2014 IEEE Annual Symposium on Foundations of Computer Science

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

Monika Henzinger
University of Vienna
Faculty of Computer Science
Vienna, Austria

Sebastian Krinninger
University of Vienna
Faculty of Computer Science
Vienna, Austria

Danupon Nanongkai
University of Vienna
Faculty of Computer Science
Vienna, Austria

***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, ϵ)-*hop set* introduced by Cohen (JACM 2000) in the PRAM literature. A (d, ϵ)-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.

Full version available at https://eprints.cs.univie.ac.at/4104/.

M. Henzinger and S. Krinninger are supported by 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. This work was partially done while D. Nanongkai was at ICERM, Brown University, USA, and Nanyang Technological University, Singapore 637371

***Keywords*-dynamic graph algorithms; single-source shortest paths;**

I. 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 $\text{dist}_G(s, x)$.

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 α-*approximation* algorithm if, for any distance query DISTANCE(x), it returns $\delta(s, x)$ such that such that $\text{dist}_G(s, x) \leq \delta(s, x) \leq \alpha \, \text{dist}_G(s, x)$. There are two time complexity measures associated with this problem: *query time* denoting the time needed to answer *each* distance query, and *total update time* denoting the time needed to process *all* edge deletions. The running time will be 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 $O(\text{poly} \log n)$ terms. In this paper, we focus on algorithms with small ($O(1)$ or $O(\text{poly} \log n)$) 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 [1], it can even

0272-5428/14 $31.00 © 2014 IEEE
DOI 10.1109/FOCS.2014.24

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 [2] 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 [3]; yet, the ES-tree has resisted many attempts to improve it for decades. Roditty and Zwick [4] 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 [5]. It is thus natural to shift the focus to *approximation algorithms*.

The first improvement was due to Bernstein and Roditty [6] 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 [7] 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 [8] observed that the ES-tree can be easily adapted to directed graphs. King [9] later extended the ES-tree to an $O(mnW)$-time algorithm on directed weighted graphs. The techniques used in recent algorithms of Bernstein [10], [11] and Mądry [12] give a $(1+\epsilon)$-approximate $\widetilde{O}(mn \log W)$-time algorithm on directed weighted graphs. Very recently, we extended our insights from [7] to obtain a $(1+\epsilon)$-approximation algorithm with roughly $\widetilde{O}(mn^{0.986})$ time for decremental approximate SSSP in directed graphs [13], 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 [14], [15], [3]. 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 [16] showed that "deamortizing" our algorithms in [13] 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 [13] 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 [7] 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 [13].

II. Overview of Techniques

A. Previous Approach

To motivate our approach, note that previous algorithms [6], [7] 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 [17] to maintain an emulator of Thorup and Zwick [18], [19] (TZ-emulator thereafter). This gives the running time of $O(n^{2+o(1)})$ if we use the ES-tree [6], 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 [20] called *lazy* ES-tree. (Note that this is an over-simplification. One difficulty faced by previous algorithms [6], [7] 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.

B. 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, ϵ)-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 $\mathrm{dist}^d_{G'}(u,v)$, be the weight of the shortest path between u and v in G' containing at most d edges. Then, E' is a (d,ϵ)-hop set, if for any u and v,

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

We call G' a *(d,ϵ)-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 [21] 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. [22], [23]). 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 [10]. 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,ϵ)-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 [10], [11], [12] 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. [6], [24], [7]). A powerful idea that usually got us around this problem is the *monotone ES-tree* introduced by us in [24]. 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 [10] with the algorithm of Roditty and Zwick [17], 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 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.

C. 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,

$$\mathrm{dist}^R_G(u,v) \leq \mathrm{dist}^d_{G'}(u,v) \leq (1+\epsilon)\,\mathrm{dist}^R_G(u,v). \quad (1)$$

In other words, a (d,ϵ)-hop set is an n-restricted (d,ϵ)-hop set. (We note that the notion of restricted hop set was also defined by Cohen [21] 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 [10], it can be easily shown that the TZ-emulator does not only provide a (d,ϵ)-hop set, but also provide an R-restricted (d,ϵ)-hop set. By adapting the algorithm of Roditty and Zwick [17], 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

[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 III for detail.

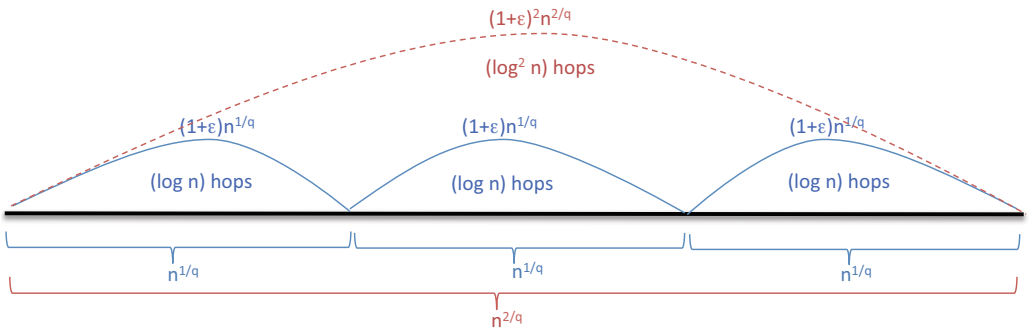

Figure 1. How our multi-layer hop set construction conceptually works: The straight thick line (in black) represents a path in the original graph G_0. Thin curved lines (in blue) represent $(\log^{O(1)} n)$-hop paths in an $n^{1/q}$-restricted $(\widetilde{O}(1), \epsilon)$-hop set of G_0. These edges 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 $\widetilde{O}(1/n^{1/q})$.

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 $d = \log^{O(q)} n$,

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

Figure 1 illustrates the idea of this multi-layer construction. By setting ϵ small enough, i.e., $\epsilon = \omega(1/q)$, the approximation factor $(1+\epsilon)^q$ will be $(1+\epsilon')$ for an arbitrarily small constant ϵ'. 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. We give the definition of this hop set in the full version of our paper.

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 $(1+\epsilon)^k$.

III. 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

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

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 ϵ 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 $\mathrm{dist}_G(u,s) \geq \beta^{(0)}/\epsilon$ there is a path from u to s in H with at most $p\,\mathrm{dist}_G(u,s)/\sqrt{n} \leq p\sqrt{n}$ many hops of approximately the same length as $\mathrm{dist}_G(u,s)$. 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.

A. Hop Set via Thorup-Zwick

The hop set in this simplified version of our algorithm comes from the construction of Thorup and Zwick. In [18], 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 [19], 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 $\mathrm{dist}_G(u, A_i) = \min_{v \in A_i} \mathrm{dist}_G(u,v)$ the distance between the node u and the set of nodes A_i and we set $\mathrm{dist}_G(u, \emptyset) = \infty$. Now, for every node u and every $0 \leq i \leq p-1$, the i-bunch of u is defined as

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

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:

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

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 [6]. 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:

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

Similarly, we define $\overline{Bunch}(u) = \bigcup_{0 \leq i \leq p-1} \overline{Bunch}_i(u)$ and $\overline{Bunch}^D(u) = \{v \in \overline{Bunch}(v) \mid \text{dist}_G(u,v) \leq D\}$. We define the edge (u,v) to be contained in the hop set H if and only if $v \in \overline{Bunch}^D(u)$. The weight of such an edge $(u,v) \in H$ is $w(u,v) = \text{dist}_G(u,v)$. 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 III.1. *The number of edges ever contained in H is $O(n^{1+o(1)})$ in expectation.*

Proof: At any time, the size of $Bunch_i(u)$ is $n^{1/p}$ in expectation [18] for every node u and every $0 \leq i \leq p-1$. As $\overline{Bunch}_i(u) \subseteq Bunch_i(u)$, this also bounds the size of $\overline{Bunch}_i(u)$. 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 $\overline{Bunch}^D(u)$ or, symmetrically, if u joins $\overline{Bunch}^D(v)$. For every node u, nodes can only join $\overline{Bunch}^D(u)$ by joining $\overline{Bunch}_i(u)$ for some $0 \leq i \leq p-2$ when the value $\lfloor \log \text{dist}_G(u, A_{i+1}) \rfloor$ increases. (Note that by the definition of $\overline{Bunch}_{p-1}(u)$ no node will ever join $\overline{Bunch}_{p-1}(u)$.) 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 $\overline{Bunch}_i(u)$, which trivially bounds the number of nodes that might join $\overline{Bunch}_i(u)$ 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 III-C we will show that this goal can be achieved with H. In our proof the following structural property of H will be essential.

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

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 $\text{dist}_G(u,v') \leq r^{(i)}$.

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

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

Thus, in both cases we have shown that there is some node $v' \in A_{i+1}$ such that $\text{dist}_G(u,v') \leq 3r^{(i)}$. 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 $\text{dist}_G(u,v'') \leq 2\,\text{dist}_G(u,v') \leq 6r^{(i)}$. If v'' is contained in $\overline{Bunch}^D(u)$, 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 $\overline{Bunch}^D(u)$ (making the edge (u,v^*) contained in H) such that $\text{dist}_G(u,v^*) \leq 2^{j^*-i-1} 3r^{(i)}$. ∎

Finally, we argue about the running time needed for maintaining the hop set H. Roditty and Zwick [17] gave an algorithm for maintaining $Bunch^D(u)$ 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 $\text{dist}_G(v, A_i)$ for every node v and every $0 \leq i \leq p-1$, which allows us to maintain $\overline{Bunch}^D(u)$ for every node u in the same running time. Thus, we can maintain our hop set in time $\widetilde{O}(mn^{1/p}D) = O(mn^{1/2+o(1)})$.

B. 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 [10]. Here, we deviate from the analysis in [10] 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 [19] 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 $(1+\epsilon)\,\text{dist}_G(u,s) + \beta^{(0)}$ using at most $p\,\text{dist}_G(u,s)/\sqrt{n}$ 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 [19].

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 III.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.

C. Dynamic Hop Reduction

The static analysis we gave above mostly follows the analysis in [19]; 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 [24], [7] 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 [10], [12], [11]. 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 φ 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 ϵ. 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 [24] on H up to distance R with a multiplicative approximation of α and an additive approximation of β 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 φ, scaling them down by dividing by φ 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 III.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 [7] 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 φ. 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 III-B 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 III.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 [24] 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 [7], we get

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

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 III.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 $\operatorname{dist}_G(u,v')\leq 3r^{(i)}$.

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

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 III.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 ϵ 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.

IV. Overview of Multi-Layer Approach

In the following we provide an abstract overview of our multi-layer approach towards a near-linear time algorithm. This algorithm works on weighted graphs and provides a $(1+\epsilon')$-approximation for single-source shortest paths from a node s up to distance $n\gamma$, where γ is a parameter of

[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.

the algorithm. A reduction using a technique of rounding edge weights [10], [12], [11] will make it possible for our algorithm to be used only for very small value of γ (namely $\gamma = 4/\epsilon'$) even when having to answer distance queries for nodes in arbitrary distance to s. The algorithm internally uses many parameters and we will only define some of them in this overview. All details can be found in the full version of the paper.

Our algorithm uses $q-2$ layers (where $q = O(\sqrt{\log n})$ and guarantees that in the k-th layer we can provide (α_k, β_k)-approximate single-source shortest paths for distances in the range from Δ_k to Δ_{k+2}, i.e., with a multiplicative error of α_k and an additive error of β_k. Here we set $\Delta_k = n^{k/q}\gamma$, $\alpha_k = 1+4k\epsilon$, where $\epsilon = \epsilon'/(1+4\sqrt{\log n})$, and guarantee that $\beta_k \leq \epsilon\Delta_{k+1}$. In the range from Δ_{k+1} to Δ_{k+2} we can even provide $(\alpha_k, 0)$-approximate single-source shortest paths without any additive error and where $\alpha_k \leq 1+\epsilon'$. 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 Δ_{k+2}, there is a path P' in G_k that provides an (α_k, β_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 (α_k, β_k)-approximation of all distances in the range from Δ_k to Δ_{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 IV.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 $p-1$ (where $p = O(\sqrt{\log n})$, called the* priority *of v.*
2) *For every node u of priority i, E_k contains either*
   - *for every node v such that $\operatorname{dist}_G(u,v) \leq r_k^{(i)}$ an edge (u,v) of weight $w_k(u,v) \leq w_k^{(i)}$ with $\operatorname{dist}_G(u,v) \leq w_k(u,v) \leq \alpha_{k-1}\operatorname{dist}_G(u,v) + \beta_{k-1}$ 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 $\operatorname{dist}_G(u,v') \leq w_k(u,v') \leq \alpha_{k-1}\operatorname{dist}_G(u,v') + \beta_{k-1}$*

   (for some specific values of $r_k^{(i)}$, $w_k^{(i)}$ and $w_k^{(i,j')}$). For every edge (u,v) contained in E_k we have $\operatorname{dist}_G(u,v) \leq w_k(u,v) \leq \alpha_{k-1}\operatorname{dist}_G(u,v) + \beta_{k-1}$.
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 $\widetilde{O}(kp^3m^{1+2/p}n^{2/q}\gamma/\epsilon)$ in expectation.*

Lemma IV.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:*

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

Given an algorithm for maintaining E_k (as in Lemma IV.1), we can maintain a distance estimate $\ell(v;s)$ for every node v that satisfies

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

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 IV.2 we can also bound the sizes of F and U by m and n, respectively and then obtain a running time of $\widetilde{O}((m + pnm^{1/p})pn^{2/q}\gamma/\epsilon)$ for a single decremental approximate SSSP algorithm. 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 we show in the full version of this paper): Let G_k be the graph that contains all edges of G and E_k. Then for all nodes u and v such that $\Delta_{k+1} \leq \operatorname{dist}_G(u,v) \leq \Delta_{k+2}$ 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 IV.1 and IV.2 in the full version of this paper is to prove them simultaneously by induction on k. The induction base is $k = 0$, for which we can show that Lemma IV.2 holds by using the classic ES-tree [9]. The induction step for $1 \leq k \leq q-2$ works as follows. We will show that Lemma IV.1 holds for k assuming that Lemma IV.2 holds for $k-1$. Similarly, we will show that Lemma IV.2 holds for k assuming that Lemma IV.1 holds for k.

As mentioned before, the hop set of Lemma IV.1 is different from the one based on Thorup-Zwick we use for the slower algorithm in Section III. We are currently not aware of a suitable modification of Thorup-Zwick that does not have access exact distances but only to approximate distances as provided by Lemma IV.2. This is the reason

why our multi-layer approach uses a different hop set that needs a somewhat more complicated analysis than the hop set in Section III.

V. 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 [13] 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 [17] 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 [25].)

References

[1] M. Thorup, "Undirected single-source shortest paths with positive integer weights in linear time," *J. ACM*, vol. 46, no. 3, pp. 362–394, 1999, announced at FOCS, 1997.

[2] S. Even and Y. Shiloach, "An on-line edge-deletion problem," *Journal of the ACM*, vol. 28, no. 1, pp. 1–4, 1981.

[3] L. Roditty, "Decremental maintenance of strongly connected components," in *SODA*, 2013, pp. 1143–1150.

[4] L. Roditty and U. Zwick, "On dynamic shortest paths problems," *Algorithmica*, vol. 61, no. 2, pp. 389–401, 2011, announced at ESA, 2004.

[5] V. Vassilevska Williams and R. Williams, "Subcubic equivalences between path, matrix and triangle problems," in *FOCS*, 2010, pp. 645–654.

[6] A. Bernstein and L. Roditty, "Improved dynamic algorithms for maintaining approximate shortest paths under deletions," in *SODA*, 2011, pp. 1355–1365.

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

[8] M. R. Henzinger and V. King, "Fully dynamic biconnectivity and transitive closure," in *FOCS*, 1995, pp. 664–672.

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

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

[11] ——, "Maintaining shortest paths under deletions in weighted directed graphs," in *STOC*, 2013, pp. 725–734.

[12] A. Mądry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *STOC*, 2010, pp. 121–130.

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

[14] L. Roditty and U. Zwick, "Improved dynamic reachability algorithms for directed graphs," *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1455–1471, 2008, announced at FOCS, 2002.

[15] J. Łącki, "Improved deterministic algorithms for decremental reachability and strongly connected components," *ACM Transactions on Algorithms*, vol. 9, no. 3, p. 27, 2013, announced at SODA, 2011.

[16] A. Abboud and V. Vassilevska Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *FOCS*, 2014, p. to appear.

[17] L. Roditty and U. Zwick, "Dynamic approximate all-pairs shortest paths in undirected graphs," *SIAM Journal on Computing*, vol. 41, no. 3, pp. 670–683, 2012, announced at FOCS, 2004.

[18] M. Thorup and U. Zwick, "Approximate distance oracles," *Journal of the ACM*, vol. 52, no. 1, pp. 74–92, 2005, announced at STOC, 2001.

[19] ——, "Spanners and emulators with sublinear distance errors," in *SODA*, 2006, pp. 802–809.

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

[21] E. Cohen, "Polylog-time and near-linear work approximation scheme for undirected shortest paths," *J. ACM*, vol. 47, no. 1, pp. 132–166, 2000, announced at STOC 1994.

[22] M. Khan, F. Kuhn, D. Malkhi, G. Pandurangan, and K. Talwar, "Efficient distributed approximation algorithms via probabilistic tree embeddings," *Distributed Computing*, vol. 25, no. 3, pp. 189–205, 2012, announced at PODC 2008.

[23] D. Nanongkai, "Distributed approximation algorithms for weighted shortest paths," in *STOC*, 2014.

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

[25] L. Roditty, M. Thorup, and U. Zwick, "Deterministic constructions of approximate distance oracles and spanners," in *ICALP*, 2005, pp. 261–272.