s while at the same time providing the desired approximation guarantee. The approximation guarantee can now formally be stated as follows.

Lemma 5.11 (Approximation Guarantee). *For every $0 \leq i \leq p-1$ and every node u of priority i with $\text{dist}_G(u, s) \leq D + \sum_{0 \leq i' \leq i-1} s_{i'}$, we have*

$$\delta(s, u) \leq (\alpha + \epsilon)\text{dist}_G(u, s) + \gamma_i + h(u, i) \cdot \varphi.$$

Once we have proved this lemma, the desired bound on the approximation error (Property **A2**) follows easily, because $h(u,i)\cdot\varphi \le \epsilon\,\mathrm{dist}_G(u,s) + 2\epsilon\Delta$ (as we show below) and $\gamma \le \epsilon n^{1/p}\Delta$ by Lemma 5.6, and thus

$$\begin{aligned}\delta(s,u) &\le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + h(u,i)\cdot\varphi\\ &\le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_0 + h(u,i)\cdot\varphi\\ &\le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_0 + \epsilon\,\mathrm{dist}_G(u,s) + 2\epsilon\Delta\\ &= (\alpha+2\epsilon)\mathrm{dist}_G(u,s) + \gamma\\ &\le (\alpha+2\epsilon)\mathrm{dist}_G(u,s) + \epsilon n^{1/p}\Delta.\end{aligned}$$

Lemma 5.12. *For every node u and every $0 \le i \le p-1$,*

$$h(u,i)\cdot\varphi \le \epsilon\,\mathrm{dist}_G(u,s) + 2\epsilon\Delta.$$

Proof. If $u = s$, then the claim is trivially true. Otherwise, we have

$$\begin{aligned}h(u,i) &= \left((p+1)\left\lceil\frac{\max(\mathrm{dist}_G(u,s)-r_i,0)}{\Delta}\right\rceil + p + 1 - i\right)\varphi\\ &\le \left((p+1)\left\lceil\frac{\mathrm{dist}_G(u,s)}{\Delta}\right\rceil + p + 1\right)\varphi\\ &\le \left((p+1)\left(\frac{\mathrm{dist}_G(u,s)}{\Delta}+1\right) + p + 1\right)\varphi\\ &= \left(\frac{(p+1)\mathrm{dist}_G(u,s)}{\Delta} + 2(p+1)\right)\varphi\\ &= \left(\frac{(p+1)\mathrm{dist}_G(u,s)}{\Delta} + 2(p+1)\right)\cdot\frac{\epsilon\Delta}{p+1}\\ &= \epsilon\,\mathrm{dist}_G(u,s) + 2\epsilon\Delta.\end{aligned}$$

□

Proof of Lemma 5.11. The proof is by double induction first on the number of updates in G and second on $h(u,i)$. Let $0 \le i \le p-1$ and let u be a node of priority i such that $\mathrm{dist}_G(u,s) \le D + \sum_{0\le i'\le i-1} s_{i'}$. Remember that $\delta(u,s) = \ell(u)\cdot\varphi$, where $\ell(u)$ is the level of u in the monotone ES-tree of s. We know that after the previous deletion in G the distance estimate gave an approximation of the true distance in G. Since distances in G are non-decreasing it must have been the case that the level of u was less than ∞ after the previous deletion in G.

If $u = s$, then the claim is trivially true, because $\ell(s) = 0$. Assume that $u \ne s$. If u is stretched in the monotone ES-tree, then the level of u has not changed since the previous deletion in G, and thus the claim is true by induction. If u is not stretched, then $\ell(u) \le \ell(v) + w_{H''}(u,v)$ for every edge (u,v) in H''. Define the nodes v and x as follows. If $\mathrm{dist}_G(u,s) \le r_i$, then $v = s$. If $\mathrm{dist}_G(u,s) > r_i$, then consider a shortest path π from u to s in G and let v be the furthest node from u on π such that $\mathrm{dist}_G(u,v) \le r_i$ (which implies $\mathrm{dist}_G(v,s) \ge \mathrm{dist}_G(u,s) - r_i$). Furthermore let x be the neighbor of v on the shortest path π that is closer to s than v is. Note that $\mathrm{dist}_G(u,x) \ge r_i$ (and thus $\mathrm{dist}_G(x,s) \le \mathrm{dist}_G(u,s) - r_i$) and in particular G contains the edge (v,x). The edge (v,x) is also contained in H (and thus in H' and H'') by the following argument: For $\mathrm{dist}_G(u,s) \le D + \sum_{0\le i'\le i-1} s_{i'}$ to hold it has to be the case that $w_G(v,x) \le D + \sum_{0\le i'\le i-1} s_{i'}$. Note that $\sum_{0\le i'\le i-1} s_{i'} \le \sum_{0\le i'\le i-1} w_{i'} \le r_{p-1} \le n^{1/p}\Delta$ by Lemma 5.7. Thus, $w_G(v,x) \le D + n^{1/p}\Delta$, which by the definition of H means that the edge (v,x) is contained in H.

Note that $s(\mathrm{dist}_G(u,v), p-1-i) \le s(r_i, p-1-i)$, since the function $s(\cdot,\cdot)$ is non-decreasing in the first argument. By Lemma 5.8, we have $s(r_i, p-1-i) \le r_{p-1}$, and by Lemma 5.7, we have

$r_{p-1} \leq n^{1/p}\Delta \leq \hat{D}$. It follows that $s(\text{dist}_G(u,v), p-1-i) \leq \hat{D}$. Thus, by Property **B2**, we know that either $v \in B(u)$ or there is a node v' of priority $j' > i$ such that $u \in B(v)$ and $\text{dist}_G(u,v') \leq s(\text{dist}_G(u,v), j'-i)$. Note that in the first case the set of edges F contains the edge (u,v) and in the second case it contains the edge (u,v').

Case 1: $v \in B(u)$

If $v \in B(u)$, then F contains an edge (u,v), such that

$$w_F(u,v) = \hat{\delta}(u,v) \leq \alpha \text{ dist}_G(u,v) + \beta. \tag{2}$$

Since $\text{dist}_G(u,v) \leq r_i$, we have $w_F(u,v) \leq \alpha r_i + \beta \leq \alpha r_{p-1} + \beta \leq n^{1/p}\Delta$, where the last inequality holds by Lemma 5.7. Thus, (u,v) is contained in H and thus also in H' and H''.

If $\text{dist}_G(u,s) \leq r_i$, then we have $v = s$. First, observe that by the definition of H'', we have $w_{H''}(u,s) = w_{H'}(u,s)/\varphi$. Furthermore, the rounding of the edge weights in H' guarantees that $w_{H'}(u,s) \leq w_H(u,s) + \varphi$. We therefore get

$$\begin{aligned}
w_{H''}(u,s) &\leq \frac{w_F(u,s) + \varphi}{\varphi} \\
&\leq \frac{\alpha \text{ dist}_G(u,s) + \beta + \varphi}{\varphi} \\
&\leq \frac{\alpha\,(D + \sum_{0 \leq i' \leq i-1} s_{i'}) + \beta + \varphi}{\varphi} \\
&\leq \frac{\alpha D + (\alpha + 1 + \epsilon) \sum_{0 \leq i' \leq p-2} w_{i'} + \beta + \varphi}{\varphi} \\
&= \frac{\alpha D + \gamma_0 + \varphi}{\varphi} \\
&= \frac{\alpha D + \gamma_0 + \frac{\epsilon\Delta}{p+1}}{\varphi} \\
&\leq \frac{\alpha D + \gamma_0 + 2\epsilon\Delta}{\varphi} \\
&= \frac{\alpha D + \gamma}{\varphi} \leq \frac{\alpha D + \epsilon n^{1/p}\Delta}{\varphi} \leq \frac{(\alpha + 2\epsilon)D}{\varphi} + (p+1)n^{1/p} = L.
\end{aligned}$$

Here, we have used the inequality $\gamma \leq \epsilon n^{1/p}\Delta$ from Lemma 5.6. Since the maximum level in the monotone ES-tree is L and u is not stretched, it follows from Lemma 5.10 that $\ell(u) \leq \ell(s) + w_{H''}(u,s) = w_{H''}(u,s)$. Together with the observations $h(u,i) \geq 1$ (since $u \neq s$) and $\beta \leq \gamma_0$, we therefore get

$$\begin{aligned}
\delta(s,u) = \ell(u) \cdot \varphi \leq w_{H''}(u,s) \cdot \varphi &\leq \alpha \text{ dist}_G(u,s) + \beta + \varphi \\
&\leq \alpha \text{ dist}_G(u,s) + \beta + h(u,i) \cdot \varphi \leq (\alpha + \epsilon)\text{dist}_G(u,s) + \gamma_0 + h(u,i) \cdot \varphi.
\end{aligned}$$

Consider now the case $\text{dist}_G(u,s) > r_i$. Let j denote the priority of x. We first prove the following inequality, which will allow us among other things to use the induction hypothesis on x.

Claim 5.13. *If* $\text{dist}_G(u,s) > r_i$, *then* $h(x,j) + 2 \le h(u,i)$.

Proof. Remember that $i \le p-1$. The assumption $\text{dist}_G(u,s) > r_i$ implies that $\text{dist}_G(x,s) \le \text{dist}_G(u,s) - r_i$. If $\text{dist}_G(x,s) < r_j$, then we have

$$h(x,j)+2 \le p+1-j+2 \le p+1+2 \le p+1+p+1-i \le (p+1)\left\lceil \frac{\text{dist}_G(u,s)-r_i}{\Delta}\right\rceil + p + 1 - i = h(u,i).$$

Here, we use the inequality $\lceil(\text{dist}_G(u,s) - r_j)/\Delta\rceil \ge 1$, which follows from the assumption $\text{dist}_G(u,s) > r_i$.

If $\text{dist}_G(x,s) \ge r_j$, then, using $r_j \ge r_0 \ge \Delta$, we get

$$\begin{aligned}
h(x,j)+2 &= (p+1)\left\lceil \frac{\text{dist}_G(x,s)-r_j}{\Delta}\right\rceil + p + 1 - j + 2\\
&\le (p+1)\left\lceil \frac{\text{dist}_G(x,s)-\Delta}{\Delta}\right\rceil + p + 1 + 2\\
&= (p+1)\left\lceil \frac{\text{dist}_G(x,s)}{\Delta} - 1\right\rceil + p + 1 + 2\\
&= (p+1)\left(\left\lceil \frac{\text{dist}_G(x,s)}{\Delta}\right\rceil - 1\right) + p + 1 + 2\\
&= (p+1)\left\lceil \frac{\text{dist}_G(x,s)}{\Delta}\right\rceil + 2\\
&\le (p+1)\left\lceil \frac{\text{dist}_G(x,s)}{\Delta}\right\rceil + p + 1 - i\\
&\le (p+1)\left\lceil \frac{\text{dist}_G(u,s)-r_i}{\Delta}\right\rceil + p + 1 - i\\
&\le (p+1)\left\lceil \frac{\max(\text{dist}_G(u,s)-r_i,0)}{\Delta}\right\rceil + p + 1 - i = h(u,i).
\end{aligned}$$

Here the last inequality follows from the trivial observation $\text{dist}_G(u,s) - r_i \le \max(\text{dist}_G(u,s) - r_i, 0)$. □

Having proved this claim, we go on with the proof of the lemma. We will now show that

$$\ell(x) + w_{H''}(v,x) + w_{H''}(u,v) \le \frac{(\alpha+\epsilon)\text{dist}_G(u,s) + \gamma_i + h(u,i)\cdot\varphi}{\varphi} \tag{3}$$

as follows. If $\text{dist}_G(u,s) > r_i$, then we have $\text{dist}_G(u,x) \ge r_i$ by the choice of x. Remember that the edge (v,x) lies on a shortest path from u to s in G. It is therefore contained in G since before the first deletion, and thus it will never be stretched. We also may apply the induction hypothesis on x, since

$$\text{dist}_G(x,s) = \text{dist}_G(u,s) - \text{dist}_G(u,x) \le \text{dist}_G(u,s) - r_i \le D + \sum_{0\le i'\le i-1} s_{i'} - r_i \le D,$$

due to $\sum_{0\le i'\le i-1} s_{i'} \le r_i$ by the definition of r_i. Therefore, we get

$$
\begin{aligned}
&(\ell(x) + w_{H''}(v,x) + w_{H''}(u,v)) \cdot \varphi \\
&\quad \le \delta(s,x) + w_{H''}(v,x) \cdot \varphi + w_{H''}(u,v) \cdot \varphi && \text{(definition of } \delta(s,x)) \\
&\quad = \delta(s,x) + w_{H'}(v,x) + w_{H'}(u,v) && \text{(definition of } H') \\
&\quad \le \delta(s,x) + w_H(v,x) + \varphi + w_H(u,v) + \varphi && \text{(property of } w_{H'}) \\
&\quad \le \delta(s,x) + w_G(v,x) + \varphi + w_F(u,v) + \varphi && ((v,x) \in E \text{ and } (u,v) \in F) \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_j + h(x,j) \cdot \varphi + w_G(v,x) + \varphi + w_F(u,v) + \varphi && \text{(induction hypothesis)} \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_j + w_F(u,v) + w_G(v,x) + (h(x,j)+2) \cdot \varphi && \text{(rearranging terms)} \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_j + w_F(u,v) + w_G(v,x) + h(u,i) \cdot \varphi && \text{(Claim 5.13)} \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_0 + w_F(u,v) + w_G(v,x) + h(u,i) \cdot \varphi && (\gamma_j \le \gamma_0) \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_0 + \alpha\ \mathrm{dist}_G(u,v) + \beta + w_G(v,x) + h(u,i) \cdot \varphi && \text{(by Inequality Equation (2))} \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_0 + \alpha\ \mathrm{dist}_G(u,v) + \beta + \mathrm{dist}_G(v,x) + h(u,i) \cdot \varphi && ((v,x) \text{ on shortest path}) \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \gamma_0 + \alpha\ \mathrm{dist}_G(u,v) + \beta + \alpha\ \mathrm{dist}_G(v,x) + h(u,i) \cdot \varphi && (\alpha \ge 1) \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \alpha\ (\mathrm{dist}_G(u,v) + \mathrm{dist}_G(v,x)) + \beta + \gamma_0 + h(u,i) \cdot \varphi && \text{(rearranging terms)} \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \alpha\ \mathrm{dist}_G(u,x) + \beta + \gamma_0 + h(u,i) \cdot \varphi && (v \text{ on shortest path}) \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \alpha\ \mathrm{dist}_G(u,x) + \beta + \gamma_0 - \gamma_i + \gamma_i + h(u,i) \cdot \varphi && \text{(zero addition)} \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \alpha\ \mathrm{dist}_G(u,x) + \epsilon r_i + \gamma_i + h(u,i) \cdot \varphi && \text{(by Lemma 5.3)} \\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(x,s) + \alpha\ \mathrm{dist}_G(u,x) + \epsilon\ \mathrm{dist}_G(u,x) + \gamma_i + h(u,i) \cdot \varphi && (\mathrm{dist}_G(u,x) \ge r_i) \\
&\quad = (\alpha+\epsilon)(\mathrm{dist}_G(u,x) + \mathrm{dist}_G(x,s)) + \gamma_i + h(u,i) \cdot \varphi && \text{(rearranging terms)} \\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + h(u,i) \cdot \varphi && (x \text{ on shortest path}).
\end{aligned}
$$

By Lemma 5.12, we have $h(u,i) \cdot \varphi \le \epsilon\ \mathrm{dist}_G(u,s) + 2\epsilon\Delta$ and thus Inequality Equation (3) implies that

$$
\begin{aligned}
\ell(x) + w_{H''}(v,x) + w_{H''}(u,v) &\le \frac{(\alpha+2\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + 2\epsilon\Delta}{\varphi} \\
&\le \frac{(\alpha+2\epsilon)\,(D + \sum_{0 \le i' \le i-1} s_{i'}) + \gamma_i + 2\epsilon\Delta}{\varphi} \\
&\le \frac{(\alpha+2\epsilon)D + (\alpha+1+\epsilon)\,(\sum_{0 \le i' \le i-1} w_{i'}) + \gamma_i + 2\epsilon\Delta}{\varphi} \\
&\le \frac{(\alpha+2\epsilon)D + \gamma_0 + 2\epsilon\Delta}{\varphi} \\
&= \frac{(\alpha+2\epsilon)D + \gamma}{\varphi} \\
&\le \frac{(\alpha+2\epsilon)D + \epsilon n^{1/p}\Delta}{\varphi} \\
&= \frac{(\alpha+2\epsilon)D}{\varphi} + (p+1)n^{1/p} = L,
\end{aligned}
$$

where the last inequality follows from Lemma 5.6. As the maximum level in the monotone ES-tree is L and the edge (v,x) is not stretched, it follows from Lemma 5.10 that $\ell(v) \le \ell(x) + w_{H''}(v,x)$, and since u is not stretched, we have

$$\ell(u) \le \ell(v) + w_{H''}(u,v) \le \ell(x) + w_{H''}(v,x) + w_{H''}(u,v),$$

and thus

$$\delta(s,u) = \ell(u) \cdot \varphi \le (\ell(x) + w_{H''}(v,x) + w_{H''}(u,v)) \cdot \varphi \le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + h(u,i) \cdot \varphi.$$

Case 2: $v \notin B(u)$

By Property **B2**, we know that there is some node v' of priority $j' > i$ such that $u \in B(v')$ and $\text{dist}_G(u, v') \leq s(\text{dist}_G(u, v), j' - i)$. By Lemma 5.8, we therefore have

$$\text{dist}_G(u, v') \leq s(r_i, j' - i) \leq s(r_{j'-1}, 1) = s_{j'-1}.$$

From the definition of F and Property **B1**, it now follows that F contains the edge (u, v') of weight

$$\text{dist}_G(u, v') \leq w_F(u, v') = \hat{\delta}(u, v') \leq \alpha \, \text{dist}_G(u, v') + \beta \leq \alpha s_{j'-1} + \beta = w_{j'-1}. \tag{4}$$

Since $j' \leq p - 1$, we have $w_{j'-1} \leq w_{p-2} \leq r_{p-1}$. As $r_{p-1} \leq n^{1/p}\Delta$, by Lemma 5.7, we conclude that the edge (u, v') is contained H and thus also in H' and H''.

We first prove the following inequality, which will allow us among other things to apply the induction hypothesis on v'.

Claim 5.14. $h(v', j') + 1 \leq h(u, i)$.

Proof. Remember that $j' \geq i + 1$. If $\text{dist}_G(v', s) < r_{j'}$, then we get

$$h(v', j') + 1 \leq p + 1 - j' + 1 \leq p + 1 - i \leq h(u, i).$$

If $\text{dist}_G(v', s) \geq r_{j'}$, then we use the inequality $r_{j'} \geq r_i + s_{j'-1}$ (which easily follows from the definition of $r_{j'}$ and the fact that $\alpha \geq 1$) and get

$$\begin{aligned}
h(v', j') + 1 &= (p + 1)\left\lceil \frac{\text{dist}_G(v', s) - r_{j'}}{\Delta} \right\rceil + p + 1 - j' + 1 \\
&\leq (p + 1)\left\lceil \frac{\text{dist}_G(v', s) - r_{j'}}{\Delta} \right\rceil + p + 1 - i - 1 + 1 \\
&\leq (p + 1)\left\lceil \frac{\text{dist}_G(v', u) + \text{dist}_G(u, s) - r_{j'}}{\Delta} \right\rceil + p + 1 - i \\
&\leq (p + 1)\left\lceil \frac{s_{j'-1} + \text{dist}_G(u, s) - r_{j'}}{\Delta} \right\rceil + p + 1 - i \\
&\leq (p + 1)\left\lceil \frac{\text{dist}_G(u, s) - r_i}{\Delta} \right\rceil + p - i \\
&\leq (p + 1)\left\lceil \frac{\max(\text{dist}_G(u, s) - r_i, 0)}{\Delta} \right\rceil + p + 1 - i = h(u, i).
\end{aligned}$$

□

Having proved this claim, we go on with the proof of the lemma. Note that we may apply the induction hypothesis on v', because by the triangle inequality, we have

$$\begin{aligned}
\text{dist}_G(v', s) \leq \text{dist}_G(u, s) + \text{dist}_G(v', u) &\leq D + \sum_{0 \leq i' \leq i-1} s_{i'} + \text{dist}_G(v', u) \\
&\leq D + \sum_{0 \leq i' \leq i-1} s_{i'} + s_{j'-1} \leq D + \sum_{0 \leq i' \leq j'-1} s_{i'}.
\end{aligned}$$

We will now show that

$$\ell(v') + w_{H''}(u, v') \leq \frac{(\alpha + \epsilon)\text{dist}_G(u, s) + \gamma_i + h(u, i) \cdot \varphi}{\varphi} \tag{5}$$

as follows:

$$
\begin{aligned}
&(\ell(v') + w_{H''}(u,v')) \cdot \varphi && (u \text{ not stretched})\\
&\quad = \delta(v',s) + w_{H''}(u,v') \cdot \varphi && (\text{definition of } \delta(v',s))\\
&\quad = \delta(v',s) + w_{H'}(u,v') && (\text{definition of } H'')\\
&\quad \le \delta(v',s) + w_H(u,v') + \varphi && (\text{property of } w_{H'}(u,v'))\\
&\quad \le \delta(v',s) + w_F(u,v') + \varphi && (\text{definition of } H)\\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(v',s) + \gamma_{j'} + h(v',j') \cdot \varphi + w_F(u,v') + \varphi && (\text{induction hypothesis})\\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(v',s) + \gamma_{j'} + w_F(u,v') + (h(v',j')+1) \cdot \varphi && (\text{rearranging terms})\\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(v',s) + \gamma_{j'} + w_F(u,v') + h(u,i) \cdot \varphi && (\text{Claim 5.14})\\
&\quad \le (\alpha+\epsilon)(\mathrm{dist}_G(v',u) + \mathrm{dist}_G(u,s)) + \gamma_{j'} + w_F(u,v') + h(u,i) \cdot \varphi && (\text{triangle inequality})\\
&\quad \le (\alpha+\epsilon)(w_F(u,v') + \mathrm{dist}_G(u,s)) + \gamma_{j'} + w_F(u,v') + h(u,i) \cdot \varphi && (\text{by Inequality Equation (4)})\\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_{j'} + (\alpha+\epsilon+1)w_F(u,v') + h(u,i) \cdot \varphi && (\text{rearranging terms})\\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_{j'} + (\alpha+\epsilon+1)w_{j'-1} + h(u,i) \cdot \varphi && (\text{by Inequality Equation (4)})\\
&\quad = (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_{j'-1} + h(u,i) \cdot \varphi && (\text{definition of } \gamma_{j'-1})\\
&\quad \le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + h(u,i) \cdot \varphi && (\gamma_i \ge \gamma_{j'-1} \text{ as } j' \ge i+1).
\end{aligned}
$$

By Lemma 5.12, we have $h(u,i) \cdot \varphi \le \epsilon\,\mathrm{dist}_G(u,s) + 2\epsilon\Delta$ and thus Inequality Equation (5) implies that

$$\ell(v') + w_{H''}(u,v') \le \frac{(\alpha+2\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + 2\epsilon\Delta}{\varphi} \le \frac{(\alpha+2\epsilon)D}{\varphi} + (p+1)n^{1/p} = L.$$

As the maximum level in the monotone ES-tree is L and u is not stretched, it follows from Lemma 5.10 that $\ell(u) \le \ell(v') + w_{H''}(u,v')$, and thus

$$\delta(s,u) = \ell(u) \cdot \varphi \le (\ell(v') + w_{H''}(u,v')) \cdot \varphi \le (\alpha+\epsilon)\mathrm{dist}_G(u,s) + \gamma_i + h(u,i) \cdot \varphi. \qquad \square$$

6 PUTTING EVERYTHING TOGETHER

In the following, we combine the results of Sections 4 and 5 to obtain decremental algorithms for approximate SSSP and approximate APSP.

6.1 Approximate SSSP

We first show how to obtain an algorithm for approximate SSSP. First, we obtain an algorithm that provides approximate distance for all nodes that are at distance of at most R from the source, where R is some range parameter. We use a hierarchical approach to obtain this algorithm: Given an algorithm for maintaining approximate shortest paths, we obtain an algorithm for maintaining approximate balls, which in turn gives us an algorithm for maintaining approximate shortest paths for a larger range of distances than the initial algorithm. This scheme is repeated several times and can be "started" with the (exact) ES-tree.

Lemma 6.1. *For every $R \ge n$ and every $0 < \epsilon \le 1$, there is a decremental approximate SSSP algorithm that, given a fixed source node s, maintains, for every node v, a distance estimate $\delta(s,v)$ such that $\delta(s,v) \ge \mathrm{dist}_G(s,v)$ and if $\mathrm{dist}_G(s,v) \le R$, then $\delta(s,v) \le (1+\epsilon)\mathrm{dist}_G(s,v)$. It has a total update time of $O(m^{1+O((\log\log R)/q)}R^{2/q} + n)$ in expectation, where*

$$q = \left\lfloor \sqrt{\left\lfloor \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{12\cdot 4^3 \log n}{\epsilon}\right)}} \right\rfloor} \right\rfloor$$

and, after every update in G, returns each node v such that $\delta(s,v)$ has changed together with the new value of $\delta(s,v)$.

Proof. In the proof, we will use the following values. We set $a = 4$,

$$p = \left\lfloor \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{12a^3 \log n}{\epsilon}\right)}} \right\rfloor,$$

and $q = \lfloor \sqrt{p} \rfloor$. Furthermore, we set $\epsilon' = \epsilon/(2(q-2))$, and for every $0 \le k \le q-2$, we set $\alpha_k = 1 + 3k\epsilon' \le 1+\epsilon$, $a_k = 2\alpha_k \le a$, $\Delta_k = R^{k/q}$, and $D_k = R^{(k+2)/q}$.

The heart of our proof is the following claim, which gives us decremental approximate SSSP algorithms for larger and larger depths, until finally the full range R is covered. □

Claim 6.2. *For every $0 \le k \le q-2$, there is a decremental approximate SSSP algorithm ApproxSSSP$_k$ with the following properties:*

- ***A1*** *$\delta(s,v) \ge \text{dist}_G(s,v)$.*
- ***A2*** *If $\text{dist}_G(s,v) \le D_k$, then $\delta(s,v) \le \alpha_k \text{ dist}_G(s,v)$.*
- ***A3*** *The expected total update time of ApproxSSSP$_k$ is*

$$T_k(m,n) = O(p \log n \log R)^k \cdot O(m^{1+k/p} R^{2/q}/\epsilon') + O(n).$$

- ***A4*** *After every update in G, ApproxSSSP$_k$ returns each node v such that $\delta(s,v)$ has changed together with the new value of $\delta(s,v)$.*

Proof. We prove the claim by induction on k. In the base case $k = 0$, we use the (exact) ES-tree (see Lemma 2.1), which for distances up to $D \le D_0$ has a total update time of $O(mD_0 + n) = O(mR^{2/q} + n)$ and thus has all claimed properties.

We now consider the induction step. We apply Proposition 4.1 to obtain a decremental algorithm ApproxBalls$_k$ (with parameters $\hat{k} = p$ and $\hat{\epsilon} = 1$) that maintains for every node $u \in V$ a set of nodes $B_k(u)$ and a distance estimate $\hat{\delta}_k(u,v)$ for every node $v \in B_k(u)$ such that:

- **B1** For every node u and every node $v \in B_k(u)$, we have $\text{dist}_G(u,v) \le \hat{\delta}_k(u,v) \le \alpha_{k-1} \text{ dist}_G(u,v)$.
- **B2** For all $x \ge 0$, set $s_k(x,0) = x$, and for all $x \ge 0$ and $l \ge 1$, set $s_k(x,l) = a_{k-1}(a_{k-1}+1)^{l-1}x$. Then for every $0 \le i \le p-1$, every node u of priority i, and every node v such that $s_k(\text{dist}_G(u,v), p-1-i) \le D_k$, either (1) $v \in B_k(u)$ or (2) there is some node v' of priority $j > i$ such that $u \in B_k(v')$ and $\text{dist}_G(u,v') \le s_k(\text{dist}_G(u,v), j-i)$.
- **B3** In expectation, $\sum_{u \in V} \mathcal{B}_k(u) = O(pm^{1+1/p} \log D_k)$, where $\mathcal{B}_k(u)$ denotes the number of nodes ever contained in $B_k(u)$.
- **B4** The total update time of ApproxBalls$_k$ is

$$t_k(m,n) = O\left(\left(pm^{1+1/p} + \sum_{0 \le i \le p-1} \frac{m}{m^{i/p}} \cdot T_{k-1}(m_i, n_i)\right) \log n \log D_k + pT_{k-1}(m,n)\right),$$

in expectation, where, for each $0 \le i \le p-1$, $m_i = O(m^{(i+1)/p})$ and $n_i = O(m^{(i+1)/p})$.

Note that $D_k \le R$ and thus $\log D_k \le \log R$, and remember that by the induction hypothesis, we have

$$T_k(m,n) = O(p \log n \log R)^{k-1} \cdot O(m^{1+(k-1)/p} R^{2/q}/\epsilon') + O(n).$$

To analyze $\frac{m}{m^{i/p}} \cdot T_{k-1}(m_i, n_i)$ for every $0 \leq i \leq p-1$, observe that $\frac{m}{m^{i/p}} \cdot (m^{(i+1)/p})^{1+(k-1)/p} \leq m^{1+k/p}$, because

$$\begin{aligned} 1 - i/p + ((i+1)/p) \cdot (1 + (k-1)/p) &= 1 + 1/p + ((i+1)/p)((k-1)/p) \\ &\leq 1 + 1/p + (k-1)/p \\ &= 1 + k/p. \end{aligned}$$

It now follows that

$$t_k(m, n) = O(p \log n \log R)^k \cdot O(m^{1+k/p} R^{2/q}/\epsilon') + O(n).$$

We now want to argue that we may apply Proposition 5.1 to obtain an approximate decremental SSSP algorithm ApproxSSSP$'_k$ (with parameters p, Δ_k D_k, and ϵ'). We first show that

$$p \leq \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4a^3}{\epsilon'}\right)}}.$$

First note that $q \leq \log n$ and thus $\epsilon' = \epsilon(2(q-2) \geq \epsilon/(2q) \geq \epsilon/(2 \log n)$. It follows that

$$\frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4a^3}{\epsilon'}\right)}} \geq \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{12a^3 \log n}{\epsilon}\right)}} \geq \left\lfloor \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{12 \cdot 4^3 \log n}{\epsilon}\right)}} \right\rfloor = p.$$

Note also that for all $x \geq 0$, we have $s_k(x, 1) = a_{k-1}x$, and for all $x \geq 0$ and $l \geq 1$, we have

$$s_k(x, l+1) = (a_{k-1} + 1)s_k(x, l) \leq 2a_{k-1}s_k(x, l) \leq (\alpha_{k-1} + 1 + \epsilon')\alpha_{k-1}a_{k-1}s_k(x, l).$$

We therefore may apply Proposition 5.1 to obtain an approximate decremental SSSP algorithm ApproxSSSP$'_k$ (with parameters p, Δ_k, D_k, and ϵ') that maintains, for every node $v \in V$, a distance estimate $\delta'(s, v)$ such that:

A1' $\delta'(s, v) \geq \text{dist}_G(s, v)$.
A2' If $\text{dist}_G(s, v) \leq D_k$, then $\delta'(s, v) \leq (\alpha_k + 2\epsilon')\text{dist}_G(s, v) + \epsilon' n^{1/p} \Delta_k$.
A3' The total update time of ApproxSSSP$'_k$ is

$$T'_k(m, n) = t_k(m, n) + O\left(p\left(\alpha_k D_k/\Delta_k + n^{1/p}\right)\left(m + \sum_{u \in V} \mathcal{B}_k(u)\right)/\epsilon' + n\right).$$

A4' After every update in G, ApproxSSSP$'_k$ returns each node v such that $\delta(s, v)$ has changed together with the new value of $\delta(s, v)$.

Note that $\alpha_k \leq 1 + \epsilon \leq 2$ and $D_k/\Delta_k = R^{2/q}$. Since $q \leq p$ and $R \geq n$, we have $n^{1/p} \leq R^{2/q}$. We also have $\sum_{u \in V} \mathcal{B}_k(u) = O(pm^{1+1/p} \log R)$ in expectation. Therefore, the expected total update time of ApproxSSSP$'_k$ is

$$T'_k(m, n) = O(p \log n \log R)^k \cdot O(m^{1+k/p} R^{2/q}/\epsilon') + O(p^2 m^{1+1/p} R^{2/q} \log R/\epsilon' + n),$$

and since $p \leq \log n$, it follows that

$$T'_k(m, n) = O(p \log n \log R)^k \cdot O(m^{1+k/p} R^{2/q}/\epsilon') + O(n).$$

Let ApproxSSSP$_k$ denote the algorithm that internally runs both ApproxSSSP$'_k$ and ApproxSSSP$_{k-1}$ and additionally maintains, for every node v, the value $\delta_k(s, v) = \min(\delta'_k(s, v), \delta_{k-1}(s, v))$. Since both ApproxSSSP$'_k$ and ApproxSSSP$_{k-1}$ return, after each update in G, every node v for which $\delta(s, v)$ has changed, and the minimum can be computed in constant

time, ApproxSSSP$_k$ has the same asymptotic total update time as ApproxSSSP$'_k$. It remains to show that $\delta_k(s,v)$ fulfills the desired approximation guarantee for every node v. Since both $\delta'_k(s,v) \geq \text{dist}_G(s,v)$ and $\delta_{k-1}(s,v) \geq \text{dist}_G(s,v)$ also $\delta_k(s,v) \geq \text{dist}_G(s,v)$. Furthermore, we know that if $\text{dist}_G(s,v) \leq D_k$, then $\delta'_k(s,v) \leq \epsilon n^{1/p}\Delta_k$. Let v be a node such that $\text{dist}_G(s,v) \leq D_k$. If $\text{dist}_G(s,v) \leq D_{k-1}$, then $\delta_k(s,v) \leq \delta_{k-1}(s,v) \leq \alpha_{k-1}\,\text{dist}_G(s,v) \leq \alpha_k\,\text{dist}_G(u,v)$. If $\text{dist}_G(s,v) \geq D_{k-1}$, then

$$\begin{aligned}\delta_k(s,v) \leq \delta'_k(s,v) &\leq (\alpha_{k-1}+2\epsilon')\text{dist}_G(s,v) + \epsilon' n^{1/p}\Delta_k \\ &\leq (\alpha_{k-1}+2\epsilon')\text{dist}_G(s,v) + \epsilon' D_{k-1} \leq (\alpha_{k-1}+3\epsilon')\text{dist}_G(s,v) \\ &= \alpha_k\,\text{dist}_G(s,v).\end{aligned}$$

This finishes the proof of the claim. □

The lemma now follows from the claim by observing that ApproxSSSP$_{q-2}$ is the desired decremental approximate SSSP algorithm. The correctness simply follows from the choice $D_{q-2} = R$. The expected total update time is

$$T_{q-2}(m,n) = O(p\log n\log R)^{q-2}\cdot O(m^{1+(q-2)/p}R^{2/q}/\epsilon') + O(n).$$

Remember that $q = \lfloor\sqrt{p}\rfloor$ and thus $(q-2)/p \leq q/p \leq 1/\sqrt{p} \leq 1/q$. By the definition of p, we have $(2/\epsilon')^p \leq n^{1/p}$ and thus $(2/\epsilon')^q \leq (2/\epsilon')^p \leq n^{1/p} \leq n^{1/q}$ and furthermore, since $p \leq \log n$ and $R \geq n, p \leq \log n \leq (\log R)^q \leq (\log R)^p = (2^p)^{\log\log R} \leq (n^{1/p})^{\log\log R} = n^{(\log\log R)/p} \leq n^{(\log\log R)/q}$. It follows that the total update time is

$$T_{q-2}(m,n) = O(m^{1+O((\log\log R)/q)}R^{2/q}) + n).$$ □

We can turn the algorithm above into an algorithm for the full distance range by using the rounding technique once more.

Theorem 6.3. *For every $0 < \epsilon \leq 1$, there is a decremental approximate SSSP algorithm that, given a fixed source node s, maintains, for every node v, a distance estimate $\delta(s,v)$ such that $\text{dist}_G(s,v) \leq \delta(s,v) \leq (1+\epsilon)\text{dist}_G(s,v)$. It has constant query time and a total update time of*

$$O(m^{1+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)}\log W + n)$$

in expectation. If $1/\epsilon = \text{polylog}\,n$, then the total update time is $O(m^{1+o(1)}\log W + n)$ in expectation.

Proof. For every $0 \leq i \leq \lfloor\log(nW)\rfloor$, we define

$$\varphi_i = \frac{\epsilon 2^i}{n}.$$

Let G'_i be the graph that has the same nodes and edges as G and in which every edge weight is rounded to the next multiple of φ_i, i.e., every edge (u,v) in G'_i has weight

$$w_{G'_i}(u,v) = \left\lceil\frac{w_G(u,v)}{\varphi_i}\right\rceil\cdot\varphi_i,$$

where $w_G(u,v)$ is the weight of (u,v) in G. This rounding guarantees that

$$w_G(u,v) \leq w_{G_i}(u,v) \leq w_G(u,v) + \varphi_i$$

for every edge (u,v) of G. Furthermore, we define G''_i to be the graph that has the same nodes and edges as G'_i and in which every edge weight is scaled down by a factor of $1/\varphi_i$, i.e., every edge (u,v) in G''_i has weight

$$w_{G''_i}(u,v) = \frac{w_{G'_i}(u,v)}{\varphi_i} = \left\lceil\frac{w(u,v)}{\varphi_i}\right\rceil.$$

The algorithm is as follows: For every $0 \le i \le \lfloor \log(nW) \rfloor$, we use the algorithm of Lemma 6.1 on the graph G_i'' with $R = 4n/\epsilon$ to maintain a distance estimate $\delta_i(s,v)$ for every node v that satisfies

- $\delta_i(s,v) \ge \text{dist}_{G_i''}(s,v)$ and
- if $\text{dist}_{G_i''}(s,v) \le R$, then $\delta_i(s,v) \le (1+\epsilon)\text{dist}_{G_i''}(s,v)$.

We let our algorithm return the distance estimate

$$\delta(s,v) = \min_{0 \le i \le \lfloor \log nW \rfloor} \varphi_i \delta_i(s,v).$$

We now show that there is some $0 \le i \le \lfloor \log(nW) \rfloor$ such that $\varphi_i \delta_i(s,v) \le (1+3\epsilon)\text{dist}_G(s,v)$. As $\delta(s,v)$ is the minimum of all the distance estimates, this implies that $\delta(s,v) \le (1+3\epsilon)\text{dist}_G(s,v)$. In particular, we know that there is some $0 \le i \le \lfloor \log(nW) \rfloor$ such that $2^i \le \text{dist}_G(s,v) \le 2^{i+1}$, since W is the maximum edge weight and all paths consist of at most n edges. Consider a shortest path π from s to v in G whose weight is equal to $\text{dist}_G(s,v)$. Let $w_G(\pi)$ and $w_{G_i'}(\pi)$ denote the weight of the path π in G and G_i', respectively. Since π consists of at most n edges, we have $w_{G_i'}(\pi) \le w(\pi) + n\varphi_i$. Therefore, we get

$$\begin{aligned}\text{dist}_{G_i'}(s,v) \le w_{G_i'}(\pi) \le w(\pi) + n\varphi_i = \text{dist}_G(s,v) + \epsilon 2^i &\le \text{dist}_G(s,v) + \epsilon\, \text{dist}_G(s,v) \\ &= (1+\epsilon)\text{dist}_G(s,v).\end{aligned}$$

Now, observe the following:

$$\begin{aligned}\text{dist}_{G_i''}(s,v) = \frac{\text{dist}_{G_i'}(s,v)}{\varphi_i} \le \frac{(1+\epsilon)\text{dist}_G(s,v)}{\varphi_i} \le \frac{2\text{dist}_G(s,v)}{\varphi_i} &= \frac{2\text{dist}_G(s,v)n}{\epsilon 2^i} \\ &\le \frac{2 \cdot 2^{i+1} n}{\epsilon 2^i} = \frac{4n}{\epsilon} = R.\end{aligned}$$

Since $\text{dist}_{G_i''}(s,v) \le R$, we get $\delta_i(s,v) \le (1+\epsilon)\text{dist}_{G_i''}(s,v)$ by Lemma 6.1. Thus, we get

$$\begin{aligned}\varphi_i \delta_i(s,v) \le \varphi_i((1+\epsilon)\text{dist}_{G_i''}(s,v)) = (1+\epsilon)\text{dist}_{G_i'}(s,v) &\le (1+\epsilon)^2 \text{dist}_G(s,v) \\ &\le (1+3\epsilon)\text{dist}_G(s,v)\end{aligned}$$

as desired.

We now analyze the running time of this algorithm. By Lemma 6.1, for every $0 \le i \le \lfloor \log(nW) \rfloor$, maintaining $\delta_i(s,v)$ on G_i'' for every node v takes time $O(m^{1+O((\log\log R)/q)} R^{2/q} + n)$, where

$$q = \left\lfloor \sqrt{\left\lfloor \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{12 \cdot 4^3 \log n}{\epsilon}\right)}} \right\rfloor} \right\rfloor.$$

By our choice of $R = 4n/\epsilon$, the total update time for maintaining all these $\lfloor \log(nW) \rfloor$ distance estimates is $O(m^{1+O((\log\log(n/\epsilon))/q)} \log W/\epsilon)$ in expectation. To obtain a $(1+\epsilon)$-approximation (instead of a $(1+3\epsilon)$-approximation, we simply run the whole algorithm with $\epsilon' = \epsilon/3$. This results in a total update time of $O(m^{1+O((\log\log(n/\epsilon))/q)} \log W/\epsilon)$, where

$$q = \left\lfloor \sqrt{\left\lfloor \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{36 \cdot 4^3 \log n}{\epsilon}\right)}} \right\rfloor} \right\rfloor.$$

Now observe that $1/\epsilon \leq n^{1/q}$ and that

$$O\left(\frac{\log\log\left(\frac{n}{\epsilon}\right)}{q}\right) = O\left(\frac{\left(\log\log\left(\frac{n}{\epsilon}\right)\right)\left(\log\left(\frac{\log n}{\epsilon}\right)\right)^{1/4}}{(\log n)^{1/4}}\right) = O\left(\frac{\left(\log\left(\frac{\log n}{\epsilon}\right)\right)^{5/4}}{(\log n)^{1/4}}\right).$$

The total update time therefore is

$$O(m^{1+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)} \log W + n).$$

If $1/\epsilon = \text{polylog} n$, then the total update time is $O(m^{1+(\log^{5/4}\log n)/\log^{1/4} n} \log W + n)$, which is $O(m^{1+o(1)} \log W + n)$, since $\lim_{x\to\infty}(\log^{5/4}\log n)/\log^{1/4} n = 0$.

The query time of the algorithm described above is $O(\log(nW))$ as it has to compute $\delta(s,v) = \min_{0\leq i\leq \lfloor \log nW\rfloor} \varphi_i \delta_i(s,v)$ when asked for the approximate distance from s to v. We can reduce the query time to $O(1)$ by using a min-heap for every node v that stores $\delta_i(s,v)$ for all $0 \leq i \leq \lfloor \log(nW)\rfloor$. This allows us to query for $\delta(s,v)$ in constant time and does not increase our asymptotic bound on the total update time. □

6.2 Approximate APSP

We now show how to use our techniques to obtain a decremental approximate APSP algorithm. This is conceptually simple now. We use the approximate SSSP algorithm from Theorem 6.3 and plug it into the algorithm for maintaining approximate balls from Proposition 4.1. By using an adequate query procedure, we can use the distance estimates maintained for the approximate balls to return the approximate distances between any two nodes.

Theorem 6.4. *For every $k \geq 2$ and every $0 < \epsilon \leq 1$, there is a decremental approximate APSP algorithm that upon a query for the approximate between any pair of nodes u and v returns a distance estimate $\delta(u,v)$ such that $\text{dist}_G(u,v) \leq \delta(u,v) \leq ((2+\epsilon)^k - 1)\text{dist}_G(u,v)$. It has a query time of $O(k^k)$ and a total update time of*

$$O(m^{1+1/k+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)} \log^2 W + n)$$

in expectation. If $1/\epsilon = \text{polylog}\ n$, then the total update time is $O(m^{1+1/k+o(1)} \log^2 W + n)$ in expectation.

Proof. We use the approximate SSSP algorithm of Theorem 6.3 that provides a $(1+\epsilon)$-approximation and has an total update time of

$$T(m,n) = O(m^{1+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)} \log W + n)$$

in expectation, and if $1/\epsilon = \text{polylog}\, n$, then the total update time is $T(m,n) = O(m^{1+o(1)} \log W + n)$ in expectation. By Proposition 4.1, we can maintain approximate balls with $D = 5^k nW$ in total update time

$$t(m,n,k,\epsilon) = O\left(\left(km^{1+1/k} + \sum_{0\leq i\leq k-1} \frac{m}{m^{i/k}} \cdot T(m_i,n_i)\right) k \log n \frac{\log(nW)}{\epsilon} + k \cdot T(m,n)\right)$$

in expectation, where, for each $0 \leq i \leq k-1$, $m_i = O(m^{(i+1)/k})$ and $n_i = O(m^{(i+1)/k})$. After simplification, we have $t(m,n,k,\epsilon) = O(m^{1+1/k+O(\log^{5/4}((\log n)/\epsilon)/\log^{1/4} n)} \log^2 W + n)$ and $t(m,n,k,\epsilon) = O(m^{1+1/k+o(1)} \log^2 W + n)$ if $1/\epsilon = \text{polylog}\, n$ as desired.

Additionally, we maintain, for every node $v \in V$ and every $1 \leq i \leq k-1$, the node $c_i(v)$, which is a node with minimum $\delta(u,v)$ among all nodes u of priority i such that $v \in B(u)$. This can be done as follows. For every node v, we maintain a heap containing all nodes u of priority i such

that $v \in B(u)$ using the key $\delta(u, v)$. Every time v joins or leaves $B(u)$, we insert or remove u from the heap of v. Every time $\delta(u, v)$ changes, we update the key of u in the heap of v. After each insert, remove, or update in the heap of some node v, we find the minimal element $c_i(v)$ of the heap. As each heap operation takes logarithmic time, the total update time of the algorithm of Proposition 4.1 only increases by a logarithmic factor, which does not alter the overall running time bound of our algorithm.

To answer a query for the approximate distance between a pair of nodes u and v, we use Procedure Query. This procedure first tests whether $v \in B(u)$ and if yes returns $\delta(u, v)$. Otherwise, it does the following for every $j \geq i + 1$, where i is the priority of u: It first computes the node $c_j(u)$, which among the nodes v' of priority j with $u \in B(v')$ is the one with the minimum value of $\delta(v', u)$. Then, it recursively queries for the approximate distance $\delta'(c_j(u), v)$ from $c_j(u)$ to v and sets the distance estimate via $c_j(u)$ to $\delta'_j(u, v) = \delta(v, c_j(u)) + \delta'(c_j(u), v)$. Finally, it returns the minimum of all distance estimates $\delta'_j(u, v)$.

Procedure Query(u, v)

```
if v ∈ B(u) then
    δ'(u, v) ← δ(u, v)
else
    Set i to the priority of u
    foreach j = i + 1 to k − 1 do
        if c_j(u) exists then
            v'' ← c_j(u)
            δ'(v'', v) ← Query (v'', v)
            δ'_j(u, v) ← δ(v, v'') + δ'(v'', v)
        else
            δ'_j(u, v) ← ∞
        end
    end
    δ'(u, v) ← min_{i+1≤j≤k−1} δ'_j(u, v)
end
return δ'(u, v)
```

Note that in each instance there are $O(k)$ recursive calls and with each recursive call the priority of u increases by at least one. Thus, the running time of the query procedure is $O(k^k)$.

Claim 6.5. *For every pair of nodes u and v the distance estimate $\delta'(u, v)$ computed by Procedure Query satisfies $\delta'(u, v) \leq (((1 + \epsilon)^2 + 1)^{k-i} - 1)\text{dist}_G(u, v)$, where i is the priority of u.*

Proof. The proof is by induction on the priority i of u. Let $\delta'(u, v)$ denote the distance estimate returned by Procedure Query. If $i = k - 1$, then by Proposition 4.1 and our choice of D, we know that $v \in B(u)$, and thus $\delta'(u, v) = \delta(u, v) \leq (1 + \epsilon)\text{dist}_G(u, v)$. If $i < k - 1$, then we distinguish between the two cases $v \in B(u)$ and $v \notin B(u)$. If $v \in B(u)$, then $\delta'(u, v) = \delta(u, v) \leq (1 + \epsilon)\text{dist}_G(u, v)$. If $v \notin B(u)$, then by Proposition 4.1 and our choice of D there is a node v' of priority $j > i$ such that $u \in B(v')$ and $\text{dist}_G(u, v') \leq (1 + \epsilon)^2((1 + \epsilon)^2 + 1)^{j-i-1}\text{dist}_G(u, v)$.

We will now argue that $\delta'_j(u, v) \leq 2((1 + \epsilon)^3 + 1)^{k-1-i} - 1)\text{dist}_G(u, v)$, which implies the same upper bound for $\delta'(u, v)$. Set $v'' \leftarrow c_j(u)$. Since both v'' and v' have priority j and $u \in B(v')$ as well as $v \in B(v'')$, we have $\delta(u, v'') \leq \delta(u, v')$ by the definition of v''. Since

$\delta(u, v') \leq (1+\epsilon)\text{dist}_G(u, v')$, we have

$$\delta(u, v'') \leq (1+\epsilon)\text{dist}_G(u, v') \leq (1+\epsilon)^3((1+\epsilon)^2+1)^{j-i-1}\text{dist}_G(u, v)$$
$$\leq (1+\epsilon)^3((1+\epsilon)^3+1)^{j-i-1}\text{dist}_G(u, v).$$

To simplify the presentation in the following, we set $a = (1+\epsilon)^3$ and thus have $\delta(u, v'') \leq a(a+1)^{j-i-1}\text{dist}_G(u, v)$. By the triangle inequality, we have

$$\text{dist}_G(v'', v) \leq \text{dist}_G(v'', u) + \text{dist}_G(u, v) \leq \delta(v'', u) + \text{dist}_G(u, v)$$
$$\leq (a(a+1)^{j-i-1}+1)\text{dist}_G(u, v),$$

and by the induction hypothesis, we have

$$\delta'(v'', v) \leq (2(a+1)^{k-1-j}-1)\text{dist}_G(v'', v)$$
$$\leq (2(a+1)^{k-1-j}-1)(a(a+1)^{j-i-1}+1)\text{dist}_G(u, v).$$

Since $j \geq i+1$, we get

$$\begin{aligned}
\delta'_j(u, v) &= \delta(u, v'') + \delta'(v'', v) \\
&\leq \left(a(a+1)^{j-i-1} + (2(a+1)^{k-1-j}-1)(a(a+1)^{j-i-1}+1)\right)\text{dist}_G(u, v) \\
&= \left(2(a+1)^{k-1-j}(a(a+1)^{j-i-1}+1)-1\right)\text{dist}_G(u, v) \\
&= \left(2a(a+1)^{k-1-(i+1)} + 2(a+1)^{k-1-j}) - 1\right)\text{dist}_G(u, v) \\
&\leq \left(2a(a+1)^{k-1-(i+1)} + 2(a+1)^{k-1-(i+1)}) - 1\right)\text{dist}_G(u, v) \\
&= \left(2(a+1)^{k-1-(i+1)}(a+1)-1\right)\text{dist}_G(u, v) \\
&= (2(a+1)^{k-1-i}-1)\text{dist}_G(u, v).
\end{aligned}$$

□

Note that $2 \leq ((1+\epsilon)^3+1)$, and therefore we have $\delta'(u, v) \leq (((1+\epsilon)^3+1)^{k-i}-1)\text{dist}_G(u, v)$. Furthermore, $(1+\epsilon)^3 \leq 1+7\epsilon$ and in the worst case $i = 0$. Thus, by running the whole algorithm with $\epsilon' = \epsilon/7$, we can guarantee that $\delta'(u, v) \leq ((2+\epsilon)^k-1)\text{dist}_G(u, v)$. □

7 CONCLUSION

In this article, we showed that single-source shortest paths in undirected graphs can be maintained under edge deletions with near-linear total update time and constant query time. The main approach is to maintain an $(n^{o(1)}, \epsilon)$-hop set of near-linear size in near-linear time. We leave two major open problems. The first problem is whether the same total update time can be achieved for directed graphs, substantially improving the current $mn^{0.9+o(1)}$ total update time by References [18, 19]. This problem is very challenging, because a suitable hop set for directed graphs is not known even in the static setting. Moreover, improving the current $\tilde{O}(m\sqrt{n})$ total update time by Reference [10] for the decremental reachability problem is already very interesting.

The second major open problem is to derandomize our algorithm. The main task here is to deterministically maintain the priority-induced balls of the nodes up to small bounded distance, which is the key to maintaining the hop set. A related question is whether the algorithm of Roditty and Zwick [33] for decrementally maintaining the priority-induced clusters of Thorup and Zwick [35] up to small bounded distance (and the corresponding spanners and emulators) can be derandomized, which is possible in the static setting [30]. We have previously demonstrated [20] how to derandomize the decremental $(1+\epsilon)$-approximate APSP algorithm of Roditty and Zwick [33], but the technique does not carry over to maintaining the clusters. Using a principally different approach, Bernstein and Chechik [7] have recently introduced a technique to deterministically

maintain approximate SSSP under edge deletions yielding total update times of $\tilde{O}(n^2 \log W)$ in weighted graphs [6] and $\tilde{O}(mn^{3/4})$ in unweighted graphs [8], respectively. Can their technique be extended to obtain a deterministic algorithm that is as fast as our randomized one in the sparse regime?

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers of FOCS and JACM for their valuable feedback.

REFERENCES

[1] Amir Abboud, Greg Bodwin, and Seth Pettie. 2017. A hierarchy of lower bounds for sublinear additive spanners. In *Proceedings of the Symposium on Discrete Algorithms (SODA'17)*. 568–576. DOI: https://doi.org/10.1137/1.9781611974782.36

[2] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'14)*. 434–443. DOI: https://doi.org/10.1109/FOCS.2014.53

[3] Ittai Abraham, Shiri Chechik, and Kunal Talwar. 2014. Fully dynamic all-pairs shortest paths: Breaking the $O(n)$ barrier. In *Proceedings of the International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'14)*. 1–16. DOI: https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2014.1

[4] Aaron Bernstein. 2009. Fully dynamic $(2 + \epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'09)*. 693–702. DOI: https://doi.org/10.1109/FOCS.2009.16

[5] Aaron Bernstein. 2016. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.* 45, 2 (2016), 548–574. DOI: https://doi.org/10.1137/130938670

[6] Aaron Bernstein. 2017. Deterministic partially dynamic single source shortest paths in weighted graphs. In *Proceedings of the International Colloquium on Automata, Languages, and Programming (ICALP'17)*. 44:1–44:14. DOI: https://doi.org/10.4230/LIPIcs.ICALP.2017.44

[7] Aaron Bernstein and Shiri Chechik. 2016. Deterministic decremental single source shortest paths: Beyond the $o(mn)$ bound. In *Proceedings of the Symposium on Theory of Computing (STOC'16)*. 389–397. DOI: https://doi.org/10.1145/2897518.2897521

[8] Aaron Bernstein and Shiri Chechik. 2017. Deterministic partially dynamic single source shortest paths for sparse graphs. In *Proceedings of the Symposium on Discrete Algorithms (SODA'17)*. 453–469. DOI: https://doi.org/10.1137/1.9781611974782.29

[9] Aaron Bernstein and Liam Roditty. 2011. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *Proceedings of the Symposium on Discrete Algorithms (SODA'11)*. 1355–1365. DOI: https://doi.org/10.1137/1.9781611973082.104

[10] Shiri Chechik, Thomas Dueholm Hansen, Giuseppe F. Italiano, Jakub Lacki, and Nikos Parotsidis. 2016. Decremental single-source reachability and strongly connected components in $\tilde{O}(m\sqrt{n})$ total update time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'16)*. 315–324. DOI: https://doi.org/10.1109/FOCS.2016.42

[11] Edith Cohen. 1998. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.* 28, 1 (1998), 210–236. DOI: https://doi.org/10.1137/S0097539794261295

[12] Edith Cohen. 2000. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM* 47, 1 (2000), 132–166. DOI: https://doi.org/10.1145/331605.331610

[13] E. A. Dinic. 1970. An algorithm for the solution of the max-flow problem with the polynomial estimation. *Doklady Akademii Nauk SSSR* 194, 4 (1970). In Russian; English translation: *Soviet Mathematics Doklady* 11, 5 (1970), 1277–1280.

[14] Yefim Dinitz. 2006. Dinitz' algorithm: The original version and Even's version. In *Essays in Memory of Shimon Even*. Springer, 218–240. DOI: https://doi.org/10.1007/11685654_10

[15] Michael Elkin and Ofer Neiman. 2017. Linear-size hopsets with small hopbound, and distributed routing with low memory. *CoRR abs/1704.08468*. arXiv:1704.08468.

[16] Shimon Even and Yossi Shiloach. 1981. An on-line edge-deletion problem. *J. ACM* 28, 1 (1981), 1–4. DOI: https://doi.org/10.1145/322234.322235

[17] Monika Henzinger and Valerie King. 1995. Fully dynamic biconnectivity and transitive closure. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'95)*. 664–672. DOI: https://doi.org/10.1109/SFCS.1995.492668

[18] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2014. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Proceedings of the Symposium on Theory of Computing (STOC'14)*. 674–683. DOI: https://doi.org/10.1145/2591796.2591869

[19] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2015. Improved algorithms for decremental single-source reachability on directed graphs. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'15)*. 725–736. DOI: https://doi.org/10.1007/978-3-662-47672-7_59

[20] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. 2016. Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization. *SIAM J. Comput.* 45, 3 (2016), 947–1006. DOI: https://doi.org/10.1137/140957299

[21] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. 2015. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Symposium on Theory of Computing (STOC'15)*. 21–30. DOI: https://doi.org/10.1145/2746539.2746609

[22] Shang-En Huang and Seth Pettie. 2017. Thorup-Zwick emulators are universally optimal hopsets. *CoRR abs/1705.00327*. arXiv:1705.00327.

[23] Maleq Khan, Fabian Kuhn, Dahlia Malkhi, Gopal Pandurangan, and Kunal Talwar. 2012. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distrib. Comput.* 25, 3 (2012), 189–205. DOI: https://doi.org/10.1007/s00446-012-0157-9

[24] Valerie King. 1999. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'99)*. 81–91. DOI: https://doi.org/10.1109/SFFCS.1999.814580

[25] Philip N. Klein and Sairam Subramanian. 1997. A randomized parallel algorithm for single-source shortest paths. *J. Algor.* 25, 2 (1997), 205–220. DOI: https://doi.org/10.1006/jagm.1997.0888

[26] Jakub Łącki. 2013. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Trans. Algor.* 9, 3 (2013), 27. DOI: https://doi.org/10.1145/2483699.2483707

[27] Aleksander Mądry. 2010. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the Symposium on Theory of Computing (STOC'10)*. 121–130. DOI: https://doi.org/10.1145/1806689.1806708

[28] Danupon Nanongkai. 2014. Distributed approximation algorithms for weighted shortest paths. In *Proceedings of the Symposium on Theory of Computing (STOC'14)*. 565–573. DOI: https://doi.org/10.1145/2591796.2591850

[29] Liam Roditty. 2013. Decremental maintenance of strongly connected components. In *Proceedings of the Symposium on Discrete Algorithms (SODA'13)*. 1143–1150. DOI: https://doi.org/10.1137/1.9781611973105.82

[30] Liam Roditty, Mikkel Thorup, and Uri Zwick. 2005. Deterministic constructions of approximate distance oracles and spanners. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP'05)*. 261–272. DOI: https://doi.org/10.1007/11523468_22

[31] Liam Roditty and Uri Zwick. 2008. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.* 37, 5 (2008), 1455–1471. DOI: https://doi.org/10.1137/060650271

[32] Liam Roditty and Uri Zwick. 2011. On dynamic shortest paths problems. *Algorithmica* 61, 2 (2011), 389–401. DOI: https://doi.org/10.1007/s00453-010-9401-5

[33] Liam Roditty and Uri Zwick. 2012. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.* 41, 3 (2012), 670–683. DOI: https://doi.org/10.1137/090776573

[34] Mikkel Thorup. 1999. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM* 46, 3 (1999), 362–394. DOI: https://doi.org/10.1145/316542.316548

[35] Mikkel Thorup and Uri Zwick. 2005. Approximate distance oracles. *J. ACM* 52, 1 (2005), 74–92. DOI: https://doi.org/10.1145/1044731.1044732

[36] Mikkel Thorup and Uri Zwick. 2006. Spanners and emulators with sublinear distance errors. In *Proceedings of the Symposium on Discrete Algorithms (SODA'06)*. 802–809. http://dl.acm.org/citation.cfm?id=1109557.1109645

[37] Virginia Vassilevska Williams and Ryan Williams. 2010. Subcubic equivalences between path, matrix and triangle problems. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS'10)*. 645–654. DOI: https://doi.org/10.1109/FOCS.2010.67

[38] Uri Zwick. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM* 49, 3 (2002), 289–317. DOI: https://doi.org/10.1145/567112.567114

Received December 2015; revised May 2018; accepted May 2018