Deterministic Partially Dynamic Single Source Shortest Paths for Sparse Graphs

Aaron Bernstein [*] Shiri Chechik [†]

November 2, 2016

Abstract

In this paper we consider the *decremental* single-source shortest paths (SSSP) problem, where given a graph G and a source node s the goal is to maintain shortest paths between s and all other nodes in G under a sequence of online adversarial edge deletions. (Our algorithm can also be modified to work in the incremental setting, where the graph is initially empty and subject to a sequence of online adversarial edge insertions.)

In their seminal work, Even and Shiloach [JACM 1981] presented an exact solution to the problem with only $O(mn)$ total update time over all edge deletions. Later papers presented conditional lower bounds showing that $O(mn)$ is optimal up to log factors.

In SODA 2011, Bernstein and Roditty showed how to bypass these lower bounds and improve upon the Even and Shiloach $O(mn)$ total update time bound by allowing a $(1 + \epsilon)$ approximation. This triggered a series of new results, culminating in a recent breakthrough of Henzinger, Krinninger and Nanongkai [FOCS 14], who presented a $(1 + \epsilon)$-approximate algorithm whose total update time is near linear: $O(m^{1+O(1/\sqrt{\log n})})$.

However, every single one of these improvements over the Even-Shiloach algorithm was randomized and assumed a non-adaptive adversary. This additional assumption meant that the algorithms were not suitable for certain settings and could not be used as a black box data structure. Very recently Bernstein and Chechik presented in STOC 2016 the first *deterministic* improvement over Even and Shiloach, that did not rely on randomization or assumptions about the adversary: in an undirected unweighted graph the algorithm maintains $(1+\epsilon)$-approximate distances and has total update time $\tilde{O}(n^2)$.

In this paper, we present a new deterministic algorithm for the problem with total update time $\tilde{O}(n^{1.25}\sqrt{m}) = \tilde{O}(mn^{3/4})$: it returns a $(1+\epsilon)$ approximation, and is limited to undirected unweighted graphs. Although this result is still far from matching the randomized near-linear total update time, it presents important progress towards that direction, because unlike the STOC 2016 $\tilde{O}(n^2)$ algorithm it beats the Even and Shiloach $O(mn)$ bound for *all* graphs, not just sufficiently dense ones. In particular, the $\tilde{O}(n^2)$ algorithm relied entirely on a new sparsification technique, and so could not hope to yield an improvement for sparse graphs. We present the first deterministic improvement for sparse graphs by significantly extending some of the ideas from the $\tilde{O}(n^2)$ algorithm and combining them with the hop-set technique used in several earlier dynamic shortest path papers.

Also, because decremental single source shortest paths is often used as a building block for fully dynamic all pairs shortest paths, using our new algorithm as a black box yields new deterministic algorithms for fully dynamic approximate all pairs shortest paths.

1 Introduction

In this paper we study the *dynamic* shortest paths problem, where the goal is to maintain shortest path information in a graph that changes over time. In the most general *fully dynamic* model, an update to the graph can insert or delete an edge, or change an edge weight. This general setting is often very difficult, so many researchers have tried to develop better results in the *partially dynamic* model, which restricts the set of possible updates: the decremental setting allows only edge deletions and edge weight increases (i.e. the graph is deteriorating), whereas the incremental setting allows only edge insertions and weight decreases.

In this paper we consider the problem of (approximate) single source shortest paths (SSSP) in unweighted undirected graphs. Our algorithm can be made to work in both the incremental and decremental settings, but for the rest of the paper we focus on the decremental setting, as it is typically the harder one. Specifically, we start with some original unweighted undirected graph G and a source node s, and the algorithm must process an online intermixed sequence of two different operations: 1) Delete(e) – delete the edge e from the graph. 2) Distance(v) – return the distance between s and v, i.e., $\mathbf{dist}(s, v)$, in the current graph G.

Fully dynamic shortest paths has a very clear motivation, as computing shortest paths in a graph is one of the fundamental problems of graph algorithms, and many shortest path applications must deal with a graph that is changing over time. The incremental setting is somewhat more restricted, but is applicable to any setting in which the network is only expanding. The decremental setting is often very important from a theoretical

[*]Department of Mathematics, Technical University of Berlin, Berlin, Germany. Email: bernstei@gmail.com. This research was supported by the Einstein Grant in Berlin.

[†]Department of Computer Science, Tel-Aviv University, Israel. Email: shiri.chechik@gmail.com. This research was supported by the ISRAEL SCIENCE FOUNDATION (grant No. 1528/15).

perspective, as decremental shortest paths (and decremental *single source* shortest paths especially) are used as a building block in a large variety of fully dynamic shortest paths algorithms; see e.g. [14, 3, 1, 2]. In fact, as we discuss in Section 1.2, using our new decremental single source shortest paths result as a black box immediately yields new results for deterministic *fully dynamic* all pairs shortest paths. Decremental shortest paths can also have applications to non-dynamic graph problems; see e.g. Madry's paper on efficiently computing multicommodity flows [17].

We say that an algorithm has an approximation guarantee of α if its output to the query Distance(v) is never smaller than the actual shortest distance and is not more than α times the shortest distance.

Dynamic algorithms are typically judge by two parameters: the time it takes the algorithm to adapt to an update (the Delete operation), and the time to process a query (the Distance operation). Typically one tries to keep the query time small (polylog or constant), while getting the update time as low as possible. All the algorithms discussed in this paper have constant query time, unless noted otherwise. In the decremental setting, which is the focus of this paper, one usually considers the aggregate sum of update times over the *entire* sequence of deletions, which is referred to as the *total update time*.

1.1 Related work The most naive solution to dynamic SSSP is to simply invoke a static SSSP algorithm after every deletion, which requires $\tilde{O}(m)$ [1] time, using e.g. Dijkstra's algorithm. Since there can be a total of m deletions, the total update time for the naive implementation is thus $\Omega(m^2)$.

For fully dynamic SSSP, nothing better than the trivial $O(m^2)$ total update time is known. That is, we do not know how to do better than reconstructing from scratch after every edge update. For this reason, researches have turned to the decremental (or incremental) case in search of a better solution.

The first improvement stems all the way back to 1981, when Even and Shiloach [8] showed how to achieve total update time $O(mn)$ in unweighted undirected graphs. A similar result was independently found by Dinitz [7]. This was later generalized to directed graphs by King [16]. This $O(mn)$ total update time bound is still the state of art, and there are conditional lower bounds [18, 13] showing that it is in fact optimal up to log factors. (The reductions are to boolean matrix multiplication and the online matrix-vector conjecture respectively).

These lower bounds motivated the study of the approximate version of this problem. In 2011, Bernstein and Roditty [5] presented the first algorithm to beyond the $O(mn)$ bound of Even and Shiloach [8]: they presented a $(1 + \epsilon)$ decremental SSSP algorithm for undirected unweighted graphs with constant query time and $O(n^{2+O(1/\sqrt{\log n})}) = O(n^{2+o(1)})$ total update time. Henzinger, Krinninger and Nanongkai [11] later improved the total update time for to $O(n^{1.8+o(1)} + m^{1+o(1)})$, and soon after the same authors [9] achieved a close to optimal total update time of $O(m^{1+o(1)} \log W)$ in undirected weighted graphs, where W is the largest weight in the graph (assuming the minimum edge weight is 1).

Henzinger, Krinninger, and Nanongkai also showed that one can go beyond $O(mn)$ total update time bound in *directed* graphs (with a $(1+\epsilon)$ approximation) [10, 12], although the state of art is still only a small improvement: total update time $O(mn^{0.9+o(1)} \log W)$.

However, every single one of these improvements over the $O(mn)$ bound relies on randomization, and has to make the additional assumption of a *non-adaptive* adversary. In particular, they all assume that the updates of the adversary are completely independent from the shortest paths or distances returned to the user, i.e. that the updates are fixed in advance. This makes these algorithms unsuitable for many settings, and also prevents us from using them as a black box data structure. For example, if we were to use dynamic shortest paths to route packages in a changing graph, then if edges on the paths used for routing were slightly more likely to deteriorate we would not have independence between updates and queries, and we could not use a non-adaptive algorithm.

Very recently, Bernstein and Chechik [4] presented the first *deterministic* algorithm to go beyond $O(mn)$ total update time: their algorithm achieves total update time $\tilde{O}(n^2)$ in undirected unweighted graphs, again with a necessary $(1 + \epsilon)$ approximation. See Section 1.2 of their paper for a detailed discussion of why it is especially important to develop deterministic algorithms for this problem (in short: to avoid the non-adaptivity assumption), but also of why an entirely new set of techniques seems to be required.

The algorithm of Bernstein and Chechik relies on a new deterministic sparsification technique. For this reason, their $\tilde{O}(n^2)$ total update time only constitutes an improvement over $O(mn)$ in dense graphs: for graphs that are already sparse, their techniques have no effect. In this paper, we set out to develop deterministic techniques that allow us to go beyond the $O(mn)$ bound in sparse graphs.

[1]The $\tilde{O}$ notation suppresses polylogarithmic factors.

1.2 Our Results

Theorem 1.1. *Given an undirected unweighted graph G subject to a sequence of edge deletions, and a fixed source s, there exists a* deterministic *algorithm that maintains $(1+\epsilon)$ approximate distances from s to every vertex in total update time $O(n^{1.25}\sqrt{m}\log(n)\epsilon^{-1.25}) = \tilde{O}(mn^{3/4}\epsilon^{-1.25})$. The query time is $O(1)$.*

We can easily extend our algorithm to work for graphs with small positive integer weights, at the cost of multiplying the update time by $O(W)$. We can also extend our algorithm to work in the *incremental* setting, where we start with an empty graph, and the adversary inserts edges one at a time.

Our result is still far behind the state of the art randomized result ($O(m^{1+o(1)})$ total update time), but it is the first deterministic algorithm to go beyond the classic $O(mn)$ total update time bound in *all* graphs, not just dense ones. In addition to being important as a proof of possibility, we believe that the techniques for sparse graphs developed here could possibly be of use in further deterministic algorithms for this problem.

Decremental SSSP algorithms are often used as a building block in fully dynamic all pairs shortest path algorithms (APSP). In fact, using our algorithm as a black box in a fully dynamic APSP algorithm of Bernstein [3] yields the theorem below. (We simply use our new decremental SSSP algorithm of Theorem 1.1 instead of the standard Even and Shiloach algorithm in the algorithm in Section 3 of [3].)

Theorem 1.2. *There exists a* d*eterministic fully dynamic APSP algorithm with an approximation error of $(2+\epsilon)$ that has update time $\tilde{O}(mn^{3/4}\epsilon^{-1.25})$ and constant query time.*

The previous best deterministic algorithm for the problem (Demetrescu and Italiano, 2004 [6]) achieves $O(n^2)$ update time (per operation), though it is significantly more general: it returns exact distances and works in weighted directed graphs. Our algorithm serves primarily as a proof of possibility, as it is the first deterministic algorithm to achieve update time $o(mn)$ for sparse graphs. The fastest randomized (and non-adaptive) algorithm is that of Bernstein [3], which achieves a $(2+\epsilon)$ approximation with $O(m^{1+o(1)})$ update time. Note that Theorem 1.2 uses our main result in Theorem 1.1 as a black box: we suspect one could significantly improve upon this bound with a more sophisticated application of the new deterministic techniques in this paper.

The next section defines some preliminaries. Then present an overview of our techniques (Section 3).

2 Preliminaries

We consider the decremental setting in which edges are being deleted one by one from an initial graph. We assume the main graph is unweighted and undirected. Let $G = (V, E)$ always refer to the *current* version of the graph. Let m be the number of edges in the original graph, and n the number of vertices.

For any pair of vertices u, v, let $\pi(u, v)$ be the shortest $u - v$ path in G (ties can be broken arbitrarily), and let $\mathbf{dist}(u, v)$ be the length of $\pi(u, v)$. Let s be the fixed source from which our algorithm must maintain approximate distances, and let ϵ refer to our approximation parameter; when the adversary queries the distance to a vertex v, the algorithm must return an approximate distance $\widehat{\mathbf{dist}}(v)$ such that $\mathbf{dist}(s,t) \leq \widehat{\mathbf{dist}}(s,t) \leq (1+O(\epsilon))\mathbf{dist}(s,t)$; if we wanted a strict $(1+\epsilon)$-approximation, we could simply run with algorithm with $\epsilon' = \epsilon/c$ for large enough constant c. Some of our auxiliary graphs will have weights, in which case we set $\omega(u, v)$ to be the weight of edge (u, v).

Let $B(v, r)$ (B for ball) for a vertex v and radius r be the set of vertices at distance at most r from v in G, that is $B(v, r) = \{u \in V \mid \mathbf{dist}(v, u) \leq r\}$. Let $N(v, r)$ (N for neighborhood) for an integer r and a vertex v be the set of nodes at distance exactly r from v, that is, $N(v, r) = \{u \in V \mid \mathbf{dist}(v, u) = r\}$. For a vertex v, let $\deg(v)$ be the degree of v in G. For a set of vertices S, let $\deg(S) = \sum_{v \in S} \deg(v)$.

We will measure the update time of the dynamic subroutines used by our algorithm in terms of their *total* update time over the entire sequence of edge changes. Note that although edges in the main graph G are only being deleted, there may be edge insertions into the auxiliary graphs used by the algorithm.

Definition 2.1. *Given a graph G subject to a sequence of edge deletions and insertions, define* MAX-EDGES(G) *to be the number of pairs (u, v) such that edge (u, v) is in G* at some *point during the update sequence. Note that if the update sequence contains only deletions, then* MAX-EDGES(G) *is simply the number of edges in the original graph.*

2.1 Even and Shiloach

We next state the result of the classic Even-Shiloach algorithm [8]. This algorithm runs faster when it only needs to maintain shortest distances up to some small distance threshold d. The basic idea is that the algorithm spends $O(\deg(v))$ whenever $\mathbf{dist}(s, v)$ changes. This is true for both insertions and deletions, but the algorithm only yields good bounds if the update sequence never decreases distances, in which case every $\mathbf{dist}(s, v)$ can only change d times before it exceeds the distance threshold, so we get the bound in

Lemma 2.1 below:

DEFINITION 2.2. *Given any number d, the function* $\textsc{bound}_d(x)$ *is equal to x if $x \leq d$, and to ∞ otherwise.*

DEFINITION 2.3. *Let G be a dynamic graph subject to a sequence of edge insertions and deletions, let s be a fixed source, and let d be some depth bound. Then, the Even-Shiloach algorithm* $\mathbf{ES}(G, s, d)$ *maintains the value* $\textsc{bound}_d(\mathbf{dist}(s, v))$ *for every vertex v over the entire sequence of changes to G. We refer to this as running an Even and Shiloach tree in G from source s up to depth d.*

LEMMA 2.1. *[8] Let $G = (V, E)$ be a graph with positive integer weights subject to an online sequence of insertions and deletions, let s be a fixed source, and say that for every vertex v we are guaranteed that* $\mathbf{dist}(s, v)$ *never decreases due to an edge insertion. Then, the* total *update time of* $\mathbf{ES}(G, s, d)$ *over the entire sequence of edge updates is $O(m \cdot d + \Delta)$, where $m = \textsc{max-edges}(G)$, and Δ is the total number of updates.*

The classic **ES** algorithm can only handle edge insertions under the assumption that they *never* decrease distances. Henzinger *et al.* developed a modification of this algorithm which can handle occasional distance decreases, which they called the Monotone Even-Shiloach algorithm, denoted **MES**. The basic idea is that **MES** simply ignores distance decreases. More precisely, the classic **ES** algorithm maintains for every vertex v a distance label $\ell(v)$ with the guarantee that we always have $\ell(v) = \mathbf{dist}(s, v)$. **MES**, on the other hand, will only guarantee that $\ell(v) \geq \mathbf{dist}(s, v)$. It does so by running classical **ES**, with one modification: whenever classic **ES** adds an edge (u, v) to the shortest path tree, it sets $\ell^{\text{NEW}}(v) = \min\left\{\ell(u) + \omega(u, v), \ell^{\text{OLD}}(v)\right\}$. **MES**, on the other hand, sets $\ell^{\text{NEW}}(v) = \max\left\{\ell(u) + \omega(u, v), \ell^{\text{OLD}}(v)\right\}$.

DEFINITION 2.4. *Let* **MES***(G,s,d) refer to running monotone Even-Shiloach from a fixed source s up to distance d. In particular, whenever we have $\ell(v) > d$, we remove v from the graph.*

We now turn to analyzing the total update time of **MES**. Note that we do not give an approximation analysis here since distance labels in **MES** are not firmly tethered to the actual shortest distances, there is no general approximation guarantee that would work for an arbitrary sequence of insertions and deletions. Every invocation of **MES** will require a separate approximation error analysis to show that for the particular graph and update sequence at hand, $\ell(v)$ remains a good approximation to $\mathbf{dist}(s, v)$.

2.2 A New Extension of Even and Shiloach

The Even-Shiloach algorithm, whether the monotone or classical version, spends $O(\deg(v))$ time whenever the distance label $\ell(v)$ changes. In particular, if we could guarantee that at all times the graph has maximum degree D, then **ES** and **MES** would only require D time per label increase. In this section, we present a slight generalization of this argument.

DEFINITION 2.5. *Given a dynamic graph $G = (V, E)$, a dynamic assignment $A : E \to V$ assigns each edge (u, v) to one of u or v. A must assign each edge (u, v) the moment the edge is inserted into the graph, and cannot change this assignment. We say than an assignment has maximum load α if at any time during the dynamic update sequence, every vertex v has at most α edges assigned to it. (Dynamic assignments are analogous to the more commonly used notion of a dynamic orientation, except that in our definition once an edge e is inserted, its assignment must remain fixed until e is deleted.)*

LEMMA 2.2. *Let $G = (V, E)$ be a directed graph with positive integer weights subject to a sequence of online insertions and deletions, and say that there is a dynamic assignment A with maximum load α. Then, there is an implementation of* $\mathbf{MES}(G, s, d)$ *with total update time $O(\Delta + nd\alpha)$, where Δ is the total number of edge updates made to G.*

Proof. Conceptually the proof only involves a small modification to the classical **ES** algorithm, but the details are rather technical because in order to make our modification, we have to go over all the details of the classical **ES** algorithm.

Let us first recall the high level of the Even-Shiloach algorithm in a setting with only deletions. The algorithm maintains a shortest path tree T from the root s. We say an edge (u, v) is a tree edge if $(u, v) \in T$, and a non-tree edge if $(u, v) \in E \setminus T$. Whenever a non-tree edge is deleted, shortest distances do not change, so the algorithm does not have to do much. Now, consider the deletion of a tree edge (u, v) where u is the parent of v in T. The algorithm checks in constant time (by storing the edges in a clever way) if v has another edge to a vertex z with $\ell(v) = \ell(z) + \omega(z, v)$. If so, the edge is added to the tree and the distances to all nodes remain the same. If not, the distance $\mathbf{dist}(s, v)$ must increase. The algorithm then examines the children of v and checks if they can be attached to the tree without increasing their distance in a similar manner. The algorithm keeps a list of all vertices such that either their label increased in the current update or the label of their parent increased. Each time the algorithm discovers that the label of a vertex v must increase, it increases it by 1 (which is the most optimistic label it can get) and returns it to the list in at attempt to reattach v with this higher label. The algorithm examines the vertices in the list in an increasing order of their label. Note that the label of a vertex may be increased many times as a result of an update. This means that the algorithm removes and adds it many times to the list. Since $\mathbf{dist}(s, v)$ can increase at most d times for any node v, this algorithm has $O(md)$ total update time. Essentially, to implement both the **ES** and the **MES** algorithms, we need to describe the implementation of two operations. First, an operation that for a vertex v checks if there is an edge that reconnects it to the tree without increasing its distance label. Second, when some $\ell(v)$ increases, we need an operation that returns a list of vertices whose label might have increased as a result of $\ell(v)$ increasing. In classical **ES**, this second operation just returns all vertices w such that (v, w) was a tree edge. In **MES**, however, since distances can decrease but distance labels cannot, it might be the case that even though (v, w) is a tree edge we have $\ell(v) + \omega(v, w) < \omega(w)$, in which case $\ell(v)$ increasing by 1 will not actually affect $\ell(w)$; we would like to be able to detect this case in advance and avoid wasting time looking at $\ell(w)$.

We start by describing the information that our implementation of the algorithm maintains. We will later explain how the algorithm maintains this information. Recall that by the assumption of the Lemma, we maintain a dynamic assignment A with maximum load α (see Definition 2.5).

For every vertex v and every distance $1 \le i \le d$, let $P_i(v)$ (P stands for potential parents) be the set of edges (u, v) such that $\ell(u) + \omega(u, v) = i$ and the edge (u, v) is not assigned to v. In addition, let $C_i(v)$ (C stands for children) be the set of edges (u, v) such that $\ell(u) - \omega(u, v) = i$ and the edge (u, v) is not assigned to v. The algorithm maintains the sets $P_i(v)$ and $C_i(v)$ for every $1 \le i \le d$. In addition, the algorithm maintains a list $P_{small}(v)$ that contains all edges (u, v) such that $\ell(u) + \omega(u, v) \le \ell(v)$. Note that, $P_{small}(v) = \cup_{i \le \ell(v)} P_i(v)$. Similarly, the algorithm maintains a list $C_{small}(v)$ of all *tree* edges (u, v) such that $\ell(u) - \omega(u, v) \le \ell(v)$. Note that, $C_{small}(v) = \cup_{i \le \ell(v)} C_i(v) \cap E(T)$. Finally, the algorithm maintains a set $A(v)$ (A stands for assigned) for every vertex v that is a subset of the edges assigned to v. Intuitively, when v loses the edge to its parent, or when the parent of v increases its label, the set $A(v)$ will contain all the edges assigned to v that could potentially replace the edge from v to its parent.

Next, we explain the implementation of the algorithm along with how to maintain these sets. It is not hard to verify that initially after the algorithm constructs a shortest path tree from s in the original graph, all these sets can be computed in $O(m)$ time.

When a new edge is added distances might decrease, but distance labels cannot change because **MES** never decreases distance labels. Thus, it is not hard to verify that in constant time we can add the newly inserted edge to all the sets it should be part of. Moreover, when a new replacement edge (u, v) is added to the tree, in constant time we can add it to the list $C_{small}(v)$ if $\ell(u) - \omega(u, v) \le \ell(v)$. Similarly, when an edge is deleted the algorithm removes it in $O(1)$ time from all sets containing it. A deletion of a tree edge, however, will also force the algorithm to look for replacement edges, and might also change distance labels. When a vertex v loses its edge to its parent or when the distance label of the parent of v increases, the algorithm does the following. First, the algorithm needs to search if there is an alternative edge to v without increasing its distance label. That is, we are looking for an edge (u, v) such that $\ell(u) + \omega(u, v) \le \ell(v)$. In $O(1)$ time the algorithm checks if there is such an edge that is not assigned to v. This can be done by simply checking if the set $P_{small}(v)$ is not empty, in which case any edge in $P_{small}(v)$ is a good replacement edge and the label of v remains the same and this edge is added to the tree. Otherwise, we need to check if there is such an edge (u, v) such that $\ell(u) + \omega(u, v) \le \ell(v)$ and the edge (u, v) is assigned to v. To do so, the algorithm starts traversing the edges (u, v) in $A(v)$. For each such edge (u, v), the algorithm checks if $\ell(u) + \omega(u, v) \le \ell(v)$; if so the desired edge has been found and this edge is added to the tree. Otherwise, delete the edge (u, v) from $A(v)$ and move to the next edge in $A(v)$. If $A(v)$ becomes empty then the label of v must increase by at least 1.

When the label of v increases by 1 the algorithm does the following:

- It adds to $A(v)$ all edges assigned to v.
- For every edge $(u, v) \in E$ that is assigned to v, the algorithm updates the list $P_i(u)$ where $i = \ell(v) + \omega(u, v)$, and the list $C_j(u)$ where $j = \ell(v) - \omega(u, v)$.
- For every edge (u, v) assigned to v, the algorithm updates the lists $P_{small}(u)$: if $\ell(v) + \omega(u, v) > \ell(u)$ and $P_{small}(u)$ contains the edge (u, v) then remove the edge (u, v) from $P_{small}(u)$.
- For every edge (u, v) assigned to v, the algorithm updates the lists $C_{small}(u)$: if $\ell(v) - \omega(u, v) > \ell(u)$ and $C_{small}(u)$ contains the edge (u, v) then remove the edge (u, v) from $C_{small}(u)$.

- Assume the label $\ell(v)$ increases from i to $i+1$, the algorithm checks if $P_{i+1}(v)$ is not empty and if so it adds $P_{i+1}(v)$ to $P_{small}(v)$.

- Assume the label $\ell(v)$ increases from i to $i+1$, the algorithm checks if $C_{i+1}(v)$ is not empty and if so it add all tree edges in $C_{i+1}(v)$ to $C_{small}(v)$.

- The algorithm for all vertices u such that either the edge (u,v) is assigned to v or the edge $(u,v) \in C_{small}(v)$ continues recursively: i.e., checks for a replacement edge to attach u without changing its label, and if no such edge is found, the algorithm increase the distance label $\ell(u)$. Note that v itself is also added to the list of vertices which need a new tree edge, though the distance label of v has increased by 1, so this time we are searching for an edge that will keep v at distance label $\ell^{\text{OLD}}(v)+1$.

In order to see that the algorithm is a valid implementation of **MES**, first note that the sets $P_i(v)$ and $C_i(v)$ are always updated as each time the label of u increases it updates the sets $P_i(v)$ and $C_i(v)$ for every edge (u,v) owned by u. As the sets $P_i(v)$ and $C_i(v)$ are always up to date, it is not hard to see that $P_{small}(v)$ and $C_{small}(v)$ are also up to date.

Note that every edge that was deleted from $A(v)$ cannot be used as a replacement edge to v; that is, if u is deleted from $A(v)$, then making u the parent of v would increase $\ell(v)$. It follows that the replacement edge of v can only be in the sets $P_{small}(v)$ or $A(v)$. Therefore if there exists a replacement edge to v that does not change the label of v, our implementation will find it. Moreover, note that when the distance label of a node v increases by 1, $\ell(u)$ can only increase for children u such that either $u \in C_{small}(v)$ or the edge (u,v) is assigned to v. It follows that our implementation increase all the required distance labels.

Finally, we bound the total update time of our implementation.

First, note that our algorithm in order to check if v has a replacement edge spends $O(1)+O(j)$ time, where j is the number of edges considered by the algorithm in $A(v)$. Let us first bound the second term for all vertices and for all updates. Note that each edge in $A(v)$ that is considered by the algorithm is also deleted afterward from $A(v)$. Hence, we only need to bound the number of insertions to $A(v)$. Note that an edge is inserted to $A(v)$ only when it is first added to the graph or when the label of v increases and the edge is assigned to v. There are always at most α edges assigned to v and its label can increase at most d times. It follows that the number of insertions to $A(v)$ is at most $O(\alpha d + \Delta(v))$, where $\Delta(v)$ is the number of edge updates made to G for edges incident v. All in all, the total update time for all vertices is $O(\Delta + nd\alpha)$.

To bound the $O(1)$ term for checking if v has a replacement edge, we charge this cost to the parent u of v whose distance label increased (similarly to the analysis of the Even-Shiloach) or to the edge deletion (u,v). The number of edge deletions is at most $O(m + \Delta)$, where m is the number of the edges in the original graph: but clearly $m \leq n\alpha$ because we have an assignment with max load α. We are left with the case where the label of u increases.

Recall that when the label of u increases, the algorithm examines all nodes z such that either the edge (u,z) is assigned to u or the edge $(u,z) \in C_{small}(u)$. There are at most α edges of the first type. As the label of u can increase at most d times, this gives $O(d\alpha)$ total update time charged to vertex u as a result of its child along an assigned edge looking for a replacement edge; therefore, at most $O(d\alpha n)$ total update time for all vertices. Consider now the edges $(u,z) \in C_{small}(u)$. Note that u is the parent z. Note also by straightforward calculations that if $(u,z) \in C_{small}(u)$ then the next time u will be the parent of z, the label of z must increase. We can therefore bound this update time by the following charging argument. Each time the label of z increases it adds a charge of 1 to all sets $C_{small}(u)$ such that (u,z) is assigned to z. As mentioned above, the edge (u,z) can be added at most once to $C_{small}(u)$ before the label of z increases again. In addition, each time a new edge is added to the graph and also to $C_{small}(u)$ the algorithm also give it a credit of 1. In order to bound the update time of this part, we need to bound the amount of credits. It is not hard to verify that this is $O(\alpha nd + \Delta)$ total update time for all vertices.

Let us now turn to bound the update time for a distance label increase. Consider a vertex v such that its distance label increased. Note that a constant number of operations are done to all edges assigned to v. This costs $O(1)$ time for every such edge. There are α such edges. The distance label of v can increase at most d times. This costs $O(d\alpha)$ time for a vertex v for all label increases and $O(nd\alpha)$ total update time for all vertices and all labels increases.

We are left with bounding two additional operations. That is, adding $P_{i+1}(v)$ to $P_{small}(v)$, when the label of v increases from i to $i+1$ and adding all tree edges in $C_{i+1}(v)$ to $C_{small}(v)$. Note that every edge (u,v) assigned to u can be added at most once to $P_{small}(v)$ and $C_{small}(v)$ before the label of u increases. Hence, this can be bounded by $O(\alpha)$ for the vertex u for each time its distance label increases and therefore at most $O(n\alpha d)$ for all vertices and for all label increases.

Corollary 2.1. *Let $G=(V,E)$ be a graph with posi-*

tive weights which are all an integer multiple of some number x, and say that there is a dynamic assignment A with maximum load α. Then, there is an implementation of $\mathbf{MES}(G, s, d)$ *with total update time $O(\Delta + nd\alpha/x)$. (Divide all weights by x and invoke Lemma 2.2.)*

3 Overview of Techniques

In their deterministic $\tilde{O}(n^2)$ total update time algorithm [4], Bernstein and Chechik partitioned the graph into light and heavy vertices according to their degree. Because light vertices had low degree they were easier to work with. The algorithm handled heavy vertices by arguing that any shortest path only contained a small number of heavy vertices, and so they only made a minor contribution the shortest distance and could be effectively ignored. In particular they showed that it is enough to maintain dynamic shortest paths on the light parts of the graph, while only maintaining dynamic connectivity information for the heavy parts.

We also partition the graph into heavy and light vertices, where the light vertices are easier to work with and the heavy vertices can be effectively ignored because there are only so many of them on a shortest path. However, we do not partition vertices according to degree, as this would be of little use in a graph that is already sparse. Instead, we say that a vertex v is *light* if the graph is not dense in the local region of v.

For the sake of intuition, let us say that all vertices in G have constant degree. In this case, both the Even and Shiloach algorithm [8] and the one of Bernstein and Chechik [4] achieve total update time $\tilde{O}(n^2)$. In this overview, we outline an algorithm that achieves total update time $O(n^{11/6})$ for this setting (our full algorithm achieves $O(n^{7/4})$).

We start by observing that short distances are easy to handle: we can run $\mathbf{ES}(G, s, 100n^{5/6}/\epsilon)$ to handle all vertices up to distance $O(n^{5/6})$. So we only need to focus on vertices v for which $\mathbf{dist}(s, v) \gg n^{5/6}$.

Say that a vertex v is light if $B(v, n^{1/4})$ contains at most $\sqrt{n}$ vertices, and heavy otherwise. Using a similar argument to the one in the algorithm of Bernstein and Chechik [4], we can show that any shortest path contains at most $O(n^{3/4})$ heavy vertices. Thus, since we only care about vertices v for which $\mathbf{dist}(s, v) \gg n^{5/6}$, we will be able to effectively ignore the heavy vertices.

Since we can ignore heavy vertices, let us assume for the rest of this section that *all* vertices are light. Unlike in the algorithm of Bernstein and Chechik [4], where an all-light graph immediately gave better bounds because it was guaranteed to be sparse, it is not immediately clear how to take advantage of low-density vertices. To this end, we take of advantage of *hop sets*, a technique used in several other shortest path algorithms.

Observe that for any light vertex v, we can use the **ES** algorithm to maintain $B(v, n^{1/4})$ in total update time $O(n^{3/4})$: since we are in a decremental setting, the ball $B(v, n^{1/4})$ is only shrinking, so v only has to maintain distances up to depth $n^{1/4}$ in a graph with $O(\sqrt{n})$ edges (recall: we are assuming that vertices have constant degree). Thus, we could maintain all light balls in total time $O(n^{7/4})$.

We avoid going into a detailed discussion of hop sets in general, since ours is especially simple. For every light vertex v and every $w \in B(v, n^{1/4})$, we add an edge (v, w) of weight $\mathbf{dist}(v, w)$. Let G^+ be the new graph with additional edges. It is easy to see that distances in G^+ are the same as in G. But notice that in G^+, for any vertex v there is a shortest path from s to v with $O(n^{3/4})$ edges. We now do the following: we take every edge in G^+, including the original edges of G, and we round up their weights to the nearest multiple of $n^{1/6}$. It is not hard to see that distances in this rounded graph G^R are off from distances in G by at most an additive error of $O(n^{5/6})$: for any vertex v, we know that there is a shortest path from s to v in G^+ with at most $O(n^{3/4})$ edges, each of which incurs an additive error of $n^{1/6}$. But since we only care about vertices v with $\mathbf{dist}(s, v) \gg n^{5/6}$, this additive error is subsumed in our $(1 + \epsilon)$ approximation. It thus suffices to maintain distances from s in G^R, which is easy because all weights are a multiple of $n^{1/6}$, so we can scale all distances down by $n^{1/6}$, so $\mathbf{ES}(G^R, s, n)$ runs in time $O(n^2/n^{1/6}) = O(n^{11/6})$ (see Corollary 2.1).

All in all, while the idea of partitioning into light and heavy vertices comes from the paper of Bernstein and Chechik [4], we partition along an entirely different criteria: density rather than degree. This allows us to achieve an improvement in sparse graphs as well. That being said, low density is much harder to take advantage of than low degree, and much of our paper must deal with these extra difficulties. Firstly, our auxiliary graph is a hopset rather than a standard sparsification, so it tends to change a lot as the main graph G changes, and we need additional tools to analyze the resulting running time and approximation error. Secondly, the graph G^+ ends up being much denser than G, so if we actually added an edge from v to every $w \in B(v, n^{1/4})$ we could not get total update time below $\tilde{O}(n^2)$. To resolve this issue, we introduce a simple new technique for reducing the number of edges in a hop set, which we believe might be useful in future applications. In particular, we observe that it is enough to add edges from v to all vertices in $N(v, i)$ for some $r(1-\epsilon) \leq i \leq r$, instead of to the whole ball $B(v, r)$. We can argue by the pigeonhole principle that we can always find an i such that $|N(v,i)|$ is relatively small. We also need additional tools to handle the fact that the appropriate level i might change over time, so we will need to perform many edge insertions and deletions.

Most of the new ideas in this paper are contained in Section 4 (partitioning along density) and Section 6.1 (analyzing the hop set); once we partition into light and heavy vertices, the details of how we separate out the heavy vertices (Section 5) are similar to those of Section 4 in [4].

4 Heavy and Light Vertices

Definition 4.1. *Given a graph G and positive integer thresholds τ_n, τ_m and a radius r, we say that a vertex v in G is (τ_n, τ_m, r)-heavy if at least one of the following is true:*

- $|N(v,1)| > \tau_n/r$ *or*
- $|B(v,r)| > \tau_n$ *or*
- $\deg(B(v,r)) > \tau_m$.

We say that v is (τ_n, τ_m, r)-light otherwise.

Let HEAVY(τ_n, τ_m, r) *be the set of all (τ_n, τ_m, r)-heavy vertices in G; note that when we say that a vertex v is (τ_n, τ_m, r)-heavy or (τ_n, τ_m, r)-light, this is* always *with respect to the main graph G, never with respect to any other graph the algorithm relies on.*

Definition 4.2. *Let* OLB(v, τ_n, τ_m, r) *for a vertex v be the ball $B(v,r)$ at the time when v first becomes (τ_n, τ_m, r)-light (*OLB*stands for original light ball). Note that for a (τ_n, τ_m, r)-light vertex v we always have $B(v,r) \subseteq$ OLB(v, τ_n, τ_m, r), $|$OLB$(v, \tau_n, \tau_m, r)| \le \tau_n$, and* $\deg($OLB$(v, \tau_n, \tau_m, r)) \le \tau_m$.

Proposition 4.1. *Given a (τ_n, τ_m, r)-light node v, one can maintain $B(v,r)$ in total update time $O(\tau_m \cdot r)$.*

Proof. The proof is a straightforward application of the **ES** algorithm. Since $B(v,r)$ is always a subset of OLB(v, τ_n, τ_m, r), vertex v only has to maintain shortest distances up to depth r in the graph induced by OLB(v, τ_n, τ_m, r), which by definition has at most τ_m edges. By Lemma 2.1 the total update time is $O(\tau_m \cdot r)$.

The next lemma shows that we can efficiently detect when a vertex transitions from heavy to light.

Lemma 4.1. *There is an algorithm that runs in $O(n \cdot \tau_m \cdot r)$ total update time that after each deletion returns a list of the vertices that were (τ_n, τ_m, r)-heavy before the update and are (τ_n, τ_m, r)-light after the update (note that because we are in a decremental setting, once a node becomes (τ_n, τ_m, r)-light it will always remain so).*

Proof. The basic idea is very simple: a vertex v cannot afford to maintain $B(v,r)$ if this ball contains more than τ_m edges, so v will simply truncate the ball the moment it reaches τ_m edges. As long as v is forced to truncate the ball at radius less than r, we know that v must be must be heavy. The formal proof is slightly more involved.

The algorithm will maintain for every vertex v the minimal index $i(v)$ such that the sum of the degrees of nodes at distance at most $i(v)$ from v is at least τ_m, that is, $\deg(\{u \in V \mid \mathbf{dist}(u,v) \le i(v)\}) > \tau_m$. In particular, for every vertex v the algorithm constructs an Even-Shiloach shortest path tree $\tilde{T}(v)$ from v one level at a time until it reaches the desired level $i(v) \le r$ for which $\deg(\{u \in V \mid \mathbf{dist}(u,v) \le i(v)\}) > \tau_m$. (If $\deg(B(v,r)) \le \tau_m$ then simply set $i(v) = r$). In addition, for every edge (x,y) store a list of all trees $\tilde{T}(v)$ containing the edge (x,y). When the edge (x,y) is deleted from the graph, the deletion operation is invoked on all trees $\tilde{T}(v)$ containing the edge (x,y) (this is done to avoid spending time on trees not containing the deleted edge). In addition, the algorithm maintains the sum of the degrees in $\tilde{T}(v)$: this can be easily maintained by simply storing the sum and once a vertex leaves $\tilde{T}(v)$ decrease the sum by the degree of that vertex. If at some point $\deg(\tilde{T}(v)) < \tau_m$ and $i(v) < r$ then increase $i(v)$ by one and add to the Even-Shiloach tree $\tilde{T}(v)$ all nodes on the next level. Once $i(v) = r$ and $|B(v,r)| \le \tau_n$ and $\deg(v) \le \tau_n/r$ add v to the list of (τ_n, τ_m, r)-light vertices (note that once $i(v) = r$ the algorithm maintains $B(v,r)$ and therefore checking if $|B(v,r)| \le \tau_n$ can be done in constant time).

We next bound the total update time of this algorithm. Recall from Lemma 2.1 that in the **ES** algorithm the update time comes from two components. Firstly, we pay $O(1)$ per deletion (we call this the *constant component*). Secondly, as a result of deletions, we must also pay $\deg(v)$ every time the label of v changes (we call this the *label increase component*).

We first bound the constant component by showing that at most $O(r \cdot \tau_m)$ edges are ever added to any $\tilde{T}(v)$. This is because at any given time the sum of degrees of all vertices at depth at most $i(v) - 1$ is bounded by τ_m; thus, when $i(v)$ increases by one, only the $O(\tau_m)$ edges incident to vertices at level $i(v) - 1$ or below can become part of the new $\tilde{T}(v)$. Hence, each time $i(v)$ increases, at most $O(\tau_m)$ new edges can be part of $\tilde{T}(v)$. As $i(v)$ can increase at most r times, we get that a total of $O(\tau_m r)$ edges are added to any given tree, and thus at most $O(n \cdot \tau_m \cdot r)$ over all trees.

We now turn to bounding the total update time of the label increase component. Consider a vertex v and $\tilde{T}(v)$. The **ES** algorithm algorithms does $O(\deg(u))$

when the label of a vertex u increases: we will charge this $O(\deg(u))$ work to the level i that was the level of u before the increase. Notice that this can only happen when $i < i(v)$ as otherwise when the algorithm finds that the distance $\mathbf{dist}(s, u)$ is larger than i (and so larger than the threshold $i(v)$ of $\tilde{T}(v)$), it simply removes u from the tree and therefore does not pay the addition $O(\deg(u))$ time. But when $i < i(v)$ we have that the sum of the degrees up to and including level i is at most τ_m. Thus, since each vertex can be charged to level i at most once, it is not hard to see that the total time spent for level i is bounded by $O(\tau_m)$. Summing over all levels we get $O(\tau_m r)$ total update time per tree $\tilde{T}(v)$, leading to $O(n \cdot \tau_m \cdot r)$ total update time for the algorithm as a whole.

The following lemma will allow us to reduce the number of edges in our final hop set.

LEMMA 4.2. *For every (τ_n,τ_m,r)-light vertex v and parameters τ_n,τ_m, and r, there exists an index* $\textsc{rad}(v,\tau_n,\tau_m,r)$ *(for convenience, when the parameters τ_n,τ_m,r are clear from the context we write* $\textsc{rad}(v)$*) which changes as the main graph G changes and which has the following properties:*

- $(1-\epsilon)r \le \textsc{rad}(v) \le r$.
- $\textsc{rad}(v)$ *is monotonically decreasing over time.*
- $|N(v,\textsc{rad}(v))| \le \frac{\tau_n}{\epsilon \cdot r}$.
- *One can maintain* $\textsc{rad}(v)$ *in total update time $O(\tau_m \cdot r)$.*

Proof. we first note that an index i such that $(1-\epsilon)r \le i \le r$ and $|N(v,i)| \le \frac{\tau_n}{\epsilon \cdot r}$ must exists. To see this, recall that as v is (τ_n,τ_m,r)-light then by definition $|B(v,r)| \le \tau_n$. In addition, note that there are ϵr levels in $[(1-\epsilon)r, r]$. By the pigeonhole principle there must be a level i such that $(1-\epsilon)r \le i \le r$ and $N(v,i) \le \frac{\tau_n}{\epsilon \cdot r}$.

Initially, we define $\textsc{rad}(v)$ to be the maximal such index. As long as $|N(v,\textsc{rad}(v))| \le \frac{\tau_n}{\epsilon \cdot r}$ we do not modify $\textsc{rad}(v)$. Once this is not the case we set the new value of $\textsc{rad}(v)$ to be the maximal index i that is smaller than the previous $\textsc{rad}(v)$ and such that $|N(v,i)| \le \frac{\tau_n}{\epsilon \cdot r}$. We need to show that such an index always exists. We claim that at all times

$$|B(v,\textsc{rad}(v))| \le \tau_n - (r - \textsc{rad}(v))\frac{\tau_n}{\epsilon \cdot r}. \tag{4.1}$$

Note that again by the pigeonhole principle, Equation 4.1 shows that the desired index always exists. We prove the equation by induction. To prove the base case, note that $|B(v,r)| \le \tau_n$ (because v is light), and that by the maximality of the initial $\textsc{rad}(v)$ we know that for every $j > \textsc{rad}(v)$ we have $|N(v,j)| > \frac{\tau_n}{\epsilon \cdot r}$. Thus for the initial $\textsc{rad}(v)$ we have

$$\begin{aligned} |B(v,\textsc{rad}(v))| &= |B(v,r)| - \sum_{\textsc{rad}(v)<j\le r} |N(v,j)| \\ &\le \tau_n - (r - \textsc{rad}(v))\frac{\tau_n}{\epsilon \cdot r}. \end{aligned}$$

For a time t let $\textsc{rad}_t(v)$ be the value of $\textsc{rad}(v)$ at time t. Similarly, let $B_t(v,r')$ (resp. $N_t(v,r')$) for radius r' be the set $B(v,r')$ (resp. $N(v,r')$) at time t. To prove the induction step, assume Equation 4.1 is true up until some time $t-1$ and consider time t. If at time t we still have $|N_t(v,\textsc{rad}_{t-1}(v))| \le \frac{\tau_n}{\epsilon \cdot r}$ then $\textsc{rad}(v)$ remains unchanged, so $\textsc{rad}_t(v) = \textsc{rad}_{t-1}(v)$; Equation 4.1 then holds by the induction hypothesis, because the right side of the equation remains the same during time t and $t-1$ ($\textsc{rad}(v)$ does no change), while the left side can only decrease because in we are in a decremental setting, so for any radius r', $|B(v,r')|$ is monotonically decreasing. So assume $|N_t(v,\textsc{rad}_{t-1}(v))| > \frac{\tau_n}{\epsilon \cdot r}$. Because we choose the new $\textsc{rad}_t(v)$ to be the maximal desired index, we have that for all indices $\textsc{rad}_t(v) < j \le \textsc{rad}_{t-1}(v)$ we have $|N_t(v,j)| > \frac{\tau_n}{\epsilon \cdot r}$. Equation 4.1 for $\textsc{rad}_t(v)$ then follows from the induction hypothesis and the fact that $|B_t(v,\textsc{rad}_t(v))| = |B_t(v,\textsc{rad}_{t-1}(v))| - \sum_{\textsc{rad}_t(v)<j\le\textsc{rad}_{t-1}(v)} |N(v,j)|$.

We can maintain $\textsc{rad}(v)$ in total update time $O(\tau_m \cdot r)$ by simply maintaining an **ES**-tree on $B(v,r)$ and using a counter to keep track of $|N_i(v,r)|$ for each $i < r$. (It is not hard to see that one can tweak the **ES** algorithm to do so without increasing the asymptotic bound on the running time). By Proposition 4.1, maintaining $B(v,r)$ requires $O(\tau_m \cdot r)$ total update time.

5 The Threshold Graph

Although the partition into heavy and light vertices is quite different than in the earlier Bernstein and Chechik result [4], once we have this partition we construct the threshold graph in a relatively similar manner, although with different edge weights and some additional edges. Note that for the sake of convenience, we contract heavy components instead of adding a dummy vertex with edges of weight 1/2 as in [4].

We define the graph $G_{\tau_n,\tau_m,r}$ as follows. The set of vertices $V_{\tau_n,\tau_m,r}$ contains all (τ_n,τ_m,r)-light vertices, as well as vertex c for every connected component C in the induced subgraph $G[\textsc{heavy}(\tau_n,\tau_m,r)]$. Note that because of this contraction, $V_{\tau_n,\tau_m,r}$ contains different vertices than V: the following definition allows us to easily switch between the two.

DEFINITION 5.1. *For a vertex $v \in V$, let* $\textsc{comp}(v)$ *be the vertex v itself if v is (τ_n,τ_m,r)-light or if v is heavy* *then* $\textsc{comp}(v)$ *is the component vertex* $c \in V_{\tau_n,\tau_m,r} \setminus V$ *that is the contraction of the connected component containing* v *in* $G[\textsc{heavy}(\tau_n, \tau_m, r)]$. *(The parameters* τ_n, τ_m, *and* r *will always be clear from context).*

The set of edges $E_{\tau_n,\tau_m,r}$ is defined as follows:

- For every (τ_n, τ_m, r)-light vertex v and for all edges (v, w) in the main graph G, add edge (v, w) if w is light and edge $(v, \textsc{comp}(w))$ if w is heavy. Set the weight of these edges to be ϵr.

- For every (τ_n, τ_m, r)-light vertex v and for all $w \in N(v, \textsc{rad}(v))$ add edge (v, w) if w is light and edge $(v, \textsc{comp}(w))$ is w is heavy. Set the weight to these edges to r.

Note that because of component contraction $E_{\tau_n,\tau_m,r}$ may contain parallel edges. For convenience of notation we assume that s is (τ_n, τ_m, r)-light. This is without loss of generality, since we can always add a new vertex s', add a path of length r from s' to s and invoke our algorithm from s'. It is not hard to see that s' is (τ_n, τ_m, r)-light and by subtracting r from the distances from s' we can get the distances from s.

We now prove some properties of $G_{\tau_n,\tau_m,r}$. The proofs of the next two lemmas are simple and analogous to those of Lemmas 4.3 and 4.5 in [4] respectively.

Lemma 5.1. *At any given time, the number of edges in the threshold graph* $G_{\tau_n,\tau_m,r}$ *is* $O(\frac{n\tau_n}{\epsilon r})$.

Proof. Recall that for every (τ_n, τ_m, r)-light vertex v, we add two types of edges to $E_{\tau_n,\tau_m,r}$. The first type is edges incident edges to v in G. By definition of (τ_n, τ_m, r)-light, v has at most $O(\tau_n/r)$ such edges.

The second type are edges between v and all vertices in $N(v, \textsc{rad}(v))$. Recall from Lemma 4.2 that there are at most $O(\frac{\tau_n}{\epsilon r})$ vertices in $N(v, \textsc{rad}(v))$, so every v has at most $O(\frac{\tau_n}{\epsilon r})$ edges of the second type.

There are n vertices, leading to a total of $O(\frac{n\tau_n}{\epsilon r})$ edges.

The previous lemma upper bounds the number of edges in $G_{\tau_n,\tau_m,r}$ at any given time. But note that the total number of edges that are ever added to $G_{\tau_n,\tau_m,r}$ throughout the update sequence may be higher, as edges are deleted and added when $\textsc{rad}(v)$ changes for some (τ_n, τ_m, r)-light vertex v.

Lemma 5.2. *The number of edges ever added to the threshold graph* $G_{\tau_n,\tau_m,r}$ *is* $O(n\tau_n \log n)$.

Proof. Notice that for a (τ_n, τ_m, r)-light vertex v, all edges that we ever add to $G_{\tau_n,\tau_m,r}$ are one of two forms: 1) For edges (v, z) where z is also (τ_n, τ_m, r)-light we always have $z \in \textsc{olb}(v, \tau_n, \tau_m, r)$; 2) For edges (v, c) where c corresponds to a heavy component in $G[\textsc{heavy}(\tau_n, \tau_m, r)]$, we always have that $c = \textsc{comp}(z)$ for some $z \in \textsc{olb}(v, \tau_n, \tau_m, r)$.

The first type of edge is easy to analyze: because $\textsc{rad}(v)$ only decreases, every edge (v, z) is added at most once, leading a total of $|\textsc{olb}(v, \tau_n, \tau_m, r)| = O(\tau_n)$ edges (v, z) that are ever added to $G_{\tau_n,\tau_m,r}$

For the second case, we must analyze how we deal with component vertices c. Consider a heavy component C in $G[\textsc{heavy}(\tau_n, \tau_m, r)]$, and let c be the corresponding vertex in $V_{\tau_n,\tau_m,r}$. Say that at some point C splits into two components C_1 and C_2. Assume w.l.o.g that $|C_1| \leq |C_2|$. In order to update $G_{\tau_n,\tau_m,r}$ as a result of this component split, we do the following: 1. add a new vertex c_1 to $G_{\tau_n,\tau_m,r}$; 2. for every $z \in C_1$ and every (τ_n, τ_m, r)-light vertex v such that $z \in N(v, 1) \cup N(v, \textsc{rad}(v))$, remove one copy of edge (c, v) and add a copy of (c_1, v) instead. Note that all in all, we added and removed $\deg(C_1)$ number of edges. Notice also that because are in a decremental setting components are only splitting apart, and each time a vertex z is part of the smaller component C_1 in a component split, we know that its component has shrunk by a factor of at least two. Therefore, this may happen at most $O(\log n)$ times for a (τ_n, τ_m, r)-heavy node z, and therefore every $z \in \textsc{olb}(v, \tau_n, \tau_m, r)$ leads to at most $O(\log(n))$ edges (v, c) (where $c = \textsc{comp}(z)$) being added to $G_{\tau_n,\tau_m,r}$.

Summing over all n possibilities for v we get a total of $O(n\tau_n \log n)$ edge insertions into $G_{\tau_n,\tau_m,r}$.

The next crucial lemma shows that distances in $G_{\tau_n,\tau_m,r}$ are close to distances in G for large distances.

Lemma 5.3. *For any positive parameters* τ_n, τ_m, r, *and any pair of vertices* $s, t \in V$:

$$\mathbf{dist}(s, t) < \mathbf{dist}_{\tau_n,\tau_m,r}(s, \textsc{comp}(t)) + \frac{10rn}{\tau_n} + \frac{5rm}{\tau_m}$$

.

Proof. The proof is analogous to the proof of Lemma 4.4 from [4], though somewhat more complicated. Recall from Definition 4.1 that a vertex v can be (τ_n, τ_m, r)-heavy vertex for three reasons. We say that it is (τ_n, τ_m, r)-degree-heavy if $|N(v, 1)| > \tau_n/r$, (τ_n, τ_m, r)-vertex-heavy if $|B(v, r)| > \tau_n$ and (τ_n, τ_m, r)-edge-heavy if $\deg(B(v, r)) > \tau_m$ (note that a vertex may belong to more than one heaviness type).

We start by giving intuition for the full proof. Note that edge weights in $G_{\tau_n,\tau_m,r}$ are actually higher than those in G, so the only reason distances in $G_{\tau_n,\tau_m,r}$ might be shorter is because we contract heavy components. The basic idea of the proof is to show that contracting heavy components does not significantly reduce shortest distances in $G_{\tau_n,\tau_m,r}$ because we can upper bound the number of heavy vertices on any shortest path $\pi(s,t)$. 1) There are at most $3rn/\tau_n$ degree-heavy vertices on $\pi(s,t)$. To see this, consider any two degree-heavy vertices v, w at distance at least 3 on $\pi(s,t)$. By definition, $|B(v,1)| \geq \tau_n/r$ and $|B(w,1)| \geq \tau_n/r$. But we also know that $B(v,1)$ and $B(w,1)$ are disjoint because otherwise there would be a $v-w$ path of length 2. Thus, there are are most $\frac{n}{\tau_n/r} = \frac{rn}{\tau_n}$ degree- heavy vertices at distance 3 on $\pi_{s,t}$, so $3rn/\tau_n$ degree-heavy vertices in total. 2) Similarly, $\pi(s,t)$ contains at most $3rn/\tau_n$ vertex-heavy vertices. Consider any two vertex-heavy vertices v, w at distance at least $3r$ on $\pi(s,t)$. The bound follows from the fact that $|B(v,r)| \geq \tau_n$ and $|B(w,r)| \geq \tau_n$, but $B(v,r)$ and $B(w,r)$ must be disjoint. 3) By the same argument, there are at most $3rm/\tau_m$ edge-heavy vertices on $\pi(s,t)$.

We now turn to the formal proof. It is a bit more involved because we have to be wary of the possibility that there is some *near*-shortest path with a lot of heavy vertices. Consider the shortest $s-t$ path $\pi(s,t) \in G$. Let $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$ be the shortest path between s and $\text{COMP}(t)$ in $G_{\tau_n,\tau_m,r}$. Let $L_{\pi_{\tau_n,\tau_m,r}}$ be the set of (τ_n,τ_m,r)-light vertices $v \in V \bigcap \pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$. Now, let $V^* \subseteq V$ be the set of vertices containing

- All the vertices in $L_{\pi_{\tau_n,\tau_m,r}}$.
- All the (τ_n,τ_m,r)-heavy vertices in G.
- All vertices $z \in N(v,1)$ for every (τ_n,τ_m,r)-degree-heavy vertex v
- All vertices $z \in B(v,r)$ for every (τ_n,τ_m,r)-vertex-heavy or (τ_n,τ_m,r)-edge-heavy vertex v.

Let G^* be the subgraph of G induced by V^*. We first show that there must exist an $s-t$ path in G^*. We construct this path by looking at $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$. $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$ contains edges between (τ_n,τ_m,r)-light vertices in G^*, as well as edges between a (τ_n,τ_m,r)-light vertex in G^* and a contracted heavy component vertex $c \in V_{\tau_n,\tau_m,r} \setminus V$. The edges between two light vertices exist in G^*, so we can follow them directly in $\pi^*(s,t)$. For every node on $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$ that is a contracted heavy component c, let v' and w' be the two (τ_n,τ_m,r)-light neighbors of c on $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$. (Technical note: if $c = \text{COMP}(t)$ then there is no w' to consider.) There must exist some $v \in C$ that is a neighbor of v' in G and some $w \in C$ that is a neighbor of w' in G. Note that since v and w belong to the same heavy connected component in G, there is a $v-w$ path in G using only heavy vertices, so that path is in G^* as well. We have thus exhibited an $s-t$ path in G^*.

Now, let $\pi^*(s,t)$ be the *shortest* $s-t$ path in G^*. Let $\textbf{dist}^*(s,t)$ be the length of $\pi^*(s,t)$. Since G^* is a subgraph of G, we know that $\textbf{dist}(s,t) \leq \textbf{dist}^*(s,t)$. We now show that

(5.2)
$$\textbf{dist}^*(s,t) < \textbf{dist}_{\tau_n,\tau_m,r}(s, \text{COMP}(t)) + \frac{10rn}{\tau_n} + \frac{5rm}{\tau_m}$$

which completes the proof of Lemma 5.3. To prove Equation 5.2, Let X^* contain all vertices in $\pi^*(s,t)$ that are NOT in $L_{\pi_{\tau_n,\tau_m,r}}$: observe that by definition of V^*, every vertex $z \in X^*$ is either (τ_n,τ_m,r)-heavy, or belongs to $B(v,r)$ for some (τ_n,τ_m,r)-heavy vertex v. Let $B^*(v,r)$ and $N^*(v,r)$ be defined analogously to $B(v,r)$ and $N(v,r)$ for G^*.

Let X_1^* contain all vertices v in $\pi^*(s,t)$ such that v is (τ_n,τ_m,r)-degree-heavy or $v \in N(w,1)$ for some (τ_n,τ_m,r)-degree-heavy vertex w. Note that for every vertex $x \in X_1^*$, we have $|B^*(x,2)| \geq \tau_n/r$.

Let X_2^* contain all vertices v in $\pi^*(s,t)$ such that v is either (τ_n,τ_m,r)-vertex-heavy, or $v \in B(w,r)$ for some (τ_n,τ_m,r)-vertex-heavy vertex w. Note that for every vertex $x \in X_2^*$, we have $|B^*(x,2r)| \geq \tau_n$.

Let X_3^* contain all vertices v in $\pi^*(s,t)$ such that v is either (τ_n,τ_m,r)-edge-heavy, or $v \in B(w,r)$ for some (τ_n,τ_m,r)-edge-heavy vertex w. Note that for every vertex $x \in X_3^*$, we have $\deg(B^*(x,2r)) \geq \tau_m$.

Note that $X^* = X_1^* \cup X_2^* \cup X_3^*$. Note also that $\textbf{dist}^*(s,t) \leq |L_{\pi_{\tau_n,\tau_m,r}}| + |X^*|$, while $\textbf{dist}_{\tau_n,\tau_m,r}(s, \text{COMP}(t)) \geq |L_{\pi_{\tau_n,\tau_m,r}}|$ because all the vertices in $L_{\pi_{\tau_n,\tau_m,r}}$ are on $\pi_{\tau_n,\tau_m,r}(s, \text{COMP}(t))$. Thus, to prove Inequality 5.2, it suffices to show that

(5.3)
$$|X_1^*| < \frac{5rn}{\tau_n} \textit{ and } |X_2^*| < \frac{5rn}{\tau_n} \textit{ and } |X_3^*| < \frac{5rm}{\tau_m}.$$

We start with the first inequality. We define the set Y_1^* to contain every 5th vertex in X_1^*, ordered according to their distance from s: that is, Y_1^* contains the vertex in X_1^* that is closest to s on $\pi^*(s,t)$, the vertex that is 6th closest to s, and so on. Recall that for every vertex $y \in Y_1^* \subset X_1^*$ we have $B^*(y,2) > \tau_n/r$. On the other hand, for any two vertices y and y' in Y_1^*, we must have that $B^*(y,2)$ and $B^*(y',2)$ are disjoint, because otherwise there would be a path of length 4 between y and y' in G^*, which contradicts our only adding every fifth vertex to Y_1^*. Thus, $|Y_1^*| < \frac{n}{\tau_n/r} = \frac{rn}{\tau_n}$, which yields the desired inequality $|X_1^*| \leq 5|Y_1^*| < \frac{5rn}{\tau_n}$.

Similarly, let Y_2^* contain every $(4r+1)$th vertex in X_2^*: that is, if we order the vertices in X_2^* according to their distance from s in $\pi^*(s,t)$, then Y_2^* contains the vertices of rank $1, 4r+2, 8r+3$ and so on. Recall that for every $y \in Y_2^* \subseteq X_2^*$ we have that $|B^*(y, 2r)| \geq \tau_n$. On the other hand, for every two vertices y and y' in Y_2^* we have that $B^*(y, 2r)$ and $B^*(y', 2r)$ are disjoint, because otherwise there would be a path of length $4r$ between y and y' in G^*, which contradicts our only adding every $(4r+1)$th vertex to Y_2^*. Thus, $|X_2^*| < 5r|Y_2^*| < \frac{5rn}{\tau_n}$.

An analogous proof also yields $|X_3^*| < \frac{5mr}{\tau_m}$.

Maintaining the threshold graph: We now show how to maintain $G_{\tau_n,\tau_m,r}$ as the main graph changes. The details are analogous to those in Lemma 4.5 of [4].

LEMMA 5.4. *Given a graph G subject to a sequence of edge deletions, and a positive integers threshold τ_n, τ_m, r, we can maintain the graph $G_{\tau_n,\tau_m,r}$ in total time $O(m \log^2(n) + n\tau_n r \log n + n\tau_m r)$.*

Proof. By Lemma 4.1, we can keep track of which vertices are (τ_n, τ_m, r)-heavy and which are (τ_n, τ_m, r)-light in $O(n \cdot \tau_m \cdot r)$ total update time.

By Proposition 4.1, maintaining for every (τ_n, τ_m, r)-light vertex the ball $B(v, r)$can be done in total update time $O(n \cdot \tau_m \cdot r)$ (for all vertices).

By Lemma 4.2 we can maintain $\text{RAD}(v)$ for a (τ_n, τ_m, r)-light vertex v in total update time $O(\tau_m \cdot r)$ and therefore $O(n \cdot \tau_m \cdot r)$ for all vertices together.

Finally, the algorithm must also maintain the connected components in $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ and the set of their incident edges in G. First off, note that $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ is simply a subgraph of G and therefore is easy to maintain in $O(m)$ total update time. Maintaining the connected components in $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ can be done by using a dynamic connectivity data structure (CDS) on the graph $G[\text{HEAVY}(\tau_n, \tau_m, r)]$. We use the CDS of Holm *et al.*[15] for this purpose, which is based on top trees. This CDS can process insertions and deletions into the graph with amortized update time of $O(\log^2(n))$. It is not hard to check that the top trees used by their algorithm can be augmented to support more than just basic connectivity queries. In particular, their CDS can answer the following queries:

- **connected(u,v):** determines whether u and v are in the same connected component in the current graph. The query time is $O(\log(n))$.

- **size(v)**: returns the size of the connected component of v. The query time is $O(\log(n))$.

- **component(v)**: Returns a list of all the vertices in the same connected component as v. The query time is $O(\log(n)$ + number of vertices returned).

We maintain the above CDS on $G[\text{HEAVY}(\tau_n, \tau_m, r)]$. Notice that $G[\text{HEAVY}(\tau_n, \tau_m, r)]$, like G, is decremental, i.e. it is only subject to edge deletions. When the adversary deletes an edge (u, v) in G where both u and v are (τ_n, τ_m, r)-heavy, we delete this this edge from $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ as well, and this deletion is processed by the CDS in time $O(\log^2(n))$. Similarly, when a vertex $v \in V$ transitions from (τ_n, τ_m, r)-heavy to (τ_n, τ_m, r)-light, we delete all its incident edges from $G[\text{HEAVY}(\tau_n, \tau_m, r)]$, and process each deletion by the CDS. Each edge is deleted from $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ at most once, so the total update time of the CDS is $O(m \log^2(n))$.

In addition, when a component in $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ breaks into two connected components, the algorithm needs to add another vertex c' to $G_{\tau_n,\tau_m,r}$ to represent the new heavy connected component. In addition the relevant incident edges to c' need to be added to $G_{\tau_n,\tau_m,r}$ and be removed from the vertex c of the previous connected component.

Whenever an edge $(u, v) \in G[\text{HEAVY}(\tau_n, \tau_m, r)]$ is deleted, we first query the CDS in $O(\log(n))$ time to check whether u and v are still part of the same connected component in $G[\text{HEAVY}(\tau_n, \tau_m, r)]$; if yes, the components of $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ do not change, and we are done. Otherwise, the deletion of (u, v) has caused the component to split into two. We now query CDS.size(u) and CDS.size(v) to determine in $O(\log(n))$ time which of the two parts is smaller. say, w.l.o.g, that CDS.size(v) $\leq$ CDS.size(u). Let C' be the original component that contained both u and v before the deletion of (u, v) and let c' be the vertex that represents C' in $G_{\tau_n,\tau_m,r}$. Let C_v be the component containing v after the deletion. We can use CDS.component(v) to find all the vertices in C_v in time $O(\log(n) + |C_v|)$. After the deletion, we add a new component vertex c_v to $G_{\tau_n,\tau_m,r}$, and for every $w \in C_v$ and every edge $(w, z) \in E(G)$ such that z is (τ_n, τ_m, r)-light and $w \in N(z, 1) \cup N(z, \text{RAD}(z))$, we remove the edge (c, z) and add the edge (c_v, z) instead (if there were multiple copies of (c, z) then remove only one of these copies).

This takes time $O(\deg(C_v))$ and makes $O(\deg(C_v))$ edge changes to $G[\text{HEAVY}(\tau_n, \tau_m, r)]$. Amortized over all edge deletions in $G_{\tau_n,\tau_m,r}$ we have $\sum \deg(C_v) \leq n\tau_n \log(n)$ because edges in $G[\text{HEAVY}(\tau_n, \tau_m, r)]$ are only being deleted, so components are only splitting apart, and each time a vertex w is part of the smaller component C_v in a component split, we know that its component has shrunk by a factor of at least two. (The factor $n\tau_n$ comes from the fact that for every (τ_n, τ_m, r)-light vertex z, every edge $(c, z) \in E_{\tau_n,\tau_m,r}$ corresponds to an edge (w, z) where w is in $\text{OLB}(z, \tau_n, \tau_m, r)$, which has size at most $O(\tau_n)$.)

6 Maintaining distances on the threshold graph

Our algorithm simply maintains $\mathbf{MES}(G_{\tau_n,\tau_m,r}, s, d)$ for some depth parameter d defined later. For a vertex $v \in V(G_{\tau_n,\tau_m,r})$, let $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \leq d$ be the distance label in monotone ES given to v. Note that when a vertex v transitions from heavy to light, a new vertex is added to $G_{\tau_n,\tau_m,r}$ ($\text{COMP}(v)$ is replaced by v). Since **MES** cannot technically handle vertex insertions, we allow $G_{\tau_n,\tau_m,r}$ to include a light copy of each vertex v, even if v is heavy. But as long as v is heavy, the light copy of v will contain no incident edges and so it will be completely disconnected from the graph and will not be touched by the **MES** algorithm. When vertex v transitions to a light vertex, we will then add all the edges in $N(v,1)$ and $N(v,\text{RAD}(v))$ to the light copy of v, and determine the distance label of v as with regular edge insertions.

6.1 Analysis of monotone ES

We start by bounding the approximation ratio of the estimated distances. We assume that all labels $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \leq d$, since otherwise v is simply removed from the graph.

In the next lemma we show that $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) \leq \epsilon r \cdot \mathbf{dist}(s,v)$. Note that even though this seems like a huge stretch, we are only going to use it for small distances. In particular if $\mathbf{dist}(s,v) \leq r$ then the lemma implies that $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \leq \epsilon r^2$.

LEMMA 6.1. *For every vertex v we have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) \leq \epsilon r \cdot \mathbf{dist}(s,v)$.*

Proof. The proof relies on the simple observation that if we consider the shortest path $\pi(s,v) \in G$, then there is a corresponding path from s to $\text{COMP}(v)$ in $G_{\tau_n,\tau_m,r}$ such that this path in $G_{\tau_n,\tau_m,r}$ contains the same or fewer edges than $\pi(s,v)$ (fewer because heavy components are contracted in $G_{\tau_n,\tau_m,r}$), and such that each edge in the new path has weight ϵr.

We prove by induction that after every update the lemma holds. For the base case, we show that initially we have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) \leq \epsilon r \cdot \mathbf{dist}(s,v)$. Consider $\pi(s,v) \in E$ and notice that every edge in $\pi(s,v)$ that is incident to a (τ_n,τ_m,r)-light vertex also exists in $G_{\tau_n,\tau_m,r}$ but with weight ϵr rather than weight of 1. To deal with edges between heavy vertices, consider any maximal subpath $\pi(x,y)$ of $\pi(s,v)$ such that all vertices on $\pi(x,y)$ are (τ_n,τ_m,r)-heavy. (Maximal in that neither the vertex before x nor the vertex after y on $\pi(s,v)$ are also heavy.) Note that all the vertices in $\pi(x,y)$ belong to the same connected component in $G[\text{HEAVY}(\tau_n,\tau_m,r)]$. Let c be the vertex in $G_{\tau_n,\tau_m,r}$ that represents this connected component. Note that in $G_{\tau_n,\tau_m,r}$ we can simply replace the path $\pi(x,y)$ by the single vertex c. Consider the vertex x' that is the neighbor of x in $\pi(s,v)$ that is closer to s, that is, the neighbor of x that is not in $\pi(x,y)$. Note that by the maximality of $\pi(x,y)$, x' is (τ_n,τ_m,r)-light and the edge $(x',x) \in E$ of weight 1 is replaced by the edge $(x',c) \in E_{\tau_n,\tau_m,r}$ with weight ϵr. Similarly, if $c \neq \text{COMP}(v)$ then let y' be the neighbor of y (closet to v) in $\pi(s,v)$. Then there is an edge $(c,y') \in E_{\tau_n,\tau_m,r}$ of weight ϵr. To summarize, every edge in $\pi(s,v)$ that is incident to a (τ_n,τ_m,r)-light vertex is replaced with a matching edge but with weight ϵr rather 1 and all other edges are contracted. Thus for the initial graph, we have that $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) \leq \epsilon r \cdot \mathbf{dist}(s,v)$.

We now turn to the induction step. *First induction hypothesis:* assume the lemma holds until the current update and consider the current update. We prove that the lemma still holds after the current update by using a second induction on $\mathbf{dist}(s,v)$. For $\mathbf{dist}(s,v) = 0$, that is $v = s$, this is clearly the case. *Second induction hypothesis:* assume the lemma holds for every vertex u such that $\mathbf{dist}(s,u) < \ell$ and consider a vertex v such that $\mathbf{dist}(s,v) = \ell$. Let $\widehat{\mathbf{dist}^{\mathbf{OLD}}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v))$ be the previous label of $\text{COMP}(v)$. Let v' be the closest vertex to v in $\pi(s,v)$ such that v is (τ_n,τ_m,r)-light. Note that the edge $(v',\text{COMP}(v))$ exists in $G_{\tau_n,\tau_m,r}$ with weight ϵr. We therefore have

$$\widehat{\mathbf{dist}_{\tau_n,\tau_m,r}}(s,\text{COMP}(v)) \leq \max\left\{\widehat{\mathbf{dist}^{\mathbf{OLD}}_{\tau_n,\tau_m,r}}(s,\text{COMP}(v)), \epsilon r + \widehat{\mathbf{dist}_{\tau_n,\tau_m,r}}(s,v')\right\}.$$

By the second induction hypothesis we have

$$\begin{aligned} \epsilon r + \widehat{\mathbf{dist}_{\tau_n,\tau_m,r}}(s,v') &\leq \epsilon r + \epsilon r \mathbf{dist}(s,v') \\ &\leq \epsilon r + \epsilon r(\mathbf{dist}(s,v) - 1) \\ &= \epsilon r \mathbf{dist}(s,v). \end{aligned}$$

In addition, by the first induction hypothesis we have $\widehat{\mathbf{dist}^{\mathbf{OLD}}_{\tau_n,\tau_m,r}}(s,\text{COMP}(v)) \leq \epsilon r \mathbf{dist}(s,v)$. This completes the proof of the lemma.

LEMMA 6.2. *For every (τ_n,τ_m,r)-light vertex v, we have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) = \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \leq (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2$. For every (τ_n,τ_m,r)-heavy vertex v we have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\text{COMP}(v)) \leq (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2 + \epsilon r$.*

Proof. Consider a vertex $v \in V$ and consider $\pi(s,v)$. If $\mathbf{dist}(s,v) \leq r$ then the claim follows by Lemma 6.1. So we only need to show the claim for v such that $\mathbf{dist}(s,v) > r$.

We first prove that the claim holds in the original graph. We prove it by induction on $\mathbf{dist}(s,v)$. For the base case, that is $s=v$, the claim trivially holds. Assume the claim holds for every $u \in V$ such that $\mathbf{dist}(s,u) < \ell$ and consider v such that $\mathbf{dist}(s,v) = \ell$. Recall that we can assume that $\mathbf{dist}(s,v) > r$. If v is (τ_n,τ_m,r)-heavy then let v' be the closest (τ_n,τ_m,r)-light vertex to v in $\pi(s,v)$. Note that $\mathbf{dist}(s,v') \le \mathbf{dist}(s,v)$ and that the graph $G_{\tau_n,\tau_m,r}$ contains an edge $(v', \textsc{comp}(v))$ of weight ϵr. In addition, by the induction hypothesis, we have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v') \le (1+3\epsilon)\mathbf{dist}(s,v') + \epsilon r^2$. We therefore have:

$$\begin{aligned}
&\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v)) \\
&\qquad \le \epsilon r + \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v') \\
&\qquad \le \epsilon r + (1+3\epsilon)\mathbf{dist}(s,v') + \epsilon r^2 \\
&\qquad \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2 + \epsilon r
\end{aligned}$$

as required. If v is (τ_n,τ_m,r)-light then let v' be the vertex in $\pi(s,v)$ at distance $\textsc{rad}(v)$ from v (see Lemma 4.2); such a vertex is guaranteed to exist because $\mathbf{dist}(s,v) > r$, and by definition of $\textsc{rad}(v)$ we have that $\mathbf{dist}(v',v) \le r(1-\epsilon)$. Note that v' might be heavy. By construction, $G_{\tau_n,\tau_m,r}$ contains the edge $(v, \textsc{comp}(v'))$ with weight r. We thus have

$$\begin{aligned}
& \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \\
\le\ & r + \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v')) \\
\le\ & r + (1+3\epsilon)\mathbf{dist}(s,v') + \epsilon r^2 + \epsilon r \\
\le\ & r + (1+3\epsilon)(\mathbf{dist}(s,v) - (1-\epsilon)r) + \epsilon r^2 + \epsilon r \\
\le\ & \\
& (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2 + \\
& \qquad [r + \epsilon r - r(1+3\epsilon)(1-\epsilon)] \\
\le\ & (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2,
\end{aligned}$$

where the second inequality follows by induction hypothesis and the last is true for a small enough ϵ.

First induction hypothesis: assume now that the lemma is true until the last update and consider the current update. We use a second induction on $\mathbf{dist}(s,v)$. For the base case where $s=v$ the claim is trivial. *Second induction hypothesis:* assume the claim is correct for all vertices u such that $\mathbf{dist}(s,u) < \ell$ and consider a vertex v such that $\mathbf{dist}(s,v) = \ell$. Recall that if $\mathbf{dist}(s,v) \le r$ the correctness follows by Lemma 6.1. So assume $\mathbf{dist}(s,v) > r$. If v is (τ_n,τ_m,r)-light then there is some vertex $v' \in \pi(s,v)$ at distance $\textsc{rad}(v)$ from v. Note that $\mathbf{dist}(s,v') \le \mathbf{dist}(s,v)$, that $\mathbf{dist}(v',v) \ge (1-\epsilon)r$, and that by construction $G_{\tau_n,\tau_m,r}$ contains the edge $(v,\textsc{comp}(v'))$ with weight r. Let $\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,v)$ be the previous distance label of v. We have

$$\begin{aligned}
& \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \le \\
& \max\{\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,v), \\
& \quad \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v')) + r\}
\end{aligned} \tag{6.4}$$

By the first induction hypothesis we assume

$$\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,v) \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2.$$

By the second induction hypothesis we have: (note the extra additive error because v' might be heavy)

$$\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v')) \le (1+3\epsilon)\mathbf{dist}(s,v') + \epsilon r^2 + \epsilon r.$$

Using the same analysis as in the base case for the first induction hypothesis, we can show that

$$\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v')) + r \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2.$$

It follows that $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2$, as required. Note that the exact same argument holds if v transitioned from heavy to light in the current update, as in this case the current update inserts all the edges that are associated with new light vertex v (namely, all the edge in $N(v,1)$ and $N(v,\textsc{rad}(v))$, so in particular there will still be a v' on $\pi(s,v)$ at distance $\textsc{rad}(v)$ from v, and an edge $(\textsc{comp}(v'),v) \in E_{\tau_n,\tau_m,r}$. Also, if v transitioned from heavy to light, then the light copy of v was previously disconnected from the graph, so this will be the first time v receives a distance label, so there is no $\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,v)$ and we simply have $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \le \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v')) + r$.

Consider now the case where v is (τ_n,τ_m,r)-heavy. Let w be the (τ_n,τ_m,r)-light vertex closest to v on $\pi(s,v)$. Let $\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v))$ be the previous distance label of $\textsc{comp}(v)$. We know that

$$\begin{aligned}
& \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v)) \le \\
& \max\left\{\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v)), \epsilon r + \widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,w)\right\}.
\end{aligned}$$

By the first induction hypothesis we have

$$\widehat{\mathbf{dist}}^{\mathrm{OLD}}_{\tau_n,\tau_m,r}(s,\textsc{comp}(v)) \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2 + \epsilon r.$$

By the second induction hypothesis:

$$\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,w) \le (1+3\epsilon)\mathbf{dist}(s,w) + \epsilon r^2.$$

Thus we have that $\widehat{\mathbf{dist}}_{\tau_n,\tau_m,r}(s,v) \le (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r^2 + \epsilon r$, as required.

COROLLARY 6.1. *For every vertex $v \in V$ if $(1 + 2\epsilon)\mathbf{dist}(s, v) \leq d - r^2 - \epsilon r$ then $\mathbf{MES}(G_{\tau_n,\tau_m,r}, s, d)$ contains* COMP(v) *(i.e.* COMP(v) *is not removed for having label greater than d).*

Finally, we bound the total update time of the algorithm.

LEMMA 6.3. *The total update time of $\mathbf{MES}(G_{\tau_n,\tau_m,r}, s, d)$ is $O(\frac{dn\tau_n}{\epsilon r^2} + n\tau_n \log n)$.*

Proof. We show a dynamic assignment from the set of edges $E_{\tau_n,\tau_m,r}$ to $V_{\tau_n,\tau_m,r}$ with maximum load $O(\tau_n/r)$ (see Definition 2.5). For every (τ_n, τ_m, r)-light vertex v assign the set of edges $\{(v, z) \mid z \in N(v, \text{RAD}(v)) \cup N(v, 1)\}$ to v. (If an edge ends up assigned to both of its endpoints, choose one arbitrarily). By definition of light vertices and RAD(v), the assignment load of each vertex is always $O(\tau_n/r)$. By Lemma 5.2 the number of edges ever added to the threshold graph $G_{\tau_n,\tau_m,r}$ is $O(n\tau_n \log n)$. We maintain the **MES** up to depth d and recall that the weights in $G_{\tau_n,\tau_m,r}$ are multiples of ϵr. Applying Corollary 2.1 then concludes the lemma.

7 Putting it all together

We now turn to proving Theorem 1.1. Let $d_{min} = \frac{n^{1.25}}{\epsilon^{1.25}\sqrt{m}}$. To handle distances $\mathbf{dist}(s, v) \leq d_{min}$, we will run $\mathbf{ES}(G, s, d_{min})$, which by Lemma 2.1 requires total update time $O(md_{min}) = O(n^{1.25}\sqrt{m}/\epsilon^{1.25})$. To handle distances larger than d_{min}, we will run **MES** on $O(\log(n))$ different threshold graphs $G_{\tau_n,\tau_m,r}$, each corresponding to a possible distance interval for $\mathbf{dist}(s, v)$. We will then argue that for any possible value of $\mathbf{dist}(s, v) > d_{min}$, one of these threshold graphs yields a good approximation to $\mathbf{dist}(s, v)$.

More formally, Let $\mathcal{I} = [1, \lceil \log(n)/\log d_{min} \rceil]$. For every integer $i \in \mathcal{I}$, let $d_i = 2^i \cdot d_{min}$, $r_i = \frac{d_i^{1/3} \cdot n^{1/3}}{\epsilon^{1/3} \cdot m^{1/3}}$, $r_{min} = \frac{d_{min}^{1/3} \cdot n^{1/3}}{\epsilon^{1/3} \cdot m^{1/3}}$, $\tau_n^i = \frac{r_i \cdot n}{\epsilon \cdot d_i}$ and $\tau_m^i = \frac{r_i \cdot m}{\epsilon \cdot d_i}$.

Our algorithm maintains for every integer $i \in \mathcal{I}$, the graph $G_{\tau_n^i,\tau_m^i,r_i}$ and runs $\mathbf{MES}(G_{\tau_n^i,\tau_m^i,r_i}, s, d_i)$. In addition, the algorithm runs $\mathbf{ES}(G, s, d_{min})$.

For every vertex v let

$$\widehat{\mathbf{dist}}_i(v) = \widehat{\mathbf{dist}}_{\tau_n^i,\tau_m^i,r_i}(s, \text{COMP}(v)) \qquad \text{and let}$$

$$\widehat{\mathbf{dist}}(v) = \min\{\text{BOUND}_{d_{min}}(\mathbf{dist}(s, v)), \min_i\{\widehat{\mathbf{dist}}_i(v) + \frac{10r_i n}{\tau_n^i} + \frac{5r_i m}{\tau_m^i}\}\}.$$

When the adversary queries the distance to a vertex v, our algorithm returns $\widehat{\mathbf{dist}}(v)$. In short, the total update time follows from Lemma 5.4 (maintaining the various $G_{\tau_n^i,\tau_m^i,r_i}$) and Lemma 6.3 (running **MES** in the threshold graphs); the fact that $\mathbf{dist}(s, v) \leq \widehat{\mathbf{dist}}(v)$ follows from Lemma 5.3; the fact that $\widehat{\mathbf{dist}}(v) \leq (1 + O(\epsilon))\mathbf{dist}(s, v)$ follows from Lemma 6.2.

We now analyze the algorithm more formally. The execution of the algorithm is simple. By Lemmas 5.4 and 6.3 for any i we can maintain $\widehat{\mathbf{dist}}_i(v)$ for all vertices v in total update time $O(m \log^2(n) + n\tau_n^i r_i \log n + n\tau_m^i r_i + \frac{d^i n\tau_n^i}{\epsilon r^2} + n\tau_n^i \log n) = O(m \log^2(n) + \frac{n^2 r_i{}^2 \log n}{\epsilon d_i} + \frac{n \cdot m r_i{}^2}{\epsilon d_i} + \frac{n^2}{\epsilon^2 r_i})$. Plugging in r_i, we get the total update time is:

$$O(m \log^2(n) + \frac{n^{5/3} m^{1/3}}{\epsilon^{5/3} d_i^{1/3}} + \frac{n^{8/3} \log n}{\epsilon^{5/3} m^{2/3} d_i^{1/3}}).$$

Plugging in $d_i = d_{min} 2^i$, we get total update time of:

$$O(m \log^2(n) + \frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}(2^i)^{1/3}} + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4} (2^i)^{1/3}})$$
$$= O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}(2^i)^{1/3}} + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4} (2^i)^{1/3}}).$$

Summing over all i's we get that the total update time for all indices is

$$\sum_i O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}(2^i)^{1/3}} + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4} (2^i)^{1/3}})$$
$$= \frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}} + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4}} \cdot \sum_i O(1/(2^i)^{1/3})$$
$$= O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}} + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4}}).$$

In addition, the total update time for maintaining Even-Shiloach $\mathbf{ES}(G, s, d_{min})$ is $O(m \cdot d_{min}) = O(m \cdot d_{min}) = O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}})$.

To maintain all the $\widehat{\mathbf{dist}}(v)$, for each vertex v we create a min-heap HEAP$_v$ containing $\widehat{\mathbf{dist}}_i(v)$ for every i. The algorithm can access any $\widehat{\mathbf{dist}}(v)$ in $O(1)$ time by looking at the minimum of the heap, thus leading to an $O(1)$ query time. Maintaining the heaps is easy: each $\widehat{\mathbf{dist}}_i(v)$ can change at most $O(\frac{d_{min} 2^i}{r_i \epsilon})$ times (because all edge weights in the corresponding threshold graph are multiples of ϵr_i), so there will be at most $\sum_i O(\frac{d_{min} 2^i}{r_i \epsilon}) = \frac{n}{\epsilon r_{min}}$ changes to each HEAP$_v$, and since each heap contain $O(\log n)$ elements, a change requires $O(\log \log n)$ time to process. Maintaining HEAP$_v$ for all v thus requires total update time only $O(\frac{n^2 \log \log n}{\epsilon r_{min}}) = O(\frac{n^{5/4} m^{1/2} \log \log n}{\epsilon^{1/4}})$.

All in all, the total update time of our algorithm is $O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}} \log \log n + \frac{n^{9/4} \log n}{m^{1/2} \epsilon^{5/4}}) = O(\frac{n^{5/4} m^{1/2}}{\epsilon^{5/4}} \log n)$.

We now turn to proving that for any vertex v, $\mathbf{dist}(s,v) \le \widehat{\mathbf{dist}}(s,v) \le (1+O(\epsilon))\mathbf{dist}(s,v)$. The fact that $\mathbf{dist}(s,v) \le \widehat{\mathbf{dist}}(s,v)$ follows directly from Lemma 5.3, because for every i

$$\mathbf{dist}(s,v) \le \mathbf{dist}_{\tau_i}(s,v) + \frac{10n}{r\tau_n^i} + + \frac{5m}{r\tau_m^i}.$$

We now turn to the second direction. If $\mathbf{dist}(s,v) \le d_{min}$ then $\mathbf{ES}(G,s,d_{min})$ returns the correct value for $\mathbf{dist}(s,v)$, so clearly $\widehat{\mathbf{dist}}(s,v) \le \mathbf{dist}(s,v)$. So assume from now that $\mathbf{dist}(s,v) > d_{min}$.

To prove that $\widehat{\mathbf{dist}}(s,v) \le (1+O(\epsilon))\mathbf{dist}(s,v)$, we need to show that for *some* i, we have $\widehat{\mathbf{dist}}_i(s,v) + \frac{10r_in}{\tau_n} + \frac{5r_im}{\tau_m} \le (1+O(\epsilon))\mathbf{dist}(s,v)$.

Let k be the index such that $d_{min}2^{k-2} \le \mathbf{dist}(s,v) \le d_{min}2^{k-1}$.

Straightforward calculations show that:

$$\frac{10r_in}{\tau_n^i} + \frac{5r_im}{\tau_m^i} = 15\epsilon d_i. \tag{7.5}$$

Straightforward calculations also show that $r_i^2 < d_i$. Note that $\mathbf{dist}(s,v) \le d_{min}2^{k-1}$, so we have that $(1+2\epsilon)\mathbf{dist}(s,v) + \epsilon r_i^2 + \epsilon r_i \le d_{min}2^k$ for a small enough ϵ. Hence, by Corollary 6.1 $\mathbf{MES}(G_{\tau_n^k,\tau_m^k,r_k}, s, d_k)$ contains $\textsc{comp}(v)$, i.e. $\textsc{comp}(v)$ is not removed due to having label greater than d. In addition by Lemma 6.2 we have

$$\begin{aligned}
& \widehat{\mathbf{dist}}_i(s,v) + \frac{10r_in}{\tau_n} + \frac{5r_im}{\tau_m} \\
\le\ & \widehat{\mathbf{dist}}_{\tau_n^k,\tau_m^k,r_k}(s,\textsc{comp}(v)) + \frac{10r_kn}{\tau_n} + \frac{5r_km}{\tau_m} \\
\le\ & (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon r_k^2 + \epsilon r_k + 15\epsilon d_k \\
\le\ & (1+3\epsilon)\mathbf{dist}(s,v) + \epsilon d_k + \epsilon d_k + 15\epsilon d_k \\
\le\ & (1+O(\epsilon))\mathbf{dist}(s,v).
\end{aligned}$$

8 Conclusions

There are two main open problems for deterministic partially dynamic SSSP. The first is whether we can improve upon our $\tilde{O}(n^{1.25}\sqrt{m}) = \tilde{O}(mn^{3/4})$ total update time. Is it possible to match the randomized state of the art of $O(m^{1+o(1)})$? We believe that perfecting our techniques (especially reducing the number of edges in the threshold graph) could potentially lead to total update time $\tilde{O}(m\sqrt{n})$, but that going beyond this would require a new set of techniques. The second question is whether there are deterministic algorithms with $o(mn)$ total update time for weighted and/or directed graphs (such results are not known even for dense graphs).

References

[1] Ittai Abraham, Shiri Chechik, and Cyril Gavoille. Fully dynamic approximate distance oracles for planar graphs via forbidden-set distance labels. In *Proceedings of the Forty-fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 1199–1218, 2012.

[2] Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the $o(n)$ barrier. In *International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 1–16, 2014.

[3] Aaron Bernstein. Fully dynamic (2 + epsilon) approximate all-pairs shortest paths with fast query and close to linear update time. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS, pages 693–702, 2009.

[4] Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the o(mn) bound. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing (STOC)*, pages 389–397, 2016.

[5] Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1355–1365, 2011.

[6] Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004.

[7] Yefim Dinitz. Dinitz' algorithm: The original version and even's version. In *Theoretical Computer Science, Essays in Memory of Shimon Even*, pages 218–240, 2006.

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

[9] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. In *Proceedings of the 55th Annual Symposium on Foundations of Computer Science*, FOCS, pages 146–155, 2014.

[10] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Proceedings of the 46th Annual ACM Symposium on Theory of Computing (STOC)*, pages 674–683, 2014.

[11] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA, pages 1053–1072, 2014.

[12] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved algorithms for decremental single-source reachability on directed graphs. In *Proceedings of the 42nd International Colloquium, ICALP*, pages 725–736, 2015.

[13] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 21–30, 2015.

[14] Monika Rauch Henzinger and Valerie King. Maintaining minimum spanning forests in dynamic graphs. *SIAM J. Comput.*, 31(2):364–374, 2001.

[15] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[16] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS, pages 81–91, 1999.

[17] Aleksander Madry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 121–130, 2010.

[18] Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011.