Near-Optimal Decremental SSSP in Dense Weighted Digraphs

Aaron Bernstein
Department of Computer Science
Rutgers University New Brunswick
New Brunswick, US
bernstei@gmail.com

Maximilian Probst Gutenberg
Department of Computer Science
University of Copenhagen and BARC
Copenhagen, Denmark
maximilian.probst@outlook.com

Christian Wulff-Nilsen
Department of Computer Science
University of Copenhagen and BARC
Copenhagen, Denmark
koolooz@di.ku.dk

***Abstract*—In the decremental Single-Source Shortest Path problem (SSSP), we are given a weighted directed graph $G = (V, E, w)$ undergoing edge deletions and a source vertex $r \in V$; let $n = |V|, m = |E|$ and W be the aspect ratio of the graph. The goal is to obtain a data structure that maintains shortest paths from r to all vertices in V and can answer distance queries in $O(1)$ time, as well as return the corresponding path P in $O(|P|)$ time.**

This problem was first considered by Even and Shiloach [JACM'81], who provided an algorithm with total update time $O(mn)$ for unweighted undirected graphs; this was later extended to directed weighted graphs [FOCS'95, STOC'99]. There are conditional lower bounds showing that $O(mn)$ is in fact near-optimal [ESA'04, FOCS'14, STOC'15, STOC'20]. In a breakthrough result, Forster et al. showed that total update time $\min\{m^{7/6}n^{2/3+o(1)}, m^{3/4}n^{5/4+o(1)}\}\textbf{polylog}(W) = mn^{0.9+o(1)}\textbf{polylog}(W)$ is possible if the algorithm is allowed to return $(1+\epsilon)$-approximate paths, instead of exact ones [STOC'14, ICALP'15]. No further progress was made until Probst Gutenberg and Wulff-Nilsen [SODA'20] provided a new approach for the problem, which yields total time $\tilde{O}(\min\{m^{2/3}n^{4/3}\log W, (mn)^{7/8}\log W\}) = \tilde{O}(\min\{n^{8/3}\log W, mn^{3/4}\log W\})$.

Our result builds on this recent approach, but overcomes its limitations by introducing a significantly more powerful abstraction, as well as a different core subroutine. Our new framework yields a decremental $(1+\epsilon)$-approximate SSSP data structure with total update time $\tilde{O}(n^2\log^4 W/\epsilon)$. Our algorithm is thus *near-optimal* for dense graphs with polynomial edge-weights. Our framework can also be applied to sparse graphs to obtain total update time $\tilde{O}(mn^{2/3}\log^3 W/\epsilon)$. Combined, these data structures dominate all previous results. Like all previous $o(mn)$ algorithms that can return a path (not just a distance estimate), our result is randomized and assumes an oblivious adversary.

Our framework effectively allows us to reduce SSSP in general graphs to the same problem in directed acyclic graphs (DAGs). We believe that our framework has significant potential to influence future work on directed SSSP, both in the dynamic model and in others.

***Keywords*-dynamic algorithm, shortest paths, single-source shortest paths, generalized topological order**

I. Introduction

In the Single-Source Shortest Paths (SSSP) problem, the input is a directed weighted graph $G = (V, E, w)$ and a dedicated source vertex $r \in V$, and the goal is to compute shortest paths from r to every other vertex v in V. Let $n = |V|$, $m = |E|$, and W be the aspect ratio of the graph, which is the ratio of maximum to minimum edge weight. The problem can be solved in $O(m + n\log(n))$ time using Dijkstra's algorithm. In this article, we study the dynamic version of the problem, where the graph changes over time. The most general model is the *fully dynamic* one, where the graph is subject to a sequence of edge insertions and deletions. Unfortunately, there are extremely strong conditional lower bound for this model [1], [2], [3], [4].

For this reason, much of the research on this problem has focused on the *decremental* setting. Formally, the algorithm is given a graph $G = (V, E, w)$ subject to a sequence of edge deletions and edge weight increases, and the goal is is to maintain shortest distances/paths from r to every $v \in V$. Although decremental SSSP only applies to a more restricted model, it is an extremely common subroutine in other dynamic algorithms (including fully dynamic ones), and has recently been used to make progress on long-standing *static* problems such as computing max-flow [5], [6], multi-commodity flow [7], expanders [6], and sparse cuts [6]. For this reason, decremental SSSP is one of the most well-studied problems in dynamic algorithms [8], [9], [10], [11], [12], [13], [14], [15], [16], [5], [6], [17], [18], [19].

The first algorithm for decremental SSSP was the Even and Shiloach tree [8], which dates back to 1981 and has total update time $O(mn)$ over the entire sequence of deletions; it was later extended to directed weighted graphs [20], [21]. $O(mn)$ is conditionally optimal for the exact version [1], [2], [3], [4], but one can do better with a $(1+\epsilon)$-approximation. In fact, recent research culminated in a near-optimal algorithm for *undirected* graphs [10] with total update time $m^{1+o(1)}\log(W)$[1]; there has also been more recent work improving upon $O(mn)$ with adaptive or even deterministic algorithms (see e.g. [14], [15], [16], [5], [6], [18], [22]).

In directed graphs, however, the decremental SSSP problem remains poorly understood. The first algo-

[1]In the decremental setting m is taken to be the number of edges in the *initial* graph G

2575-8454/20/$31.00 ©2020 IEEE
DOI 10.1109/FOCS46700.2020.00107

rithms to improve upon the classic $O(mn)$ bound (with a $(1+\epsilon)$-approximation) were by Henzinger, Forster and Nanongkai [11] in 2014; the total update time is $\min\{m^{7/6}n^{2/3+o(1)}, m^{3/4}n^{5/4+o(1)}\}\text{polylog}(W) = mn^{0.9+o(1)}\text{polylog}(W)$ [12]. Since then, the only progress on the problem is a very recent algorithm of Probst Gutenberg and Wulff-Nilsen [17] with total update time $\tilde{O}(\min\{m^{2/3}n^{4/3}, mn^{3/4}\}\log(W))$, or slightly better in unweighted graphs. Very recently, it was further shown in [19] that a total update time of $n^{2+2/3+o(1)}$ can even be obtained deterministically. Besides this recent progress, state-of-the-art algorithms for directed graphs still lag far behind the ones for undirected ones, and only achieve small improvements beyond $O(mn)$.

Related Work: We also point out that the simpler problem of Single-Source Reachability where the data structure only has to report whether there exists a path from the fixed vertex s to some vertex $v \in V$ has been solved to near-optimality in [23] which improved on the breakthrough results in [24], [25], [11], [12], [26]. The algorithm in [23] further even allows to maintain the Strongly-Connected Components of G. A recent result by Bernstein et al. [19] further attempts to derandomize the algorithm in [23] and deterministically achieves total update time $mn^{2/3+o(1)}$ to maintain Strongly-Connected Components, which constitutes the first deterministic improvement over the long-standing $O(mn)$ bound.

We further point out that the problem of maintain Single-Source Shortest Paths has also been considered quite recently in the incremental setting, i.e. the setting where a graph only undergoes edge insertions. In [4], the authors give a deterministic algorithm that achieves total update time $\tilde{O}(n^2\text{polylog}(W))$. Unfortunately, the techniques proposed in this algorithm cannot be extended to the more interesting decremental setting.

There is also a wide literature on the related problems of All-Pairs Shortest Paths and All-Pairs Reachability both in the decremental [27], [9], [10], [13], [28], [29], [30] and in the fully-dynamic setting [20], [21], [31], [32], [1], [33], [34], [35], [36], [13], [37], [38], [39].

Finally, for planar graphs, decremental algorithms are known to solve Single-Source Reachability deterministically in near-linear update time [40] and SSSP in directed graphs in total update time $\tilde{O}(n^{4/3})$ as shown by [41].

Our Contribution: We make significant progress on the problem of decremental SSSP in directed (dense) graphs and present the first *near-optimal* algorithm.

Theorem I.1. *Given a decremental input graph $G = (V, E, w)$ with $n = |V|, m = |E|$ and aspect ratio W, a dedicated source $r \in V$ and $\epsilon > 0$, there is a randomized algorithm that maintains a distance estimate $\widetilde{\mathbf{dist}}(r, x)$, for every $x \in V$, such that*

$$\mathbf{dist}_G(r,x) \leq \widetilde{\mathbf{dist}}(r,x) \leq (1+\epsilon)\mathbf{dist}_G(r,x)$$

at any stage w.h.p. The algorithm has total expected update time $\tilde{O}(n^2 \log^4 W/\epsilon)$. Distance queries are answered in $O(1)$ time, and a corresponding path P can be returned in $O(|P|)$ time.

We also present the currently fastest algorithm for sparse graphs. Combined, our two results significantly improve upon all previous work.

Theorem I.2. *Given a decremental input graph $G = (V, E, w)$ with $n = |V|, m = |E|$ and aspect ratio W, a dedicated source $r \in V$ and $\epsilon > 0$, there is a randomized algorithm that maintains a distance estimate $\widetilde{\mathbf{dist}}(r, x)$, for every $x \in V$, such that*

$$\mathbf{dist}_G(r,x) \leq \widetilde{\mathbf{dist}}(r,x) \leq (1+\epsilon)\mathbf{dist}_G(r,x)$$

at any stage w.h.p. The algorithm has total expected update time $\tilde{O}(mn^{2/3} \log^3 W/\epsilon)$. Distance queries are answered in $O(1)$ time, and a corresponding path P can be returned in $O(|P|)$ time.

Adaptive Versus Oblivious Adversaries: Both our results are randomized, and assume an oblivious adversary whose update sequence does not depend on the distance estimates and query-paths returned by the algorithm. This assumption is also necessary for all previous algorithms in directed graphs that go beyond the classic $O(mn)$ bound of Even and Shiloach [11], [12], [17], and achieving such a result adaptively remains a major open problem. (The result of [17] allows for adaptive *distance* queries, but still assumes the adversary is oblivious to the *paths* returned by the algorithm.) More generally, achieving truly fast adaptive algorithms seems implausible until we understand the problem well enough to at least have fast oblivious algorithms, and our result makes significant progress on this front. In fact, arguably more than any other dynamic graph algorithm, our $\tilde{O}(n^2)$ result highlights the gap between these two models: if one could design an *adaptive* algorithm that matches this efficiency, then plugging it into the existing framework of [5] (based in turn on the multiplicative-weight update method as described in [42], [43], [7]) would yield an $\tilde{O}(n^2)$ algorithm for *static* directed maximum flow, which would constitute an enormous breakthrough in the field of graph algorithms.

Techniques: The key contribution of our paper is a general technique for converting algorithms on directed acyclic graphs (DAGs) into algorithms on general graphs. Earlier techniques in [16], [17] lead to two simple algorithms for DAGs with total update times $\tilde{O}(n^2)$ and $\tilde{O}(mn^{2/3})$; our conversion then extends these bounds to general graphs. We first introduce the concept *approximate topological order* ($\mathcal{ATO}$), which loosely speaking imposes a DAG-like structure on *any* graph. We then show that an $\mathcal{ATO}$ always exists and can be maintained efficiently.

At a high-level, the conversion is as follows. Let $\mathcal{A}_{DAG}$ be

a decremental SSSP algorithm for DAGs and let $T(\mathcal{A}_{DAG})$ be the total update time. The first (easier) step is to convert $\mathcal{A}_{DAG}$ to an algorithm $\mathcal{A}^*_{DAG}$ that works on any graph with an $\mathcal{ATO}$ and has total update time $T(\mathcal{A}^*_{DAG}) \sim T(\mathcal{A}_{DAG})$. The second (harder) step is to build an algorithm $\mathcal{A}_{ATO}$ that maintains an $\mathcal{ATO}$ in G by recursively applying $\mathcal{A}^*_{DAG}$ as a subroutine: the basic idea is that an $\mathcal{ATO}$ of better "quality" can be built by using $\mathcal{A}^*_{DAG}$ to maintain shortest paths in an $\mathcal{ATO}$ of worse quality. Using this layered approach, we can achieve $T(\mathcal{A}_{ATO}) \sim T(\mathcal{A}^*_{DAG}) \sim T(\mathcal{A}_{DAG})$, and combining $\mathcal{A}_{ATO}$ with $\mathcal{A}^*_{DAG}$ gives an algorithm for general graphs.

Neither the step from $\mathcal{A}_{DAG}$ to $\mathcal{A}^*_{DAG}$ nor the step from $\mathcal{A}^*_{DAG}$ to $\mathcal{A}_{ATO}$ are black-box, but the techniques are quite modular and flexible, as evidenced by the fact that we were able to apply this conversion to both of the state-of-the-art algorithms for DAGs. We believe our conversion has strong potential to influence future work on directed shortest paths, in both the dynamic model and in others, by allowing researchers to focus on the simpler case of DAGs.

II. Preliminaries

We let a graph H refer to a weighted, directed graph with vertex set denoted by $V(H)$, edge set $E(H)$ and weight function $w_H : E(H) \to [1, W] \cup \{\infty\}$[2]. We say that H is a *decremental* graph if it is undergoing a sequence of edge deletions and edge weight increases (also referred to as updates), and refer to *version* t of H, or H at *stage* t as the graph H obtained after the first t updates have been applied. In this article, we denote the (decremental) input graph by $G = (V, E, w)$ with $n = |V|$ and $m = |E|$ (where m refers to the number of edges of G at stage 0). In all subsequent definitions, we often use a subscript to indicate which graph we refer to, however, when we refer to G, we often omit the subscript.

Cuts, Neighborhoods and Subgraphs: For graph H, and any two disjoint subsets $X, Y \subseteq V(H)$, we let $E_H(X)$ be the set of edges in $E(H)$ with an endpoint in X, and $E_H(X, Y)$ denote the set of edges in $E(H)$ with tail in X and head in Y. We let $\mathcal{N}^{in}_H(v) = \{u \mid (u, v) \in E(H)\}$ and $\mathcal{N}^{out}_H(v) = \{u \mid v \in \mathcal{N}^{in}_H(u)\}$ denote the in-neighborhood and out-neighborhoods of $v \in V$. We further let $H[X]$ refer to the subgraph of H induced by X, i.e. $H[X] = (X, E_H(X, X), w_H)$. We use $H \subseteq G$ to denote that $V(H) = V(G)$ and $E(H) \subseteq E(G)$.

Contractions: We define the graph H/X to be the graph obtained from H by contracting all vertices in X into a single node (we use the word *node* instead of vertex if it was obtained by contractions). Similarly, for a set of pairwise disjoint vertex sets $X_1, X_2, \ldots, X_k$, we let $H/\{X_1, X_2, \ldots, X_k\}$ denote the graph $((((H/X_1)/X_2)\ldots)/X_k)$. If $\mathcal{V}$ forms a partition of V, we use the convention to denote by X^v the node in $G/\mathcal{V}$ that contains $v \in V$, i.e. $v \in X^v$.

Reachability and Strong-Connectivity: For graph H and any two vertices $u, v \in V(H)$, we let $u \rightsquigarrow_H v$ denote that u can reach v in H, and $u \rightleftarrows_H v$ that u can reach v and vice versa (in the latter case, we also say u and v are strongly-connected). For any sets $X, Y \subseteq V(H)$, we say that $X \rightsquigarrow_H Y$ if there exists some $x \in X$, $y \in Y$ such that $x \rightsquigarrow_H y$; we define $X \rightleftarrows_H Y$ analogously. We say that the partition of $V(H)$ induced by the equivalence relation $\rightleftarrows_H$ is the set of *strongly-connected components* (SCCs).

Generalized Topological Order: We define a *generalized topological order* GeneralizedTopOrder(H) to be a tuple $(\mathcal{V}, \tau)$ where $\mathcal{V}$ is the set of SCCs of H and $\tau : \mathcal{V} \to [0, n)$ is a function that maps any sets $X, Y \in \mathcal{V}$ such that $\tau(X) < \tau(Y)$, if $X \rightsquigarrow_H Y$ and such that $[\tau(X), \tau(X) + |X|) \cap [\tau(Y), \tau(Y) + |Y|) = \emptyset$. Thus, τ effectively establishes a one-to-one correspondence between $|X|$-sized intervals and SCCs X in H. We point out that a GeneralizedTopOrder(H) can always be computed in $O(|E(H)|)$ time [44]. In fact, a generalized topological order can also be maintained efficiently in a decremental graph H. Here, we say that $(\mathcal{V}, \tau)$ is a dynamic tuple that forms a generalized topological order of H if it is a topological order for all versions of H. Further, we say that $(\mathcal{V}, \tau)$ has the *nesting* property, if for any set $X \in \mathcal{V}$ and a set $Y \supseteq X$ that was in $\mathcal{V}$ at an earlier stage, we have $\tau(X) \in [\tau(Y), \tau(Y) + |Y| - |X|]$; in other words, the interval $[\tau(X), \tau(X) + |X|)$ is entirely contained in the interval $[\tau(Y), \tau(Y) + |Y|)$. Thus, the associated interval with X is contained in the interval associated with Y. We refer to the following result that can be obtained straightforwardly by combining the data structure given in [23] and the static procedure by Tarjan [44] as described in [17].

Theorem II.1 (see [44], [23], [17])**.** *Given a decremental digraph H, there exists an algorithm that can maintain the generalized topological order $(\mathcal{V}, \tau)$ of H where τ has the nesting property. The algorithm runs in expected total update time $O(m \log^4 n)$, is randomized and works against an adaptive adversary.*

Distances and Diameter: We let $\mathbf{dist}_H(u, v)$ denote the distance from vertex u to vertex v in graph H and denote by $\pi_{u,v,H}$ the corresponding shortest path (we assume uniqueness by implicitly referring to the lexicographically shortest path). We define the weak diameter of $X \subseteq V(H)$ in H by $\mathbf{diam}(X, H) = \max_{x,y \in X} \mathbf{dist}_H(x, y)$.

Exponential Distribution: Finally, we make use of the exponential distribution, that is we use random variables X with cumulative distribution function $F_X(x, \lambda) = 1 - e^{-\lambda x}$ for all $x \geq 0, \lambda > 0$, which we denote by the shorthand $X \sim \text{Exp}(\lambda)$. If $X \sim \text{Exp}(\lambda)$ is clear, we also use $F_X(x)$ in place of $F_X(x, \lambda)$. The exponential distribution has the special

[2]In this article, we restrict our attention to integers and let $[a, b]$ denote the set of integers $a, a + 1, a + 2, \ldots, b$.

property of being *memoryless*, that is if $X \sim \text{EXP}(\lambda)$, then

$$\mathbb{P}[X > s + t \mid X > t] = \mathbb{P}[X > s].$$

III. Overview

We now give an overview of our algorithm. In order to illustrate the main concepts, we start by giving a simple algorithm to obtain total update time $O(n^2 \log n/\epsilon)$ in directed acyclic graphs (DAGs). While this algorithm was previously not explicitly mentioned, it follows rather directly from the techniques developed in [16], [17]. We present this algorithm to provide intuition for our approach and motivates our novel notion of approximate topological orders. In the light of approximate topological orders, we then shed light on limitations of the previous approach in [17] and present techniques to surpass these limitations to obtain Theorem I.1. In this overview, we focus on obtaining an SSSP algorithm that runs in $\tilde{O}(n^2 \log^4 W/\epsilon)$ expected update time. The final paragraph then sketches our improvement for sparse graphs (Theorem I.2); this result combines our directed framework in a non-trivial way with ideas from an earlier result for sparse SSSP in *undirected, unweighted* graphs [15].

A. A Fast Algorithm for DAGs

The Topological Order Difference: Let $G = (V, E, w)$ be a DAG and let τ be the function returned by GeneralizedTopOrder(G) computed on the initial version of G (since G is a DAG, this is just a standard topological order). Let us now make an almost trivial observation: for any shortest s-to-t path $\pi_{s,t}$, in any version of G, the sum of topological order differences is bounded by n. More formally:

$$\mathcal{T}(\pi_{s,t}, \tau) \stackrel{\text{def}}{=} \sum_{(u,v)\in\pi_{s,t}} \tau(v) - \tau(u) = \tau(t) - \tau(s) \leq n. \quad (1)$$

Observe that every path in G can only contain few edges (u, v) with large topological order difference, i.e. with $\tau(v) - \tau(u)$ large, by the pigeonhole principle.

Reviewing the ES-tree: To understand how this fact can be exploited, let us review the classic ES-tree (see [8]): We maintain distances from a fixed source $r \in V$ to each vertex v in V up to distance $\delta > 0$ by storing a distance estimate $\widetilde{\mathbf{dist}}(r, v)$ that is initialized to the distance between r and v in G along with a shortest-path tree T rooted at r. On update (u, v), we delete the edge from G and possibly from T. Then, if possible, we extract $w_{min} \in V \setminus \{r\}$, the vertex with smallest distance estimate among vertices without incoming edge in T. (For the first extraction, we always have $w_{min} = v$.) For this vertex w_{min}, we then try to find a vertex $x \in \mathcal{N}^{in}(w_{min})$ such that adding (x, w_{min}) to T implies $\mathbf{dist}_T(r, w_{min}) \leq \widetilde{\mathbf{dist}}(r, w_{min})$. We therefore search in $\mathcal{N}^{in}(v)$ for an x that satisfies

$$\widetilde{\mathbf{dist}}(r, x) + w(x, w_{min}) \leq \widetilde{\mathbf{dist}}(r, w_{min}). \quad (2)$$

If no such x exists, then $\widetilde{\mathbf{dist}}(r, w_{min})$ has to be incremented, and we set T to $T \backslash N^{out}(w_{min})$. If $\widetilde{\mathbf{dist}}(r, w_{min}) > \delta$, we set it to ∞ and remove w_{min} from the tree. We iterate the process until T is spanning for vertices with distance estimate $< \infty$. Since for each distance estimate value $\widetilde{\mathbf{dist}}(r, v)$, the in-neighourhood $\mathcal{N}^{in}(v)$ has to be scanned once (in an efficient implementation), and since estimates increase monotonically, the total update time can be bound by $O(\sum_{v\in V} |\mathcal{N}^{in}(v)|\delta) = O(m\delta)$. Further, observe that if a vertex v was only allowed to scan a certain vertex $x \in \mathcal{N}^{in}(v)$ every i distance estimates (for example whenever $\widetilde{\mathbf{dist}}(r, v)$ is divisible by i) then this corresponds to enforcing that at all times $\widetilde{\mathbf{dist}}(r, x) + w(x, v) + i - 1 \leq \widetilde{\mathbf{dist}}(r, v)$ since we check equation 2 every i steps (in particular, for $i = 1$, we get an exact algorithm). Consequently, we get at most $i - 1$ additive error in the distance estimate for any t whose shortest r-to-t path $\pi_{r,t}$ contains (x, v). On the other hand, we only need to scan and check the edge (x, v), by the argument above, δ/i times[3].

Improving the Running Time: Let us now exploit Inequality 1. We therefore define

$$B_j(v) = \{u \in \mathcal{N}^{in}(v) \text{ with } 2^j \leq \tau(v) - \tau(u) < 2^{j+1}\}$$

for every $v \in V$ and $0 \leq j \leq \lg n$; the $B_j(v)$ partition the in-neighborhood of v according to topological order difference to v. Observe that $|B_j(v)| \leq 2^j$. Now, consider the algorithm as above where for every vertex v, instead of checking all $\mathcal{N}^{in}(v)$, we only check edge (x, v) for $x \in B_j(v)$ if $\widetilde{\mathbf{dist}}(r, v)$ is divisible by $\lceil 2^j \frac{\epsilon\delta}{n} \rceil$. By the arguments above the total running time now sums to

$$\begin{aligned} &O\left(\sum_{v\in V} \sum_{0\leq j\leq \lg n} |B_j(v)| \frac{\delta}{2^j \frac{\epsilon\delta}{n}}\right) \\ &= O\left(\sum_{v\in V} \sum_{0\leq j\leq \lg n} 2^{j+2} \frac{\delta}{2^j \frac{\epsilon\delta}{n}}\right) = \tilde{O}(n^2/\epsilon). \end{aligned} \quad (3)$$

Bounding the Error: Fix a shortest r-to-t path $\pi_{r,t}$, and consider any edge $(u, v) \in \pi_{r,t}$ with $u \in B_{j+1}(v)$. We observe that the edge (u, v) contributes at most an additive error of $2^{j+1} \frac{\epsilon\delta}{n}$ to $\widetilde{\mathbf{dist}}(r, t)$ since it is scanned every $\lceil 2^j \frac{\epsilon\delta}{n} \rceil$ distance values and if $\lceil 2^j \frac{\epsilon\delta}{n} \rceil$ is equal to 1 it does not induce any error.

On the other hand, since $u \in B_{j+1}(v)$ we also have $\tau(v) - \tau(u) \geq 2^j$. We can thus charge $n/(2\epsilon\delta)$ units from $\mathcal{T}(\pi_{r,t}, \tau)$ for each additive error unit; we know from Equation 1 that $\mathcal{T}(\pi_{r,t}, \tau) \leq n$, so the total additive error is at most $\frac{n}{n/(2\epsilon\delta)} = 2\epsilon\delta$. Thus, for all distances $\approx \delta$ (say in $[\delta/2, \delta)$), we obtain a $(1 + 4\epsilon)$-multiplicative distance estimate[4].

[3] This trade-off was first observed in [16].

[4] Technically, we run to depth $(1 + 4\epsilon)\delta$ to ensure that vertices' distance estimates are not set to ∞ too early.

Working with Multiple Distance Scales: Observe that the data structure above has no running time dependency on δ. Thus, to obtain a data structure that maintains a $(1+2\epsilon)$-approximate distance estimate from r to any vertex x, we can simply use $\lg(nW)$ data structures in parallel where the i^{th} data structure has $\delta = 2^i$. A query can then be answered by returning the smallest distance estimate from any data structure, using a min-heap data structure to obtain this smallest estimate in constant time[5]. The running time for all data structures is then bounded by $\tilde{O}(n^2 \log W/\epsilon)$.

B. Extending the Result to General Graphs

We now encourage the reader to verify that in the data structure for DAGs, we used at no point that the graph was acyclic, but rather only used that $\mathcal{T}(\pi_{s,t}, \tau)$ is bounded by n for any path $\pi_{s,t}$. In this light, it might be quite natural to ask whether such a function τ might exist for general decremental graphs. Surprisingly, it turns out that after carrying out some contractions in G that only distort distances slightly, we can find such a function τ that comes close in terms of guarantees. We call such a function τ an *approximate topological order* (this function will no longer encode guarantees about reachability, but it helps for intuition to think of τ as being similar to a topological order).

The Approximate Topological Order: We start with the formal definition:

Definition III.1. *Given a decremental weighted digraph $G = (V, E, w)$ and parameter $\eta_{diam} \geq 0$. We call a dynamic tuple $(\mathcal{V}, \tau)$ an* approximate topological order of G of quality $q > 1$ *(abbreviated $\mathcal{ATO}(G, \eta_{diam})$ of quality q), if at any stage*

1) *$\mathcal{V} = \{X_1, X_2, .., X_k\}$ forms a partition of V and a refinement of all earlier versions of $\mathcal{V}$, and*
2) *$\tau : \mathcal{V} \to [0, n)$ is a function that maps each $X \in \mathcal{V}$ to a value $\tau(X)$. If some set $X \in \mathcal{V}$ is split at some stage into disjoint subsets $X_1, X_2, .., X_k$, then we let $\tau(X_{\pi(1)}) = \tau(X)$ and $\tau(X_{\pi(j+1)}) = \tau(X_{\pi(j)}) + |X_{\pi(j)}|$ for each $j < k$ and some permutation π of $[1, k]$, and*
3) *each $X \in \mathcal{V}$ has weak diameter $\mathbf{diam}(X, G) \leq \frac{|X|\eta_{diam}}{n}$, and*
4) *for any two vertices $s, t \in V$, the shortest path $\pi_{s,t}$ in G satisfies $\mathcal{T}(\pi_{s,t}, \tau) \leq q \cdot w(\pi_{s,t}) + n$ where we define $\mathcal{T}(\pi_{s,t}, \tau) \stackrel{def}{=} \sum_{(u,v)\in\pi_{s,t}} |\tau(X^u) - \tau(X^v)|$.*

We say that $(\mathcal{V}, \tau)$ is an $\mathcal{ATO}(G, \eta_{diam})$ of expected *quality q, if $(\mathcal{V}, \tau)$ satisfies properties 1-3, and at any stage, for every $s, t \in V$, $\mathbb{E}[\mathcal{T}(\pi_{s,t}, \tau)] \leq q \cdot w(\pi_{s,t}) + n$.*

Let us expound the ideas captured by this definition. We remind the reader that such a function τ is required by a data structure that only considers distances in $[\delta/2, \delta)$. Let us consider a tuple $(\mathcal{V}, \tau)$ that forms an $\mathcal{ATO}(G, \epsilon\delta)$ of quality q. Then, for any s-to-t shortest path $\pi_{s,t} = \langle s = v_1, v_2, \ldots v_\ell = t\rangle$ in G, let s_i and t_i be the first and last vertex on the path in $X_i \in \mathcal{V}$ (see property 1) if there are any. Observe that by property 3, the vertices s_i and t_i are at distance at most $\frac{|X_i|\epsilon\delta}{n}$ in G. It follows that if we contract the SCC X_i, the distance $\mathbf{dist}_{G/X_i}(s, t)$ is at least the distance from s to t in G minus an additive error of at most $\frac{|X_i|\epsilon\delta}{n}$. It follows straight-forwardly, that after contracting all sets in $\mathcal{V}$, we have that distances in $G/\mathcal{V}$ correspond to distances in G up to a negative additive error of at most $\sum_{X_i\in\mathcal{V}} \frac{|X_i|\epsilon\delta}{n} = \frac{n\cdot\delta\epsilon}{n} = \epsilon\delta$. Thus, maintaining the distances in $G/\mathcal{V}$ $(1 + 2\epsilon)$-approximately is still sufficient for getting a $(1 \pm 2\epsilon)$-approximate distance estimate.

Property 1 simply ensures that the vertex sets forming the elements of $\mathcal{V}$ decompose over time. Property 2 states that τ assigns each node in $G/\mathcal{V}$ a distinct number in $[0, n)$. It also ensures that if a set $X \in \mathcal{V}$ receives $\tau(X)$ that every later subset of X will obtain a number in the interval $[\tau(X), \tau(X) + |X|)$. Moreover, τ effectively establishes a one-to-one correspondence between nodes in a version of $G/\mathcal{V}$ and intervals in $[0, n)$ of size equal to their underlying vertex set. Once a set $X \in \mathcal{V}$ decomposes into sets $X_1, X_2, \ldots$, property 2 stipulates that the intervals that τ maps $X_1, X_2, \ldots$ to are disjoint subintervals of $[\tau(X), \tau(X) + |X|)$. We point out that once $\mathcal{V}$ consists of singletons, each vertex is essentially assigned a single number.

Finally, property 4 gives an upper bound on the topological order difference. Observe that we redefine $\mathcal{T}(\pi_{s,t}, \tau)$ in a way that is consistent with Definition 1. In our algorithm, for $\eta_{diam} \approx \epsilon\delta$, we obtain a quality of $\tilde{O}(n/\epsilon\delta)$. Thus, any path π of weight $\approx \delta$ has $\mathcal{T}(\pi/\mathcal{V}, \tau) \leq \tilde{O}(n/\epsilon)$ which is very close to the upper bound obtained by the topological order function in DAGs. We summarize this result in the theorem below which is one of our main technical contributions and whose proof is found in the full version of the paper.

Theorem III.2. *For any $0 \leq i \leq \lg(Wn)$, given a decremental digraph $G = (V, E, w)$, we can maintain an $\mathcal{ATO}(G, 2^i)$ of expected quality $\tilde{O}(n/2^i)$. The algorithm runs in total expected update time $\tilde{O}(n^2)$ against a non-adaptive adversary with high probability.*

Combining the theorem above and the theorem below which is obtained by generalizing the above decremental SSSP algorithm for DAGs, we obtain our main result Theorem I.1 (again the proof of the theorem is deferred to the full version).

Theorem III.3. *Given $G = (V, E, w)$ and $(\mathcal{V}, \tau)$ an $\mathcal{ATO}(G, \eta_{diam} \approx \epsilon\delta)$, for some depth parameter $\delta > 0$, of quality q, a dedicated source r in V, and an approximation parameter $\epsilon > 0$. Then, there exists a deterministic data structure $\mathcal{E}_r$ that maintains a distance estimate $\widetilde{\mathbf{dist}}(r, v)$ for*

[5]Here, we exploit that all distance estimates are overestimates, and at least one of them is $(1 + 2\epsilon)$-approximate.

each $v \in V$, which is guaranteed to be $(1+\epsilon)$-approximate if $\mathbf{dist}(r, v) \in [\delta, 2\delta)$. Distance queries are answered in $O(1)$ time and a corresponding path P can be returned in $O(|P|)$ time. The total update time is $\tilde{O}(n\delta q/\epsilon + n^2)$.

C. The Framework by [17]

Before we describe our new result, we review the framework in [17] to construct and maintain an $\mathcal{ATO}(G, \eta_{diam})$. We point out that while the abstraction of an approximate topological order is new to our paper, analyzing the technique in [17] through the $\mathcal{ATO}$-lens is straight-forward and gives a first non-trivial result. Throughout this review section, we assume that the graph G is unweighted to simplify presentation. This allows us to make use of the following result which states that for vertex sets that are far apart, one can find deterministically a vertex separator that is small compared to the smaller side of the induced partition (to obtain an algorithm for weighted graphs a simple edge rounding trick is sufficient to generalize the ideas presented below).

Definition III.4. *Given graph $G = (V, E, w)$, then we say a partition of V into sets A, S_{Sep}, B is a one-way vertex separator if $A \not\rightarrow_{G \setminus S_{Sep}} B$ and A and B are non-empty.*

Lemma III.5 (see Definition 5 and Lemma 6 in [26])**.** *Given an unweighted graph G of diameter $\mathbf{diam}(G)$. Then we can find sets A, S_{Sep}, B that form a one-way vertex separator such that $|S_{Sep}| \leq \tilde{O}(\frac{\min\{|A|,|B|\}}{\mathbf{diam}(G)})$ in time $O(m)$.*

High-level Framework: The main idea of [17] is to maintain a tuple $(\mathcal{V}, \tau)$ which is an $\mathcal{ATO}(G, \eta_{diam})$ by setting $(\mathcal{V}, \tau)$ to be GENERALIZEDTOPORDER(G') of some decremental graph $G' \subseteq G$ (over the same vertex set, i.e. $V(G') = V(G)$). It is straight-forward to see that $(\mathcal{V}, \tau)$ satisfies property 1 in Definition III.1, since SCCs in the decremental graph G' decompose. Further, it is not hard to extend the existing algorithm for maintaining SCCs in a decremental graph G' given in [23] to also maintain function τ that obeys property 2 in Definition III.1. The algorithm to maintain $(\mathcal{V}, \tau)$ given G' runs in total update time $\tilde{O}(m)$ (same as in [23]).

So far, we have not given any reason why G' needs to be a subgraph of G. To see why we cannot use the above strategy on G directly, recall property 3 in the $\mathcal{ATO}$-definition III.1, which demands that each SCC X has weak diameter at most $\frac{|X|\eta_{diam}}{n}$. This property might not hold in the main graph G. In order to resolve this issue, G' is initialized to G and then the diameter of SCCs in G' is monitored. Whenever an SCC X violates property 3, a vertex separator S_{Sep} is found in the graph $G'[X]$ as described in Lemma III.5 and all edges incident to S_{Sep} are removed from G'. Letting S denote the union of all such separators S_{Sep}, we can now write $G' = G \setminus E(S)$.

Establishing the Quality Guarantee: To establish a quality q of the $\mathcal{ATO}$ as described in property 4 in Definition III.1, let us first partition the set S into sets $S_0, S_1, \ldots, S_{\lg n}$ where each S_i contains all separator vertices found on a graph of size $[n/2^i, n/2^{i+1})$, thus it was found when the procedure from Lemma III.5 was invoked on a graph with diameter at least $\frac{(n/2^{i+1})\eta_{diam}}{n} = \frac{\eta_{diam}}{2^{i+1}}$. Since separators are further balanced, i.e. there size is controlled by the smaller side of the induced cut, we can further use induction and Lemma III.5 to establish that there are at most $\tilde{O}(\frac{2^i n}{\eta_{diam}})$ vertices in S_i. Next, observe that since any separator that was added to S_i was found in a graph $G'[X]$ with $|X| \in [n/2^i, n/2^{i+1})$, we have by property 2 that nodes $X' \subseteq X$ that are in the current version of $\mathcal{V}$ are assigned a $\tau(X')$ from the interval $[\tau(X), \tau(X) + |X|)$. Thus, any edge (x, s) or (s, x) with $x \in X, s \in S_i \cap X$ has topological order difference $|\tau(X^x) - \tau(X^s)| \leq |X| \leq n/2^i$.

Finally, let us define

$$\mathcal{T}'(\pi_{s,t}, \tau) \stackrel{\text{def}}{=} \sum_{(u,v) \in \pi_{s,t}} \min\{0, \tau(X^v) - \tau(X^u)\} \qquad (4)$$

the function similar to $\mathcal{T}(\pi_{s,t}, \tau)$ that only captures negative terms, i.e. sums only over edges that go "backwards" in τ. Observe that $\mathcal{T}(\pi_{s,t}, \tau) \leq 2\mathcal{T}'(\pi_{s,t}, \tau) + n$. Now, since $(\mathcal{V}, \tau)$ is a GENERALIZEDTOPORDER(G'), we have that (u, v) occurs in the sum of $\mathcal{T}'(\pi_{s,t}, \tau)$ if and only if $(u, v) \in G \setminus G'$, so one endpoint is in a set S_i and therefore $\tau(X^v) - \tau(X^u) \leq n/2^i$. Since we only have two edges on any shortest path incident to the same vertex, we can establish that

$$\mathcal{T}'(\pi_{s,t}, \tau) \leq \sum_i 2|S_i|n/2^i = \tilde{O}(n^2/\eta_{diam}).$$

We obtain that $(\mathcal{V}, \tau)$ is a $\mathcal{ATO}(G, \eta_{diam})$ of quality $\tilde{O}(\frac{n^2}{2^i \eta_{diam}})$ for all paths of weight at least 2^i. Thus, when the distance scale $\delta \geq \sqrt{n}/\epsilon$, Theorem III.3 requires total update time $\approx n^{2.5}$ to maintain $(1+\epsilon)$-approximate SSSP. For distance scales where $\delta < \sqrt{n}/\epsilon$, a classic ES-tree has total update time $\approx m\sqrt{n} \leq n^{2.5}$.

Limitations of the Framework: Say that the goal is to maintain shortest paths of length around $\sqrt{n}$. The first step in the framework of [17] is to find separator S such that all SCCs of $G' = G \setminus E(S)$ have diameter at most $\epsilon\sqrt{n}$ and then maintain $(\mathcal{V}, \tau) =$ GENERALIZEDTOPORDER(G'). Every edge $(u, v) \notin E(S)$ will only go forward in τ, but each edge (u, s), for $s \in S$, can go "backwards" in τ. By the nesting property of generalized topological orders, the amount that (u, s) goes backwards – i.e. the quantity $|\tau(X^u) - \tau(X^s)|$ – is upper bounded by the size of the SCC in G' from which s was chosen: the original SCC has size n, but as we add vertices to S, the SCCs of $G' = G \setminus E(S)$ decompose and new vertices added to S may belong to smaller SCCs. Define $S^* \subseteq S$ to contain all vertices $s \in S$ that were

chosen in an SCC of size $\Omega(n)$. Intuitively, S^* is the top-level separator chosen in G, before SCCs decompose into significantly smaller pieces. Every edge in $E(S^*)$ may go backwards by as much as n in τ, so for any path $\pi_{x,y}$ in G, the best we can guarantee is that $\mathcal{T}(\pi_{x,y}) \sim n \cdot |\pi_{x,y} \cap S^*|$.

The framework of [17] tries to find a *small* separator S^* and then uses the trivial upper bound $|\pi_{x,y} \cap S^*| \leq |S^*|$. In fact, one can show that given *any* deterministic separator procedure, the adversary can pick a sequence of updates where $|\pi_{x,y} \cap S^*| \sim |S^*|$. But now, say that G is a $\sqrt{n} \times \sqrt{n}$-grid graph with bidirectional edges. It is not hard to check that $|S^*| = \Omega(\sqrt{n})$, because every balanced separator of a grid has $\Omega(\sqrt{n})$ vertices. The framework of [17] can thus at best guarantee $\mathcal{T}(\pi_{x,y}) \sim n \cdot |\pi_{x,y} \cap S^*| \sim n|S^*| = \Omega(n^{1.5})$, which is a $\sqrt{n}$ factor higher than it would be in a DAG, and thus leads to running time $\tilde{O}(n^{2.5})$ instead of $\tilde{O}(n^2)$.

Our algorithm uses an entirely different random separator procedure. We allow S^* to be arbitrarily large, but use randomness to ensure that $|\pi_{x,y} \cap S^*|$ is nonetheless small.

D. Our Improved Framework

We now introduce our new separator procedure and then show how it can be used in a recursive algorithm that uses ATOs of worse quality (large q) to compute ATOs of better quality (small q). (By contrast, the framework of [17] could not benefit from a multi-layered algorithm because it would still hit upon the fundamental limitation outlined above.)

A New Separator Procedure: Before we describe the separator procedure, let us formally define the guarantees that we obtain. In the lemma below, think of $\zeta = \Theta(\log(n))$.

Lemma III.6. *There exists a procedure* OUTSEPARATOR(r, G, d, ζ) *where G is a weighted graph, $r \in V$ a root vertex, and integers $d, \zeta > 0$. Then, with probability at least $1 - e^{-\zeta}$, the procedure computes a tuple (E_{Sep}, V_{Sep}) where edges $E_{Sep} \subseteq E$, and vertices $V_{Sep} = \{v \in V | r \rightsquigarrow_{G \setminus E_{Sep}} v\}$ such that*

1) *for every vertex $v \in V_{Sep}$, $\mathbf{dist}_{G \setminus E_{Sep}}(r, v) \leq d$, and*
2) *for every $e \in E$, we have $\mathbb{P}[e \in E_{Sep} | r \rightsquigarrow_{G \setminus E_{Sep}} \mathbf{tail}(e)] \leq \frac{\zeta}{d} w(e)$.*

Otherwise, it reports **Fail**. *The running time of* OUTSEPARATOR$(\cdot)$ *can be bounded by $O(|E(V_{Sep})| \log n)$.*

In fact, Algorithm 1 gives an extremely simple implementation of procedure OUTSEPARATOR$(\cdot)$. Here, we pick a ball $B = B^{out}(r, X)$ in the graph G from r to random depth X, and then simply return the tuple $(E_{Sep}, V_{Sep}) = (E(B, \overline{B}), B)$ where $E(B, \overline{B})$ are the edges (u, v) with $u \in B$ but $v \notin B$. The procedure thus only differs from a standard edge separator procedure in that we choose X according to the exponential distribution EXP$(\frac{\zeta}{d})$.

A proof of Lemma III.6 is now straightforward. We return **Fail** in Line 2 with probability $\mathbb{P}[X \geq d] = 1 - F_X(d) = 1 - (1 - e^{-\frac{\zeta}{d} \cdot d}) = e^{-\zeta}$ (recall from section II that $F_X(d)$ is shorthand for $F(x, \frac{\zeta}{d})$, the cumulative distribution function of an exponential distribution with parameter $\frac{\zeta}{d}$). Assuming no failure, we have $E_{Sep} = E(B, \overline{B})$, and it is easy to see that $V_{Sep} = \{v \in V | r \rightsquigarrow_{G \setminus E_{Sep}} v\} = B$, so Property 1 of Lemma III.6 holds by definition of B. Moreover, we can compute B in the desired $O(|E(B)| \log n)$ time by using Dijkstra's algorithm by only extracting a vertex from the heap if it is at distance at most X. Finally, for property 2, note that $e \in E_{sep}$ iff $\mathbf{dist}(r, \mathbf{tail}(e)) \leq X < \mathbf{dist}(r, \mathbf{tail}(e)) + w(e)$. Thus,

$$\begin{aligned}
&\mathbb{P}[e \in E_{Sep} | r \rightsquigarrow_{G \setminus E_{Sep}} \mathbf{tail}(e)] \\
&= \mathbb{P}[X < \mathbf{dist}(r, \mathbf{tail}(e)) + w(e) | X \geq \mathbf{dist}(r, \mathbf{tail}(e))] \\
&= \mathbb{P}[X < w(e)] = F_X(w(e)) = 1 - e^{-\frac{\zeta}{d} w(e)} \\
&\leq 1 - \left(1 - \frac{\zeta}{d} w(e)\right) = \frac{\zeta}{d} w(e)
\end{aligned}$$

where the second equality follows from the memory-less property of the exponential distribution, and the inequality holds because $1 + x \leq e^x$ for all $x \in \mathbb{R}$. We point out that the technique of random ball growing using the exponential distribution is not a novel contribution in itself and has be previously used in the context of low-diameter decompositions [45], [46], [47], [48] which have recently also been adapted to dynamic algorithms [49], [50].

Algorithm 1: OUTSEPARATOR(r, G, d, ζ)

1 Choose $X \sim$ EXP$(\frac{\zeta}{d})$.
2 **if** $X \geq d$ **then return Fail**
3 Compute the Ball
$B = B^{out}(r, X) = \{v \in V | \mathbf{dist}(r, v) \leq X\}$
4 **return** $(E(B, \overline{B}), B)$

A New Framework: Let us now outline how to use Lemma III.6 to derive Theorem III.2 which is stated below again. The full details and a rigorous proof are provided in the full version.

Theorem III.2. *For any $0 \leq i \leq \lg(Wn)$, given a decremental digraph $G = (V, E, w)$, we can maintain an $\mathcal{ATO}(G, 2^i)$ of expected quality $\tilde{O}(n/2^i)$. The algorithm runs in total expected update time $\tilde{O}(n^2)$ against a non-adaptive adversary with high probability.*

As in [17], we maintain a graph $G' \subseteq G$ and its generalized topological order $(\mathcal{V}, \tau)$. Whenever the diameter of an SCC X in G' is larger than $\frac{|X| \eta_{diam}}{n}$, we now use the separator procedure described in Lemma III.6 with $d = \frac{|X| \eta_{diam}}{2n}$ from some vertex r in X with $|B^{out}(r, d = \frac{|X| \eta_{diam}}{2n})| \leq |X|/2$. Such a vertex exists since by definition of diameter, as we can find two vertices with disjoint balls. We obtain an edge separator E_{Sep} and update G' by removing the edges in E_{Sep}. Let the union of all edge separators be denoted by F

and again observe that $G' = G \setminus F$. It is not hard to see that our scheme still ensures properties 1-3 in Definition III.1. We now argue that the quality improved to $\tilde{O}(n/2^i)$.

Partition F into sets $F_0, F_1, \ldots, F_{\lg n}$ where each F_j contains all separator edges found on a graph of size $[n/2^{j+1}, n/2^j)$. We again have that every edge $(u,v) \in F_j$ has $|\tau(X^u) - \tau(X^v)| \leq n/2^j$. Let us establish an upper bound on the number of edges in F_j on any shortest path $\pi_{s,t}$. Consider therefore any edge $e \in \pi_{s,t}$, at a stage where both endpoints of e are in a SCC X of size $[n/2^{j+1}, n/2^j)$ and where we compute a tuple (V_{Sep}, E_{Sep}) from some $r \in X$. Now, observe that if $\mathbf{tail}(e) \notin V_{Sep}$, then e can not be in E_{Sep}. So assume that $\mathbf{tail}(e) \in V_{Sep}$. Then, we have e joining F_j with probability

$$\begin{aligned}\mathbb{P}[e \in E_{Sep} | r \rightsquigarrow_{H \setminus E_{Sep}} \mathbf{tail}(e)] &= \frac{\zeta}{2^i |X|/n} w(e) \\ &= \tilde{O}\left(\frac{2^j w(e)}{2^i}\right)\end{aligned}$$

according to Lemma III.6, where we set $\zeta = \tilde{O}(1)$ to obtain high success probability. But if e did not join F_j at that stage, then it is now in a SCC of size at most $n/2^{j+1}$ (recall we chose $|B^{out}(r,d)| \leq |X|/2$, and we have $V_{Sep} \subseteq B^{out}(r,d)$). Thus, e cannot join F_j at any later stage.

Now, it suffices to sum over edges on the path $\pi_{s,t}$ and indices j to obtain that

$$\begin{aligned}\mathcal{T}'(\pi_{s,t}, \tau) &\leq \sum_{e \in \pi_{s,t}} \sum_j \mathbb{P}[e \in F_j] \cdot n/2^j \\ &= \sum_{e \in \pi_{s,t}} \sum_j \tilde{O}\left(\frac{2^j w(e)}{2^i}\right) \cdot n/2^j \\ &= \tilde{O}\left(\frac{w(\pi_{s,t}) n}{2^i}\right)\end{aligned}$$

giving quality $\tilde{O}\left(\frac{n}{2^i}\right)$.

Efficiently Maintaining G': As shown above, maintaining an $\mathcal{ATO}(G, 2^i)$ requires detecting when any SCC in $G' = G \setminus E(S)$ has diameter above 2^i. We start by showing how to do this efficiently if we are given a black-box algorithm $\mathcal{A}_{SSSP}$ that maintains distance estimates up to depth threshold 2^i (i.e. if a vertex is at distance less than 2^i from the source vertex, there is a distance estimate with good approximation ratio).

We use the random source scheme introduced in [24] along with some techniques developed in [26], [23], [17]: we choose for each SCC X in G' a center vertex $\textsc{Center}(X) \in X$ uniformly at random, and use $\mathcal{A}_{SSSP}$ to maintain distances from $\textsc{Center}(X)$ to depth $2^i |X|/n \leq 2^i$. Since the largest distances between the vertex $\textsc{Center}(X)$ and any other vertex in X is a 2-approximation on the diameter of $G[X]$, this is sufficient to monitor the diameter and to trigger the separator procedure in good time.

Cast in terms of our new ATO-framework, the previous algorithm of [17] used a regular Even and Shiloach tree for the algorithm $\mathcal{A}_{SSSP}$. We instead use a recursive structure, where $\mathcal{ATO}s$ of bad quality (large q) are used to build $\mathcal{ATO}s$ of better quality (small q). Recall that our goal is to build an $\mathcal{ATO}(G, 2^i)$ of quality $\tilde{O}(n/2^i)$ and say that X is some SCC of $G' = G \setminus E(S)$ whose diameter we are monitoring. Now, using the lower level of the recursion, we inductively assume that we can maintain an $\mathcal{ATO}(G'[X], 2^{i-1})$ of quality $\tilde{O}(n/2^{i-1})$ in time $\tilde{O}(|X|^2)$. Plugging this $\mathcal{ATO}$ into Theorem III.3 gives us an algorithm for maintaining distances up to depth 2^i in X with total update time $\tilde{O}(|X|^2)$, Summing over all components X in G', we get an $\tilde{O}(n^2)$ total update time to maintain $\mathcal{ATO}(G, 2^i)$, as desired.

We actually cheated a bit in the last calculation, because the scheme above could incur an additional logarithmic factor for computing $\mathcal{ATO}(G, 2^i)$ from all the $\mathcal{ATO}(G'[X], 2^{i-1})$, so we can only afford a sublogarithmic number of levels, which leads to an extra $n^{o(1)}$ factor in the running time. However, a careful bootstrapping argument allows us to avoid this extra term.

E. A Framework for Sparse SSSP

Similarly to Theorem III.2, we can prove a similar theorem with better running time for sparse graphs.

Theorem III.7. *For any $0 \leq i \leq \log(Wn)$, given a decremental digraph $G = (V, E, w)$, we can maintain an $\mathcal{ATO}(G, 2^i)$ of expected quality $\tilde{O}(n/2^i)$. The algorithm runs in total expected update time $\tilde{O}(mn^{2/3})$ against a non-adaptive adversary with high probability.*

In the remaining section, let us sketch how we obtain an efficient algorithm to compute SSSP given the result from Theorem III.7. To simplify the presentation let us assume for the rest of the section that the graph $G = (V, E)$ is unweighted.

Reducing SSSP to Hopset Maintenance: Let us introduce the notion of a $(1+\epsilon, h)$-hopset H which is a weighted decremental graph on the same vertex set as G such that at any stage, for any two vertices $s, t \in V$, we have

$$\mathbf{dist}_G(s,t) \leq \mathbf{dist}^h_{G \cup H}(s,t) \leq (1+\epsilon)\mathbf{dist}_G(s,t) \quad (5)$$

where we use the notation $\mathbf{dist}^\ell_F(s,t)$ to denote the shortest s to t path in the graph F consisting of at most ℓ edges. Using a well-known rounding technique, we can then run a slightly modified ES-tree data structure from a root vertex r on the graph $G \cup H$ to depth h to obtain $(1+\epsilon)$-approximate distances in G from r. The data structure only requires time $\tilde{O}((m + |E(H)|)h)$.

In our paper, we show how to construct for every $\lg(n^{2/3} \log n) \leq i \leq \lg n$, a $(1+\epsilon, n^{2/3})$-hopset F_i with $\tilde{O}(n)$ edges that satisfies equation Equation (5) for all pairs

at distance $[2^i, 2^{i+1})$, so the total update time of the ES-tree becomes $\tilde{O}(mn^{2/3})$, as desired. For $i < \lg(n^{2/3}\log n)$, we can maintain distances up to 2^i in total update time $\tilde{O}(mn^{2/3})$ by simply running a classic ES-tree, without any hop-set. We obtain final distance estimates by running the above algorithm for each i; then, for any pair s, t, we find the smallest distance estimate among these $O(\log(n))$ data structures and output that as the final distance estimate.

Maintaining a Hopset F_i: For each $\lg(n^{2/3}\log n) \leq i \leq \lg n$, we want to maintain F_i as a weighted graph with the guarantee that for every vertices s and t at distance at $[2^i, 2^{i+1})$, we have a $(1+\epsilon)$-approximate shortest path in $F_i \cup G$ of hop at most $\tilde{O}(n^{2/3})$.

To maintain such a graph F_i, we use $(\mathcal{V}_i, \tau_i)$ an $\mathcal{ATO}(G, \eta_{diam} = \epsilon 2^i)$ as given in Theorem III.7. We first sample each vertex $v \in V$ with probability $\tilde{\Theta}(n^{1/3}/2^i)$. We let S be the set of sampled vertices and have $|S| = \tilde{O}(n^{1+1/3}/2^i)$ with high probability. We then run from each vertex $s \in S$, with node $X^s \in \mathcal{V}_i$, an ES-tree to depth $2^i/n^{2/3}$ on the graph $G/\mathcal{V}_i$ induced by the set of nodes $Y \in \mathcal{V}_i$ such that $|\tau_i(X^s) - \tau_i(Y)| = \tilde{O}(n^{2/3})$. We then add an edge (s, t) for every vertex t that is in a node in the ES-tree of the vertex s where we use corresponding distance estimate as an edge weight.

Hopset Sparsity and Running Time: It can be shown that every edge only participates in $\tilde{O}(|S|)$ ES-trees and therefore the total running time of all ES-trees can be bound by $\tilde{O}(m|S| \cdot 2^i) = \tilde{O}(mn^{2/3})$. Further, each ES-tree from a vertex $s \in S$ has with high probability at most $\tilde{O}(n^{1/3}/2^i)$ vertices in S in its tree and therefore the set F_i has at most $\tilde{O}(|S|n^{1/3}/2^i) = \tilde{O}(n)$ edges.

Correctness: Finally, to show that $G \cup F_i$ contains a path of hop at most $\tilde{O}(n^{2/3})$ between any two vertices s, t at distance 2^i let us focus on their corresponding shortest path $\pi_{s,t}$. We can partition the path into segments of length at most $2^{i-1}/n^{2/3}$, and by a standard hitting set argument, we have with high probability a vertex in S in every segment. Let $s_1, s_2, \ldots, s_k$ be such that each s_j is a vertex in S in the j^{th} segment. We observe that $k \leq \frac{2^i}{2^{i-1}/n^{2/3}} = 2n^{2/3}$. Now, for every s_j we either have s_{j+1} in the ES-tree, in which case we have a direct edge between the vertices in F_i. Otherwise, some vertex v on the path segment from s_j to s_{j+1} is in a node Y such that $|\tau_i(X^s) - \tau_i(Y)| \gg n^{2/3}$. But since the quality of the $\mathcal{ATO}$ is $\tilde{O}(n/2^i)$, the total sum of topological difference of the path $\pi_{s,t}$ is only $\tilde{O}(n)$. Thus, this can occur at most $n^{1/3}$ times. Every time, we can use the shortest path in G between s_j and s_{j+1} consisting of at most $2^i/n^{2/3} \leq n^{1/3}$ edges. Adding the paths from s to s_1 and from s_k to t, the total number of edges is at most $\tilde{O}(n^{2/3})$ as desired. Finally, we observe that when running the ES-data structure on $G \cup F_i$, we have to add an additive term of η_{diam} since the contractions in the graph $G/\mathcal{V}_i$ might decrease distances by this additive term. But for distances in the range $[2^i, 2^{i+1})$, this term can be subsumed in the $(1+\epsilon)$ multiplicative error (after rescaling ϵ by a constant factor).

IV. Conclusion

In this article, we gave the first near-optimal algorithm to solve the decremental SSSP problem on dense digraphs. Combined with the recent result by Gutenberg et al [4], this establishes an $\tilde{O}(n^2 \log^4 W/\epsilon)$ complexity for the partially-dynamic SSSP problem in directed weighted graphs. Moreover, we gave a simple new technique to derive a data structure for sparse graphs using our framework, which runs in total time $\tilde{O}(mn^{2/3}\log^3 W/\epsilon)$ and thereby vastly improves over the best existing result by Probst Gutenberg and Wulff-Nilsen as recently shown in [17].

This substantial progress on the problem motivates the following two open questions:

- Can we obtain a near-optimal algorithm for partially-dynamic SSSP in directed graphs for any sparsity? Recent work by Bernstein et al. [23] has shown that this is at least possible for the simpler problem of maintaining single-source reachability.
- Can we derandomize our results or make them work against an adaptive adversary? Partial progress on this question has been made in [4] where the incremental SSSP data structure presented is already deterministic which gave the first deterministic improvement in directed graphs over the ES-tree. In decremental graphs, a recent result by Bernstein et al. [19] gives deterministic total update time $O(n^{2+2/3})$ following a result by Probst Gutenberg and Wulff-Nilsen [17] that broke the $O(mn)$ bound randomized but against an adaptive adversary. However, this is far from the near-optimal bound achieved in this paper. We point out that even in the undirected, unweighted setting, the best data structure requires total update time $\tilde{O}(\min\{mn^{0.5+o(1)}, n^2\})$ [18], [14] as opposed to $m^{1+o(1)}$ total update time in the non-oblivious setting [11].

Acknowledgment

Aaron Bernstein is supported by NSF CAREER Grant 1942010 and the Simons Group for Algorithms & Geometry. Maximilian Probst Gutenberg is supported by Basic Algorithms Research Copenhagen (BARC), supported by Thorup's Investigator Grant from the Villum Foundation under Grant No. 16582. Christian Wulff-Nilsen is supported by the Starting Grant 7027-00050B from the Independent Research Fund Denmark under the Sapere Aude research career programme. The authors thank Thatchaphol Saranurak for insightful comments and corrections, and anonymous FOCS reviewers for their helpful feedback.

REFERENCES

[1] L. Roditty and U. Zwick, "On dynamic shortest paths problems," in *European Symposium on Algorithms*. Springer, 2004, pp. 580–591.

[2] A. Abboud and V. V. Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 434–443.

[3] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. ACM, 2015, pp. 21–30.

[4] M. Probst Gutenberg, V. Vassilevska Williams, and N. Wein, "New algorithms and hardness for incremental single-source shortest paths in directed graphs," in *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, 2020.

[5] J. Chuzhoy and S. Khanna, "A new algorithm for decremental single-source shortest paths with applications to vertex-capacitated flow and cut problems," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2019, pp. 389–400.

[6] J. Chuzhoy, Y. Gao, J. Li, D. Nanongkai, R. Peng, and T. Saranurak, "A deterministic algorithm for balanced cut with applications to dynamic connectivity, flows, and beyond," *arXiv preprint arXiv:1910.08025*, 2019.

[7] A. Madry, "Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms," in *Proceedings of the forty-second ACM symposium on Theory of computing*. ACM, 2010, pp. 121–130.

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

[9] A. Bernstein and L. Roditty, "Improved dynamic algorithms for maintaining approximate shortest paths under deletions," in *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2011, pp. 1355–1365.

[10] M. Henzinger, S. Krinninger, and D. Nanongkai, "Decremental single-source shortest paths on undirected graphs in near-linear total update time," in *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*. IEEE, 2014, pp. 146–155.

[11] ——, "Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs," in *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. ACM, 2014, pp. 674–683.

[12] ——, "Improved algorithms for decremental single-source reachability on directed graphs," in *International Colloquium on Automata, Languages, and Programming*. Springer, 2015, pp. 725–736.

[13] ——, "Dynamic approximate all-pairs shortest paths: Breaking the o(mn) barrier and derandomization," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 947–1006, 2016.

[14] A. Bernstein and S. Chechik, "Deterministic decremental single source shortest paths: beyond the o (mn) bound," in *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. ACM, 2016, pp. 389–397.

[15] ——, "Deterministic partially dynamic single source shortest paths for sparse graphs," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2017, pp. 453–469.

[16] A. Bernstein, "Deterministic partially dynamic single source shortest paths in weighted graphs," in *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2017, p. 44.

[17] M. P. Gutenberg and C. Wulff-Nilsen, "Decremental sssp in weighted digraphs: Faster and against an adaptive adversary," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 2542–2561.

[18] ——, "Deterministic algorithms for decremental approximate shortest paths: Faster and simpler," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 2522–2541.

[19] A. Bernstein, M. Probst Gutenberg, and T. Saranurak, "Deterministic decremental reachability, scc, and shortest paths via directed expanders and congestion balancing," in *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2020.

[20] M. R. Henzinger and V. King, "Fully dynamic biconnectivity and transitive closure," in *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*. IEEE, 1995, pp. 664–672.

[21] V. King, "Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs," in *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*. IEEE Computer Society, 1999, pp. 81–91.

[22] A. Bernstein, J. v. d. Brand, M. P. Gutenberg, D. Nanongkai, T. Saranurak, A. Sidford, and H. Sun, "Fully-dynamic graph sparsifiers against an adaptive adversary," *arXiv preprint arXiv:2004.08432*, 2020.

[23] A. Bernstein, M. Probst, and C. Wulff-Nilsen, "Decremental strongly-connected components and single-source reachability in near-linear time," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2019, pp. 365–376.

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

[25] J. Łkacki, "Improved deterministic algorithms for decremental reachability and strongly connected components," *ACM Transactions on Algorithms (TALG)*, vol. 9, no. 3, p. 27, 2013.

[26] S. Chechik, T. D. Hansen, G. F. Italiano, J. Łkacki, and N. Parotsidis, "Decremental single-source reachability and strongly connected components in o ($m\sqrt{}$ n) total update time," in *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*. IEEE, 2016, pp. 315–324.

[27] S. Baswana, R. Hariharan, and S. Sen, "Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths," *Journal of Algorithms*, vol. 62, no. 2, pp. 74–92, 2007.

[28] A. Bernstein, "Maintaining shortest paths under deletions in weighted directed graphs," *SIAM Journal on Computing*, vol. 45, no. 2, pp. 548–574, 2016.

[29] S. Chechik, "Near-optimal approximate decremental all pairs shortest paths," in *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE, 2018, pp. 170–181.

[30] A. Karczmarz and J. Łkacki, "Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs," in *Symposium on Simplicity in Algorithms*. SIAM, 2020, pp. 106–120.

[31] C. Demetrescu and G. F. Italiano, "Fully dynamic all pairs shortest paths with real edge weights," in *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 2001, pp. 260–267.

[32] ——, "A new approach to dynamic all pairs shortest paths," *Journal of the ACM (JACM)*, vol. 51, no. 6, pp. 968–992, 2004.

[33] M. Thorup, "Worst-case update times for fully-dynamic all-pairs shortest paths," in *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*. ACM, 2005, pp. 112–119.

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

[35] I. Abraham and S. Chechik, "Dynamic decremental approximate distance oracles with $(1+\epsilon, 2)$ stretch," *arXiv preprint arXiv:1307.1516*, 2013.

[36] L. Roditty and U. Zwick, "A fully dynamic reachability algorithm for directed graphs with an almost linear update time," *SIAM Journal on Computing*, vol. 45, no. 3, pp. 712–733, 2016.

[37] I. Abraham, S. Chechik, and S. Krinninger, "Fully dynamic all-pairs shortest paths with worst-case update-time revisited," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2017, pp. 440–452.

[38] J. v. d. Brand and D. Nanongkai, "Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time," *arXiv preprint arXiv:1909.10850*, 2019.

[39] M. Probst Gutenberg and C. Wulff-Nilsen, "Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds," in *Proceedings of the Thirty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020.

[40] G. F. Italiano, A. Karczmarz, J. Łkacki, and P. Sankowski, "Decremental single-source reachability in planar digraphs," in *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*. ACM, 2017, pp. 1108–1121.

[41] A. Karczmarz, "Decrementai transitive closure and shortest paths for planar digraphs and beyond," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, pp. 73–92.

[42] L. K. Fleischer, "Approximating fractional multicommodity flow independent of the number of commodities," *SIAM Journal on Discrete Mathematics*, vol. 13, no. 4, pp. 505–520, 2000.

[43] N. Garg and J. Koenemann, "Faster and simpler algorithms for multicommodity flow and other fractional packing problems," *SIAM Journal on Computing*, vol. 37, no. 2, pp. 630–652, 2007.

[44] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.

[45] N. Linial and M. Saks, "Low diameter graph decompositions," *Combinatorica*, vol. 13, no. 4, pp. 441–454, 1993.

[46] Y. Bartal, "Probabilistic approximation of metric spaces and its algorithmic applications," in *Proceedings of 37th Conference on Foundations of Computer Science*. IEEE, 1996, pp. 184–193.

[47] G. L. Miller, R. Peng, and S. C. Xu, "Parallel graph decompositions using random shifts," in *Proceedings of the twenty-fifth annual ACM symposium on Parallelism in algorithms and architectures*. ACM, 2013, pp. 196–203.

[48] J. Pachocki, L. Roditty, A. Sidford, R. Tov, and V. V. Williams, "Approximating cycles in directed graphs: Fast algorithms for girth and roundtrip spanners," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2018, pp. 1374–1392.

[49] S. Forster and G. Goranci, "Dynamic low-stretch trees via dynamic low-diameter decompositions," in *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, 2019, pp. 377–388.

[50] S. Chechik and T. Zhang, "Dynamic low-stretch spanning trees in subpolynomial time," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 463–475.