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

Monika Henzinger[*] Sebastian Krinninger[*] Danupon Nanongkai[†]

## Abstract

The decremental single-source shortest paths (SSSP) problem concerns maintaining the distances between a given source node $s$ to every node in an $n$-node $m$-edge graph $G$ undergoing edge deletions. While its static counterpart can be easily 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 (JACM 1981) has been the fastest known algorithm for three decades. With the loss of a $(1+\epsilon)$-approximation factor, the running time was recently improved to $O(n^{2+o(1)})$ by Bernstein and Roditty (SODA 2011), and more recently to $O(n^{1.8+o(1)} + m^{1+o(1)})$ by Henzinger, Krinninger, and Nanongkai (SODA 2014). In this paper, we finally bring the running time of this case down to near-linear: We give a $(1+\epsilon)$-approximation algorithm with $O(m^{1+o(1)})$ total update time, thus obtaining *near-linear time*. Moreover, we obtain $O(m^{1+o(1)} \log W)$ time for the weighted case, where the edge weights are integers from 1 to $W$. The only prior work on weighted graphs in $o(mn \log W)$ time is the $O(mn^{0.986} \log W)$-time algorithm by Henzinger, Krinninger, and Nanongkai (STOC 2014) which works for the general weighted directed case.

In contrast to the previous results which rely on maintaining a sparse emulator, our algorithm relies on maintaining a so-called *sparse* $(d, \epsilon)$*-hop set* introduced by Cohen (JACM 2000) in the PRAM literature. A $(d, \epsilon)$-hop set of a graph $G = (V, E)$ is a set $E'$ of weighted edges such that the distance between any pair of nodes in $G$ can be $(1+\epsilon)$-approximated by their $d$-hop distance (given by a path containing at most $d$ edges) on $G' = (V, E \cup E')$. 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 the distances on this hop set, we develop a *monotone bounded-hop Even-Shiloach tree*. It results from extending and combining the monotone Even-Shiloach tree of Henzinger, Krinninger, and Nanongkai (FOCS 2013) with the bounded-hop SSSP technique of Bernstein (STOC 2013). These two new tools might be of independent interest.

[*] University of Vienna, Faculty of Computer Science, Austria. Supported by the Austrian Science Fund (FWF): P23499-N23 and the University of Vienna (IK I049-N). 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 and from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 317532.

[†] University of Vienna, Faculty of Computer Science, Austria. Work partially done while at ICERM, Brown University, USA, and Nanyang Technological University, Singapore 637371, and while supported in part by the following research grants: Nanyang Technological University grant M58110000, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 grant MOE2010-T2-2-082, and Singapore MOE AcRF Tier 1 grant MOE2012-T1-001-094.

# Contents

# 1 Introduction

Dynamic graph algorithms refer to data structures on a graph that support update and query operations. They are classified according to what 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 paper, 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$(x)$: return the distance between node $s$ and node $x$ in the current graph $G$, denoted by $\operatorname{dist}_G(s, x)$.

The *weighted* case allows an additional operation Increase$(u, v, i)$ which increases the weight of the edge $(u, v)$ by $i$. 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 *$\alpha$-approximation* algorithm if, for any distance query Distance$(x)$, it returns $\delta(s, x)$ such that such that $\operatorname{dist}_G(s, x) \leq \delta(s, x) \leq \alpha \operatorname{dist}_G(s, x)$. 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 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 have $W$, the maximum edge weight. We use the $\widetilde{O}$-notation to hide $O(\operatorname{poly}\log n)$ terms. In this paper, we focus on algorithms with small ($O(1)$ or $O(\operatorname{poly}\log n)$) query time, and the main goal is to minimize the total update time, which will simply referred to as *time* when the context is clear.

**Previous Work.** The static version of SSSP can be easily solved in $\widetilde{O}(m)$ time using, e.g., Dijkstra's algorithm. Moreover, due to the deep result of Thorup [22], it can even be solved in linear ($O(m)$) time. This implies that we can naively solve decremental SSSP in $O(m^2)$ time by running the static algorithm after every deletion. The first non-trivial decremental algorithm is due to Even and Shiloach [6] from 1981 and takes $O(mn)$ time on unweighted undirected graph. This algorithm will be referred to as *ES-tree* throughout this paper. It has many potential applications such as for decremental strongly-connected components and multicommodity flow problems [17]; yet, the ES-tree has resisted many attempts to improve it for decades. Roditty and Zwick [20] explained this phenomenon by showing 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 [26]. It is thus natural to shift the focus to *approximation algorithms.*

The first improvement was due to Bernstein and Roditty [4] who presented a $(1+\epsilon)$-approximation algorithm with $O(n^{2+O(1/\sqrt{\log n})})$ expected time for the case of undirected unweighted graphs. This time bound is only slightly larger than quadratic time and beats the $O(mn)$ time of the ES-tree unless the input graph is very sparse. Recently, we [10] beat this near-quadratic time with an $O(n^{1.8+O(1/\sqrt{\log n})} + m^{1+O(1/\sqrt{\log n})})$-time algorithm. For the more general cases, Henzinger and King [11] observed that the ES-tree can be easily adapted to directed graphs. King [13] later extended the ES-tree to an $O(mnW)$-time algorithm on directed weighted graphs. The techniques used in recent algorithms of Bernstein [2, 3] and Mądry [15] give a $(1 + \epsilon)$-approximate $\widetilde{O}(mn \log W)$-time algorithm on directed weighted graphs. Very recently, we extended our insights from [10] to obtain a $(1+\epsilon)$-approximation algorithm with roughly $\widetilde{O}(mn^{0.986})$ time for decremental approximate SSSP in directed graphs [9], giving 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 [19, 14, 17].

This algorithm can also be extended to achieve a total update time of $O(mn^{0.986}\log^2 W)$ for the weighted case. Also very recently, Abboud and Vassilevska Williams [1] showed that "deamortizing" our algorithms in [9] might not be possible: a combinatorial algorithm with *worst case* update time and query time of $O(n^{2-\delta})$ (for some $\delta > 0$) 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 [9] can do) a worst case update and query time of $O(m^{1-\delta})$ (for some $\delta > 0$) will falsify the strong exponential time hypothesis.

**Our Results.** Given the significance of the decremental SSSP problem, it is important to understand the time complexity of this problem. Recent progress on this problem leads to the hope for a near-linear time algorithm – as fast as the static case – for the general directed weighted case and many exciting potential applications. This goal, however, is still far from reality: even in the undirected unweighted case, we only have an $O(n^{1.8+O(1/\sqrt{\log n})} + m^{1+O(1/\sqrt{\log n})})$ time which is near-linear only when the graph is very dense ($m = \Omega(n^{1.8})$).

In this paper, we make one step toward this goal. We obtain a near-linear time algorithm for $(1+\epsilon)$-approximate decremental SSSP in undirected graphs. Our algorithm is correct with high probability and has an expected total update time of $O(m^{1+O(\sqrt{\log\log n/\log n})})$ for unweighted graphs and $O(m^{1+O(\sqrt{\log\log n/\log n})}\log W)$ for weighted graphs. It maintains an estimate of the distance between the source node $s$ and every other node, guaranteeing constant query time in the worst case. In the unweighted case, our algorithm significantly improves our previous algorithm in [10] as discussed above. There was no previous algorithm designed specifically for weighted undirected graphs, and the previous best running time for this case come from our $O(mn^{0.986}\log^2 W)$ time for weighted directed graphs [9].

# 2 Overview of Techniques

## 2.1 Previous Approach

To motivate our approach, note that previous algorithms [4, 10] rely on maintaining an ES-tree on a *sparse emulator* – a sparse graph that preserves the distances of the original graph – and running the ES-tree or its variants on top of this emulator. The time of the ES-tree is $O(m'n)$ where $m'$ is the number of edges ever contained in the emulator. We can maintain an emulator with $m' = O(n^{1+o(1)})$ efficiently by using the algorithm of Roditty and Zwick [21] to maintain an emulator of Thorup and Zwick [23, 24] (TZ-emulator thereafter). This gives the running time of $O(n^{2+o(1)})$ if we use the ES-tree [4], and this running time can be improved to $O(n^{1.8+o(1)} + m^{1+o(1)})$ by using a variant of the ES-tree we introduced in [8] called *lazy* ES-tree. (Note that this is an over-simplification. One difficulty faced by previous algorithms [4, 10] is the fact that the emulator has edge insertions and the original ES-tree cannot deal with this. This difficulty is also one factor that makes the present paper fairly technical.) Extending this approach further, especially to get a near-linear-time algorithm, is seemingly very difficult. Consider, for example, an input graph that is already sparse. In this case, the above emulator approach does not help at all, and a completely new idea is needed.

## 2.2 Main Tools

**Hop Set.** Our algorithm uses a rather different approach based on a so-called *sparse hop set*. The *d-hop distance* between any nodes $u$ and $v$ is the weight of the shortest path among all paths between $u$ and $v$ containing at most $d$ edges. Given any graph $G = (V, E)$, we say that a set of weighted edges $E'$ is a $(d, \epsilon)$-*hop set* if the distance between any two nodes in $G$ can be $(1+\epsilon)$-approximated by a path between them in $G' = (V, E \cup E')$ containing at most $d$ edges. To be precise, let the *$d$-hop distance* between any nodes $u$ and $v$ in $G'$, denoted by $\operatorname{dist}^d_{G'}(u, v)$, be the weight of the shortest path between $u$ and $v$ in $G'$ containing at most $d$ edges. Then, $E'$ is a $(d, \epsilon)$-hop set, if for any $u$ and $v$,

$$\operatorname{dist}_G(u, v) \leq \operatorname{dist}^d_{G'}(u, v) \leq (1+\epsilon) \operatorname{dist}_G(u, v).$$

We call $G'$ a $(d, \epsilon)$-*shortcut graph.* As a part of our algorithm, we maintain an $(n^{o(1)}, \epsilon)$-hop set of size $O(m^{1+o(1)})$ in $O(m^{1+o(1)})$ time. (The situation is actually more complicated than this, as we will explain later.)

The notion of hop set was first introduced by Cohen [5] in the PRAM literature and is conceptually related to the notion of emulator. It is also related to the notion of *shortest-paths diameter* used in distributed computing (e.g. [12, 16]). To the best of our knowledge, the only place that this hop set concept was used before in the dynamic algorithm literature (without the name mentioned) is Bernstein's fully dynamic $(2+\epsilon)$-approximation algorithm for all-pairs shortest paths [2]. There, an $(n^{o(1)}, \epsilon)$-hop set is essentially recomputed from scratch after every edge update, and a single-source shortest-paths data structure is maintained on top of this hop set.

**Monotone Bounded-Hop ES-tree.** Suppose that we can maintain a $(d, \epsilon)$-hop set efficiently. We now wish to maintain the $d$-hop distances from $s$ in the corresponding shortcut graph. A tool that can almost do this job is Bernstein and Mądry's extension of the ES-tree [2, 3, 15] which we will call a *$d$-hop ES-tree.* This algorithm can maintain the $d$-hop distance from $s$ in weighted $m'$-edge graphs undergoing edge deletions in $\widetilde{O}(m'd)$ time. We cannot use this algorithm directly because we sometimes might have to *insert* edges into the hop set we maintain, which the $d$-hop ES-tree cannot handle. Fortunately, this situation has been already encountered many times in the past when we want to maintain distances on an emulator (e.g. [4, 7, 10]). A powerful idea that usually got us around this problem is the *monotone ES-tree* introduced by us in [7]. The idea is quite simple: If an edge insertion will make the distance between two nodes that we maintain smaller, we simply ignore it, thus keeping the maintained distance estimate monotonically non-decreasing. Analyzing the running time of this algorithm is fairly easy (it follows from the monotonicity of the maintained distance). Analyzing that it gives a good distance estimate is, however, quite challenging.

In this paper, we apply this idea to extend Bernstein's $d$-hop ES-tree to a *monotone $d$-hop ES-tree.* We again successfully analyze its distance estimate using the hop set we constructed. To do this, we need some new (although not groundbreaking) ideas since the previous analyses for monotone ES-trees only work when we maintain an ES-tree up to some distance value, regardless of the number of hops. To maintain the distances up to some number of hops, we need to extend the previous induction-based arguments and use an induction on the number of hops instead. In this way we can argue that the errors caused from Bernstein's technique do not aggregate too much.

**A Showcase: $\mathbf{O(mn^{1/2+o(1)})}$-Time Algorithm via TZ-Emulator.** To illustrate the advantage of using a hop set, we note that by combining Bernstein's analysis [2] with the algorithm of Roditty and Zwick [21], it can be shown that the TZ-emulator is an $(\widetilde{O}(d), \epsilon)$-hop set of size $O(n^{1+o(1)})$ that can be maintained in $O(mn^{1+o(1)}/d)$ time, for any $d \geq 1$. By maintaining $d$-hop distances between $s$ and every other node on the corresponding shortcut graph using our monotone $d$-hop ES-tree, we can maintain $(1+\epsilon)$-approximate single-source shortest-paths in $O(mn^{1+o(1)}/d + md)$ time[1] which is $O(mn^{1/2+o(1)})$ when we set $d = n^{1/2}$. This approach already gives a huge improvement over the

[1] We note one technical detail here: In fact, we cannot maintain $(1+\epsilon)$-approximate single-source shortest-paths on such TZ-emulator in $O(mn^{1+o(1)}/d + md)$ time, and we have to slightly modify the TZ-emulator; see Section 3 for detail.

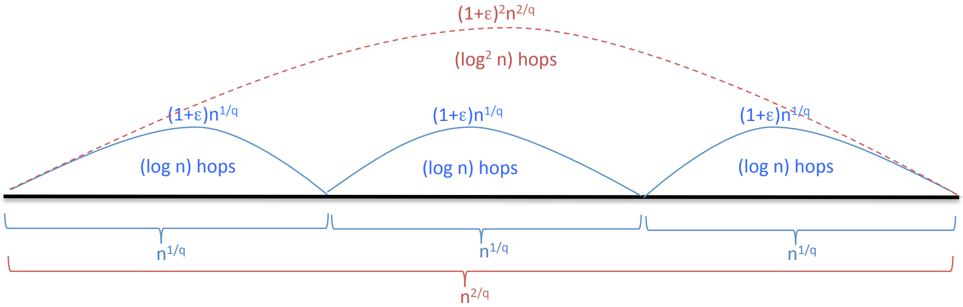

**Figure 1:** How our multi-layer hop set construction conceptually works: The straight thick line (in black) represents a path in the original graph $G_0$. Thin curved lines (in blue) represent $(\log^{O(1)} n)$-hop paths in an $n^{1/q}$-restricted $(\widetilde{O}(1), \epsilon)$-hop set of $G_0$. These edges provide a $\widetilde{O}(n^{1-1/q})$-hop path that $(1+\epsilon)$-approximate the original path; they shrink the original path by a factor of $\widetilde{O}(1/n^{1/q})$. If we compute an $n^{1/q}$-restricted $(\widetilde{O}(1), \epsilon)$-hop set of these edges, we get edges like the dashed curved lines (in red). These edges further shrink the previous path by a factor of $\widetilde{O}(1/n^{1/q})$.

previous total update time of $O(n^{1.8+o(1)} + m^{1+o(1)})$ for sparse graphs. To maintain an $(n^{o(1)}, \epsilon)$-hop set, we unfortunately cannot simply use the TZ-emulator since we have to set $d = n^{o(1)}$ which will make the maintenance time too large (roughly $O(mn)$). We need some more ideas for this as explained next.

### 2.3 Towards Efficient Maintenance of $(n^{o(1)}, \epsilon)$-Hop Set

**Restricted Hop Set.** To get an $(n^{o(1)}, \epsilon)$-hop set, we will compute an *$R$-restricted* $(n^{o(1)}, \epsilon)$-hop set as a subroutine. The $R$-restricted hop set is similar to the standard hop set except that the guarantee holds only for paths containing at most $R$ hops. That is, a shortcut graph $G'$ corresponding to an *$R$-restricted* $(n^{o(1)}, \epsilon)$-hop set of a graph $G$ has the property that for any pair of nodes $u$ and $v$,

$$\operatorname{dist}^R_G(u,v) \le \operatorname{dist}^d_{G'}(u,v) \le (1+\epsilon)\operatorname{dist}^R_G(u,v). \tag{1}$$

In other words, a $(d, \epsilon)$-hop set is an $n$-restricted $(d, \epsilon)$-hop set. (We note that the notion of restricted hop set was also defined by Cohen [5] and used to construct a hop set. Our definition is different from Cohen's in that the guarantee of Cohen's restricted hop set holds only for paths containing at most $R$ hops and *at least* $R/2$ hops.) Using Bernstein's proof [2], it can be easily shown that the TZ-emulator does not only provide a $(d, \epsilon)$-hop set, but also provide an $R$-restricted $(d, \epsilon)$-hop set. By adapting the algorithm of Roditty and Zwick [21], we can maintain an $R$-restricted $(d \log^{O(1)} n, \epsilon)$-hop set in $O(mn^{o(1)}R/d)$ total update time.

**A Multi-Layer Construction.** The above restricted hop set naturally leads to the following idea for constructing a $(n^{o(1)}, \epsilon)$-hop set. Let $G_0 = (V, E_0)$ be the original graph and set $q = \log^{1/2} n$ (so $1/q = o(1)$). We first maintain an $R$-restricted $(\log^{O(1)} n, \epsilon)$-hop set denoted by $E_1$, where $R = n^{1/q}$. This can be done in $O(mn^{o(1)})$ time by maintaining the TZ-emulator. Let $G_1 = (V, E_0 \cup E_1)$ be the resulting shortcut graph. Note that $G_1$ has $m + n^{1+o(1)}$ edges. Intuitively, since $G_1$ contains a shortcut of length $\log^{O(1)} n$ for shortest paths having at most $R$ hops in $G_0$, it "shrinks" the number hops of every shortest path by a factor of $(\log^{O(1)} n)/R$ (with the cost of $(1+\epsilon)$ multiplicative error). (See Figure 1 for an illustration.) If we again maintain an $R$-restricted $(\log^{O(1)} n, \epsilon)$-hop set on $G_1$, say $E_2$, then we will further shrink the number of hops in paths in $G_0$ by another factor of $(\log^{O(1)} n)/R$ in $G_2 = (V, E_0 \cup E_1 \cup E_2)$. If we keep computing an $R$-restricted $(\log^{O(1)} n, \epsilon)$-hop set on $G_k = (V, E_0 \cup \ldots \cup E_k)$ to get a shortcut graph $G_{k+1} = (V, E_0 \cup \ldots \cup E_{k+1})$, we will eventually arrive at a graph $G_q$ where the number of hops in paths in $G_0$ is shrunk by a factor of $(\log^{O(q)} n)/R^q = (\log^{O(q)} n)/n$ with the cost of $(1+\epsilon)^q$ multiplicative error. In other words, every shortest path in $G_0$ can be $(1+\epsilon)^q$-approximated by a $(\log^{O(q)} n)$-hop path in $G_q$. Thus, for $d = \log^{O(q)} n$,

$$\operatorname{dist}_{G_0}(u,v) \leq \operatorname{dist}^d_{G_q}(u,v) \leq (1+\epsilon)^q \operatorname{dist}_{G_0}(u,v).$$

Figure 1 illustrates the idea of this multi-layer construction. By setting $\epsilon$ small enough, i.e., $\epsilon = \omega(1/q)$, the approximation factor $(1+\epsilon)^q$ will be $(1+\epsilon')$ for an arbitrarily small constant $\epsilon'$. This implies that $E_q$ is the $(n^{o(1)}, \epsilon')$-hop set that we want. Since computing an $n^{1/q}$-restricted $(\log^{O(1)} n, \epsilon)$-hop set on each $G_k$ takes $O(mn^{o(1)})$ time the total update time needed to maintain this hop set is $O(m^{1+o(1)})$ as desired.

**Beyond the TZ-Emulator.** The multi-layer construction explained above almost works perfectly except for one problem: we do not know how to maintain the TZ-emulator on the shortcut graphs $G_k$. One important reason is that each $G_k$ is a dynamic graph with both edge deletions and *insertions* (edge insertions are caused by the restricted hop set $E_k$). We do not have many tools to deal with this type of change. One exception is the monotone bounded-hop ES-tree that we explained earlier. It is in fact not clear to us how to modify the decremental algorithm of Roditty and Zwick to maintain the TZ-emulator in this situation. For this reason, we will abandon the TZ-emulator and construct our own restricted hop set.

Our hop set has the special property that it can be efficiently maintained simply by maintaining some monotone bounded-hop ES-trees. It also has properties similar to the TZ-emulator; i.e., it is an $R$-restricted $(d \log^{O(1)} n, \epsilon)$-hop set that can be maintained in $O(mn^{o(1)}R/d)$ total update time, and on which we can maintain a monotone bounded-hop ES-tree that provides good distance estimates. In fact, it is also an emulator that can guarantee some additive error on an unweighted undirected graph. Its construction is similar to the TZ-emulator; i.e, nodes are assigned priorities based on some random sampling, and we will have an edge between two nodes if some certain rule is satisfied. The main difference is that we introduce the notion of *active* and *inactive* nodes. Roughly speaking, inactive nodes are nodes for which it is too expensive to maintain (monotone) ES-trees rooted at them. Our construction and analysis make sure that the guarantees hold even when we do not maintain (monotone) ES-trees for inactive nodes. Additionally, to handle errors caused by monotone bounded-hop ES-trees and shortcut graphs, our hop set needs a more sophisticated rule for when we will have an edge between two nodes. The description of our hop set is quite complicated, and the most difficult part of this work is to come up with the right hop set that has all the properties we want. This hop set is described in Section 5.1.

**Note on Some Technical Details.** For a technical reason, when we actually implement and analyze the multi-layer construction, we will *not* argue that $E_k$ is an $R$-restricted $(\log^{O(1)} n, \epsilon)$-hop set of $G_{k-1}$ as intuitively explained above. This is hard to argue since $G_k$ might have edge insertions. Instead, we will directly argue that, for every $k$, every path in $G_0$ of weight at most $R^k$ can be $(1+\epsilon)^k$-approximated by a $(\log^{O(k)} n)$-hop path in the shortcut graph $G_k$ (i.e., they are shrunk by a factor of $\log^{O(k)} n/R^k$). Moreover, due to errors caused by many parts of the algorithm, the approximation factor will not be exactly $(1+\epsilon)^k$.

# 3 Warm Up: $\mathbf{O(mn^{1/2+o(1)})}$ Total Update Time

In the following we give a simplified version of our algorithm for unweighted, undirected graphs with a total update time of $O(mn^{1/2+o(1)})$. Our goal is to maintain $(1+\epsilon)$-approximate SSSP from a node $s$ in a graph $G$ undergoing edge deletions. We assume that $0 < \epsilon \leq 1$ is a constant. For technical reasons we further assume in this section that $1/\epsilon$ is integer.

In the algorithm, we use the following parameters. We set

$$p = \sqrt{\log n} / \sqrt{\log 9/\epsilon} \, .$$

With this choice of $p$ we have $(9/\epsilon)^p = n^{1/p}$. Note that $p = O(\sqrt{\log n})$ and $1/p = o(1)$ since $\epsilon$ is a constant. We set

$$r^{(0)} = \sqrt{n} \text{ and } r^{(i)} = (9/\epsilon) \sum_{0 \leq j \leq i-1} r^{(j)} \text{ for } 1 \leq i \leq p-1 \, .$$

Furthermore, we set

$$\beta^{(i)} = 9 \sum_{i \leq j \leq p-2} r^{(j)} \text{ for all } 0 \leq i \leq p-1$$

(in particular $\beta^{(p-1)} = 0$). Note that for each $0 \leq i \leq p-1$, we can bound $r^{(i)}$ by $r^{(i)} \leq (9/\epsilon)^{2i}\sqrt{n}$. We also have $\beta^{(0)}/\epsilon \leq (9/\epsilon)^{2p}\sqrt{n}$ and by our choice of $p$ this means that $\beta^{(0)}/\epsilon \leq n^{2/p}\sqrt{n} \leq n^{1/2+o(1)}$ and $r^{(i)} \leq r^{(p-1)} \leq n^{1/2+o(1)}$.

The first part of the algorithm is to maintain an ES-tree up to depth $\beta^{(0)}/\epsilon$ from $s$, which allows us to determine distances from $s$ up to $\beta^{(0)}/\epsilon$ exactly. This takes time $O(m\beta^{(0)}/\epsilon) = O(mn^{1/2+o(1)})$. We deal with larger distances as follows. Our goal is to dynamically maintain a graph $H$ such that for every node $u$ with $\operatorname{dist}_G(u, s) \geq \beta^{(0)}/\epsilon$ there is a path from $u$ to $s$ in $H$ with at most $p \operatorname{dist}_G(u, s)/\sqrt{n} \leq p\sqrt{n}$ many hops of approximately the same length as $\operatorname{dist}_G(u, s)$. On this graph we run a variant of the ES-tree from $s$, called monotone ES-tree, to maintain the approximate distances for nodes that are at distance more than $\beta^{(0)}/\epsilon$ from $s$. As the monotone ES-tree never underestimates the true distance, we can simply return the minimum of the distance estimates returned by both trees to obtain a $(1+\epsilon)$-approximation for every node.

## 3.1 Hop Set via Thorup-Zwick

The hop set in this simplified version of our algorithm comes from the construction of Thorup and Zwick. In [23], Thorup and Zwick provide, for every $k \geq 1$ a distance oracle for undirected weighted graphs of size $O(kn^{1+1/k})$ and multiplicative stretch $2k-1$ that has a preprocessing time of $O(kmn^{1/k})$ and a query time of $O(k)$. Their construction immediately implies a $(2k-1)$-spanner for weighted graphs with $O(kn^{1+1/k})$ many edges that can be constructed in time $O(kmn^{1/k})$. In [24], they show that in *unweighted* graphs exactly the same construction provides a spanner with both a multiplicative error of $(1+\epsilon)$ and an additive error of $2(1+2/\epsilon)^{k-2}$. We extend their analysis to our new setting to give the desired hop set.

We now review their construction and define the hop set we want to use. We define a hierarchy of sets $A_0 \supseteq A_1 \supseteq \ldots \supseteq A_p$ as follows (where $p$ is as defined above). We let $A_0 = V$ and $A_p = \emptyset$, and for $1 \leq i \leq p-1$ we obtain $A_i$ by picking each node from $A_{i-1}$ with probability $(\ln n/n)^{1/p}$. We say that a node $v$ has *priority* $i$ if $v \in A_i \setminus A_{i+1}$ (for $0 \leq i \leq p-1$). The central notion of Thorup and Zwick is the *bunch* of a node $u$. As usual, we denote by $\operatorname{dist}_G(u, A_i) = \min_{v \in A_i} \operatorname{dist}_G(u, v)$ the distance between the node $u$ and the set of nodes $A_i$ and we set $\operatorname{dist}_G(u, \emptyset) = \infty$. Now, for every node $u$ and every $0 \leq i \leq p-1$, the $i$-bunch of $u$ is defined as

$$\mathit{Bunch}_i(u) = \{v \in A_i \setminus A_{i+1} \mid \operatorname{dist}_G(u, v) < \operatorname{dist}_G(u, A_{i+1})\}$$

and the bunch of $u$ is

$$\mathit{Bunch}(u) = \bigcup_{0 \leq i \leq p-1} \mathit{Bunch}_i(u) \,.$$

Intuitively, a node $v$ of priority $i$ is in the bunch of $u$ if $v$ is closer to $u$ than any node of priority greater than $i$. We will only need the following "truncated" version of the bunches:

$$\mathit{Bunch}^D(u) = \{ v \in \mathit{Bunch}(v) \mid \operatorname{dist}_G(u, v) \leq D \} \,.$$

In our case we set $D = r^{(p-1)} \leq n^{1/2+o(1)}$.

Intuitively, we would now like to use the hop set containing all edges $(u, v)$ such that $v \in \mathit{Bunch}^D(u)$. This would give us a hop set of size $O(n^{1+o(1)})$ providing the desired hop reduction and approximation guarantee. Note that as edges are deleted from $G$, distances in $G$ might increase between certain nodes which results in nodes joining and leaving the bunches. This means that the edges in the hop set we would like to use might undergo insertions, deletions, and edge weight increases. However, the only bound on the number of edges inserted into this hop set we are aware of is $O(n^{1+1/2+o(1)})$ following the analysis of [4]. This is too inefficient as the number of inserted edges shows up in the running time of the monotone ES-tree. In the following we show how to avoid this problem by using a slightly modified definition of the hop set that guarantees that the number of inserted edges is $O(n^{1+o(1)})$.

We define the *rounded $i$-bunch* of a node $u$ as follows:

$$\overline{\mathit{Bunch}}_i(u) = \{ v \in A_i \setminus A_{i+1} \mid \log \operatorname{dist}_G(u, v) < \lfloor \log \operatorname{dist}_G(u, A_{i+1}) \rfloor \} \,.$$

Similarly, we define $\overline{\mathit{Bunch}}(u) = \bigcup_{0 \leq i \leq p-1} \overline{\mathit{Bunch}}_i(u)$ and $\overline{\mathit{Bunch}}^D(u) = \{ v \in \overline{\mathit{Bunch}}(v) \mid \operatorname{dist}_G(u, v) \leq D \}$. We define the edge $(u, v)$ to be contained in the hop set $H$ if and only if $v \in \overline{\mathit{Bunch}}^D(u)$. The weight of such an edge $(u, v) \in H$ is $w(u, v) = \operatorname{dist}_G(u, v)$. We slightly abuse notation and denote by $H$ also the graph that has only the edges of the hop set and the same nodes as $G$.

**Lemma 3.1.** *The number of edges ever contained in $H$ is $O(n^{1+o(1)})$ in expectation.*

*Proof.* At any time, the size of $\mathit{Bunch}_i(u)$ is $n^{1/p}$ in expectation [23] for every node $u$ and every $0 \leq i \leq p - 1$. As $\overline{\mathit{Bunch}}_i(u) \subseteq \mathit{Bunch}_i(u)$, this also bounds the size of $\overline{\mathit{Bunch}}_i(u)$. The initial number of edges of $H$ can thus be bounded by $O(pn^{1+1/p})$.

An edge $(u, v)$ is only inserted into $H$ if $v$ joins $\overline{\mathit{Bunch}}^D(u)$ or, symmetrically, if $u$ joins $\overline{\mathit{Bunch}}^D(v)$. For every node $u$, nodes can only join $\overline{\mathit{Bunch}}^D(u)$ by joining $\overline{\mathit{Bunch}}_i(u)$ for some $0 \leq i \leq p - 2$ when the value $\lfloor \log \operatorname{dist}_G(u, A_{i+1}) \rfloor$ increases. (Note that by the definition of $\overline{\mathit{Bunch}}_{p-1}(u)$ no node will ever join $\overline{\mathit{Bunch}}_{p-1}(u)$.) This happens at most $\log D$ times until the threshold $D$ is exceeded. Every time this happens at most $n^{1/p}$ nodes will be contained in $\overline{\mathit{Bunch}}_i(u)$, which trivially bounds the number of nodes that might join $\overline{\mathit{Bunch}}_i(u)$ by $n^{1/p}$. It follows that the number of edges inserted into $H$ is $O(pn^{1+1/p} \log D)$. This dominates the number of edges initially contained in $H$. Since $p \leq \log n$, $1/p = o(1)$, and $D \leq n$, the number of edges ever contained in $H$ is $O(n^{1+o(1)})$. □

The set of edges $H$ we defined will only be useful if it can be used to provide a good approximation of shortest paths using only a small number of hops. In Section 3.3 we will show that this goal can be achieved with $H$. In our proof the following structural property of $H$ will be essential.

**Lemma 3.2.** *For every node $u$ of priority $i$ and every node $v$ such that $\operatorname{dist}_G(u, v) = r^{(i)}$, $H$ contains either the edge $(u, v)$ or an edge $(u, v')$ to a node $v'$ of priority $j' \geq i + 1$ such that $\operatorname{dist}_G(u, v') \leq 2^{j'-i-1} 3 r^{(i)}$.*

*Proof.* Assume that $(u,v)$ is not contained in $H$ and let $j$ denote the priority of $v$. We first show that there is some node $v' \in A_{i+1}$ such that $\operatorname{dist}_G(u,v') \leq r^{(i)}$.

Case 1: $j \geq i+1$. Since $H$ does not contain the edge $(u,v)$, we know that $v \notin \overline{\mathit{Bunch}}^D(u)$. As $\operatorname{dist}_G(u,v) = r^{(i)} \leq D$ it follows that $v \notin \overline{\mathit{Bunch}}(u)$ and in particular $v \notin \overline{\mathit{Bunch}}_j(u)$. By the definition of $\overline{\mathit{Bunch}}_j(u)$ we have $\lfloor \log \operatorname{dist}_G(u,A_{j+1}) \rfloor \leq \log \operatorname{dist}_G(u,v)$ and thus $\operatorname{dist}_G(u,A_{j+1}) \leq 2\operatorname{dist}_G(u,v)$. This means that there is a node $v' \in A_{j+1} \subseteq A_{i+1}$ such that $\operatorname{dist}_G(u,v') \leq 2\operatorname{dist}_G(u,v) = 2r^{(i)}$.

Case 2: $j \leq i$. Following similar arguments as above, we conclude that $u \notin \overline{\mathit{Bunch}}^D(v)$ which means that there is some node $v' \in A_{i+1}$ such that $\operatorname{dist}_G(v,v') \leq 2r^{(i)}$. By the triangle inequality we have $\operatorname{dist}_G(u,v') \leq \operatorname{dist}_G(u,v) + \operatorname{dist}_G(v,v') \leq 3r^{(i)}$.

Thus, in both cases we have shown that there is some node $v' \in A_{i+1}$ such that $\operatorname{dist}_G(u,v') \leq 3r^{(i)}$. If $v'$ is contained in the bunch of $u$, then $H$ contains the edge $(u,v')$ as desired. Otherwise, we use the argument from above again and get that there is some node $v'' \in A_{i+2}$ such that $\operatorname{dist}_G(u,v'') \leq 2\operatorname{dist}_G(u,v') \leq 6r^{(i)}$. If $v''$ is contained in $\overline{\mathit{Bunch}}^D(u)$, then $H$ contains the edge $(u,v')$ as desired. This argument can be repeated until eventually a node of priority $p-1$ is reached. It follows that we will find a node $v^*$ of priority $j^* \geq i+1$ contained in $\overline{\mathit{Bunch}}^D(u)$ (making the edge $(u,v^*)$ contained in $H$) such that $\operatorname{dist}_G(u,v^*) \leq 2^{j^*-i-1}3r^{(i)}$. □

Finally, we argue about the running time needed for maintaining the hop set $H$. Roditty and Zwick [21] gave an algorithm for maintaining $\mathit{Bunch}^D(u)$ for every node $u$ with a total update time of $\widetilde{O}(mn^{1/p}D)$. This algorithm also maintains the distances of a node to the nodes in its truncated bunch as well as $\operatorname{dist}_G(v,A_i)$ for every node $v$ and every $0 \leq i \leq p-1$, which allows us to maintain $\overline{\mathit{Bunch}}^D(u)$ for every node $u$ in the same running time. Thus, we can maintain our hop set in time $\widetilde{O}(mn^{1/p}D) = O(mn^{1/2+o(1)})$.

### 3.2 Static Hop Reduction

We first explain how the set of edges $H$ based on the Thorup-Zwick construction provides a hop set in the static setting. A hop set based on Thorup-Zwick has implicitly been used already in [2]. Here, we deviate from the analysis in [2] because we do not know how to transfer it to the decremental setting. Instead, our analysis mainly follows the argument for proving the approximation guarantee of the spanner in [24] with the hop reduction being an additional aspect.

Consider a shortest path from some node $u$ to $s$. We show that in $H$ there is a path from $u$ to $s$ of total weight at most $(1+\epsilon)\operatorname{dist}_G(u,s) + \beta^{(0)}$ using at most $p\operatorname{dist}_G(u,s)/\sqrt{n}$ edges. Intuitively, we divide the shortest path into subpaths of length at least $\sqrt{n}$ and try to replace each such subpath by a path that has approximately the same length but uses at most $p$ hops. In the following we explain how to "walk" from $u$ to $s$ using only few edges of $H$, similar to the argument used in [24].

Assume that $u$ has priority 0. We first try to walk to the node $v$ at distance $r^{(0)} = \sqrt{n}$ from $u$ on the shortest path from $u$ to $s$ in $G$. If $H$ contains the edge $(u,v)$ we have replaced the subpath from $u$ to $v$ by a single edge without any approximation error, reducing the distance to $s$ by $r^{(0)}$. (In this case we can continue the argument from $v$). If $H$ does not contain the edge $(u,v)$, then we know by Lemma 3.2 that $H$ contains an edge $(u,u_1)$ where $u_1$ has priority $j \geq 1$ and is at distance at most $3^{j-1}r^{(0)}$ to $u$. To understand the intuition behind the argument we assume here that $u_1$ has priority 1 and is at distance at most $3r^{(0)}$ to $u$. Following this edge, we start a detour on our way to $s$ whose weight we want to compensate. At $u_1$ we try to reduce the distance to $s$ by $r^{(1)} = (9/\epsilon)r^{(0)}$. Let $v_1$ denote the node at distance $r^{(1)}$ to $u_1$ on the shortest path from $u_1$ to $s$ in $G$. If $H$ contains the edge $(u_1,v_1)$, the path consisting of the edges $(u,u_1)$ and $(u_1,v_1)$ has weight at most $3r^{(0)} + r^{(1)}$ and reduces the distance to $s$ by at least $r^{(1)} - 3r^{(0)}$. Note that $3r^{(0)} + r^{(1)} \leq (1+\epsilon)(r^{(1)} - 3r^{(0)})$, thus this detour consisting of two hops gives a multiplicative error of $(1+\epsilon)$. If $H$ does not contain the edge $(u_1, v_1)$, we can again argue that $H$ contains an edge $(u_1, u_2)$ to a node $u_2$ of priority at least 2. We can repeat our arguments until we eventually reach a node $u_{p-1}$ of priority $p-1$. As a node of priority $p-1$ is contained in the bunch of every node at distance at most $r^{(p-1)}$, we can guarantee that at this stage the distance to $s$ can be reduced by $r^{(p-1)}$.

In addition to the multiplicative error of $(1+\epsilon)$, using only the edges of $H$ to reach $s$ also gives us an additional additive error of $\beta^{(0)}$. The reason is that the last subpath we replace on the way to $s$ might not be long enough to compensate the additional weight we have accumulated by using edges of $H$.

### 3.3 Dynamic Hop Reduction

The static analysis we gave above mostly follows the analysis in [24]; the difficulty now comes from the fact that edges might be inserted into $H$ over time. This usually requires a fully dynamic algorithm that can handle both insertions and deletions. Using known fully dynamic algorithms would however be too inefficient and we would like to retain the efficiency of the decremental ES-tree. We solve this problem by using a monotone ES-tree [7, 10] that can handle certain insertions of edges. In contrast to previous applications of the monotone ES-tree, this time we want to bound its running time not in terms of distance from the source, but in terms of the number of hops used in a weighted graph. For purely decremental ES-trees this can be done by rounding the edge weights, a technique that has already been used in the context of dynamic shortest paths algorithm before [2, 15, 3]. In the following we show that the rounding also works for the monotone ES-tree with the hop set $H$.

We round every weight of an edge in $H$ up to a multiple of $\varphi = \epsilon\sqrt{n}/p$. Thus, instead of using the hop set $H$ in which every edge $(u, v)$ has weight $w(u, v)$, we use the hop set $H'$ containing the same edges as $H$ in which every edge $(u, v)$ has weight $w'(u, v) = \lceil w(u, v)/\varphi \rceil \cdot \varphi$. This gives an additive error of $\varphi$ for every edge we use on the path from $s$ to $t$. We would like to argue that due to the hop reduction there are at most $p \operatorname{dist}_G(u, s)/\sqrt{n} \leq p\sqrt{n}$ such edges and thus the error can be converted into a multiplicative error of $\epsilon$. Again, this would be simple to argue in the static setting, but we have to make sure that this is also the case for the monotone ES-tree.

**Running Time** The rounding makes maintaining the monotone ES-tree more efficient in the following sense. Maintaining a monotone ES-tree [7] on $H$ up to distance $R$ with a multiplicative approximation of $\alpha$ and an additive approximation of $\beta$ would take time $O(\mathcal{E}(H)(\alpha R + \beta) + \mathcal{W}(H))$, where $\mathcal{E}(H)$ is the number of edges ever contained in $H$ and $\mathcal{W}(H)$ is the number of weight increases of edges in $H$. As the maximum distance of a node from $s$ in $G$ is $n$, we would have to set $R = n$. However, if we run the algorithm on $H'$ we can maintain the monotone ES-tree up to a smaller depth. As all edge weights in $H'$ are multiples of $\varphi$, scaling them down by dividing by $\varphi$ yields integer weights again. On this scaled-down version of the graph, we get a more efficient running time of $O(\mathcal{E}(H)(\alpha n + \beta)/\varphi + \mathcal{W}(H)) = O(p\mathcal{E}(H)(\alpha n + \beta)/(\epsilon\sqrt{n}) + \mathcal{W}(H))$. Remember that $\mathcal{E}(H) = O(n^{1+o(1)})$ by Lemma 3.1. Furthermore we have the bound $\mathcal{W}(H) \leq \mathcal{E}(H) r^{(p-1)}$ as $r^{(p-1)}$ is the maximum edge weight we use. Since $r^{(p-1)} \leq n^{1/2+o(1)}$ we get $\mathcal{W}(H) \leq O(n^{1.5+o(1)})$. Using $\alpha = (1+\epsilon)$ and $\beta = \beta^{(0)} \leq n^{1/2+o(1)}$, the approximation guarantee we will obtain below, we get total update time of $O(n^{1.5+o(1)})$ for the monotone ES-tree.

**Approximation Guarantee** We now argue about the approximation guarantee of the monotone ES-tree using only the edges of $H'$. For every node $u$, let $\ell(u; s)$ denote the level of a node $u$ in the monotone ES-tree with root $s$.

Without the rounding a similar analysis to the one we gave in [10] would show that $\ell(u; s) \leq$

$(1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(0)} \leq (1+\epsilon)\operatorname{dist}_G(u,s)+n^{1/2+o(1)}$. We will show that in the rounded graph we have $\ell(u;s) \leq (1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(0)}+\varphi p \operatorname{dist}_G(u,s)/\sqrt{n}$, where $p\operatorname{dist}_G(u,s)/\sqrt{n}$ bounds the number of edges from $H'$ we use, each adding an error of $\varphi$. To prove this we will use an inductive argument that needs an explicit upper bound on the number of hops on the path to $s$ in $H'$ for every node $u$. Intuitively, the number of hops used on the path from a node $u$ to $s$ should depend on the distance from $u$ to $s$ and on the priority of $u$. If, for example, the distance from $u$ to $s$ is $\sqrt{n}$ and $u$ has priority $i$, the analysis from Section 3.2 suggests that the number of edges needed to go from $u$ to $s$ in $H'$ is $p-i$. It turns out that the following estimate of the hop count works:

$$h(u,i)=\begin{cases}0 & \text{if } u=s\\ p\cdot\left\lceil\frac{\max(\operatorname{dist}_G(u,s)-r^{(i)},0)}{\sqrt{n}}\right\rceil+p-i & \text{otherwise}\end{cases}.$$

**Lemma 3.3.** *For every node $u$ of priority $i$ we have the following bound on the level of $u$ in the monotone ES-tree on $H'$ with root $s$:*

$$\ell(u;s) \leq (1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(i)}+h(u,i)\cdot\varphi\,.$$

*Proof.* The first case is that $\ell(u;s) \leq \ell(v;s)+w'(u,v)$ for every neighbor $v$ of $u$, where $w'(u,v)$ is the weight of the edge $(u,v')$ after the rounding. If this is not the case, i.e., if there is a neighbor $v$ of $u$ such that $\ell(u;s) > \ell(v;s)+w'(u,v)$, then we know by properties of the monotone ES-tree [7] that the edge $(u,v)$ has been inserted at some previous point in time and since then the level of $u$ has not changed. As the desired inequality was fulfilled for $u$ at that time, it is still fulfilled right now. The more involved case is when $\ell(u;s) \leq \ell(v;s)+w'(u,v)$ for every neighbor $v$ of $u$, which we analyze in the following.

Let $v$ be the node on the shortest path from $u$ to $s$ in $G$ that is at distance $r^{(i)}$ to $u$. We separately analyze the two cases $(u,v)\in H$ and $(u,v)\notin H$. In the first case, we further distinguish whether $\operatorname{dist}_G(u,s) \leq r^{(i)}$ or $\operatorname{dist}_G(u,s) > r^{(i)}$. If $\operatorname{dist}_G(u,s) \leq r^{(i)}$, then $v=s$ and it is straightforward to show that $\ell(u;s) \leq \operatorname{dist}_G(u,s)+\varphi$, which immediately implies $\ell(u;s) \leq (1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(i)}+h(u,i)\varphi$. If $\ell(u;s) \leq \operatorname{dist}_G(u,s)+\varphi$ we apply the induction hypothesis on $v$, which gives $\ell(v;s) \leq (1+\epsilon)\operatorname{dist}_G(v,s)+\beta^{(j)}+h(v,j)\varphi$. Furthermore we know that $\ell(u;s) \leq \ell(v;s)+w'(u,v)$ since $v$ is a neighbor of $u$ using the edges of the hop set. Remember also that $w'(u,v) \leq w(u,v)+\varphi$ due to the rounding. Following similar arguments as in [10], we get

$$\begin{aligned}\ell(u;s) &\leq \ell(v;s)+w'(u,v)\\ &\leq \ell(v;s)+w(u,v)+\varphi\\ &= \ell(v;s)+\operatorname{dist}_G(u,v)+\varphi\\ &\leq (1+\epsilon)\operatorname{dist}_G(v,s)+\beta^{(j)}+h(v,j)\varphi+\operatorname{dist}_G(u,v)+\varphi\\ &\leq (1+\epsilon)\operatorname{dist}_G(v,s)+\beta^{(0)}+\operatorname{dist}_G(u,v)+(h(v,j)+1)\varphi\\ &= (1+\epsilon)\operatorname{dist}_G(v,s)+\beta^{(i)}+\epsilon r^{(i)}+\operatorname{dist}_G(u,v)+(h(v,j)+1)\varphi\\ &= (1+\epsilon)\operatorname{dist}_G(v,s)+\beta^{(i)}+\epsilon\operatorname{dist}_G(u,v)+\operatorname{dist}_G(u,v)+(h(v,j)+1)\varphi\\ &= (1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(i)}+(h(v,j)+1)\varphi\,.\end{aligned}$$

Here we have used the fact that $\beta^{(0)}=\beta^{(i)}+\epsilon r^{(i)}$. We omit the straightforward verification of the inequality $h(v,j)+1 \leq h(u,i)$ and conclude that $\ell(u;s) \leq (1+\epsilon)\operatorname{dist}_G(u,s)+\beta^{(i)}+h(u,i)\varphi$ in this case.

Consider now the case $(u,v)\notin H$. By Lemma 3.2 we know that there is an edge $(u,v')$ to a node $v'$ of priority $j' \geq i+1$ such that $\operatorname{dist}_G(u,v') \leq 2^{j'-i-1}3r^{(i)}$. Let us first argue about the case

$j' = i + 1$ where $\operatorname{dist}_G(u, v') \leq 3r^{(i)}$.

$$\begin{aligned}
\ell(u; s) &\leq \ell(v'; s) + w'(u, v') \\
&\leq \ell(v; s) + w(u, v') + \varphi \\
&= \ell(v; s) + \operatorname{dist}_G(u, v') + \varphi \\
&\leq \ell(v; s) + 3r^{(i)} + \varphi \\
&\leq (1+\epsilon) \operatorname{dist}_G(v', s) + \beta^{(j')} + h(v', j')\varphi + 3r^{(i)} + \varphi \\
&\leq (1+\epsilon) \operatorname{dist}_G(u, s) + (1+\epsilon) \operatorname{dist}_G(u, v') + \beta^{(j')} + 3r^{(i)} + (h(v', j') + 1)\varphi \\
&\leq (1+\epsilon) \operatorname{dist}_G(u, s) + 6r^{(i)} + 3r^{(i)} + \beta^{(j')} + (h(v', j') + 1)\varphi \\
&= (1+\epsilon) \operatorname{dist}_G(u, s) + 9r^{(i)} + \beta^{(j')} + (h(v', j') + 1)\varphi \\
&= (1+\epsilon) \operatorname{dist}_G(u, s) + 9r^{(i)} + \beta^{(i+1)} + (h(v', i+1) + 1)\varphi \\
&= (1+\epsilon) \operatorname{dist}_G(u, s) + \beta^{(i)} + (h(v', i+1) + 1)\varphi
\end{aligned}$$

Here we have used the fact that $\beta^{(i)} = 9r^{(i)} + \beta^{(i+1)}$. Again we omit the straightforward verification of the inequality $h(v', i+1) + 1 \leq h(u, i)$.[2] In general, if $v'$ has priority $j' \geq i+1$ and we know that $\operatorname{dist}_G(u, v') \leq 2^{j'-i-1} 3r^{(i)}$, the same argument requires us to verify the inequality $2^{j'-i-1} 9r^{(i)} + \beta^{(j')} \leq \beta^{(i)}$. This is equivalent to $2^{j'-i-1} r^{(i)} \leq \sum_{i \leq i' \leq j-1} r^{(i')}$ and holds by the exponential growth of the $r^{(i)}$'s. Thus, we have proved the inequality $\ell(u; s) \leq (1+\epsilon) \operatorname{dist}_G(u, s) + \beta^{(i)} + h(u, i)\varphi$. □

**Obtaining $(1+\epsilon)$-approximation** Finally, we explain how to obtain the $(1+\epsilon)$-approximation. In Lemma 3.3 we showed that the monotone ES-tree with root $s$ provides, for every node $u$, an approximation guarantee of $\ell(u; s) \leq (1+\epsilon) \operatorname{dist}_G(u, s) + \beta^{(i)} + h(u, i)\varphi$, where $i$ is the priority of $u$. Note that $\beta^{(i)} \leq \beta^{(0)}$ and $h(u, i) \leq p \operatorname{dist}_G(u, s)/\sqrt{n}$. Thus, $h(u, i)\varphi \leq \epsilon \operatorname{dist}_G(u, s)$ since we have defined $\varphi = \epsilon\sqrt{n}/p$. Furthermore, we have assumed that $\operatorname{dist}_G(u, s) \geq \beta^{(0)}/\epsilon$, which is equivalent to $\beta^{(0)} \leq \epsilon \operatorname{dist}_G(u, s)$. Thus, in total we get $\ell(u; s) \leq (1+\epsilon) \operatorname{dist}_G(u, s) + \beta^{(0)} + h(u, i)\varphi \leq (1+3\epsilon) \operatorname{dist}_G(u, s)$. We now run the whole algorithm with $\epsilon' = \epsilon/3$ to get the desired $(1+\epsilon)$-approximation. Since we have assumed that $\epsilon$ is a constant, this does not affect the bound of $O(mn^{1/2+o(1)})$ on the total update time that we have argued for all parts of our algorithm.

# 4 Preliminaries

## 4.1 Notation and Basic Lemmas

We are given an undirected graph $G$ 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 $G$ before the first edge deletion. We set $n = |V|$ and $m = |E|$.

For every graph $H$, we denote the weight of an edge $(u, v)$ in $H$ by $w_H(u, v)$. The *distance* $\operatorname{dist}_H(u, v)$ between a node $x$ and a node $y$ is the weight of the shortest path, i.e., the minimum-weight path, between $u$ and $v$ in $H$. If there is no path between $x$ and $y$ in $H$, we set $\operatorname{dist}_H(x, y) = \infty$. When we refer to the *current* version of $G$ (after the last update), we usually omit the index $G$. We say that a distance estimate $\delta(u, v)$ is an $(\alpha, \beta)$-approximation of the true distance $\operatorname{dist}(u, v)$ if $\operatorname{dist}(u, v) \leq \delta(u, v) \leq \alpha \operatorname{dist}(u, v) + \beta$, i.e., $\delta(u, v)$ never underestimates the true distance and

[2] Technical note: To verify $h(v', i+1) + 1 \leq h(u, i)$ we need the inequality $2r^{(i)} \leq r^{(i+1)}$, which holds by our choice of the $r^{(i)}$'s.

overestimates it with a multiplicative error of at most $\alpha$ and an additive error of at most $\beta$. If there is no additive error, we simply say $\alpha$-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 edge weight increases 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 edge weight increases in $F$. For every set of nodes $U$, we denote by $H|U$ the subgraph of $H$ induced by the nodes in $U$, i.e., $H|U$ contains all edges $(u, v)$ such that $(u, v)$ is contained in $H$ and $u$ and $v$ are both contained in $U$. Similarly, for every set of edges $F$ and every set of nodes $U$ we denote by $F|U$ the subset of $F$ induced by $U$.

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

**Lemma 4.1** ([6, 11, 13])**.** *There is a data structure called ES-tree that, given a weighted directed graph 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 $\delta(v, s) = \operatorname{dist}(v, s)$ if $\operatorname{dist}(v, s) \leq D$ and $\delta(v, s) = \infty$ if $\operatorname{dist}(v, s) > D$. It has constant query time and a total update time of $O(mD)$.*

Recent approaches for solving approximate decremental SSSP and APSP use special graphs called *emulators.* An $(\alpha, \beta)$-emulator $H$ of a graph $G$ is a graph containing the nodes of $G$ such that $\operatorname{dist}_G(u, v) \leq \operatorname{dist}_H(u, v) \leq \alpha \operatorname{dist}_G(u, v) + \beta$ for all nodes $u$ and $v$.[3] Maintaining exact distances on $H$ provides an $(\alpha, \beta)$-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 maintained while $G$ undergoes updates. For unweighted, undirected graphs undergoing edge deletions, the emulator of Thorup and Zwick, which provides a relatively good approximation, can be maintained quite efficiently. 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 [4] 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 [7] 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 $(\alpha, \beta)$-approximation on *any* $(\alpha, \beta)$-approximate emulator as we need to exploit the structure of the emulator. In [10] we gave an analysis of the monotone ES-tree when run on the Thorup-Zwick emulator and in the current paper we extend this analysis for our new algorithms. The running time of the monotone ES-tree as analyzed in [7] is as follows.

**Lemma 4.2.** *For every $D \geq 1$, the total update time of a monotone ES-tree up to depth $(\alpha D + \beta)$ on a graph $H$ undergoing edge deletions, edge insertions, and edge weight increases is $O(\mathcal{E}(H)(\alpha D + \beta) + \mathcal{W}(H))$.*

Our algorithms will heavily use randomization. It is well-known, and exploited by many other algorithms for dynamic (approximate) shortest paths and reachability, that by sampling a set of nodes with a sufficiently large probability we can guarantee that certain sets of nodes contain at

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

least one of the sampled nodes whp. To the best of our knowledge, the first use of this technique in graph algorithms goes back to Ullman and Yannakakis [25].

**Lemma 4.3.** *Let $T$ be a set of size $t$ and let $S_1, S_2, \dots, S_l$ be subsets of $T$ of size at least $s$. Let $U$ be a subset of $T$ that was obtained by choosing each element of $T$ independently with probability $p = (a \ln lt)/s$, where $a$ is a constant. Then, for every $1 \leq i \leq l$, the set $S_i$ contains a node of $U$ with high probability (whp), i.e. probability at least $1 - 1/t^a$, and the size of $U$ is $O((t \log{(lt)})/s)$ in expectation.*

## 4.2 Definition of Variables Used by Our Algorithm

In the following we define how to set the values of the variables used by our algorithm in Section 5. The right choice of these values is crucial for our approach to work. The algorithm has two parameters. The first one is the approximation parameter $\epsilon'$ which is restricted to $0 < \epsilon' \leq 1$. The second one is the multiplier $\gamma$ which is restricted to $\gamma \geq 1$. Given $\epsilon'$, the goal of our algorithm is to provide a $(1 + \epsilon')$-approximation for all distances up to $n\gamma$. In unweighted graphs we can set $\gamma = 1$ and in weighted graphs we will use a reduction that sets $\gamma = O(1/\epsilon')$.

The first value we define for our algorithm is

$$\epsilon = \frac{\epsilon'}{1 + 4\sqrt{\log n}} \,.$$

We will use $\epsilon$ as our "basic unit of multiplicative error". In particular, the multiplicative error will sum up to at most $1 + (1 + 4\sqrt{\log n})\epsilon$ which results in $1 + \epsilon'$.

Our algorithm uses $q - 2$ layers where

$$q = \frac{\sqrt{\log n}}{2\sqrt{\log{(\frac{4 \cdot 44}{\epsilon})}}}$$

which is $O(\sqrt{\log n})$. In the $k$-th layer (where $0 \leq k \leq q - 2$), the algorithm will provide $(\alpha_k, \beta_k)$-approximate single-source shortest paths for the distance range from $\Delta_k$ to $\Delta_{k+2}$. We set

$$\alpha_k = 1 + 4k\epsilon$$

and explain how to set $\beta_k$ below, but in particular we set set $\alpha_0 = 1$ and $\beta_0 = 0$. Thus, for $k = 0$ we provide exact distances. Note that $\alpha_k$ increases with $k$ (and $\beta_k$ will also show this behavior). The distance range is defined by

$$\Delta_k = n^{k/q}\gamma$$

Note $\Delta_k$ increases with $k$ and $\Delta_q = n\gamma$.

To achieve the desired approximation guarantee in every layer, our algorithm constructs and maintains in the $k$-th layer (where $1 \leq k \leq q - 2$) a certain set of edges $E_k$. For this purpose it will assign to every node an integer from 0 to $p - 1$ called the priority of the node. We set

$$p = \frac{\sqrt{\log n}}{\sqrt{\log{(\frac{4 \cdot 44}{\epsilon})}}} \,.$$

Note that this definition guarantees that $p = 2q$ and $p = O(\sqrt{\log n})$. We now define a few more values that are needed in the algorithm for constructing and maintaining $E_k$. We set

$$r_k^{(0)} = \Delta_k \,.$$

To define $r_k^{(i)}$ for every for every $1 \leq i \leq p-1$, we also need two auxiliary variables $s_k^{(i,i)}$ and $w_k^{(i,i)}$ for every $0 \leq i \leq p-1$. For every $1 \leq i \leq p-1$, we set

$$r_k^{(i)} = (4/\epsilon) \sum_{0 \leq i' \leq i-1} w_k^{(i',i'+1)} + \beta_{k-1}/\epsilon$$

where $\beta_{k-1}$ is the additive term in the approximation provided by the $(k-1)$-th layer, a value that we will define below. For every $0 \leq i \leq p-1$ we set

$$s_k^{(i,i+1)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1}$$

and

$$w_k^{(i,i+1)} = \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,i+1)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \,.$$

Note that these three definitions are not circular since for defining $r_k^{(i)}$ for the $k$-th layer we need $\beta_{k-1}$ from the *previous* layer and as $r_k^{(0)} = \Delta_k$ we can calculate all values in the order $r_k^{(0)}$, $s_k^{(0,1)}$, $w_k^{(0,1)}$, $r_k^{(1)}$, and so on. Furthermore, $r_k^{(i)}$ increases as the priority $i$ increases, i.e., $r_k^{(i)} \leq r_k^{(i+1)}$ for all $0 \leq i \leq p-2$ and thus $r_k^{(i)} \geq \Delta_k$ for all $0 \leq i \leq p-1$. We will show that, for every $0 \leq i \leq p-1$, $r_k^{(i)} \leq (c/\epsilon)^{2i} \Delta_k$ for some constant $c$ which will imply $r_k^{(p-1)} \leq n^{1/q} \Delta_k = n^{(k+1)/q} \gamma$. Intuitively, $r_k^{(i)}$ (for $1 \leq i \leq p-1$) tells us for a node $v$ of priority $i$ up to which weight the algorithm will try to replace paths starting from $v$ by a single edge in $E_k$. We will show that the choice $r_k^{(0)} = \Delta_k$ results in a "hop reduction" that replaces paths of weight $\Delta_k$ by a paths with $O(p)$ edges and still guarantees an $(\alpha_k, \beta_k)$-approximation.

We also set, for every $0 \leq i \leq p-1$

$$s_k^{(i,i)} = 0$$

and, for all $0 \leq i < j < p-1$,

$$s_k^{(i,j)} = \sum_{i \leq i' < j} s_k^{(i',i'+1)} \,.$$

Remember that we have already defined $s_k^{(i,i+1)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1}$ above. We will use $s_k^{(i,j)}$ as an estimate of the true distance of a node $u$ of priority $i$ to the node $v$ of priority $j > i$ "dominating" $u$. Furthermore, we set, for every $0 \leq i \leq p-1$,

$$w_k^{(i)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1}$$

and, for all $0 \leq i < j \leq p-1$

$$w_k^{(i,j)} = \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}$$

Note that the definition of $w_k^{(i,i+1)}$ we gave above exactly matches the definition here. These two values bound the weights of edges in $E_k$. We will show that $w_k^{(i,j)} \leq r_k^j$.

Finally, we set

$$\beta_k^{(p-1)} = \beta_{k-1}$$

and, for every $0 \leq i \leq p-2$,

$$\beta_k^{(i)} = \beta_k^{(i+1)} + 4 w_k^{(i,i+1)} = 4 \sum_{i \leq i' \leq p-2} w_k^{(i',i'+1)} \,.$$

The value $\beta_k^{(i)}$ tells us the magnitude of the additive error of a node of priority $i$ in the $k$-th layer. Note that $\beta_k^{(i)}$ increases as the priority $i$ decreases, i.e., $\beta_k^{(i)} \geq \beta_k^{(i+1)}$ for all $0 \leq i \leq p-2$ and thus $\beta_k^{(i)} \geq \beta_k^{(0)}$ for all $0 \leq i \leq p-1$. We set $\beta_k$, the additive approximation guarantee in layer $k$ to

$$\beta_k = \beta_k^{(0)}\,.$$

Thus, $\beta_k$ is set to the worst value $\beta_k^{(i)}$, namely the one for $i = 0$. We will show that $\beta_k \leq (c/\epsilon)^{2p-1}\Delta_k$ for some constant $c$. Observe that

$$r_k^{(i)} = (\beta_k^{(0)} + \beta_{k-1} - \beta_k^{(i)})/\epsilon\,.$$

### 4.3 Bounds on Variables Used by Our Algorithm

In the following we state several equations and inequalities that we will later on, in Section 5, need prove the correctness of our main algorithm and bound its running time.

**Lemma 4.4.** *For all* $0 \leq k \leq q-2$, $\alpha_k \leq 1 + \epsilon' \leq 2$.

*Proof.*

$$\alpha_k = 1 + 4k\epsilon \leq 1 + 4q\epsilon \leq 1 + \sqrt{\log n}\epsilon = 1 + \frac{\sqrt{\log n}\epsilon'}{1 + 4\sqrt{\log n}} \leq 1 + \epsilon' \leq 2$$

□

**Lemma 4.5.** $q \leq p \leq \sqrt{\log n}$

*Proof.* Since $\epsilon' \leq 1$, we have

$$\frac{4 \cdot 44}{\epsilon} \geq 2\,.$$

It follows that

$$\log\left(\frac{4 \cdot 44}{\epsilon}\right) \geq 1\,.$$

and

$$\sqrt{\log\left(\frac{4 \cdot 44}{\epsilon}\right)} \geq 1\,.$$

By multiplying both sides with $\log n$ we get

$$q = \frac{p}{2} \leq p = \frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4\cdot 44}{\epsilon}\right)}} \leq \sqrt{\log n}\,.$$

□

**Lemma 4.6.** $(44(4/\epsilon))^p = n^{1/p}$

*Proof.* We only need to simplify both expressions as follows:

$$n^{1/p} = 2^{1/p \cdot \log n} = 2^{\frac{\sqrt{\log\left(\frac{4\cdot 44}{\epsilon}\right)}}{\sqrt{\log n}} \cdot \log n} = 2^{\sqrt{\log\left(\frac{4\cdot 44}{\epsilon}\right)} \cdot \sqrt{\log n}}$$

$$(44(4/\epsilon))^p \leq \left(\frac{4 \cdot 44}{\epsilon}\right)^p = 2^{p \cdot \log\left(\frac{4\cdot 44}{\epsilon}\right)} = 2^{\frac{\sqrt{\log n}}{\sqrt{\log\left(\frac{4\cdot 44}{\epsilon}\right)}} \cdot \log\left(\frac{4\cdot 44}{\epsilon}\right)} = 2^{\sqrt{\log n} \cdot \sqrt{\log\left(\frac{4\cdot 44}{\epsilon}\right)}}$$

□

**Lemma 4.7.** *For every $1 \le k \le q-2$ and all $0 \le i < j \le p-1$, $w_k^{(i,j)} \le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)}$.*

*Proof.* We simply plug in the definition of $w_k^{(i,j)}$:

$$\begin{aligned}
w_k^{(i,j)} &= \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \\
&= \alpha_{k-1}\left(\alpha_{k-1}\left(r_k^{(i)} + \alpha_{k-1}\left(r_k^{(i)} + \sum_{i \le i' < j} s_k^{(i',i'+1)}\right) + \beta_{k-1}\right) + \beta_{k-1}\right) + \beta_{k-1} \\
&\le \sum_{i \le i' < j} (\alpha_{k-1}(\alpha_{k-1}(r_k^{(i')} + \alpha_{k-1}(r_k^{(i')} + s_k^{(i',i'+1)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1})
\end{aligned}$$

□

**Lemma 4.8.** *For every $1 \le k \le q-2$ and all $0 \le i < j \le p-1$, $w_k^{(i,j)} + r_k^{(i)} \le r_k^{(j)}$.*

By Lemma 4.7 we have $w_k^{(i,j)} \le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)}$. We therefore get the following chain of inequalities:

$$\begin{aligned}
w_k^{(i,j)} + r_k^{(i)} &\le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} + r_k^{(i)} \\
&\le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} + w_k^{(i,i+1)} \\
&\le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} + \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} \\
&= 2 \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} \\
&\le (4/\epsilon) \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} + \beta_{k-1}/\epsilon = r_k^{(j)} .
\end{aligned}$$

**Lemma 4.9.** *For every $1 \le k \le q-2$ and all $0 \le i < j \le p-1$, $\beta_k^{(j)} + 4w_k^{(i,j)} \le \beta_k^{(i)}$.*

*Proof.* By Lemma 4.7 we have $w_k^{(i,j)} \le \sum_{i \le i' \le j-1} w_k^{(i',i'+1)}$. Together with the definitions of $\beta_k^{(i)}$ and $\beta_k^{(j)}$ we get

$$\begin{aligned}
\beta_k^{(j)} + 4w_k^{(i,j)} &= 4 \sum_{j \le i' \le p-2} w_k^{(i',i'+1)} + 4w_k^{(i,j)} \\
&\le 4 \sum_{j \le i' \le p-2} w_k^{(i',i'+1)} + 4 \sum_{i \le i' \le j-1} w_k^{(i',i'+1)} \\
&= 4 \sum_{i \le i' \le p-2} w_k^{(i',i'+1)} = \beta_k^{(i)} .
\end{aligned}$$

□

**Lemma 4.10.** *For every $1 \le k \le q-2$ and every $0 \le i \le p-1$,*

$$s_k^{(i,i+1)} \le 2r_k^{(i)} + \beta_{k-1}$$

*and*

$$w_k^{(i,i+1)} \le 28r_k^{(i)} + 15\beta_{k-1} .$$

*Proof.* Remember that $\alpha_{k-1} \leq 2$. Therefore we get

$$s_k^{(i,i+1)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1} \leq 2r_k^{(i)} + \beta_{k-1}$$

and

$$\begin{aligned} w_k^{(i,i+1)} &= \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,i+1)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \\ &\leq 2(2(r_k^{(i)} + 2(r_k^{(i)} + s_k^{(i,i+1)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \\ &\leq 2(2(r_k^{(i)} + 2(r_k^{(i)} + 2r_k^{(i)} + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \\ &= 2(2(7r_k^{(i)} + 3\beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \\ &= 28r_k^{(i)} + 15\beta_{k-1}\,. \end{aligned}$$

□

**Lemma 4.11.** *For every $1 \leq k \leq q-2$, the following inequalities hold:*

- $r_k^{(i)} \leq (44(4/\epsilon))^{2i}\Delta_k$,
- $\beta_k \leq (44(4/\epsilon))^{2p-1}\Delta_k$,
- $\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1} \leq \Delta_{k+1}$, *and*
- $\beta_k/\epsilon \leq \Delta_{k+1}$.

*Proof.* The proof is by induction on $k$. We first prove the inequality $r_k^{(i)} \leq (44(4/\epsilon))^{2i}\Delta_k$ for every $0 \leq i \leq p-1$. If $k = 1$, then trivially $r_k^{(i)} \geq \beta_{k-1}/\epsilon$ because $\beta_{k-1} = 0$. If $k > 1$, then by the induction hypothesis we have $\Delta_k \geq \beta_{k-1}/\epsilon$ and since $r_k^{(i)} \geq \Delta_k$ we get $r_k^{(i)} \geq \beta_{k-1}/\epsilon$. Therefore we get

$$\begin{aligned} r_k^{(i)} &= (4/\epsilon) \sum_{0 \leq i' \leq i-1} w_k^{(i',i'+1)} + \beta_{k-1}/\epsilon \\ &\leq (4/\epsilon) \sum_{0 \leq i' \leq i-1} (28r_k^{(i)} + 15\beta_{k-1}) + \beta_{k-1}/\epsilon \\ &\leq (4/\epsilon) \sum_{0 \leq i' \leq i-1} (28r_k^{(i)} + 15\beta_{k-1}) + (4/\epsilon)\beta_{k-1} \\ &\leq (4/\epsilon) \sum_{0 \leq i' \leq i-1} (28r_k^{(i)} + 16\beta_{k-1}) \\ &\leq (4/\epsilon) \sum_{0 \leq i' \leq i-1} 44r_k^{(i)} \\ &= (44(4/\epsilon)) \sum_{0 \leq i' \leq i-1} r_k^{(i)} \\ &\leq (44(4/\epsilon)) \sum_{0 \leq i' \leq i-1} (44 + 152/\epsilon)^{2i'}\Delta_k \\ &\leq (44(4/\epsilon))(44(4/\epsilon))^{2(i-1)+1}\Delta_k \\ &= (44(4/\epsilon))^{2i}\Delta_k\,. \end{aligned}$$

Now we prove the bound on $\beta_k$:

$$\begin{aligned}\beta_k = \beta_k^{(0)} = 4 \sum_{0 \leq i \leq p-2} r_k^{(i)} &\leq 4 \sum_{0 \leq i \leq p-2} (44(4/\epsilon))^{2i} \Delta_k \\ &\leq 4(44(4/\epsilon))^{2(p-1)} \Delta_k \\ &\leq (44(4/\epsilon))^{2p-1} \Delta_k \,.\end{aligned}$$

Finally we prove the bounds on $\beta_k/\epsilon$ and $\alpha_{k-1} r_k^{(p-1)} + \beta_k$. As shown above we have $\beta_k = (44(4/\epsilon))^{2p-1}\Delta_k$ and by Lemma 4.6 we have $(44(4/\epsilon))^p \leq n^{1/p}$. Therefore we get

$$\begin{aligned}\beta_k/\epsilon &\leq \frac{1}{\epsilon}(44(4/\epsilon))^{2p-1}\Delta_k \\ &\leq (44(4/\epsilon))^{2p}\Delta_k \\ &= \left((44(4/\epsilon))^p\right)^2 \Delta_k \\ &\leq \left(n^{1/p}\right)^2 \Delta_k \\ &= n^{2/p}\Delta_k \\ &= n^{1/q} n^{k/q} \gamma \\ &= n^{(k+1)/q}\gamma = \Delta_{k+1} \,.\end{aligned}$$

Similarly we get

$$\begin{aligned}\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1} &\leq 2 r_k^{(p-1)} + \Delta_k \\ &\leq 2(44(4/\epsilon))^{2(p-1)}\Delta_k + \Delta_k \\ &\leq (44(4/\epsilon))^{2p}\Delta_k \\ &\leq n^{2/p}\Delta_k \\ &= n^{1/q}\Delta_k \\ &= \Delta_{k+1}\end{aligned}$$

□

# 5 Maintaining the Hop Set

We now explain our main algorithm which uses $q-2$ layers and guarantees that in the $k$-th layer we can provide $(\alpha_k, \beta_k)$-approximate single-source shortest paths for distances in the range from $\Delta_k$ to $\Delta_{k+2}$. In the range from $\Delta_{k+1}$ to $\Delta_{k+2}$ we can even provide $(\alpha_k, 0)$-approximate single-source shortest paths, i.e., without any additive error. This approximation guarantee can be achieved by running a monotone ES-tree on the graph $G_k$ which contains all edges of $G$ and all edges of a hop set $E_k$ constructed by our algorithm. We need a hop set $E_k$ such that for every path $P$ in $G$ of weight at most $\Delta_{k+2}$, there is a path $P'$ in $G_k$ that provides an $(\alpha_k, \beta_k)$-approximation of $P$ and has a relatively small number of edges ("hops"). This property will be implied by a certain property of the edges contained in $E_k$ and allows us to show that the monotone ES-tree provides an $(\alpha_k, \beta_k)$-approximation of all distances in the range from $\Delta_k$ to $\Delta_{k+2}$ when we restrict it to a relatively small depth after some rounding. Since the depth parameter shows up linearly in the running time of the monotone ES-tree, a small depth is necessary to obtain an almost linear running time.

An important part of our algorithm are the procedures for constructing and maintaining the hop set $E_k$. In particular, the edges of $E_k$ can be maintained by using several $(\alpha_{k-1}, \beta_{k-1})$-approximate SSSP data structures from the *previous* layer. As we may only spend almost linear time for maintaining $E_k$, we need a careful analysis of the running time of the monotone ES-tree on $G_k$ as well as the number of edges ever contained in $E_{k-1}$, which in turn influences the running time of the monotone ES-tree on $E_k$.

Thus, our algorithm consists of two parts as stated in the following two lemmas.

**Lemma 5.1.** *For every $1 \leq k \leq q-2$, there is an algorithm for maintaining a set of weighted edges $E_k$ such that:*

1. *Every node $v \in V$ is assigned a number from $0$ to $p-1$, called the* priority *of $v$.*
2. *For every node $u$ of priority $i$, $E_k$ contains either*
    - *for every node $v$ such that $\operatorname{dist}(u,v) \leq r_k^{(i)}$ an edge $(u,v)$ of weight $w_k(u,v) \leq w_k^{(i)}$ with $\operatorname{dist}(u,v) \leq w_k(u,v) \leq \alpha_{k-1} \operatorname{dist}(u,v) + \beta_{k-1}$ or*
    - *an edge $(u,v')$ to a node $v'$ of priority $j' > i$ of weight $w_k(u,v') \leq w_k^{(i,j')}$ with $\operatorname{dist}(u,v') \leq w_k(u,v') \leq \alpha_{k-1} \operatorname{dist}(u,v') + \beta_{k-1}$.*

    *For every edge $(u,v)$ contained in $E_k$ we have $\operatorname{dist}(u,v) \leq w_k(u,v) \leq \alpha_{k-1} \operatorname{dist}(u,v) + \beta_{k-1}$.*
3. *For every subset $E_k|U$ of $E_k$ induced by a set of nodes $U$, the number of edges ever contained in $E_k|U$ is $\mathcal{E}(E_k|U) = \widetilde{O}(|U|pm^{1/p})$ in expectation.*
4. *The total time needed for maintaining $E_k$ is $\widetilde{O}(kp^3 m^{1+2/p} n^{2/q} \gamma/\epsilon)$ in expectation.*

**Lemma 5.2.** *Let $0 \leq k \leq q-2$, let $D \leq \Delta_{k+2}$, let $s$ be a source node and let $U$ and $F$ be the following sets of nodes and edges, respectively:*

$$U = \{v \in V \mid \operatorname{dist}(v,s) \leq D\}$$
$$F = \{(u,v) \in E \mid \operatorname{dist}(u,s) \leq D \text{ or } \operatorname{dist}(v,s) \leq D\}.$$

*Given an algorithm for maintaining $E_k$ (as in Lemma 5.1), we can maintain a distance estimate $\ell(v;s)$ for every node $v$ that satisfies*

- *$\ell(v;s) \geq \operatorname{dist}(s,v)$*
- *If $\Delta_k \leq \operatorname{dist}(s,v) \leq D$, then $\ell(v;s) \leq \alpha_k \operatorname{dist}(s,v) + \beta_k$*

*The total update time is $\widetilde{O}((|F| + p|U|m^{1/p})pn^{2/q}\gamma/\epsilon)$, excluding the time needed for maintaining $E_k$.*

Note that in Lemma 5.2 we can also bound the sizes of $F$ and $U$ by $m$ and $n$, respectively and then obtain a running time of $\widetilde{O}((m + pnm^{1/p})pn^{2/q}\gamma/\epsilon)$. Furthermore, $E_k$ (for $1 \leq k \leq q-2$) is a set of edges that undergoes edge deletions, edge insertions, and edge weight increases. Given a constant $0 < \epsilon' \leq 1$ and a parameter $\alpha \geq 1$, $E_k$ is a restricted hop set in the following sense (as follows from the results in Section 5.2): Let $G_k$ be the graph that contains all edges of $G$ and $E_k$. Then for all nodes $u$ and $v$ such that $\Delta_{k+1} \leq \operatorname{dist}(u,v) \leq \Delta_{k+2}$ there is a path from $u$ to $v$ in $G_k$ of weight at most $(1+\epsilon')$ whose number of edges ("hops") is at most $(p+1)(\Delta_2+2) = O(n^{o(1)}\gamma)$.

Our strategy for proving Lemmas 5.1 and 5.2 is to prove them simultaneously by induction on $k$. The induction base is $k=0$, for which we can show that Lemma 5.2 holds by using the classic

ES-tree. The induction step for $1 \leq k \leq q-2$ works as follows. We will show that Lemma 5.1 holds for $k$ assuming that Lemma 5.2 holds for $k-1$. Similarly, we will show that Lemma 5.2 holds for $k$ assuming that Lemma 5.1 holds for $k$. Sections 5.1 and 5.2 are devoted to proving the first part and the second part of this induction step, respectively.

## 5.1 Maintaining Restricted Hop Set

In the following we prove Lemma 5.1 for a fixed value of $k$ (where $1 \leq k \leq q-2$) under the assumption that Lemma 5.2 and Lemma 5.1 itself hold for $k-1$. Thus, we show how to maintain the hop set $E_k$ using $(\alpha_{k-1}, \beta_{k-1})$-approximate SSSP data structures for distances up to $\Delta_{k+1}$.

The algorithm for maintaining $E_k$ is as follows:

1. We set $A_0 = V$ and $A_p = \emptyset$ and for $1 \leq i \leq p-1$, $A_i$ is a set of nodes constructed as follows:

   (a) Sample a set of edges from $E$ with probability $(a \ln n)/m^{i/p}$ (for a large enough constant $a$) and add all endpoints of sampled edges to $A_i$

   (b) Sample a set of nodes from $V$ with probability $(a \ln n)/n^{i/p}$ (for a large enough constant $a$) and add all sampled nodes to $A_i$.

   (c) Sample a set of nodes from $A_{i-1}$ with probability $(a \ln n)/m^{1/p}$ (for a large enough constant $a$) and add all sampled nodes to $A_i$.

   The number of nodes in $A_i$ is $\widetilde{O}(m^{1-i/p})$ in expectation. The priority of a node $v$ is the maximum $i$ such that $v \in A_i$. Note that $A_{i+1}$ might not be a subset of $A_i$.

2. For $0 \leq j \leq p-1$ we maintain a monotone ES-tree up to depth $\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1}$ from an artificial source node $s_j$ that has an edge of weight 0 to every node in $A_j$. The level of a node $u$ in this tree is denoted by $\ell(u; A_j)$. Note that $\operatorname{dist}(u, A_j) \leq \ell(u; A_j) \leq \alpha_{k-1} \operatorname{dist}(u, A_j) + \beta_{k-1}$ for every node $u$ with $\operatorname{dist}(u, A_j) \leq r_k^{(p-1)}$.[4]

   For every node $u$ (of priority $i$) we maintain

   - $\operatorname{dom}(u)$ which is the maximum $j$ such that $i \leq j \leq p-1$ and[5]

     $$\ell(u; A_j) \leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1} \text{ }^{6}$$

     and

   - for every $i' \geq i$, $\operatorname{cov}(u, i')$ which is the maximum $j$ such that $i \leq j \leq p-1$ and[7]

     $$\ell(u; A_j) \leq \alpha_{k-1}(r_k^{(i')} + s_k^{(i',j)}) + \beta_{k-1} \text{ .}^{8}$$

[4] For every $0 \leq i \leq p-1$ we define $\operatorname{dist}(u, A_i) = \min_{v \in A_i} \operatorname{dist}(u, v)$.

[5] The notation $\operatorname{dom}(u)$ comes from "dominates" and intuitively every inactive node is "dominated" by a nearby node of higher priority.

[6] By Lemma 4.8 we have $\alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1} \leq w_k^{(i,j)} \leq r_k^{(j)} \leq \alpha_{k-1} r_k^{(p-1)} + \beta_{k-1}$, the depth of the ES-tree of $A_j$.

[7] The notation $\operatorname{cov}(u, j)$ comes from "covers" and intuitively, for every $j$, a node has edges to nodes of the highest priority by which it is "$j$-covered".

[8] If $j = i'$, then $s_k^{(i',j)} = 0$ and therefore $\alpha_{k-1}(r_k^{(i')} + s_k^{(i',j)}) + \beta_{k-1} = \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$, the depth of the ES-tree of $A_j$. If $j > i$, then by Lemma 4.8 $\alpha_{k-1}(r_k^{(i')} + s_k^{(i',j)}) + \beta_{k-1} \leq w_k^{(i,j)} \leq r_k^{(j)} \leq \alpha_{k-1} r_k^{(p-1)} + \beta_{k-1}$, the depth of the ES-tree of $A_j$.

Note that $\operatorname{dom}(u)$ and $\operatorname{cov}(u,i')$ (for every $i' \geq i$) are non-increasing over the course of the algorithm since the levels in monotone ES-trees are non-decreasing. Furthermore, $\operatorname{dom}(u) \geq i$ and $\operatorname{cov}(u,i') \geq i$ (for every $i' \geq i$).

We say that a node $u$ of priority $i < p-1$ is *active* if $\ell(u; A_{i+1}) > s_k^{(i,i+1)}$. Nodes of priority $p-1$ (the highest priority) are always active. Note that when a node is active it remains active. Furthermore, if $u$ has priority $i$, then $\operatorname{dom}(u) = i$ implies that $u$ is active.

3. For every active node $v$ of priority $i$, we maintain a monotone ES-tree up to depth $\alpha_{k-1} r_k^{(i)} + \beta_{k-1}$. The level of a node $u$ in this tree is denoted by $\ell(u; v)$. Note that $\operatorname{dist}(u,v) \leq \ell(u;v) \leq \alpha_{k-1} \operatorname{dist}(u,v) + \beta_{k-1}$ for every node $u$ with $\operatorname{dist}(u,v) \leq r_k^{(i)}$.

4. For every node $u$ of priority $i$, the edge $(u,v)$ is contained in $E_k$ iff $v$ is active and

   (a) there is some $j > i$ such that $v \in A_j$, $j \geq \operatorname{dom}(u)$, and
   $$\ell(u;v) \leq w_k^{(i,j)} = \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} \text{ [9]}$$
   or

   (b) there is some $j \geq i$ such that $v \in A_j$, $j \geq \operatorname{cov}(u,j)$, and
   $$\ell(u;v) \leq w_k^{(j)} = \alpha_{k-1} r_k^{(j)} + \beta_{k-1} \text{ .[10]}$$

   In both cases, we set the weight of the edge $(u,v)$ in $E_k$ to $w_k(u,v) = \ell(u;v)$. In the first case we have $w_k(u,v) \leq w_k^{(i,j)}$ and in the second case we have $w_k(u,v) \leq w_k^{(j)}$. Furthermore we have $\operatorname{dist}(u,v) \leq w_k(u,v) \leq \alpha_{k-1} \operatorname{dist}(u,v) + \beta_{k-1}$ for every edge $(u,v)$ in $E_k$ due to $w_k(u,v) = \ell(u;v)$. Edge deletions in $G$ might cause edge deletions, edge weight increases, and edge *insertions* in $E_k$.

In the following we will show that this algorithm for maintaining $E_k$ fulfills the specifications of Lemma 5.1. First, in Section 5.1.1 we show that the edges in $E_k$ fulfill the desired structural property, which we will use in Section 5.2 prove the approximation guarantee. Second, in Section 5.1.2 we show that $E_k$ is sparse, i.e., we can bound the number of edges ever contained in $E_k$ over the course of the algorithm. Third, in Section 5.1.3 we analyze the running time of our algorithm for maintaining $E_k$. Before we give these proofs, we provide a useful lemma that allows us for every node to find a nearby active node.

**Lemma 5.3** (Domination chain)**.** *For every node $u$ of priority $i$ there is an active node $v \in A_j$ for some $j \geq i$ such that* $\operatorname{dist}(u,v) \leq s_k^{(i,j)}$.

*Proof.* The proof is by induction on $i$. If $u$ itself is active, then the claim is certainly true. If $i = p-1$, then $u$ is active by definition and the claim is true. If $i < p-1$ and $u$ is inactive, then $\ell(u; A_{i+1}) \leq s_k^{(i,i+1)}$ by the definition of being active. Since $\operatorname{dist}(u, A_{i+1}) \leq \ell(u; A_{i+1})$, there is a node $v \in A_{i+1}$ such that $\operatorname{dist}(u,v) = \operatorname{dist}(u; A_{i+1}) \leq s_k^{(i,i+1)}$. Let $j \geq i+1$ be the priority of $v$. By the induction hypothesis we know that there is an active node $v \in A_{j'}$ for some $j' \geq j$ such that $\operatorname{dist}(v,v') \leq s_k^{(j,j')}$. By the triangle inequality it follows that
$$\operatorname{dist}(u,v') \leq \operatorname{dist}(u,v) + \operatorname{dist}(v,v') \leq s_k^{(i,i+1)} + s_k^{(j,j')} \leq s_k^{(i,j)} + s_k^{(j,j')} = s_k^{(i,j')}$$
as desired. □

[9] Note that by Lemma 4.8 we have $w_k^{(i,j)} \leq r_k^{(j)} \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$, the depth of the ES-tree of $v$.

[10] Note that $r_k^{(i)} \leq r_k^{(j)}$ and therefore $w_k^{(i)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1} \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$ the depth of the ES-tree of $v$.

### 5.1.1 Structural Property

We first show that $E_k$ has a certain structural property that we need for proving the approximation guarantee of monotone ES-trees in the shortcut graph containing the edges of $E_k$ (as done in Section 5.2).

**Lemma 5.4.** *Let $u$ be a node of priority $i$ and let $v$ be a node such that $\operatorname{dist}(u, v) \leq r_k^{(i)}$. Then $E_k$ contains either*

- *for every node $v$ such that $\operatorname{dist}(u, v) \leq r_k^{(i)}$ an edge $(u, v)$ of weight $w_k(u, v) \leq w_k^{(i)}$ with $\operatorname{dist}(u, v) \leq w_k(u, v) \leq \alpha_{k-1} \operatorname{dist}(u, v) + \beta_{k-1}$ (if $\operatorname{dom}(u) = i$) or*
- *an edge $(u, v')$ to a node $v' \in A_{j'}$ for some $j' > i$ of weight $w_k(u, v) \leq w_k^{(i,j')}$ with $\operatorname{dist}(u, v') \leq w_k(u, v') \leq \alpha_{k-1} \operatorname{dist}(u, v') + \beta_{k-1}$ (if $\operatorname{dom}(u) > i$).*

*Proof.* Let $j$ denote the priority of $v$. We distinguish two cases: the first case is that $\operatorname{dom}(u) = i$ and the second case is that $\operatorname{dom}(u) > i$.

Case 1: $\operatorname{dom}(u) = i$

In this case $u$ has to be active because if $u$ were inactive, then $\ell(u; A_{i+1}) \leq s_k^{(i,i+1)}$, which implies that $\operatorname{dom}(u) \geq i + 1$, contradicting the assumption $\operatorname{dom}(u) = i$. Furthermore it has to be the case that $i \geq j$: Suppose that $i < j$. Then

$$\begin{aligned}\ell(u; A_j) &\leq \alpha_{k-1} \operatorname{dist}(u, A_j) + \beta_{k-1} \leq \alpha_{k-1} \operatorname{dist}(u, v) + \beta_{k-1} \\ &\leq \alpha_{k-1} r_k^{(i)} + \beta_{k-1} \leq \alpha_{k-1} s_k^{(i,i+1)} + \beta_{k-1} \leq \alpha_{k-1} s_k^{(i,j)} + \beta_{k-1}\end{aligned}$$

which implies that $\operatorname{dom}(u) \geq j > i$, contradicting the assumption $\operatorname{dom}(u) = i$.

We now argue that $\operatorname{cov}(v, i) = i$. Note that if $\operatorname{cov}(v, i) = i$, then $E_k$ contains the edge $(v, u)$ of weight $w_k^{(i)}$ since $u$ is active, $i \geq j$, $u \in A_i$, and $\ell(v; u) \leq \alpha_{k-1} \operatorname{dist}(v, u) + \beta_{k-1} \leq \alpha_{k-1} r_k^{(i)} + \beta_{k-1} = w_k^{(i)}$. Note that $\operatorname{cov}(v, i) \geq i$: since $\operatorname{dist}(u, v) \leq r_k^{(i)}$ and $u \in A_i$, we have $\ell(v; A_i) \leq \alpha_{k-1} \operatorname{dist}(v, A_i) + \beta_{k-1} \leq \alpha_{k-1} r_k^{(i)} + \beta_{k-1}$. If $i = p - 1$ then $\operatorname{cov}(v, i) = i$. If $i < p - 1$, then suppose that $\operatorname{cov}(v, i) > i$. It follows that $j' > i$ such that $\ell(v; A_{j'}) \leq \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')}) + \beta_{k-1}$. Since $\operatorname{dist}(v, A_{j'}) \leq \ell(v; A_{j'})$, there is some node $v' \in A_{j'}$ such that $\operatorname{dist}(v, v') = \operatorname{dist}(v, A_{j'}) \leq \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')}) + \beta_{k-1}$. By the triangle inequality we have

$$\operatorname{dist}(u, A_{j'}) \leq \operatorname{dist}(u, v') \leq \operatorname{dist}(u, v) + \operatorname{dist}(v, v') \leq r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}\,.$$

It follows that

$$\ell(u; A_{j'}) \leq \alpha_{k-1} \operatorname{dist}(u, A_{j'}) + \beta_{k-1} \leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}$$

and thus $\operatorname{dom}(u) \geq j' > i$, which contradicts the assumption $\operatorname{dom}(u) = i$.

Case 2: $\operatorname{dom}(u) > i$

Set $j' = \operatorname{dom}(u)$. By the definition of $\operatorname{dom}(u)$ we have

$$\ell(u; A_{j'}) \leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')}) + \beta_{k-1}) + \beta_{k-1}\,.$$

Since $\operatorname{dist}(u, A_{j'}) \leq \ell(u; A_{j'})$, there is a node $v' \in A_{j'}$ such that

$$\operatorname{dist}(u, v') = \operatorname{dist}(u, A_{j'}) \leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')}) + \beta_{k-1}) + \beta_{k-1}\,.$$

By Lemma 5.3 there is an active node $v'' \in A_{j''}$ for some $j'' \geq j'$ such that $\operatorname{dist}(v', v'') \leq s_k^{(j',j'')}$. By the triangle inequality we have

$$\begin{aligned}
\operatorname{dist}(u, v'') &\leq \operatorname{dist}(u, v') + \operatorname{dist}(v', v'') \\
&\leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')}) + \beta_{k-1}) + \beta_{k-1} + s_k^{(j',j'')} \\
&\leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j')} + s_k^{(j',j'')}) + \beta_{k-1}) + \beta_{k-1} \\
&= \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j'')}) + \beta_{k-1}) + \beta_{k-1}
\end{aligned}$$

It follows that

$$\begin{aligned}
\ell(u; v'') &\leq \alpha_{k-1} \operatorname{dist}(u, v'') + \beta_{k-1} \\
&\leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j'')}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} = w_k^{(i,j'')} .
\end{aligned}$$

Since $j'' \geq j' = \operatorname{dom}(u) > i$, $E_k$ therefore contains the edge $(u, v'')$ of weight at most $w_k^{(i,j'')}$. □

### 5.1.2 Sparsity

We now show that $E_k$ is sparse, which means that we have to bound $\mathcal{E}(E_k)$, the number of edges ever contained in $E_k$, when edges are deleted from $G$. We can show that $\mathcal{E}(E_k)$ is $\widetilde{O}(pm^{1+1/p})$. This bound however is *not sufficient* for our purposes. Instead, we show that for every set $U \subseteq V$ of nodes there are at most $\widetilde{O}(|U|pm^{1/p})$ many edges adjacent to the nodes in $U$ over the whole course of the algorithm. This also allows us to bound the number of edges ever contained in certain subsets of $E_k$. For simplicity we view the edges contained in $E_k$ before the first edge deletion in $G$ as being inserted into $E_k$ at its initialization.

**Lemma 5.5.** *For every node $u$ of priority $i$ and every $i < j \leq p - 2$, during the time interval when $\operatorname{dom}(u) \leq j$ there will be at most $m^{1/p}$ active nodes $v \in A_j$ such that $\ell(u; v) \leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}$ whp.*

*Proof.* Remember that $\operatorname{dom}(u)$ is non-increasing. Let $G'$ be the earliest graph in the sequence of graph generated by edge deletions where $\operatorname{dom}(u) \leq j$. Define $U = \{v \in A_j \mid \operatorname{dist}_{G'}(u, v) \leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}\}$. Suppose that there will be more than $m^{1/p}$ active nodes $v \in A_j$ such that $\ell(u; v) \leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}$ during the time interval when $\operatorname{dom}(u) \leq j$. Since $\operatorname{dist}(u, v) \leq \ell(u; v)$ at any time and $\operatorname{dist}_{G'}(u, v) \leq \operatorname{dist}(u, v)$ as distances are non-decreasing under edge deletions, we have $\operatorname{dist}_{G'}(u, v) \leq \ell(u; v) \leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}$ and thus every such node $v$ is contained in $U$. Therefore $U$ contains more than $m^{1/p}$ nodes. As $U$ is a subset of $A_j$ it contains some node $v' \in A_{j+1}$ whp by Lemma 4.3. Since $w_k^{(i,j)} \leq r_k^{(j)}$ by Lemma 4.8 we get

$$\begin{aligned}
\operatorname{dist}(u, A_{j+1}) \leq \operatorname{dist}(u, v') &\leq \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1} = w_k^{(i,j)} \\
&\leq r_k^{(j)} \leq \alpha_{k-1}(\alpha_{k-1} r_k^{(j)} + \beta_{k-1}) + \beta_{k-1} = s_k^{(j,j+1)} \leq s_k^{(i,j+1)} \\
&\leq r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j+1)}) + \beta_{k-1} .
\end{aligned}$$

Thus, $\ell(u, A_{j+1}) \leq \alpha_{k-1} \operatorname{dist}(u, A_{j+1}) + \beta_{k-1} \leq \alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j+1)}) + \beta_{k-1}) + \beta_{k-1}$. It follows that $\operatorname{dom}(u) \geq j + 1 > j$ which contradicts the assumption $\operatorname{dom}(u) \leq j$. □

**Lemma 5.6.** *For every node $u$ of priority $i$ and every $i \leq j \leq p-2$, during the time interval when $\operatorname{cov}(u,j) \leq j$ there will be at most $m^{1/p}$ active nodes $v \in A_j$ such that $\ell(u;v) \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$ whp.*

*Proof.* Remember that $\operatorname{cov}(u,j)$ is non-increasing. Let $G'$ be the earliest graph in the sequence of graph generated by edge deletions where $\operatorname{cov}(u,j) \leq j$. Define $U = \{v \in A_j \mid \operatorname{dist}_{G'}(u,v) \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}\}$. Suppose that there will be more than $m^{1/p}$ active nodes $v \in A_j$ such that $\ell(u;v) \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$ during the time interval when $\operatorname{cov}(u,j) \leq j$. Since $\operatorname{dist}(u,v) \leq \ell(u;v)$ at any time and $\operatorname{dist}_{G'}(u,v) \leq \operatorname{dist}(u,v)$ as distances are non-decreasing under edge deletions, we have $\operatorname{dist}_{G'}(u,v) \leq \ell(u;v) \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1}$ and thus every such node $v$ is contained in $U$. Therefore $U$ contains more than $m^{1/p}$ nodes. As $U$ is a subset of $A_j$ it contains some node $v' \in A_{j+1}$ whp by Lemma 4.3. We therefore get

$$\operatorname{dist}(u, A_{j+1}) \leq \operatorname{dist}(u,v') \leq \alpha_{k-1} r_k^{(j)} + \beta_{k-1} \leq \alpha_{k-1}(\alpha_{k-1} r_k^{(j)} + \beta_{k-1}) + \beta_{k-1} = s_k^{(j,j+1)} \leq r_k^{(j)} + s_k^{(j,j+1)}$$

and thus $\ell(u, A_{j+1}) \leq \alpha_{k-1} \operatorname{dist}(u, A_{j+1}) + \beta_{k-1} \leq \alpha_{k-1}(r_k^{(j)} + s_k^{(j,j+1)}) + \beta_{k-1}$. It follows that $\operatorname{cov}(u,j) \geq j+1 > j$ which contradicts the assumption $\operatorname{cov}(u,j) \leq j$. □

We can now bound the number of edges ever contained in $E_k$ and its subsets.

**Lemma 5.7.** *For every subset $E_k|U$ of $E_k$ induced by a set of nodes $U$, the number of edges ever contained in $E_k|U$ is $\mathcal{E}(E_k|U) = \widetilde{O}(|U| p m^{1/p})$ in expectation.*

*Proof.* We want to charge $O(pm^{1/p})$ insertions to each node of $U$. To this end we view every undirected edge $(u,v)$ that we insert as an ordered pair in which $u$ has some priority $i$ and $v \in A_j$ for some $j \geq i$. Thus, we need to show that for every node $u$ there are at most $O(pm^{1/p})$ such pairs $(u,v)$ over the course of the algorithm.

For every node $u$ of priority $i$ we only insert an edge $(u,v)$ to a node $v \in A_j$ (for some $j \geq i$) if $v$ is active and

1. $j \geq \operatorname{dom}(u)$, $j > i$, and $\ell(u;v) \leq w_k^{(i,j)} = \alpha_{k-1}(\alpha_{k-1}(r_k^{(i)} + \alpha_{k-1}(r_k^{(i)} + s_k^{(i,j)}) + \beta_{k-1}) + \beta_{k-1}) + \beta_{k-1}$ or
2. $j \geq \operatorname{cov}(u,j)$ and $\ell(u;v) \leq w_k^{(i)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1}$.

If $j = p-1$, then the number of insertions of edges adjacent to nodes in $A_{p-1}$ is $\widetilde{O}(|U| m^{1/p})$ since $|A_{p-1}| = \widetilde{O}(m^{1/p})$ in expectation. If $j < p-1$, then by Lemma 5.5 the number of insertions of the first type of edges is at most $m^{1/p}$ and by Lemma 5.6 the number of insertions of the second type of edges is at most $m^{1/p}$. As there are $p$ possible values for $j$, we can charge $\widetilde{O}(pm^{1/p})$ insertions to $u$ and thus the desired bound follows. □

### 5.1.3 Running Time

Finally we analyze the running time needed to maintain $E_k$. By our induction hypotheses we may assume that Lemmas 5.1 and 5.2 hold for $k-1$. Thus, we can maintain $E_{k-1}$ in time $\widetilde{O}((k-1)p^3 m^{1+2/p} n^{2/q} \gamma/\epsilon)$ and using $E_{k-1}$ we can maintain $(\alpha_{k-1}, \beta_{k-1})$-approximate SSSP up to depth $\Delta_{k+1}$.

**Lemma 5.8.** *Maintaining the monotone ES-trees of all active nodes takes time $\widetilde{O}(p^3 m^{1+2/p} n^{2/q} \gamma/\epsilon)$ in expectation.*

*Proof.* We first bound the time needed for maintaining the monotone ES-trees of active nodes of priority $p-1$, the highest priority. These trees have depth $\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1}$. Since $r_k^{(p-1)} \leq \Delta_{k+1}$ by Lemma 4.11, we may apply Lemma 5.2 and get that maintaining one such tree takes time $\widetilde{O}((m + pnm^{1/p}) pn^{2/q} \gamma / \epsilon)$. As there are $\widetilde{O}(m^{1/p})$ nodes of priority $p-1$, the total time needed for maintaining these trees is $\widetilde{O}(p^2 m^{1+1/p} n^{2/q} \gamma / \epsilon)$.

We now bound the time needed for maintaining the monotone ES-trees of active nodes $u$ of priority $i < p-1$. Consider the time when $u$ becomes active and let $U$ be the following set of nodes and edges, respectively:

$$U = \{ v \in V \mid \operatorname{dist}(v, u) \leq r_k^{(i)} \}$$
$$F = \{ (v, v') \in E \mid \operatorname{dist}(v, u) \leq r_k^{(i)} \text{ or } \operatorname{dist}(v', u) \leq r_k^{(i)} \} .$$

Since $r_k^{(i)} \leq \Delta_{k+1}$, maintaining the monotone ES-tree of $u$ takes time $\widetilde{O}((|F| + p|U|m^{1/p}) pn^{2/q} \gamma / \epsilon)$ by Lemma 5.2.

By the random sampling of nodes and edges we can bound bound $|U|$ and $|F|$ as follows. Suppose that $|U| > n / n^{1-(i+1)/p} = n^{(i+1)/p}$ or $|F| > m / m^{1-(i+1)/p} = m^{(i+1)/p}$. If $|U| > n^{(i+1)/p}$, then $U$ contains a node $v \in A_{i+1}$ whp by Lemma 4.3. If $|F| > m^{(i+1)/p}$, then $F$ contains an edge $(v, v')$ such that $v$ and $v'$ are in $A_{i+1}$ whp by Lemma 4.3.. Thus, in both cases we find a node $v \in A_{i+1}$ such that $\operatorname{dist}(u, v) \leq r_k^{(i)}$. We therefore get

$$\ell(u; A_{i+1}) \leq \alpha_{k-1} \operatorname{dist}(u, A_{i+1}) + \beta_{k-1} \leq \alpha_{k-1} \operatorname{dist}(u, v) + \beta_{k-1} \leq \alpha_{k-1} r_k^{(i)} + \beta_{k-1} = s_k^{(i,i+1)} .$$

which implies that $u$ is not active, contradicting our assumption. It follows that $|U| = n^{(i+1)/p} \leq m^{(i+1)/p}$ and $|F| \leq m^{(i+1)/p}$.

Thus, maintaining the monotone ES-tree of an active node of priority $i < p-1$ takes time

$$\widetilde{O}((m^{(i+1)/p} + pn^{(i+1)/p} m^{1/p}) pn^{2/q} \gamma / \epsilon) = \widetilde{O}(p^2 m^{(i+1)/p + 1/p + 2/q} \gamma / \epsilon) .$$

Remember that there are $\widetilde{O}(m^{1-i/p})$ nodes of priority $i$ in expectation. Since $m^{1-i/p} \cdot m^{(i+1)/p} = m^{1+1/p}$, the total time needed for maintaining the ES-trees of active nodes of priority $i < p-1$ is $\widetilde{O}(p^2 m^{1+2/p} n^{2/q} \gamma / \epsilon)$ and the total time for maintaining the ES-trees of all active nodes is $\widetilde{O}(p^3 m^{1+2/p} n^{2/q} \gamma / \epsilon)$. □

**Lemma 5.9.** *Maintaining the monotone ES-trees of all sets $A_i$ takes time $\widetilde{O}(p^2 m^{1+1/p} n^{2/q} \gamma / \epsilon)$.*

*Proof.* For every $0 \leq j \leq p-1$, we maintain a monotone ES-tree of $A_j$ up to depth $\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1}$. Since $r_k^{(p-1)} \leq \Delta_{k+1}$, maintaining the monotone ES-tree of a fixed $A_j$ takes time $\widetilde{O}((m + pnm^{1/p}) pn^{2/q} \gamma / \epsilon)$ by Lemma 5.2. As there are $p$ such trees, maintaining all of them takes time $\widetilde{O}(p^2 m^{1+1/p} n^{2/q} \gamma / \epsilon)$. □

Note that the running time of Lemma 5.9 is dominated by the running time of Lemma 5.8. It follows that the time needed for maintaining $E_k$ is the sum of $\widetilde{O}((k-1) p^3 m^{1+2/p} n^{2/q} \gamma / \epsilon)$ (for maintaining $E_{k-1}$) and $\widetilde{O}(p^3 m^{1+2/p} n^{2/q} \gamma / \epsilon)$ (for maintaining approximate SSSP of active nodes), which is $\widetilde{O}(kp^3 m^{1+2/p} n^{2/q} \gamma / \epsilon)$ in total.

## 5.2 Monotone ES-tree Using Shortcut Graph

In the following we prove Lemma 5.2 for a fixed value of $k$ (where $1 \leq k \leq q-2$) under the assumption that Lemma 5.1 holds for $k$. Thus, we show how to maintain $(\alpha_k, \beta_k)$-approximate

SSSP for distances up to $\Delta_{k+2}$ assuming that we can maintain the hop set $E_k$. For this purpose, we will use the monotone ES-tree on the shortcut graph $G_k$ containing all edges of $G$ and $E_k$. However, we cannot run the monotone ES-tree on $G_k$ itself as the edge weights in $G_k$ might be relatively large, which makes the monotone ES-tree inefficient. Instead, we scale down the edge weights and run the algorithm on the resulting graph.

Remember that $w(u, v)$ denotes the weight of an edge $(u, v)$ in $G$ and $w_k(u, v)$ denotes the weight of an edge $(u, v)$ in $E_k$. We define $G'_k$ to be the graph with the nodes $V$, all edges from $E_k$ and all edges from $G$ such that $w(u, v) \leq \Delta_{k+2}$. We round every edge weight in $G'_k$ to the next multiple of $\varphi_k$ where

$$\varphi_k = \frac{\epsilon \Delta_k}{p+1}\,.$$

Let $w'_k(u, v)$ denote the edge weight in $G'_k$. For every edge $(u, v)$ of $G$ with $w(u, v) \leq \Delta_{k+2}$ we have

$$w'_k(u, v) = \left\lceil \frac{w(u, v)}{\varphi_k} \right\rceil \varphi_k$$

and for every edge $(u, v)$ of $E_k$ we have

$$w'_k(u, v) = \left\lceil \frac{w_k(u, v)}{\varphi_k} \right\rceil \varphi_k\,.$$

This kind of rounding guarantees that

$$w(u, v) \leq w'_k(u, v) \leq w(u, v) + \varphi_k\,.$$

for every edge $(u, v)$ of $G$ with $w(u, v) \leq \Delta_{k+2}$ and

$$w_k(u, v) \leq w'_k(u, v) \leq w_k(u, v) + \varphi_k$$

for every edge $(u, v)$ of $E_k$.

We now analyze the approximation error of a monotone ES-tree maintained on $G'_k$. 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 $E_k$. The second part is the approximation error we get from rounding the edge weights. The analysis of the first part follows arguments in [10], whereas the analysis of the second part is entirely new to this paper. 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 [7] for the full algorithm). Similar to the classic ES-tree, the monotone ES-tree with root $s$ maintains a level $\ell(v; s)$ for every node $v$. Remember that a single deletion or edge weight increase in $G$ might result in a bunch of deletions, weight increases and insertions in $E_k$. 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. The procedure for handling the insertion of a node $(u, v)$ is almost trivial and in particular does *not* change $\ell(u; s)$ or $\ell(v; s)$ We say that an edge $(u, v)$ is *stretched* if $\ell(u; s) > \ell(v; s) + w'_k(u, v)$. 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 $\ell(u; s) \leq \ell(v; s) + w'_k(u, v)$ for every edge $(u, v)$ contained in $G'_k$. In our proof we will use the following properties of the monotone ES-tree [7]:

- The level of a node never decreases.
- An edge can only become stretched when it is inserted.
- As long as a node $v$ is stretched, its level does not change.
- The monotone ES-tree never underestimates the true distance.

**Lemma 5.10.** *For every node $u$ of priority $i$ with $\operatorname{dist}(u,s) \leq \Delta_{k+2}$ in the monotone ES-tree of a node $s$ in $G'_k$ we have*

$$\ell(u;s) \leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(u,s) + \beta_k^{(i)} + \left( (p+1) \left\lceil \frac{\max(\operatorname{dist}(u,s) - r_k^{(i)}, 0)}{\Delta_k} \right\rceil + p + 1 - i \right) \varphi_k \,.$$

*Proof.* For every node $u$ and every $0 \leq i \leq p-1$ we define $h(u,i)$ as follows:

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

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. We will now show that, for every node $u$ of priority $i$,

$$\ell(u;s) \leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(u,s) + \beta_k^{(i)} + h(u,i) \cdot \varphi_k \,.$$

This clearly implies the lemma. The proof is by double induction first on the number of deletions in $G$ and second on $h(u,i)$.

Let $u$ be a node of priority $i$. If $u = s$, the claim is trivially true because $\ell(s;s) = 0$. Assume that $u \neq s$. 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 $\ell(u;s) \leq \ell(v;s) + w'_k(u,v)$ for every edge $(u,v)$ in $E_k$. Define the nodes $v$ and $x$ as follows. If $\operatorname{dist}(u,s) \leq r_k^{(i)}$, then $v = s$. If $\operatorname{dist}(u,s) > r_k^{(i)}$, consider a shortest path from $u$ to $s$ and let $v$ be the furthest node from $u$ on the shortest path such that $\operatorname{dist}(u,v) \leq r_k^{(i)}$ (which implies $\operatorname{dist}(v,s) \geq \operatorname{dist}(u,s) - r_k^{(i)}$). Furthermore let $x$ be the neighbor of $v$ on the shortest path such that $\operatorname{dist}(u,x) \geq r_k^{(i)}$ (and thus $\operatorname{dist}(x,s) \leq \operatorname{dist}(u,s) - r_k^{(i)}$) and $G$ contains the edge $(v,x)$. Note that $(v,x)$ is also contained in $G'_k$ because for $\operatorname{dist}(u,s) \leq \Delta_{k+2}$ to hold it has to be the case that $w(u,s) \leq \Delta_{k+2}$. By Lemma 5.4 we know that $E_k$ either contains an edge $(u,v)$ of weight $w_k(u,v) \leq w_k^{(i)}$ or an edge $(u,v')$ to a node $v' \in A_{j'}$ for some $j' > i$ such that $w_k(u,v') \leq w_k^{(i,j')}$.

Case 1: $E_k$ contains $(u,v)$

If $\operatorname{dist}(u,s) \leq r_k^{(i)}$, then we have $v = s$ and therefore get

$$\begin{aligned}
\ell(u;s) &\leq \ell(v;s) + w'_k(u,v) && (u \text{ not stretched}) \\
&= \ell(s;s) + w'_k(u,s) && (v = s) \\
&= w'_k(u,s) && (\ell(s;s) = 0) \\
&\leq w_k(u,s) + \varphi_k && (\text{property of } w'_k(u,v)) \\
&\leq w_k(u,s) + h(u,i) \cdot \varphi_k && (h(u,i) \geq 1 \text{ since } u \neq s) \\
&\leq \alpha_{k-1}\operatorname{dist}(u,s) + \beta_{k-1} + h(u,i) \cdot \varphi_k && (\text{property of weight in } E_k) \\
&\leq \alpha_k \operatorname{dist}(u,s) + \beta_k + h(u,i) \cdot \varphi_k && (\alpha_{k-1} \leq \alpha_k \text{ and } \beta_{k-1} \leq \beta_k) \,.
\end{aligned}$$

Consider now the case $\mathrm{dist}(u, s) > r_k^{(i)}$. Let $j$ denote the priority of $x$. We first prove the following inequality, which will allow us among others to use the induction hypothesis on $x$.

**Claim 5.11.** *If* $\mathrm{dist}(u, s) > r_k^{(i)}$, *then* $h(x, j) + 2 \leq h(u, i)$.

*Proof.* Remember that $i \leq p - 1$.

The assumption $\mathrm{dist}(u, s) > r_k^{(i)}$ implies that $\mathrm{dist}(x, s) \leq \mathrm{dist}(u, s) - r_k^{(i)}$. If $\mathrm{dist}(x, s) < r_k^{(j)}$, we have

$$\begin{aligned} h(x, j) + 2 &\leq p + 1 - j + 2 \leq p + 1 + 2 \leq p + 1 + p + 1 - i \\ &\leq (p + 1) \left\lceil \frac{\mathrm{dist}(u, s) - r_k^{(i)}}{\Delta_k} \right\rceil + p + 1 - i = h(u, i)\,. \end{aligned}$$

Here we use the inequality $\lceil (\mathrm{dist}(u, s) - r_k^{(j)})/\Delta_k \rceil \geq 1$ which follows from the assumption $\mathrm{dist}(u, s) > r_k^{(i)}$.

If $\mathrm{dist}(x, s) \geq r_k^{(j)}$, then, using $r_k^{(j)} \geq r_k^{(0)} \geq \Delta_k$, we get

$$\begin{aligned} h(x, j) + 2 &= (p + 1) \left\lceil \frac{\mathrm{dist}(x, s) - r_k^{(j)}}{\Delta_k} \right\rceil + p + 1 - j + 2 \\ &\leq (p + 1) \left\lceil \frac{\mathrm{dist}(x, s) - \Delta_k}{\Delta_k} \right\rceil + p + 1 + 2 \\ &= (p + 1) \left\lceil \frac{\mathrm{dist}(x, s)}{\Delta_k} - 1 \right\rceil + p + 1 + 2 \\ &= (p + 1) \left( \left\lceil \frac{\mathrm{dist}(x, s)}{\Delta_k} \right\rceil - 1 \right) + p + 1 + 2 \\ &= (p + 1) \left\lceil \frac{\mathrm{dist}(x, s)}{\Delta_k} \right\rceil + 2 \\ &\leq (p + 1) \left\lceil \frac{\mathrm{dist}(x, s)}{\Delta_k} \right\rceil + p + 1 - i \\ &\leq (p + 1) \left\lceil \frac{\mathrm{dist}(u, s) - r_k^{(i)}}{\Delta_k} \right\rceil + p + 1 - i \\ &\leq (p + 1) \left\lceil \frac{\max(\mathrm{dist}(u, s) - r_k^{(i)}, 0)}{\Delta_k} \right\rceil + p + 1 - i = h(u, i)\,. \end{aligned}$$

Here the last inequality follows from the trivial observation $\mathrm{dist}(u, s) - r_k^{(i)} \leq \max(\mathrm{dist}(u, s) - r_k^{(i)}, 0)$. □

Having proved this claim, we go on with the proof of the lemma. If $\mathrm{dist}(u, s) > r_k^{(i)}$, then we have $\mathrm{dist}(u, x) \geq r_k^{(i)}$ by the choice of $x$. Note that the edge $(v, x)$ will never be stretched because it is contained in $G$ since before the first deletion and therefore is not an inserted edge. Therefore we get

$$
\begin{aligned}
\ell(u;s) &\leq \ell(v;s) + w'_k(u,v) && (u \text{ not stretched})\\
&\leq \ell(x;s) + w'_k(v,x) + w'_k(u,v) && (v \text{ not stretched})\\
&\leq \ell(x;s) + w(v,x) + \varphi_k + w_k(u,v) + \varphi_k && (\text{property of } w'_k)\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(j)} + w(v,x) + \varphi_k + w_k(u,v) + \varphi_k && (\text{induction hypothesis})\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(j)} + w_k(u,v) + w(v,x) + (h(x,j)+2)\cdot\varphi_k && (\text{rearranging terms})\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(j)} + w_k(u,v) + w(v,x) + h(u,i)\cdot\varphi_k && (\text{Claim 5.11})\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(0)} + w_k(u,v) + w(v,x) + h(u,i)\cdot\varphi_k && (\beta_k^{(j)} \leq \beta_k^{(0)})\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(0)} + \alpha_{k-1}\operatorname{dist}(u,v) + \beta_{k-1} + w(v,x) + h(u,i)\cdot\varphi_k && (\text{property of weight in } E_k)\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(0)} + \alpha_{k-1}\operatorname{dist}(u,v) + \beta_{k-1} + \operatorname{dist}(v,x) + h(u,i)\cdot\varphi_k && ((v,x) \text{ on shortest path})\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \beta_k^{(0)} + \alpha_{k-1}\operatorname{dist}(u,v) + \beta_{k-1} + \alpha_{k-1}\operatorname{dist}(v,x) + h(u,i)\cdot\varphi_k && (\alpha_{k-1} \geq 1)\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \alpha_{k-1}(\operatorname{dist}(u,v) + \operatorname{dist}(v,x)) + \beta_{k-1} + \beta_k^{(0)} + h(u,i)\cdot\varphi_k && (\text{rearranging terms})\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \alpha_{k-1}\operatorname{dist}(u,x) + \beta_{k-1} + \beta_k^{(0)} + h(u,i)\cdot\varphi_k && (v \text{ on shortest path})\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \alpha_{k-1}\operatorname{dist}(u,x) + \beta_{k-1} + \beta_k^{(0)} - \beta_k^{(i)} + \beta_k^{(i)} + h(u,i)\cdot\varphi_k && (\text{rearranging terms})\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \alpha_{k-1}\operatorname{dist}(u,x) + \epsilon r_k^{(i)} + \beta_k^{(i)} + h(u,i)\cdot\varphi_k && (\text{definition of } r_k^{(i)})\\
&\leq (\alpha_{k-1}+\epsilon)\operatorname{dist}(x,s) + \alpha_{k-1}\operatorname{dist}(u,x) + \epsilon\operatorname{dist}(u,x) + \beta_k^{(i)} + h(u,i)\cdot\varphi_k && (\operatorname{dist}(u,x) \geq r_k^{(i)})\\
&= (\alpha_{k-1}+\epsilon)(\operatorname{dist}(u,x) + \operatorname{dist}(x,s)) + \beta_k^{(i)} + h(u,i)\cdot\varphi_k && (\text{rearranging terms})\\
&= (\alpha_{k-1}+\epsilon)\operatorname{dist}(u,s) + \beta_k^{(i)} + h(u,i)\cdot\varphi_k && (x \text{ on shortest path})\,.
\end{aligned}
$$

Case 2: $E_k$ contains $(u,v')$ $(v' \neq v)$

We first prove the following inequality, which will allow us among others to use the induction hypothesis on $x$.

**Claim 5.12.** $h(v',j') + 1 \leq h(u,i)$

*Proof.* Remember that $j' \geq i+1$. If $\operatorname{dist}(v',s) < r_k^{(j')}$, we get

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

If $\operatorname{dist}(v', s) \geq r_k^{(j')}$, then we use the inequality $w_k^{(i,j')} + r_k^{(i)} \leq r_k^{(j')}$ from Lemma 4.8 and get

$$\begin{aligned} h(v', j') + 1 &= (p+1) \left\lceil \frac{\operatorname{dist}(v', s) - r_k^{(j')}}{\Delta_k} \right\rceil + p + 1 - j' + 1 \\ &\leq (p+1) \left\lceil \frac{\operatorname{dist}(v', s) - r_k^{(j')}}{\Delta_k} \right\rceil + p + 1 - i - 1 + 1 \\ &\leq (p+1) \left\lceil \frac{\operatorname{dist}(u, s) + \operatorname{dist}(v', u) - r_k^{(j')}}{\Delta_k} \right\rceil + p + 1 - i \\ &\leq (p+1) \left\lceil \frac{\operatorname{dist}(u, s) + w_k(u, v') - r_k^{(j')}}{\Delta_k} \right\rceil + p + 1 - i \\ &\leq (p+1) \left\lceil \frac{\operatorname{dist}(u, s) + w_k^{(i,j')} - r_k^{(j')}}{\Delta_k} \right\rceil + p + 1 - i \\ &\leq (p+1) \left\lceil \frac{\operatorname{dist}(u, s) - r_k^{(i)}}{\Delta_k} \right\rceil + p - i \\ &\leq (p+1) \left\lceil \frac{\max(\operatorname{dist}(u, s) - r_k^{(i)}, 0)}{\Delta_k} \right\rceil + p + 1 - i = h(u, i)\,. \end{aligned}$$

Here we use the inequality $w_k^{(i,j')} + r_k^{(i)} \leq r_k^{(j')}$ we proved above. □

Having proved this claim, we go on with the proof of the lemma.

$$\begin{aligned} \ell(u) &\leq \ell(v') + w'_k(u, v') && (u \text{ not stretched}) \\ &\leq \ell(v') + w_k(u, v') + \varphi_k && (\text{property of } w'_k(u, v')) \\ &\leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(v', s) + \beta_k^{(j')} + h(v', j') \cdot \varphi_k + w_k(u, v') + \varphi && (\text{induction hypothesis}) \\ &= (\alpha_{k-1} + \epsilon) \operatorname{dist}(v', s) + \beta_k^{(j')} + w_k(u, v') + (h(v', j') + 1) \cdot \varphi_k && (\text{rearranging terms}) \\ &\leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(v', s) + \beta_k^{(j')} + w_k(u, v') + h(u, i) \cdot \varphi_k && (\text{Claim 5.12}) \\ &\leq (\alpha_{k-1} + \epsilon)(\operatorname{dist}(v', u) + \operatorname{dist}(u, s)) + \beta_k^{(j')} + w_k(u, v') + h(u, i) \cdot \varphi_k && (\text{triangle inequality}) \\ &\leq (\alpha_{k-1} + \epsilon)(w_k(u, v') + \operatorname{dist}(u, s)) + \beta_k^{(j')} + w_k(u, v') + h(u, i) \cdot \varphi_k && (\text{property of weight in } E_k) \\ &= (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k^{(j')} + (\alpha_{k-1} + \epsilon + 1) w_k(u, v') + h(u, i) \cdot \varphi_k && (\text{rearranging terms}) \\ &\leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k^{(j')} + 4 w_k(u, v') + h(u, i) \cdot \varphi_k && (\alpha_{k-1} \leq 2 \text{ and } \epsilon \leq 1) \\ &\leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k^{(j')} + 4 w_k^{(i,j')} + h(u, i) \cdot \varphi_k && (\text{property of weight in } E_k,\ v' \neq v) \\ &\leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k^{(i)} + h(u, i) \cdot \varphi_k && (\text{Lemma 4.9}) \end{aligned}$$

□

Having analyzed the dependence of approximation error on the priorities of the nodes and their distances to the root of the monotone ES-tree, the desired general approximation guarantee easily follows.

**Lemma 5.13.** *For every node $u$, if $\Delta_k \leq \operatorname{dist}(u, s) \leq \Delta_{k+2}$, then $\ell(u; s) \leq \alpha_k \operatorname{dist}(u, s) + \beta_k$.*

*Proof.* Let $i$ be the priority of $u$. By Lemma 5.10 we have

$$\ell(u; s) \leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k^{(i)} + \left( (p+1) \left\lceil \frac{\max(\operatorname{dist}(u, s) - r_k^{(i)}, 0)}{\Delta_k} \right\rceil + p + 1 - i \right) \varphi_k \,.$$

Note that $\beta_k^{(i)} \leq \beta_k^{(0)} = \beta_k$. Furthermore we have

$$\begin{aligned} \left( (p+1) \left\lceil \frac{\max(\operatorname{dist}(u, s) - r_k^{(i)}, 0)}{\Delta_k} \right\rceil + p + 1 - i \right) \varphi_k &\leq \left( (p+1) \left\lceil \frac{\operatorname{dist}(u, s)}{\Delta_k} \right\rceil + p + 1 \right) \varphi_k \\ &\leq \left( (p+1) \left( \frac{\operatorname{dist}(u, s)}{\Delta_k} + 1 \right) + p + 1 \right) \varphi_k \\ &= \left( \frac{(p+1) \operatorname{dist}(u, s)}{\Delta_k} + 2(p+1) \right) \varphi_k \\ &= \left( \frac{(p+1) \operatorname{dist}(u, s)}{\Delta_k} + 2(p+1) \right) \cdot \frac{\epsilon \Delta_k}{p+1} \\ &= \epsilon \operatorname{dist}(u, s) + 2\epsilon \Delta_k \\ &\leq \epsilon \operatorname{dist}(u, s) + 2\epsilon \operatorname{dist}(u, s) \\ &= 3\epsilon \operatorname{dist}(u, s) \,. \end{aligned}$$

By combining these two inequalities we get

$$\ell(u; s) \leq (\alpha_{k-1} + \epsilon) \operatorname{dist}(u, s) + \beta_k + 3\epsilon \operatorname{dist}(u, s) = (\alpha_{k-1} + 4\epsilon) \operatorname{dist}(u, s) + \beta_k = \alpha_k \operatorname{dist}(u, s) + \beta_k \,. \quad \square$$

Finally we provide the running time analysis. We run the algorithm in a graph in which we scale down the edge weights by a factor of $\varphi_k$. This makes the algorithm efficient.

**Lemma 5.14.** *Let $D \leq \Delta_{k+2}$, let $s$ be a source node and let $U$ and $F$ be the following sets of nodes and edges, respectively:*

$$\begin{aligned} U &= \{ v \in V \mid \operatorname{dist}(v, s) \leq D \} \\ F &= \{ (u, v) \in E \mid \operatorname{dist}(u, s) \leq D \text{ or } \operatorname{dist}(v, s) \leq D \} \,. \end{aligned}$$

*Then with a total update time of $\widetilde{O}((|F| + p|U|m^{1/p}) p n^{2/q} \gamma / \epsilon)$ we can maintain a distance estimate $\ell(v; s)$ for every node $v$ that satisfies*

- $\ell(v; s) \geq \operatorname{dist}(s, v)$
- *If $\Delta_k \leq \operatorname{dist}(s, v) \leq D$, then $\ell(v; s) \leq \alpha_k \operatorname{dist}(s, v) + \beta_k$*

*Proof.* Remember that $G'_k$ is the graph containing all edges of $G$ of weight at most $\Delta_{k+2}$ and all edges of $E_k$ with edge weights rounded to multiples of $\varphi_k$. Ideally, we would like to maintain a monotone ES-tree up to depth $\alpha_k D + \beta_k$ on the graph $G'_k$ to get the desired distance estimates. However, for efficiency reasons we will maintain the monotone ES-tree only on $G'_k | U$, the subgraph of $G'_k$ induced by $U$.

Using Dijkstra's algorithm, we can initially compute the set $U$ in time $\widetilde{O}(|F|)$ as we only have to visit edges incident to nodes in $U$. We maintain a monotone ES-tree from $s$ in the graph $G'_k | U$, which is the subgraph of $G'_k$ induced by the nodes in $U$. Since distances in $G$ are non-decreasing under edge deletions we know for every node $v$ not contained in $U$ that $\operatorname{dist}(v, s) > D$ and thus we can set $\ell(v; s) = \infty$ for such a node.

We first consider the case $k \geq 1$. We define $G''_k$ to be the graph with the same nodes and edges as $G'_k$ in which the edge weights of $G'_k$ are scaled down by a factor of $\varphi_k$, i.e., every edge $(u, v)$ in $G''_k$ has weight

$$w''_k(u, v) = \frac{w'_k(u, v)}{\varphi_k}$$

where $w'_k(u, v)$ is the weight of $(u, v)$ in $G'_k$. Maintaining the monotone ES-tree on $G'_k|U$ is equivalent to maintaining it on $G''_k|U$; the levels only differ by a factor of $\varphi_k$. By the definition of $G'_k$, the edge weights of $G''_k|U$ are integer. In $G'_k|U$ we would have to maintain the monotone ES-tree up to depth $\alpha_k D + \beta_k$, whereas in $G''_k|U$ we only have to maintain it up to depth $(\alpha_k D + \beta_k)/\varphi_k$. By Lemma 4.2 the total time needed for maintaining the monotone ES-tree up to depth $(\alpha_k D + \beta_k)/\varphi_k$ on $G''_k|U$ is

$$O(\mathcal{E}(G''_k|U)(\alpha_k D + \beta_k)/\varphi_k + \mathcal{W}(G''_k|U)) \tag{2}$$

where $\mathcal{E}(G''_k|U)$ is the number of edges ever contained in $G''_k|U$ and $\mathcal{W}(G''_k|U)$ is the number of edge weight increases in $G''_k|U$.

We first bound $\mathcal{E}(G''_k|U)$, the number of edges ever contained in $G''_k|U$. Note that $\mathcal{E}(G''_k|U) = \mathcal{E}(G'_k|U)$ by the definition of $G''_k$. Every edge contained in $G'_k|U$ is contained in $G|U$ (the subgraph of $G$ induced by $U$) or in $E_k|U$ (the subset of $E_k$ induced by $U$). We never insert any edges into $G$ and all edges of $G|U$ are contained in $F$ and by Lemma 5.7 we have $\mathcal{E}(E_k|U) = \widetilde{O}(p|U|m^{1/p})$. Therefore we get $\mathcal{E}(G''_k|U) = \mathcal{E}(G'_k|U) = \widetilde{O}(|F| + p|U|m^{1/p})$.

We now bound the depth of the monotone ES-tree. Remember that $\alpha_k \leq 2$. Furthermore $D \leq \Delta_{k+2}$ by the assumption and $\beta_k \leq \Delta_{k+1}$ by Lemma 4.11. Therefore we get

$$\begin{aligned} \frac{\alpha D + \beta_k}{\varphi_k} \leq \frac{2\Delta_{k+2} + \Delta_{k+1}}{\varphi_k} \leq \frac{3\Delta_{k+2}}{\varphi_k} \leq \frac{3\Delta_{k+2}(p+1)}{\epsilon \Delta_k} \\ = \frac{3n^{(k+2)/q}\gamma(p+1)}{\epsilon n^{k/q}\gamma} = \frac{3(p+1)n^{2/q}}{\epsilon} = O(pn^{2/q}/\epsilon)\,. \end{aligned}$$

Finally, we bound $\mathcal{W}(G''_k|U)$, the number of edge weight increases in $G''_k|U$. Note that $\mathcal{W}(G''_k|U) = \mathcal{W}(G'_k|U)$ by the definition of $G''_k$. By the following argument we can show that the maximum edge weight in $G'_k|U$ is at most $\Delta_{k+2} + \varphi_k$:

- The edges from $G$ are included in $G'_k$ only if they have weight at most $\Delta_{k+2}$
- Every edge $(u, v)$ from $E_k$ that is included in $G'_k$ has weight $w_k(u, v) \leq w_k^{(i)}$ for some $0 \leq i \leq p - 1$ or $w_k(u, v) \leq w_k^{(i,j)}$ for some $0 \leq i < j \leq p - 1$ by Lemma 5.1. By Lemma 4.11 we have $\alpha_{k-1} r_k^{(p-1)} + \beta_{k-1} \leq \Delta_{k+1} \leq \Delta_{k+2}$ and by Lemma 4.8 we have $w_k^{(i,j)} \leq r_k^{(j)}$. Therefore we get $w_k^{(i)} = \alpha_{k-1} r_k^{(i)} + \beta_{k-1} \leq \alpha_{k-1} r_k^{(p-1)} + \beta_{k-1} \leq \Delta_{k+2}$ and $w_k^{(i,j)} \leq r_k^{(j)} \leq \Delta_{k+2}$.

Since all edge weights in $G'_k|U$ are multiples of $\varphi_k$, it follows that the number of different edge weights in $G'_k|U$ is at most $\lceil (\Delta_{k+2} + \varphi_k)/\varphi_k \rceil$. We now bound the number of edge weight increases in $G'_k|U$ by that number of different edge weights in $G'_k|U$ times the number of edges ever contained in $G'_k|U$. Thus, we have

$$\begin{aligned}
\mathcal{W}(G''_k|U) = \mathcal{W}(G'_k|U) &\leq \mathcal{E}(G'_k|U) \cdot \left\lceil \frac{\Delta_{k+2} + \varphi_k}{\varphi_k} \right\rceil \\
&= \mathcal{E}(G'_k|U) \cdot \left\lceil \frac{(p+1)\Delta_{k+2}}{\epsilon \Delta_k} + 1 \right\rceil \\
&= \mathcal{E}(G'_k|U) \cdot \left\lceil \frac{(p+1)n^{2/q}}{\epsilon} + 1 \right\rceil \\
&= \widetilde{O}(|F| + p|U|m^{1/p}) \cdot \left\lceil \frac{(p+1)n^{2/q}}{\epsilon} + 1 \right\rceil \\
&= \widetilde{O}((|F| + p|U|m^{1/p})pn^{2/q}/\epsilon) \,.
\end{aligned}$$

Using the three bounds we just showed in Equation (2), we obtain a total update time of $\widetilde{O}((|F| + p|U|m^{1/p})pn^{2/q}/\epsilon)$.

Finally, we consider the case $k = 0$, for which we have $\alpha_0 = 1$ and $\beta_k = 0$. Thus, we have to maintain exact distances. We use an exact ES-tree of depth $D$ on $G|U$ for this purpose. It is well-known that the total time needed for maintaining this ES-tree is $O(E(G|U) \cdot D)$, where $E(G|U)$ is the number of edges contained in $G|U$ before the first deletion. Note that $E(G|U) \subseteq F$ and $D \leq \Delta_2 = n^{2/q}\gamma$. Therefore the time needed for maintaining this tree is $O(|F|n^{2/q}\gamma)$.

Both running times, $\widetilde{O}((|F| + p|U|m^{1/p})pn^{2/q}/\epsilon)$ and $O(|F|n^{2/q}\gamma)$ are dominated by $\widetilde{O}((|F| + p|U|m^{1/p})pn^{2/q}\gamma/\epsilon)$. □

# 6 $(1+\epsilon)$-approximate SSSP and $(O(n^{o(1)}), \epsilon)$-Hop Set

In this section we explain how to use the hierarchy edge sets of of Section 5 to maintain approximate SSSP and hop sets in the decremental setting. We first obtain an algorithm that maintains approximate SSSP up to distance $n\gamma$ for some parameter $\gamma \geq 1$. Afterwards we reduce the general problem to this special case. For hop sets we use the same approach.

**Theorem 6.1.** *Given $0 < \epsilon' \leq 1$, $\gamma \geq 1$, a source node $s$, and a weighted graph $G$ with positive integer edge weights undergoing edge deletions and edge weight increases, there is a data structure that maintains a distance estimate $\delta(v, s)$ that satisfies*

- $\delta(v, s) \geq \operatorname{dist}(v, s)$ *and*
- *if* $\operatorname{dist}(v, s) \leq n\gamma$, *then* $\delta(v, s) \leq (1 + \epsilon) \operatorname{dist}(v, s)$.

*It has constant query time and a total update time of $\widetilde{O}(m^{1+6/p}\gamma/\epsilon')$, where*

$$\frac{1}{p} = \frac{\sqrt{\log\left(\frac{880\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}} \,.$$

*Proof.* Let $s$ denote the source node. The algorithm consists of two parts. First, it maintains the hierarchy of graphs of Lemma 5.1, i.e., for every $1 \leq k \leq q - 2$ we maintain the set of edges $E_k$ as in Section 5.1. Second, it maintains the following estimates of the distance to $s$: For every $0 \leq k \leq q - 2$, we use the data structure of Lemma 5.2 to maintain a distance estimate $\ell_k(v; s)$ for every node $v$ that satisfies

- $\ell_k(v; s) \geq \operatorname{dist}(v, s)$
- If $\Delta_k \leq \operatorname{dist}(v, s) \leq \Delta_{k+2}$, then $\ell_k(v; s) \leq \alpha_k \operatorname{dist}(s, v) + \beta_k$

If $\operatorname{dist}(v, s) \leq \Delta_2$, then $\ell_0(v; s) \leq \operatorname{dist}(s, v)$ since $\alpha_0 = 1$ and $\beta_0 = 0$. If $\Delta_{k+1} \leq \operatorname{dist}(v, s) \leq \Delta_{k+2}$ for some $1 \leq k \leq q - 2$, then $\ell_k(v; s)$ provides a $(1 + \epsilon')$-approximation of $\operatorname{dist}(s, v)$, which follows from the following chain of inequalities:

$$\begin{aligned}
\ell_k(v; s) &\leq \alpha_k \operatorname{dist}(s, v) + \beta_k \\
&\leq \alpha_k \operatorname{dist}(v, s) + \epsilon \Delta_{k+1} \\
&\leq \alpha_k \operatorname{dist}(s, v) + \epsilon \operatorname{dist}(v, s) \\
&= (1 + 4k\epsilon) \operatorname{dist}(s, v) + \epsilon \operatorname{dist}(v, s) \\
&\leq (1 + 4q\epsilon) \operatorname{dist}(s, v) + \epsilon \operatorname{dist}(v, s) \\
&\leq (1 + 4\sqrt{\log n}\epsilon) \operatorname{dist}(s, v) + \epsilon \operatorname{dist}(v, s) \\
&= (1 + (4\sqrt{\log n} + 1)\epsilon) \operatorname{dist}(s, v) \\
&= (1 + \epsilon') \operatorname{dist}(s, v)\,.
\end{aligned}$$

As the distance estimates never underestimate the true distance, we therefore only have to return $\min_{0 \leq k \leq q-2} \ell_k(v; s)$ to obtain a $(1 + \epsilon')$-approximate estimate of $\operatorname{dist}(v, s)$.

We now analyze the running time of the algorithm. Maintaining $E_k$ for some $1 \leq k \leq q - 2$ takes time $\widetilde{O}(kp^3 m^{1+2/p} n^{2/q} \gamma/\epsilon)$ by Lemma 5.1 and thus maintaining all $q$ of them takes time $\widetilde{O}(p^3 q^2 m^{1+2/p} n^{2/q} \gamma/\epsilon)$. Maintaining the distance estimates from $s$ using monotone ES-trees takes time $\widetilde{O}((m + pnm^{1/p}) pn^{2/q} \gamma/\epsilon) = \widetilde{O}(p^2 m^{1+1/p} n^{2/q} \gamma/\epsilon)$ per tree by Lemma 5.2 and thus time $\widetilde{O}(p^2 q m^{1+1/p} n^{2/q} \gamma/\epsilon)$ in total. Note that the time needed for maintaining the hierarchy of graphs dominates the running time. Thus, the total update time is $\widetilde{O}(p^3 q^2 m^{1+2/p} n^{2/q} \gamma/\epsilon)$.

Remember that $q \leq \sqrt{\log n}$, $p \leq \sqrt{\log n}$, $q = p/2$, and $\epsilon = \epsilon'/(1 + 4\sqrt{\log n})$. Therefore we get that the running time is $\widetilde{O}(m^{1+6/p} \gamma/\epsilon')$. Now observe that

$$1/p = \frac{\sqrt{\log\left(\frac{4 \cdot 44}{\epsilon}\right)}}{\sqrt{\log n}} = \frac{\sqrt{\log\left(\frac{4 \cdot 44 \cdot (1 + 4\sqrt{\log n})}{\epsilon'}\right)}}{\sqrt{\log n}} \leq \frac{\sqrt{\log\left(\frac{4 \cdot 44 \cdot 5\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}} = \frac{\sqrt{\log\left(\frac{880\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}}\,.$$

The algorithm described above has a query time of $O(q) = O(\log n)$ as it has to compute $\min_{0 \leq k \leq q-2} \ell_k(v; s)$ when asked for the approximate distance from $v$ to $s$. We can reduce the query time to $O(1)$ by using a min-heap for every node $v$ that stores $\ell_k(v; s)$ for all $0 \leq k \leq q - 2$. This allows us to retrieve the minimum in constant time. □

The data structure above only provides a $(1 + \epsilon)$-approximation for distances up to $n\gamma$. Setting $\gamma = 1$ this immediately gives a $(1 + \epsilon)$-approximate decremental SSSP data structure for unweighted graphs because in unweighted graphs the maximum distance to the root is $n - 1$. Using Bernstein's rounding technique [3], we can also remove the restriction to distance ranges up to $n\gamma$ for weighted graphs as we show in the next theorem.

**Theorem 6.2.** *Given an approximation parameter $0 < \epsilon' \leq 1$ and a weighted graph $G$ with positive integer edge weights undergoing edge deletions and edge weight increases, there is a $(1 + \epsilon')$-approximate SSSP data structure that maintains a distance estimate $\delta(v, s)$ for every node $v$ that satisfies*

$$\operatorname{dist}(v, s) \leq \delta(v, s) \leq (1 + \epsilon') \operatorname{dist}(v, s)\,.$$

*It has constant query time and a total update time of*

$$\widetilde{O}\left(m^{1+6\cdot\frac{\sqrt{\log\left(\frac{2640\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}}}\log W/(\epsilon')^2\right)$$

*where $W$ is the maximum edge weight in $G$. If $\epsilon'$ is a constant, then the total update time is $\widetilde{O}(m^{1+o(1)}\log W)$.*

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

$$\rho_l = \frac{\epsilon' 2^l}{n}\,.$$

Let $\widetilde{G}_l$ 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 $\rho_l$, i.e., every edge $(u, v)$ in $\widetilde{G}_l$ has weight

$$\widetilde{w}_l(u,v) = \left\lceil \frac{w(u,v)}{\rho_l} \right\rceil \cdot \rho_l$$

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

$$w(u,v) \leq \widetilde{w}_l(u,v) \leq w(u,v) + \rho_l$$

for every edge $(u, v)$ of $G$. Furthermore we define $\widehat{G}_l$ to be the graph that has the same nodes and edges as $\widetilde{G}_l$ and in which every edge weight is scaled down by a factor of $\rho_l$, i.e., every edge $(u, v)$ in $\widehat{G}_l$ has weight

$$\widehat{w}_l(u,v) = \frac{\widetilde{w}_l(u,v)}{\rho_l} = \left\lceil \frac{w(u,v)}{\rho_l} \right\rceil\,.$$

The algorithm is as follows: For every $0 \leq l \leq \lfloor \log(nW) \rfloor$ we use the data structure of Theorem 6.1 on the graph $\widehat{G}_l$ with $\gamma = 4/\epsilon'$ to maintain a distance estimate $\delta_l(v, s)$ for every node $s$ that satisfies

- $\delta_l(v,s) \geq \operatorname{dist}_{\widehat{G}_l}(v,s)$ and
- if $\operatorname{dist}_{\widehat{G}_l}(v,s) \leq n\gamma$, then $\delta_l(v,s) \leq (1+\epsilon')\operatorname{dist}_{\widehat{G}_l}(v,s)$.

The distance estimate returned by our data structure is

$$\delta(v,s) = \min_{0 \leq l \leq \lfloor \log nW \rfloor} \rho_l \delta_l(v,s)\,.$$

We now show that there is some $0 \leq l \leq \lfloor \log(nW) \rfloor$ such that $\rho_l \delta_l(v,s) \leq (1+3\epsilon')\operatorname{dist}(v,s)$. As $\delta(v,s)$ is the minimum of all the distance estimates, this implies that $\delta(v,s) \leq (1+3\epsilon')\operatorname{dist}(v,s)$. In particular, we know that there is some $0 \leq l \leq \lfloor \log(nW) \rfloor$ such that $2^l \leq \operatorname{dist}(v,s) \leq 2^{l+1}$ since $W$ is the maximum edge weight and all paths consist of at most $n$ edges. Consider a shortest path $P$ from $v$ to $s$ in $G$ whose weight is equal to $\operatorname{dist}(v,s)$. Let $w(P)$ and $\widetilde{w}_l(P)$ denote the weight of the path $P$ in $G$ and $G_l$, respectively. Since $P$ consists of at most $n$ edges we have $\widetilde{w}_l(P) \leq w(P) + n\rho_l$. Therefore we get

$$\operatorname{dist}_{\widetilde{G}_l}(v,s) \leq \widetilde{w}_l(P) \leq w(P) + n\rho_l = \operatorname{dist}(v,s) + \epsilon' 2^l \leq \operatorname{dist}(v,s) + \epsilon' \operatorname{dist}(v,s) = (1+\epsilon')\operatorname{dist}(v,s)\,.$$

Now observe the following:

$$\operatorname{dist}_{\widehat{G}_l}(v,s) = \frac{\operatorname{dist}_{\widetilde{G}_l}(v,s)}{\rho_l} \leq \frac{(1+\epsilon')\operatorname{dist}(v,s)}{\rho_l} \leq \frac{2\operatorname{dist}(v,s)}{\rho_l} = \frac{2\operatorname{dist}(v,s)n}{\epsilon' 2^l} \leq \frac{2 \cdot 2^{l+1} n}{\epsilon' 2^l} = \frac{4n}{\epsilon'} = n\gamma\,.$$

Since $\operatorname{dist}_{\widehat{G}_l}(v,s) \leq n\gamma$ we get $\operatorname{dist}_l(v,s) \leq (1+\epsilon')\operatorname{dist}_{\widehat{G}_l}(v,s)$ by Theorem 6.1. Thus, we get

$$\rho_l \delta_l(v,s) \leq \rho_l((1+\epsilon')\operatorname{dist}_{\widehat{G}_l}(v,s)) = (1+\epsilon')\operatorname{dist}_{\widetilde{G}_l}(v,s) \leq (1+\epsilon')^2\operatorname{dist}(v,s) \leq (1+3\epsilon')\operatorname{dist}(v,s)$$

as desired.

We now analyze the running time of this algorithm. By Theorem 6.1, for every $0 \leq l \leq \lfloor \log{(nW)} \rfloor$, maintaining $\delta_l(v,s)$ on $\widehat{G}_l$ for every node $s$ takes time $\widetilde{O}(m^{1+6/p}\gamma/\epsilon')$, where

$$\frac{1}{p} = \frac{\sqrt{\log\left(\frac{880\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}}\,.$$

By our choice of $\gamma = 4/\epsilon'$, the total update time for maintaining all these $\lfloor \log{(nW)} \rfloor$ distance estimates is $\widetilde{O}(m^{1+6/p}\log W/(\epsilon')^2)$.

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

$$\widetilde{O}\left(m^{1+6\cdot\frac{\sqrt{\log\left(\frac{2640\sqrt{\log n}}{\epsilon'}\right)}}{\sqrt{\log n}}}\log W/(\epsilon')^2\right)\,.$$

If $\epsilon'$ is a constant, then the total update time is

$$\widetilde{O}\left(m^{1+O\left(\frac{\sqrt{\log\log n}}{\sqrt{\log n}}\right)}\log W\right)$$

Since $\lim_{x\to\infty}(\sqrt{\log\log n})/(\sqrt{\log n}) = 0$, this is $\widetilde{O}(m^{1+o(1)}\log W)$.

The query time of the algorithm described above is $O(\log(nW))$ as it has to compute $\delta(v,s) = \min_{0 \leq l \leq \lfloor \log nW \rfloor} \rho_l \delta_l(v,s)$ when asked for the approximate distance from $v$ to $s$. We can reduce the query time to $O(1)$ by using a min-heap for every node $v$ that stores $\delta_l(v,s)$ for all $0 \leq l \leq \lfloor \log(nW) \rfloor$. This allows us to compute $\delta(v,s)$ in constant time. □

In the proof of Lemma 5.1 we did not explicitly state the guarantees of the hop set maintained by our algorithm. The main reason for this is that we actually had to prove a stronger property, namely that the monotone ES-tree on the shortcut graph using the hop set gives the desired approximation. We now explain how to maintain an $(O(n^{o(1)}), \epsilon')$-hop set, as defined in Section 2.2.

**Corollary 6.3.** *Given a constant $0 < \epsilon' \leq 1$ and a weighted graph $G$ with positive integer edge weights undergoing edge deletions and edge weight increases, we can maintain an $(O(n^{o(1)}), \epsilon')$-hop set of size $O(m^{1+o(1)}\log W)$ in total time $O(m^{1+o(1)}\log W)$.*

*Proof.* Given some $\gamma \geq 1$, the algorithm of Lemma 5.1 provides an $(n^{o(1)}, \epsilon')$-hop set $E^*$ for distances up to $n\gamma$, which can be seen as follows. We define $E^* = \bigcup_{1 \leq k \leq q-2} E_k$. By Lemma 5.1 and the definition of $q$ we know that $E^*$ has size $O(m^{1+o(1)})$. Let $G_k$ be the shortcut graph that contains all edges of $G$ and all edges of $E_k$, and let $G^*$ be the graph that contains all edges of $G$ and all edges of $E^*$.

For every $k$ ($1 \leq k \leq q-2$), the approximation guarantee of the monotone ES-tree in Lemma 5.10 in particular holds when we initialize the tree, i.e., when there is no dynamic behavior at all. Furthermore the approximation guarantee also holds for the graph $G_k$, as in the proof we use the graph $G'_k$ in which we round the edge weights of $G_k$ to the next multiple of some parameter $\varphi_k$, i.e., the edge weights in $G'_k$ are never smaller than the edge weights in $G_k$. The bound on the additive error introduced by rounding to multiples of $\varphi_k$ tells us the number of edges ("hops") that are necessary to provide the approximation guarantee. Thus, for all nodes $u$ and $v$ such that $\Delta_k \leq \operatorname{dist}(u, v) \leq \Delta_{k+2}$, there is a path between $u$ and $v$ in $G_k$ of weight at most $\alpha_k \operatorname{dist}(u, v) + \beta_k$ using at most $(p+1)(\Delta_2+2) = O(n^{o(1)})$ edges ("hops"). If $\operatorname{dist}(u, v) \geq \Delta_{k+1}$, then $\beta_k \leq \epsilon \operatorname{dist}(u, v)$ and by the definition of $\alpha_k$ we have $\alpha_k + \epsilon \leq 1 + \epsilon'$. Thus, for all nodes $u$ and $v$ such that $\Delta_{k+1} \leq \operatorname{dist}(u, v) \leq \Delta_{k+2}$, there is a path between $u$ and $v$ in $G_k$ of weight at most $(1+\epsilon') \operatorname{dist}(u, v)$ using at most $O(n^{o(1)})$ edges. Note that for all nodes $u$ and $v$ such that $\operatorname{dist}(u, v) \leq \Delta_2$ the shortest path between $u$ and $v$ in $G$ has at most $\Delta_2 = n^{o(1)}$ edges as the minimum edge weight in $G$ is 1. This means that $E^*$ is a hop set restricted to distances up to $n\gamma$: For all nodes $u$ and $v$ such that $\operatorname{dist}(u, v) \leq \Delta_q = n\gamma$, there is a path between $u$ and $v$ in $G^*$ of weight at most $(1 + \epsilon') \operatorname{dist}(u, v)$ using at most $O(n^{o(1)})$ edges.

We can now use the same same rounding technique as in the proof of Theorem 6.2. This will provide us, for every $0 \leq l \leq \lfloor \log(nW) \rfloor$, with an $(O(n^{o(1)}), \epsilon')$-hop set $E^*_l$ of size $O(m^{1+o(1)})$ for all distances in the range from $2^l$ to $2^{l+1}$. It follows that $\bigcup_{0 \leq l \leq \lfloor \log(nW) \rfloor} E^*_l$ is an $(O(n^{o(1)}), 3\epsilon')$-hop set of size $O(m^{1+o(1)})$ for *all* distances. Thus, we can maintain an $(O(n^{o(1)}), \epsilon')$-hop set of size $O(m^{1+o(1)} \log W)$ in total time $O(m^{1+o(1)} \log W)$.

□

# 7 Conclusion

In this paper, we show that single-source shortest paths in undirected graphs can be maintained under edge deletions in 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. This problem is very challenging because such a hop set does not exist even in the static setting. Moreover, improving the current $\widetilde{O}(mn^{0.984})$ total update time by [9] for the decremental reachability problem is already very interesting. The second major open problem is to derandomize our algorithm. The major task here is to deterministically maintain our variant of the TZ-emulator, which is the key to maintaining the hop set. In fact, it is also not known whether the algorithm of Roditty and Zwick [21] for decrementally maintaining the original distance oracle of Thorup and Zwick (and the corresponding spanners and emulators) can be derandomized. (Note however that the distance oracle of Thorup and Zwick can be constructed deterministically in the static setting [18].)

# References

[1] A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *FOCS*, page to appear, 2014.

[2] A. Bernstein. Fully dynamic $(2 + \epsilon)$ approximate all-pairs shortest paths with fast query and close to linear update time. In *FOCS*, pages 693–702, 2009.

[3] A. Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. In *STOC*, pages 725–734, 2013.

[4] A. Bernstein and L. Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *SODA*, pages 1355–1365, 2011.

[5] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000. Announced at STOC 1994.

[6] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the ACM*, 28(1):1–4, 1981.

[7] M. Henzinger, S. Krinninger, and D. Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization. In *FOCS*, pages 538–547, 2013.

[8] M. Henzinger, S. Krinninger, and D. Nanongkai. Sublinear-time maintenance of breadth-first spanning tree in partially dynamic networks. In *ICALP*, pages 607–619, 2013.

[9] M. Henzinger, S. Krinninger, and D. Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *STOC*, pages 674–683, 2014.

[10] M. Henzinger, S. Krinninger, and D. Nanongkai. A subquadratic-time algorithm for dynamic single-source shortest paths. In *SODA*, pages 1053–1072, 2014.

[11] M. R. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *FOCS*, pages 664–672, 1995.

[12] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar. Efficient distributed approximation algorithms via probabilistic tree embeddings. *Distributed Computing*, 25(3):189–205, 2012. Announced at PODC 2008.

[13] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *STOC*, pages 81–91, 1999.

[14] J. Łącki. Improved deterministic algorithms for decremental reachability and strongly connected components. *ACM Transactions on Algorithms*, 9(3):27, 2013. Announced at SODA, 2011.

[15] A. Mądry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *STOC*, pages 121–130, 2010.

[16] D. Nanongkai. Distributed approximation algorithms for weighted shortest paths. In *STOC*, 2014.

[17] L. Roditty. Decremental maintenance of strongly connected components. In *SODA*, pages 1143–1150, 2013.

[18] L. Roditty, M. Thorup, and U. Zwick. Deterministic constructions of approximate distance oracles and spanners. In *ICALP*, pages 261–272, 2005.

[19] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008. Announced at FOCS, 2002.

[20] L. Roditty and U. Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. Announced at ESA, 2004.

[21] L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM Journal on Computing*, 41(3):670–683, 2012. Announced at FOCS, 2004.

[22] M. Thorup. Undirected single-source shortest paths with positive integer weights in linear time. *J. ACM*, 46(3):362–394, 1999. Announced at FOCS, 1997.

[23] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):74–92, 2005. Announced at STOC, 2001.

[24] M. Thorup and U. Zwick. Spanners and emulators with sublinear distance errors. In *SODA*, pages 802–809, 2006.

[25] J. D. Ullman and M. Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM Journal on Computing*, 20(1):100–125, 1991.

[26] V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *FOCS*, pages 645–654, 2010.