2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)

Dynamic Approximate Shortest Paths and Beyond: Subquadratic and Worst-Case Update Time

Jan van den Brand
KTH Royal Institute of Technology
Stockholm, Sweden

Danupon Nanongkai
KTH Royal Institute of Technology
Stockholm, Sweden

***Abstract*—Consider the following distance query for an n-node graph G undergoing edge insertions and deletions: given two sets of nodes I and J, return the distances between every pair of nodes in $I \times J$. This query is rather general and captures several versions of the dynamic shortest paths problem. In this paper, we develop an efficient $(1+\epsilon)$-approximation algorithm for this query using fast matrix multiplication. Our algorithm leads to answers for some open problems for Single-Source and All-Pairs Shortest Paths (SSSP and APSP), as well as for Diameter, Radius, and Eccentricities. Below are some highlights. Note that all our algorithms guarantee *worst-case* update time and are randomized (Monte Carlo), but do not need the oblivious adversary assumption.**

***Subquadratic update time for SSSP, Diameter, Centralities, ect.:* When we want to maintain distances from a single node explicitly (without queries), a fundamental question is to beat trivially calling Dijkstra's static algorithm after each update, taking $\Theta(n^2)$ update time on dense graphs. A better time complexity was not known even with amortization. It was known to be improbable for *exact* algorithms and for *combinatorial* any-approximation algorithms to polynomially beat the $\Omega(n^2)$ bound (under some conjectures) [Roditty, Zwick, ESA'04; Abboud, V. Williams, FOCS'14].[1] Our algorithm with $I = \{s\}$ and $J = V(G)$ implies a $(1 + \epsilon)$-approximation algorithm for this, guaranteeing $\tilde{O}(n^{1.823}/\epsilon^2)$ worst-case update time for directed graphs with positive real weights in $[1, W]$.[2] With ideas from [Roditty, V. Williams, STOC'13], we also obtain the first subquadratic worst-case update time for $(5/3 + \epsilon)$-approximating the eccentricities and $(1.5 + \epsilon)$-approximating the diameter and radius for unweighted graphs (with small additive errors). We also obtain the first subquadratic worst-case update time for $(1 + \epsilon)$-approximating the closeness centralities for undirected unweighted graphs.**

***Worst-case update time for APSP:* When we want to maintain distances between all-pairs of nodes explicitly, the $\tilde{O}(n^2)$ *amortized* update time by Demetrescu and Italiano [STOC'03] already matches the trivial $\Omega(n^2)$ lower bound. A fundamental question is whether it can be made *worst-case*. The state-of-the-art algorithm takes $\tilde{O}(n^{2+2/3})$ worst-case update time to maintain the distances exactly [Abraham, Chechik, Krinninger, SODA'17; Thorup STOC'05]. When it comes to $(1+\epsilon)$ approximation, this bound is still higher than calling the $\tilde{O}(n^\omega/\epsilon)$-time static algorithm of Zwick [FOCS'98], where $\omega \approx 2.373$. Our algorithm with $I = J = V(G)$ implies nearly tight bounds for this, namely $\tilde{O}(n^2/\epsilon^{1+\omega})$ for undirected unweighted graphs and $\tilde{O}(n^{2.045}/\epsilon^2)$ for directed graphs with positive real weights. Besides this, we also obtain the first dynamic APSP algorithm with subquadratic update time and sublinear query time.**

***Keywords*-data-structures; dynamic algorithms; diameter; radius; eccentricities; single-source distances; all-pairs distances; approximate;**

I. INTRODUCTION

Dynamic graph algorithms generally concern maintaining properties of a graph under a sequence of updates, typically in the form of an edge insertion, deletion or weight update. Among basic primitives extensively studied are various distance information; e.g., the all-pairs shortest paths (APSP), the single-source shortest paths (SSSP), and the st-shortest path (st-SP) concern the distances between all-pairs of nodes, from a single node to every node, and between a pair of nodes, respectively.[3] These problems have been studied in settings where distances can be *queried* (e.g. [2]–[7]) or are *explicitly* maintained (e.g. [8]–[10]). In this paper, we study the problem that captures many aforementioned problems as special cases. In this problem, one can query the distances between any two sets of nodes I and J, as follows.

Problem 1.1 (Dynamic batch-query distances)**.** *An algorithm for this problem supports the following operations.*

- PREPROCESS*(G): Process an input n-node graph G with positive real edge weights from $[1, W]$.*
- UPDATE*($(u, v), w$): Update the weight of edge (u, v) to $w \in [1, W] \cup \{\infty\}$.*[4]

The full version of this paper is available as [1] at https://arxiv.org/abs/1909.10850.

[1] The conditional lower bounds of [Roditty, Zwick, ESA'04; Abboud, V. Williams, FOCS'14] hold for algorithms with $O(n^{3-\delta})$ preprocessing time for some constant $\delta > 0$. Our preprocessing time is also in this form. "Combinatorial algorithms" is a vague term referring to algorithms that do not use fast matrix multiplication.

[2] **Notations:** Throughout, n and m denote the number of nodes and edges respectively. Our focus is on dense graphs with $m = \Theta(n^2)$ edges. Let $V(G)$ denote the set of nodes in a graph G. Unless specified otherwise, "weighted graphs" refer to graphs with positive real edge weights. The $\tilde{O}$ notation hides $\operatorname{poly}\log n$, $\operatorname{poly}\log(1/\epsilon)$ (for $(1 + \epsilon)$-approximation algorithms), and during the introduction a single $\log W$ factor.

[3] Note that some works also considered returning the shortest paths, not just the distances, and a *node update* where all edges incident to the same nodes are updated together. This paper does not consider this.

[4] Setting edge weight to ∞ is equivalent to deleting an edge.

2575-8454/19/$31.00 ©2019 IEEE
DOI 10.1109/FOCS.2019.00035

- Query*(I, J): Given sets of nodes I and J, return the distance from i to j, denoted by* $\operatorname{dist}(i, j)$*, for each $(i, j) \in I \times J$.*

For example, the explicit dynamic SSSP where we maintain the distances from a pre-specified node s to every node after changing the weight of an edge is a special case of the above problem where we fix $I = \{s\}$ and $J = V(G)$ (where $V(G)$ is the set of all nodes in G), and the query is made after every update.

Sometimes, we call the query in Problem 1.1 the *batch query* to distinguish it from a typical query of the distance between two nodes. We refer to the case where $W = 1$ as the *unweighted case* (setting an edge weight to 1 and ∞ corresponds to inserting and deleting an edge). To keep things short, we use *weighted graphs* to refer to graphs with positive real edge weights in $[1, W]$ (like in Problem 1.1) throughout.

The performance of algorithms for Problem 1.1 is measured by *preprocessing time*, *update time* and *query time*. The update time can be categorized into two types: A more desirable one is the *worst-case* update time which holds for every single update. This is to contrast with an *amortized* update time which holds "on average".[5] Our focus is on the *worst-case* update time and *dense* graphs with $m = \Theta(n^2)$ edges.

In this paper, we present a fast $(1+\epsilon)$*-approximation* algorithm for Problem 1.1. By $(1+\epsilon)$-approximation, we mean that it answers $\text{Query}(I, J)$ with $d'(i, j) \in [\operatorname{dist}(i, j), (1+\epsilon)\operatorname{dist}(i, j)]$ for every $(i, j) \in I \times J$. We state our algorithm's performance below for completeness but recommend the reader to skip on the first read. It might also be helpful to focus the bounds below for unweighted undirected graphs, where $\omega = 2$, $\varepsilon = 0.01$, and $s \leq 1/4$. In this case, the update and query time complexities in Theorem 1.2 become $\tilde{O}(n^{2-s})$ and $\tilde{O}(|I||J| + |I|n^{2s} + |J|n^{2s})$, which can be traded-off using the parameter s.

Theorem 1.2 (Main General Result)**.** *For any $0 < s < 1$, there exists a randomized (Monte Carlo) $(1+\epsilon)$-approximation algorithm for Problem 1.1 whose preprocessing time,* worst-case *update time, and query time on directed weighted graphs (respectively undirected unweighted graphs) are*

- *Preprocessing: $\tilde{O}(n^{\omega+s}/\varepsilon)$ (respectively $\tilde{O}(n^{\omega+s})$),*
- *Update: $\tilde{O}((n^{1.5286+s} + n^{\omega(1,1,1-s)-1+2s} + n^{\omega(1,1-s,1-s)})/\varepsilon^2)$ (respectively $\tilde{O}((n^{1.5286+s} + n^{\omega(1,1,\mu+s)-\mu} + n^{\omega(1,\mu+s,1-s)} + n^{(1-s)\omega})/\varepsilon^{\omega+1})$, and*
- *Query: $\tilde{O}(n^{\omega(\delta_1,1-s,\delta_2)}/\varepsilon^2)$ (respectively $\tilde{O}(n^{\omega(\delta_1,s+\mu,\delta_2)}/\varepsilon)$), for $|I| = n^{\delta_1}$ and $|J| = n^{\delta_2}$).*

The $\tilde{O}$ notation hides $\operatorname{poly}\log(n)$ and $\operatorname{poly}\log(1/\epsilon)$. To simplify our discussions, we hide the $\log W$ term in this section; we emphasize that our algorithms have only a single $\log W$ factor in their complexities. Here ω is the matrix multiplication exponent, i.e. multiplying two $n \times n$ matrices requires $O(n^\omega)$ time (the current best bound is $\omega \leq 2.3729$ [11], [12]) and $\omega(a, b, c)$ is the exponent for multiplying an $n^a \times n^b$ matrix with an $n^b \times n^c$ matrix. For bounds on $\omega(a, b, c)$ we refer to the appendix of the full version of this paper [1]. The claimed algorithm, as well as other algorithms that follow, guarantee worst-case update and query times and are randomized in that they return the results within the guaranteed approximation ratio with high probability[6]. Note that unlike typical randomized dynamic algorithms, our algorithms do *not* need the oblivious adversary assumption; i.e. an edge update can depend on the algorithms' prior outputs.

A. Consequences

Our result leads to improved algorithms for several variants of dynamic distance maintenance. This includes maintaining the *diameter*, the *radius*, and the *eccentricities*. It answers some key questions in the studies of dynamic shortest paths. Below are some of these questions (more explanations will follow).

Question 1.3 (Beating static algorithms)**.** *Can we beat trivially calling static algorithms after every update? In particular, can we (i) explicitly maintain the SSSP in* subquadratic *update time (even with amortization), and (ii) explicitly and $(1+\epsilon)$-approximately maintain the APSP faster than $\tilde{O}(n^\omega)$* worst-case *update time?*

Question 1.4 (De-amortization)**.** *Can we achieve a worst-case update time comparable to the best known amortized update time? In particular, can we explicitly, and perhaps approximately, maintain the APSP in $\tilde{O}(n^2)$ worst-case update time?*

SSSP: Beating static algorithms (Question 1.3) is the first step in tackling any dynamic problems. It has been achieved for a great number of problems, but unfortunately not for a basic problem like SSSP where we want to explicitly maintain distances between a pre-specified node t and all other nodes: There was no exact or $(1+\epsilon)$-approximation algorithm that beats calling Dijkstra's $O(m + n\log n)$-time static algorithm after every update, causing $\Theta(n^2)$ update time on dense graphs. This is in contrast to the *partially-dynamic* setting, where a $(1+\epsilon)$-approximation algorithm with $\tilde{O}(n)$ amortized update time was known essentially since 1981 [13] (see, e.g. [14]–[20] for some recent im-

---

[5] More precisely, for any t, an algorithm is said to have an amortized update time of t if, for any k, the total time it spends to process the first k updates is at most kt.

[6] With high probability (w.h.p.) means with probability at least $1 - 1/n^c$ for any constant $c > 1$.

provements).[7] The question for the fully-dynamic setting was raised in, e.g., [8].

It was known that polynomially beating this bound is *improbable* for exact algorithms: there is no *exact* dynamic SSSP algorithm with $O(n^{3-\delta})$ preprocessing time and $O(n^{2-\delta})$ amortized update time for any constant $\delta > 0$, assuming the so-called APSP conjecture [21], [22].[8] A natural question is whether this can be achieved with a $(1+\epsilon)$-*approximation* algorithm. Theorem 1.2 shows that this is the case: it implies $\tilde{O}(n^{1.863})$ worst-case update time for SSSP (after the $\tilde{O}(n^{2.708})$-time preprocessing) for directed weighted graphs. This is simply because, for $|I| = 1$, $|J| = n$, and an appropriate choice of s, the preprocessing time, update time, and query time in Theorem 1.2 become $\tilde{O}(n^{2.708})$, $\tilde{O}(n^{1.863})$, $\tilde{O}(n^{1.666})$, respectively for directed weighted graphs. For unweighted directed graphs, the bounds become $O(n^{2.621})$, $\tilde{O}(n^{1.823})$, and $\tilde{O}(n^{1.45})$, implying slightly lower time complexities. By adjusting a small part of the proof of Theorem 1.2, we get slightly better bounds:

Theorem 1.5. SSSP in subquadratic update time; Details in Theorem 5.8): *There is a $(1+\epsilon)$-approximation algorithm for maintaining SSSP explicitly for directed weighted graphs in $\tilde{O}(n^{1.823}/\varepsilon^2)$ worst-case update time after the $\tilde{O}(n^{2.621})$-time preprocessing.*

In the best future scenario, when $\omega = 2$, the update time in Theorem 1.5 would become $\tilde{O}(n^{1.75}/\varepsilon^2)$. Prior to our work, the only method to beat the $\Theta(n^2)$ update time was to run a static algorithm (e.g. Dijkstra's) on top of a *dynamic sparse spanner*, giving approximation factors of three or more on undirected graphs. For example, Bernstein, Forster, and Henzinger [23] can maintain a $(2k-1)$-spanner of $\tilde{O}(n^{1+1/k})$ edges in $O(1)^k \log^3(n)$ worst-case update time, for any constant $k \geq 1$ and for undirected graphs. This allows us to 3-approximate SSSP on undirected graphs in $\tilde{O}(n^{1.5})$ worst-case update time. Since spanners work only for *undirected* graphs and cause the distances to increase by a multiplicative factor of at least three, its use is fundamentally limited to such graphs and approximation guarantee. Our algorithm avoids spanners completely and works for directed graphs with much lower approximation ratio.

APSP: The algorithm with $\tilde{O}(n^{2.75})$ worst-case update time of Thorup [9] was among the first that addressed the issue of worst-case update time (Question 1.4). Despite much recent effort and progress on this issue[9], the only improvement over Thorup's bound was the $\tilde{O}(n^{2+2/3})$ worst-case update time by Abraham, Chechik, and Krinninger [10]. This bound holds for directed weighted graphs and can be improved to $\tilde{O}(n^{2.5})$ on directed *unweighted* graphs. (Update: Recently, after our publication, Wulff-Nilsen and Probst analyzed worst-case APSP in the deterministic and Las Vegas setting [33].)

In fact, the above results are all for maintaining distances *exactly*. When it comes to maintaining $(1+\epsilon)$-*approximate* distances, they are not better than trivially running a static algorithm after every update. On dense directed weighted graphs, this takes $\tilde{O}(n^{\omega}/\epsilon)$ worst-case update time due to Zwick's algorithm [34]. (Again, $\omega \approx 2.3729$.) In other words, there was no $(1+\epsilon)$-approximation algorithm that beats static algorithms (Question 1.3). Theorem 1.2 implies algorithms that do not only break the static $\tilde{O}(n^{\omega}/\epsilon)$ bound, but are also nearly tight:

Theorem 1.6 (APSP in almost-quadratic worst-case update time; Details in Theorem 5.7)**.** *There is a $(1+\epsilon)$-approximation algorithm for maintaining all-pairs-distances explicitly in (i) $\tilde{O}(n^2/\varepsilon^{\omega+1})$ worst-case update time after the $\tilde{O}(n^{2.53})$-time preprocessing for undirected unweighted graphs, and (ii) $\tilde{O}(n^{2.045}/\varepsilon^2)$, worst-case update time after the $O(n^{2.873})$-time preprocessing for directed weighted graphs.*

This is simply because for $I = J = V(G)$ and an appropriate choice of s, the preprocessing time, worst-case update time, and query times in Theorem 1.2 become $O(n^{2.53})$, $\tilde{O}(n^2)$, and $\tilde{O}(2)$ for undirected unweighted graphs, and $O(n^{2.873})$, $\tilde{O}(n^{2.045})$ and $\tilde{O}(n^{2.045})$ for directed weighted graphs. Prior to our algorithms, the only way to beat Zwick's static algorithm was via dynamic spanners; e.g., the aforementioned algorithm of Bernstein et al. [23] implies a 5-approximation algorithm with $\tilde{O}(n^{2+1/3})$ worst-case update time and an $O(1/\epsilon)$-approximation algorithm with $\tilde{O}(n^{2+\epsilon})$ worst-case update time. As mentioned earlier, this approach is fundamentally limited to undirected graphs and rather large approximation ratios.

Note that while previous algorithms [9], [10] can also return the shortest path connecting two nodes in time proportional to the length of the path, our algorithms only maintain the distances. Also, previous algorithms can handle a more general update where the weights of all edges incident to the same node are updated at once. Our algorithms only handle the standard edge updates. Due to the so-called "Johnson transformation"[35], [36], previous algorithms can also handle negative edge weights when there are no negative cycles. Since this transformation applies only for exact distance computation, it does not apply to our algorithms.

For a version of APSP where we can make a query for

---

[7] Recall that the partially-dynamic setting is when edge weighs can be only increased or only decreased. [13] originally presented an exact algorithm with $O(n)$ amortized update time for unweighted undirected graphs. It was observed later than this can be easily extended to $\tilde{O}(n)$ amortized update time for weighted directed graphs.

[8] The result of [21], [22] does not rule out an exact algorithm with higher preprocessing time and $O(n^{2-\delta})$ update time. Finding such algorithm remains a major open problem.

[9] See, e.g., [9], [10], [23]–[32].

a distance between two nodes, Theorem 1.2 implies a $(1+\epsilon)$-approximation algorithm with *subquadratic* update and *sublinear* query time complexities (both are in the worst case).

Theorem 1.7 (APSP in subquadratic update and sublinear query time; Details in Theorem 5.1)**.** *There is a $(1+\epsilon)$-approximation algorithm for maintaining APSP with $\tilde{O}(n^{1.863}/\varepsilon^2)$ worst-case update time and $\tilde{O}(n^{0.666}/\varepsilon^2)$ worst-case query time after the $\tilde{O}(n^{2.708})$-time preprocessing for directed weighted graphs, and (ii) $\tilde{O}(n^{1.823}/\varepsilon^{\omega+1})$ worst-case update time and $\tilde{O}(n^{0.45}/\varepsilon^{\omega+1})$ worst-case query time after the $O(n^{2.621})$-time preprocessing for undirected unweighted graphs.*

The only previously known algorithm with subquadratic update and query time complexities (but not with a sublinear query time) was by Sankowski [7]. It outputs exact distances for directed unweighted graphs. Our algorithm works on weighted graphs with much lower query time, but only returns approximate distances.

Diameter, Radius, and Eccentricities: The eccentricities (the eccentricity of a node v is the largest distance from v to any another node), the diameter (the maximum over all eccentricities) and the radius (the minimum over all eccentricities) can be easily maintained in $\tilde{O}(n^2)$ amortized update time using Demetrescu and Italiano's dynamic APSP algorithm [8]. An important challenge here is to *break the $\tilde{O}(n^2)$ bound*. This captures a fundamental question of whether we really need APSP to maintain less informative measurements like the diameter. It was known that algorithms with $n^{2-\delta}$ amortized update time might not exist for $(1.5-\epsilon)$-approximating the diameter, $(1.5-\epsilon)$-approximating the radius, and $(5/3-\epsilon)$-approximating the eccentricities for any constants $\delta, \epsilon > 0$ (assuming either the Strong Exponential Time Hypothesis (SETH) or a version of the Hitting Set Hypothesis) [37].[10] In other words, likely we cannot break the $O(n^2)$ bound with such approximation guarantees.

In this paper, we show that with slightly higher approximation guarantees we can break the $O(n^2)$ bound: by essentially simulating the static algorithm of Roditty and V. Williams [39] (with some small adjustments), using the algorithm in Theorem 1.2 to compute distances when needed, we obtain algorithms with a subquadratic worst-case update time that nearly $(1.5+\epsilon)$-approximate the diameter and radius, and nearly $(5/3+\epsilon)$-approximate the eccentricities for dynamic graphs. By "nearly", we mean that there are some additive errors smaller than one. The results, when we slightly adjust the proof of Theorem 1.2 to get some slight improvements, are as follows.[11]

Theorem 1.8 (Approximating Diameter, Radius, and Eccentricities; Details in Theorem 6.1)**.** *We write $\operatorname{diam}(G)$, $\operatorname{radius}(G)$ for the diameter and radius of graph G respectively, and let $ecc(v, G)$ be the eccentricity of node v in G. There exist algorithms that can maintain the following values with the following time complexities for a dynamic graph G.*

1) $\tilde{D} \in \left[\left(\frac{2}{3}-\varepsilon\right)\operatorname{diam}(G) - 1/3, (1+\varepsilon)\operatorname{diam}(G)\right]$ *in $\tilde{O}(n^{1.779}/\varepsilon^{1+\omega})$ worst-case update time after the $O(n^{2.624})$-time preprocessing,*
2) $\tilde{R} \in [\operatorname{radius}(G)/(1+\varepsilon), (1.5+\varepsilon)\operatorname{radius}(G) + 2/3]$ *in $\tilde{O}(n^{1.779}/\varepsilon^{1+\omega})$ worst-case update time after the $O(n^{2.624})$-time preprocessing, and*
3) $\widetilde{ecc}(v) \in \left[\left(\frac{3}{5}-\varepsilon\right) ecc(v, G) - 4/7, ecc(v, G)\right]$ *for all nodes v in $\tilde{O}(n^{1.823}/\varepsilon^{1+\omega})$ worst-case update time after the $O(n^{2.621})$-time preprocessing.*

The algorithm for Diameter works for directed unweighted graphs, while the others work for undirected unweighted graphs.

Prior to our algorithms, one can guarantee similar approximation ratios by running the static algorithms after each update (e.g. the nearly 1.5-approximation $\tilde{O}(m^{3/2})$-time algorithms of [39], [40] for Diameter and Radius, and the nearly $(5/3)$-approximation $\tilde{O}(m^{3/2})$-time algorithms of [39], [40] for Eccentricities). (See, e.g., [39]–[43] for results in the static setting.) Obviously, even a static linear-time algorithm cannot break the $\tilde{O}(n^2)$ bound for dense graphs. The only prior method to break the $\tilde{O}(n^2)$ bound was to run static algorithms that are subquadratic-time on sparse graphs on top of a dynamic sparse spanner; e.g., the aforementioned spanner algorithm of Bernstein et al. [23] allows us to nearly-7.5-approximate the diameter on directed unweighted graphs in $\tilde{O}(n^{1+1/2+1/3})$ worst-case update time (using the static 1.5-approximation $\tilde{O}(m\sqrt{n})$-time algorithm of [39] for Diameter). As mentioned earlier, this method is limited to large approximation ratio and undirected graphs.

Recently, Ancona et al. presented *partially-dynamic* algorithms with approximation guarantees similar to us, e.g. a nearly-$(1.5+\epsilon)$-approximation decremental algorithm with $m^{1+o(1/\epsilon)}\sqrt{n}/\epsilon^2$ expected total update time for unweighted undirected graphs [37, Cor. 1.1]. (A slower algorithm was presented in [44].) Both Ancona et al.'s and our algorithms essentially simulate the algorithms of Roditty and V, Williams [39] (with small adjustments to the algorithms and analyses). The main difference is that our algorithms rely on our new result in Theorem 1.2, while Ancona et al.'s

---

[10] [37] also ruled out algorithms with $m^{1-\delta}$ update time that is $(2-\epsilon)$-approximation on undirected unweighted graphs (under SETH), and finite approximation on directed unweighted graphs (under the k-Cycle Hypothesis), strengthening [38].

[11] Without the adjustment, our algorithms guarantee the following: nearly $(1.5+\epsilon)$ approximation factor and $O(n^{1.863}/\varepsilon^{\omega+1})$ time for Diameter, nearly $(1.5+\epsilon)$ approximation factor and $O(n^{1.823}/\varepsilon^{\omega+1})$ time for Radius, and nearly $(5/3+\epsilon)$ approximation factor and $O(n^{1.823}/\varepsilon^{\omega+1})$ time for Eccentricities. Note that the adjustment does not improve the update time for Eccentricities.

algorithms rely on the recent developments on partially-dynamic shortest paths (e.g. [3], [14]).

In addition to the above, we can also maintain the diameter *exactly* for directed unweighted graphs (or with small positive integer weights bounded by W). It is the first that improves trivially running the static exact algorithms by Shoshan and Zwick[45] or Cygan, Gabow and Sankowski [46]. These algorithms take $\tilde{O}(Wn^{\omega}) \approx \tilde{O}(Wn^{2.3729})$ worst-case update time, and ours improves this to $\tilde{O}(Wn^{2.3452})$. The algorithm can be found in the full version.

Further, we give the first subquadratic dynamic algorithm for $(1+\varepsilon)$-approximating all-closeness-centralities with $O(n^{1.823}/\varepsilon^{\omega+1})$ update time the full version. The closeness centrality of a node v is the inverse of the average distance from v. The technique can be used to obtain a static $(1+\varepsilon)$-approximate algorithm with $\tilde{O}(mn^{2/3}/\varepsilon^2)$ time algorithm, which improves upon the $\tilde{O}(m \cdot \operatorname{diam}(G)^2/\varepsilon^2)$ algorithm from [47] for large diameter graphs.

Techniques: At the heart of our result is a combination of the standard hitting argument and an algorithm that exploits fast matrix multiplication to quickly answer queries about approximate *bounded-hop* distances between two sets of nodes. An exact counterpart of this algorithm was already known due to Sankowski [7], but the query time of Sankowski's algorithm is too slow for our purpose. Our approximation algorithm's query time is faster, but its update time is slower. Interestingly, we run Sankowski's algorithm in parallel with our algorithm, since our algorithm needs some information from it during the updates. Our algorithm needs to make a small number of queries to Sankowski's algorithm, thus does not suffer from its slow query time. We explain this more in Section II.

We note that our algorithms heavily rely on fast matrix multiplication algorithms. This is known to be necessary even for st-Reachability and $(1+\epsilon)$-approximating st-distance on undirected unweighted graphs [22].[12] It is an intriguing question, however, whether fast matrix multiplication is necessary for other problems (see Section VII). Also note that fast matrix multiplication has been used in many previous shortest paths data-structures (e.g. [6], [7], [48]–[53]).

II. TECHNICAL OVERVIEW

In this section we outline how to obtain Theorem 1.2. For simplicity we will only consider the case of directed, unweighted graphs. The algorithm outlined here can easily be extended to small integer weights, which then allows an extension to real weights via weight rounding – a technique previously used in [34], [54]–[57].

We will outline the proof of the following Theorem. It is equivalent to Theorem 1.2, except that it only supports unweighted graphs.

Theorem 2.1 (Theorem 1.2 restricted to directed, unweighted graphs)**.** *For any $0 < s < 1$, there exists a randomized (Monte Carlo) $(1+\epsilon)$-approximation algorithm for Problem 1.1 whose preprocessing time,* worst-case *update time, and query time on directed unweighted graphs are*

- *Preprocessing: $\tilde{O}(n^{\omega+s})$,*
- *Update: $\tilde{O}(n^{1.5286+s} + n^{\omega(1,1,1-s)-1+2s}/\varepsilon + n^{\omega(1,1-s,1-s)}/\varepsilon)$, and*
- *Query: $\tilde{O}(n^{\omega(\delta_1,1-s,\delta_2)})$, for $|I| = n^{\delta_1}$ and $|J| = n^{\delta_2}$.*

The outline of this section is as follows: We first give the high-level idea of how to reduce Theorem 2.1 to the algebraic problem of maintaining the inverse of some polynomial matrix modulo X^h for some $h > 0$. This reduction is outlined in the first subsection II-A and uses common techniques used in many other algorithms (i.e. weight rounding, hitting sets, and inverse of polynomial matrices). Readers familiar with dynamic algebraic algorithms for maintaining graph distances can skip ahead to the next subsection II-B. In section II-B we highlight the difference and new techniques that allow us to maintain approximate distances quickly, whereas previous algebraic data-structures could only maintain exact distances. We also give a high-level description of our new data-structure and explain how we are able to break the long-standing $\Omega(n^2)$-bottleneck for problems such as single-source distances.

A. Basic Tools

In this subsection we outline how to reduce Theorem 2.1 to some algebraic problem. For that we first reduce the problem to maintaining only h-hops distances for some $h > 0$, and then reduce the problem further to the algebraic problem of maintaining the inverse of some polynomial matrix modulo X^h.

Restriction to short hops: In order to create an algorithm as stated in Theorem 2.1, it is enough to construct an algorithm that only maintains h-hops distances. Specifically, it is enough to prove the following Lemma.

Lemma 2.2 (Proven as Theorem 4.2 in Section IV)**.** *Let G be an unweighted graph with n nodes. Then for any $0 \leq \mu$, $0 \leq s \leq 1$ and $\varepsilon > 0$ there exists a dynamic algorithm that maintains $(1+\varepsilon)$-approximate n^s-hops all-pairs-distances of G. Each edge update requires $\tilde{O}(n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + n^{1.5286+s})$ time. For any $I, J \subset V$, we can query the approximate distances for the pairs $I \times J$ in $\tilde{O}(n^{\omega(\delta_1,\mu+s,\delta_2)}/\varepsilon)$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.*

We now outline why Lemma 2.2 is enough to obtain Theorem 2.1. The formal proof is given in Section V.

---

[12] Triangle detection can be reduced to $(6/5-\varepsilon)$-approximate st-distance and $(3/2-\varepsilon)$-approximate SSSP.

A common technique for graph algorithms is a so-called "hitting-sets". More accurately, for some $h \in \mathbb{N}$, a uniformly at random chosen set $H \subset V$ of size $\tilde{O}(n/h)$ has w.h.p. the property that every shortest path with at least h hops can be decomposed into segments $s \to h_1 \to ... \to h_k \to t$, where each $h_i \in H$ and each segment uses at most h hops. This technique goes back to Ullman and Yannakakis [58].

Let $\hat{D}$ be an h-hop distance matrix, and denote for any $I, J \subset V$ with $\hat{D}_{I,J}$ the submatrix corresponding to the pairs $(u, v) \in I \times J$. Then the length of shortest paths, using more than h hops, can be computed via the $(\min, +)$-product $\hat{D}_{V,H}\hat{D}^{|H|}_{H,H}\hat{D}_{H,V}$. Using techniques from [34], we can compute a $(1 + \varepsilon)$-approximation of $\hat{D}_{V,H}\hat{D}^{|H|}_{H,H}$ for $h = n^s$ in $\tilde{O}(n^{\omega(1,1-s,1-s)}/\varepsilon)$ time during each update (this is the third term in the update time in Theorem 2.1).

Whenever we must answer a query for the pairs $I \times J$, we then compute $(\hat{D}_{V,H}\hat{D}^{|H|}_{H,H})_{I,H}\hat{D}_{H,J}$ in $\tilde{O}(n^{\omega(\delta_1,1-s,\delta_2)})$ time and return the element-wise minimum with $\hat{D}_{I,J}$. Thus for $\mu := 1 - 2s$ Lemma 2.2 implies Theorem 2.1.

From Graphs to Polynomial Matrices: The task of maintaining h-hop distances can be reduced to maintaining the *inverse* of a polynomial matrix $M \in (\mathbb{F}[X]/\langle X^{h+1}\rangle)^{n\times n}$ (i.e. a matrix whose entries are polynomials modulo X^{h+1}). In general the inverse of M might not exist, because $\mathbb{F}[X]/\langle X^{h+1}\rangle$ is a ring, not a field. However, for the special case that M is of the form $M = \mathbb{I} - A \cdot X$, where $\mathbb{I}$ is the identity matrix, the inverse is given by $M^{-1} = \sum_{i=0}^{h} A^i \cdot X^i$.[13] A similar technique was previously used by Sankowski in [7], where Sankowski used the adjoint of a polynomial matrix, instead of the inverse. For the reduction of Lemma 2.3 below, we need the following notation: For a polynomial matrix $M \in (\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$ we define $M^{[k]}$ to be the coefficients of X^k, so $M^{[k]} \in \mathbb{F}^{n\times n}$ for any $k < d$ and $M = \sum_{i=0}^{d-1} M^{[k]}X^k$.

Lemma 2.3 (Proven in the full version)**.** *Let G be a directed graph with positive integer edge weights $(c_{u,v})_{(u,v)\in E}$ and let d be some positive integer. We define $A(G) \in (\mathbb{F}[X]/\langle X^d\rangle)^{n\times n}$, such that $A_{u,v} = a_{u,v}X^{c_{u,v}}$ for each edge (u, v), $A_{v,v} = a_{v,v}X$ for every $v \in V$, and $A_{u,v} = 0$ otherwise. Here each $a_{u,v}$ is an independent and uniformly at random chosen element from $\mathbb{F}$.* [14]

Then, the matrix $M = \mathbb{I} - A(G)$ is invertible and with probability at least $1 - dn^2/|\mathbb{F}|$ the following property holds: For every $u, v \in V$ and $0 \le k < d$ the entry $(M^{-1})^{[k]}_{u,v} \neq 0$ if and only if $\operatorname{dist}(u, v) \le k$.

Assume we have a data-structure that allows us to maintain $(M^{-1})^{[k]}$ for $k \in S = \{\lfloor(1 + \varepsilon)^i\rfloor \mid 0 \le i \le \lceil\log_{(1+\varepsilon)} n^s\rceil\}$. Then the smallest $k \in S$ with $(M^{-1}_{s,t})^{[k]} \neq 0$ is a $(1 + \varepsilon)$-approximation of $\operatorname{dist}(s, t)$ according to Lemma 2.3. Thus Lemma 2.2 can be obtained by creating a data-structure that maintains $(M^{-1})^{[k]}$ for $k \in S$.

For the field $\mathbb{F}$ we will use $\mathbb{Z}_p$ for a prime of bit-length $\Theta(\log(dn))$, then the result is correct with high probability. The variable d will typically be polynomial in n, so each arithmetic operation in $\mathbb{Z}_p$ can be performed in $\tilde{O}(1)$ time. In summary, we can obtain an algorithm as stated in Lemma 2.2 by proving the following lemma:

Lemma 2.4. *Let $0 \le \mu$, $0 \le s$, $h := n^{s+1}$ and $M = (\mathbb{I} - A \cdot X) \in (\mathbb{F}[X]/\langle X^h\rangle)^{n\times n}$ be a polynomial matrix modulo X^h. Let $S \subset \{1, ..., h\}$ be any set.*

Then there exists a dynamic algorithm that supports element updates to A, that requires $O(n^{s+\omega})$ field operations for the pre-processing and $\tilde{O}(|S|n^{\omega(1,s+\mu,1)-\mu} + n^{1.5286+s})$ operations per update. The algorithm supports queries to $(M^{-1})_{I,J}$ for any $I, J \subset [n]$, where it returns $(M^{-1})^{[d]}_{I,J}$ for all d $\in S$ in $\tilde{O}(|S|n^{\omega(\delta_1,\mu+s,\delta_2)})$ operations. Here δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.

B. Proof Sketch of Lemma 2.4

As outlined near the end of the previous subsection, we can obtain Lemma 2.2 by proving Lemma 2.4 via some chain of reductions.

Before we will prove this result, we first want to compare it to other dynamic algebraic algorithms by explaining how our algorithm manages to break a long-standing bottleneck of dynamic algebraic distance algorithms. The main differences of Lemma 2.4 compared to previous dynamic algebraic algorithms for distances (e.g. [7], [52]) is that our algorithm maintains $(M^{-1})^{[k]}$ for $k \in S$ for any set $S \subset [n^s]$, whereas previous algebraic algorithms for distances maintain $(M^{-1})^{[k]}$ for *all* $k = 1, 2, ..., n^s$, i.e. they were restricted to the special case $S = [n^s]$.

This difference allows us to circumvent a long-standing $\Omega(n^2)$ bottleneck for algebraic algorithms that maintain single-source distances. Remember from the previous subsection, that these algebraic data-structures are used to maintain the n^s-hop distances of some $\tilde{O}(n^{1-s})$-sized hitting set H to all other nodes V. Following the reduction of Lemma 2.3, this means maintaining the submatrix $M^{-1}_{H,V}$, which unfortunately means that just the output-size is already $\Omega(n^2)$: We have a $|H| \times |V| = \Omega(n^{1-s}) \times n$ sized submatrix, where each entry is a polynomial with n^s coefficients, i.e. a total of $\Omega(n^2)$ field elements. However, our algorithm does not maintain all coefficients of $M^{-1}_{H,V}$; we just maintain the coefficients of monomials of degree $k \in S$. So for $|S| \ll n^s$, the output size is smaller than $\Omega(n^s)$. For example, when maintaining $(1+\varepsilon)$-approximate distances we have $|S| = O(\varepsilon^{-1}\log n)$ as outlined at the end of the previous subsection II-A. Previous dynamic algebraic algorithms could not break this $\Omega(n^2)$ bottleneck as they were restricted to the special case $S = [n^s]$.

---

[13] This is because $X^{h+1} = 0$ in $\mathbb{F}[X]/\langle X^{h+1}\rangle$, so $(\sum_{i=0}^{h} A^i \cdot X^i) \cdot (\mathbb{I} - A \cdot X) = \sum_{i=0}^{h} A^i \cdot X^i - \sum_{i=1}^{h+1} A^i \cdot X^i = \mathbb{I}$.

[14] Note that for $c_{u,v} \ge d$ we have $A_{u,v} = 0$, as if the edge would not exist.

Reference	Element update	Batch query	Remark
[7]	$\tilde{O}(n^{\omega(1,\mu,1)+s-\mu} + n^{1.5286+s})$	$\tilde{O}(n^{\omega(\delta_1,\mu,\delta_2)+s})$	Maintains $(M^{-1})^{[k]}$ for $k \in [n^s]$.
Lemma 2.4	$\tilde{O}(\|S\|n^{\omega(1,s+\mu,1)-\mu} + n^{1.5286+s})$	$\tilde{O}(\|S\|n^{\omega(\delta_1,\mu+s,\delta_2)})$	Maintains $(M^{-1})^{[k]}$ for $k \in S \subset [n^s]$.

Figure 1. Comparison of data-structures for maintaining $(M^{-1})^{[k]}$ for $k \in S \subset [n^s]$. The data-structure of [7] supports only the special case $S = [n^s]$.

When comparing Lemma 2.4 with [7] specifically (see Figure 1), then our algorithm manages to replace the n^s factor by having it as an argument to $\omega(\cdot,\cdot,\cdot)$, i.e. the term $n^{\omega(1,\mu,1)+s-\mu}$ in the update time of [7] changes to $|S|n^{\omega(1,s+\mu,1)-\mu})$. This allows us to greatly exploit fast matrix multiplication. Consider for example the case $\omega = 2$, then $|S|n^{\omega(1,s+\mu,1)-\mu} = |S|n^{2-\mu}$, but $n^{\omega(1,\mu,1)+s-\mu} = n^{2-\mu+s}$. So when the set $S \subset [n^s]$ has $|S| \ll n^s$, then our algorithm is a lot faster.

Proof idea of Lemma 2.4: In order to complete our proof sketch of Theorem 2.1, we are left with proving Lemma 2.4. The algorithm is based on the following modified variant of the Sherman-Morrison-Woodbury identity [59], [60]:

Lemma 2.5 (Paraphrased, Formal statement in Lemma 4.6)**.** *Let M be an invertible $n \times n$ matrix and let $M_{(t)}$ be the matrix M after changing any t entries, such that $M_{(t)}$ is invertible.*

Then there exists n-dimensional vectors $u_{(1)}, ..., u_{(t)}$, $v_{(1)}, ..., v_{(t)}$ that are given by rescaled rows and columns of M^{-1}, such that

$$M_{(t)}^{-1} = M^{-1} - \sum_{i=1}^{t} u_{(i)} v_{(i)}^{\top}.$$

Assume we know M^{-1} because we computed it during the pre-processing, then one can create a dynamic algorithm by simply adding one new pair $u_{(t+1)}, v_{(t+1)}$ for every update. This new pair can be computed quickly, as they are just a row/column of the already computed M^{-1}. For answering queries to $M_{(t)}^{-1}$, one must simply compute some entries of the sum $\sum_{i=1}^{t} u_{(i)} v_{(i)}^{\top}$ and subtract them from the corresponding entries of M^{-1}. From time to time, when the sum grows too large and the queries too slow, the data-structure will reset by computing $M_{(t)}^{-1}$ explicitly, i.e. we set $M \leftarrow M_{(t)}$. (This is where the trade-off parameter n^{μ} for update and query time comes from.)

Performing the queries and reset operation naively results in the data-structure of [7] as presented in Figure 1. We now outline how we are able to speed-up both of these operations.

Query operation: Consider the sum $\sum_{i=1}^{t} u_{(i)} v_{(i)}^{\top}$, then we can write it as a matrix product $UV^{\top}$ of $(n \times t)$-matrices U, V, where the ith columns are $u_{(i)}$ and $v_{(i)}$ respectively. For answering the query of some submatrix $(M_{(t)}^{-1})_{I,J} = M_{I,J}^{-1} - (UV^{\top})_{I,J}$, we must multiply the rows I of U with the columns J of $V^{\top}$. Note however, that we only need to compute the coefficients of degree k for $k \in S$, i.e. $(UV^{\top})_{I,J}^{[k]}$ instead of $(UV^{\top})_{I,J}$. This allows for some speed-up via the following lemma:

Lemma 2.6. *Let $0 \leq a, b, c, d$ and let $U \in \mathbb{F}[X]^{n^a \times n^b}, V \in \mathbb{F}[X]^{n^c \times n^b}$ be polynomial matrices of degree at most n^d. Then we can compute for any k the kth coefficient $(UV^{\top})^{[k]}$ in $\tilde{O}(n^{\omega(a,b+d,c)})$ field operations.*

Proof: We have $(UV^{\top})^{[k]} = \sum_{i=0}^{d} U^{[i]} V^{[k-i]\top} = \left[U^{[0]} \mid U^{[1]} \mid ... \mid U^{[k]}\right] \left[V^{[k]} \mid V^{[k-1]} \mid ... \mid V^{[0]}\right]^{\top}$. This product can be computed in $\tilde{O}(n^{\omega(a,b+d,c)})$ field operations. ■

This way we are able to compute $(M_{(t)}^{-1})_{I,J}^{[k]} = (M_{I,J}^{-1})^{[k]} - (UV^{\top})_{I,J}^{[k]}$ for all $k \in S \subset [n^s]$ in $\tilde{O}(|S|n^{\omega(\delta_1,s+\mu,\delta_2)})$ operations, when $|I| = n^{\delta_1}$, $|J| = n^{\delta_2}$, $t \leq n^{\mu}$. This is exactly the query complexity of Lemma 2.4.

Reset operation: After the data-structure received n^{μ} updates, the matrices U, V are too large to answer the queries quickly enough. Thus we "reset" the algorithm by assigning $M \leftarrow M_{(t)}$ and computing M^{-1} explicitly.

Note that for answering queries to

$$(M_{(t)}^{-1})^{[k]} = (M^{-1})^{[k]} - (UV^{\top})^{[k]}$$

for $k \in S$, we do not need to know the entire M^{-1}. It is enough to know only $(M^{-1})^{[k]}$ for all $k \in S$. Thus we can speed-up the reset by computing only those coefficients. The number of operations required for that is just $\tilde{O}(|S|n^{\omega(1,\mu+s,1)})$, as this is equivalent to answering a query for $I = J = [n]$. Since this reset happens only after every n^{μ} updates, this yields the $\tilde{O}(|S|n^{\omega(1,\mu+s,1)-\mu})$ term in the update time of Lemma 2.4.

This idea of the improved reset operations leads to the following problem: If we do not know all coefficients of M^{-1}, then we can not obtain the new vectors $u_{(t+1)}, v_{(t+1)}$. Previously we said that those vectors can be trivially obtained by just reading rows/columns of M^{-1}. As we do not have all coefficients of M^{-1}, we then do not have all coefficients of $u_{(t+1)}, v_{(t+1)}$ or U, V either. Thus we can not use Lemma 2.6 to answer queries anymore.

The solution to this problem is to run Sankowski's algorithm in parallel. This algorithm was used in [7] to maintain exact distances, so in graph algorithms context, this means we are running the exact distance algorithm from [7] inside our approximate distance algorithm. At first, this might be a bit counter-intuitive, as one might wonder how an approximate algorithm can be fast, if it needs to run an exact algorithm anyway. We explain in the next paragraph, why for many problems (APSP, SSSP, diameter etc.) our approximate algorithm is faster than Sankowski's exact one, even though we run it internally.

Running exact and approximate algorithms in parallel: By running both an exact and an approximate algorithm in parallel, we are able to exploit their benefits to fix the other algorithm's disadvantages.

Dynamic algorithms often allow for a trade-off between update and query time, which is something we exploit in our dynamic approximate distance algorithm, by running the two algorithms in parallel:

- Sankowski's algorithm from [7] is exact, with improved update time at the cost of a slower query (i.e. large choice of μ in Figure 1), but it only needs to answer very few queries (one row/column per update) to obtain $u_{(t+1)}, v_{(t+1)}$.
- The other algorithm is approximate, with improved query time (so we can answer large hitting set queries), at the cost of a slower update. However, because of the approximation, this "slower" update time is still quite fast.

By combining the two algorithms, they are able to compensate each other's disadvantages: As outlined before, our approximate algorithm can not run on its own, so we run Sankowski's exact algorithm in parallel. However, if one were to run only Sankowski's algorithm, then it would be very slow for maintaining the distances of the hitting-set to all other nodes (i.e. $O(n^2/h)$ entries of the inverse, when maintaining h-hop distances). [15] By running it internally inside our approximate algorithm, the exact algorithm only needs to compute a single row and column of the inverse, so only $O(n)$ entries per update as opposed to $O(n^2/h)$.

III. PRELIMINARIES

Complexity Measures: Most of our algorithms work over any field $\mathbb{F}$ and their complexity is measured in the number of arithmetic operations performed over $\mathbb{F}$, i.e. the *arithmetic complexity*. This does not necessarily equal the *time complexity* of the algorithm as one arithmetic operation could require more than $O(1)$ time, e.g. very large rational numbers could require many bits for their representation. This is why our algebraic lemmas and theorems will always state "in $O(\cdot)$ operations" instead of "in $O(\cdot)$ time". Further, $\tilde{O}(\cdot)$ hides $\operatorname{polylog} n$ and $\operatorname{polylog} \varepsilon$ factors, so unlike the introduction, the formal proofs will no longer hide any $\log W$ factor.

For the graph applications however, when having an n node graph, we will typically use the field $\mathbb{Z}_p$ for some prime p of order n^c for some c. This means each field element requires only $O(c \log n)$ bits to be represented and all field operations can be performed in $\tilde{O}(c)$ time in the standard model. For our final results this c will be constant, i.e. all arithmetic operations can be performed in $\tilde{O}(1)$.

Notation: Identity and Submatrices: The identity matrix is denoted by $\mathbb{I}$. Let $I, J \subset [n] := \{1, ..., n\}$ and A be an $n \times n$ matrix, then the term $A_{I,J}$ denotes the submatrix of A consisting of the rows with index in I and columns with index in J. For some $i \in [n]$ we may also just use the index i instead of the set $\{i\}$. For example the term $A_{[n],i}$ refers to the ith column of A.

Matrix Multiplication: We denote with $O(n^\omega)$ the arithmetic complexity of multiplying two $n \times n$ matrices. Currently the best bound is $\omega < 2.3729$ [11], [12].

For rectangular matrices we denote the complexity of multiplying an $n^a \times n^b$ matrix with an $n^b \times n^c$ matrix with $O(n^{\omega(a,b,c)})$ for any $0 \le a, b, c$. Note that $\omega(\cdot,\cdot,\cdot)$ is a symmetric function, so we can reorder the arguments. Also by splitting a matrix product into several smaller products, we have $O(n^{\omega(a,b,c+d)}) = O(n^{\omega(a,b,c)+d})$. The current best bounds for $\omega(1,1,c)$ can be found in [61]. To see how to bound general $\omega(a,b,c)$, we refer to the full version [1].

Polynomials modulo X^d: All our algebraic results use polynomials modulo X^d for some positive integer d. The ring of such polynomials is denoted by $\mathbb{F}[X]/\langle X^d \rangle$. Given two polynomials $p, q \in \mathbb{F}[X]/\langle X^d \rangle$, we can add and subtract the two polynomials in $O(d)$ operations in $\mathbb{F}$. We can multiply the two polynomials in $O(d \log d)$ using fast-fourier-transformations. If q is of the form $c - X \cdot h$, $c \in \mathbb{F} \setminus \{0\}, h \in \mathbb{F}[X]/\langle X^d \rangle$, then q is invertible with $q^{-1} = c^{-1} \sum_{k=0}^{d-1} (Xh/c)^k = c^{-1} \prod_{k=0}^{\log d} (1 + (Xh/c)^{2^k}$ so the inverse can be computed in $O(d(\ln d)^2)$ operations. Since we typically hide polylog factors in the $\tilde{O}(\cdot)$ notation, all arithmetic operations with polynomials from $\mathbb{F}[X]/\langle X^d \rangle$ can be performed in $\tilde{O}(d)$ operations in $\mathbb{F}$.

Polynomial Matrices: We will work with polynomial matrices and vectors $M \in (\mathbb{F}[X]/\langle X^d \rangle)^{n \times n}$, $\vec{v} \in (\mathbb{F}[X]/\langle X^d \rangle)^n$, so matrices and vectors whose entries are polynomials modulo X^d. Products of such matrices/vectors can be performed as usual, but since each arithmetic operations of two entries requires $\tilde{O}(d)$ field operations, the complexity increases by a factor of $\tilde{O}(d)$. For example two $n \times n$ matrices can be multiplied in $\tilde{O}(dn^\omega)$ field operations.

Note that not every matrix $M \in (\mathbb{F}[X]/\langle X^d \rangle)^{n \times n}$ has an inverse, (even if $\det(M) \neq 0$) as $\mathbb{F}[X]/\langle X^d \rangle$ is a ring. However, we will only invert matrices of the form $M = \mathbb{I} - X \cdot A$, where $A \in (\mathbb{F}[X]/\langle X^d \rangle)^{n \times n}$. The inverse of these matrices is given via $\sum_{i=0}^{d-1} X^k A^k = \prod_{k=0}^{\log d} (\mathbb{I} + A^{2^k})$, which can be computed in $\tilde{O}(dn^\omega)$.

For a polynomial matrix $M \in (\mathbb{F}[X]/\langle X^d \rangle)^{n \times n}$ we define $M^{[k]}$ to be the matrix of coefficients of X^k. So $M = \sum_{i=0}^{d-1} M^{[k]} X^k$ and $M^{[k]} \in \mathbb{F}^{n \times n}$.

$(1+\varepsilon)$-approximate h-hop distance matrix: Given an n-node graph G we call an $n \times n$ matrix D a $(1+\varepsilon)$-approximate h-hop distance matrix, if

---

[15] One can apply Lemma 2.6 to turn Sankowski's algorithm into an approximate algorithm and thus improving the query time a bit, but this approach will not result in an algorithm as fast as our approximate one, because the time required to reset Sankowski's algorithm is a lot larger than ours.

- $\operatorname{dist}_G(u,v) \leq D_{u,v} \leq (1+\varepsilon)\operatorname{dist}_G(u,v)$, if the shortest uv-path uses at most h hops.
- $\operatorname{dist}_G(u,v) \leq D_{u,v}$ for all other pairs $u,v \in V$.

IV. ALGEBRAIC DYNAMIC SHORT HOP DISTANCES

In this section we prove the main tool used for our new results. This new tool allows us to maintain approximate bounded hop distances in a dynamic graph. We will later extend this algorithm to work on paths of any hop length in Section V.

The main result in this section will be the following theorem:

Theorem 4.1 (n^s-hop distances, approximate, positive real weights)**.** *Let G be a graph with n nodes and real edge weights from $[1,W]$. Then for any $0 \leq \mu, s \leq 1$ and $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate n^s-hops all-pairs-distances of G.*

The preprocessing time is $\tilde{O}((n^{s+\omega}/\varepsilon)\log W)$. Each edge update requires $\tilde{O}((n^{\omega(1,s+\mu,1)-\mu}/\varepsilon^2 + n^{1.5286+s}/\varepsilon)\log W)$ time.

For any $I, J \subset V$, we can query the approximate distances for the pairs $I \times J$ in $\tilde{O}(n^{\omega(\delta_1,\mu+s,\delta_2)}/\varepsilon^2 \log W)$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.

The proof will be split into two parts: First, we will prove an equivalent result for graphs with integer edge weights in Section IV-A. Then we will extend the result to work on graphs with real edge weights in Section IV-B.

A. Exact and Approximate Distances for Integer Weights

We first start with the case of integer weights, we we will later extend the algorithm to real weights. The integer version of Theorem 4.1 can be formulated as follows:

Theorem 4.2. *Let G be a graph with n nodes and positive integer edge weights. Then for any $0 \leq s, \mu$, there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G upto n^s.*

The preprocessing time is $\tilde{O}(n^{s+\omega})$. Each edge update requires $\tilde{O}(s^2 n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + sn^{1.5286+s})$ time.

For any $I, J \subset V$, we can query the approximate distances upto n^s for the pairs $I \times J$ in $\tilde{O}(s^2 n^{\omega(\delta_1,\mu+s,\delta_2)}/\varepsilon)$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$. For pairs $P \subset I \times J$ with distance larger than n^s, the returned distance is ∞.

The high-level idea of the algorithm for Theorem 4.2 was already given in Section II.

As previously stated, Theorem 4.2 is the result of maintaining the inverse of a polynomial matrix and using the reduction of Lemma 2.3. As such, our first task is to create a new algorithm for maintaining the inverse of a polynomial matrix.

Lemma 4.3. *Let $0 \leq \mu, s$ and $M = (\mathbb{I} - A \cdot X) \in (\mathbb{F}[X]/\langle X^{n^s}\rangle)^{n \times n}$ be a polynomial matrix modulo X^{n^s}. Let $S \subset [n^s]$ be any set and let $u(d,n)$ be a bound on update and query time of Lemma 4.4 for an $n \times n$ polynomial matrix modulo X^d. Then there exists a dynamic algorithm that performs $\tilde{O}(n^{s+\omega})$ operations during the pre-processing and $\tilde{O}(|S| n^{\omega(1,s+\mu,1)-\mu} + u(n^s,n))$ operations per element update to A.*

The algorithm supports queries to $(M^{-1})^{[d]}_{I,J}$ for any $I, J \subset [n]$ and $d \in S$ in $\tilde{O}(n^{\omega(\delta_1,\mu+s,\delta_2)})$ operations, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.

As already stated in Lemma 4.3 and Section II, we build our algorithm upon the following result by Sankowski [7]:

Lemma 4.4 ([7, Theorem 3])**.** [16]

Let $0 \leq \nu, s$ and $M = (\mathbb{I} - A \cdot X) \in (\mathbb{F}[X]/\langle X^{n^s}\rangle)^{n \times n}$ be a polynomial matrix modulo X^{n^s}.

Then there exists a dynamic algorithm that supports element updates to A in $\tilde{O}(n^{s+\omega(1,1,\nu)-\nu} + n^{1+s+\nu})$ operations and both row and column queries to M^{-1} in $\tilde{O}(n^{1+s+\nu})$ operations.

The pre-processing requires $\tilde{O}(n^{s+\omega})$ operations.

For current ω the update and query time are $O(n^{1.5286+s})$ for $\nu \approx 0.5285$.

The algorithm of Lemma 4.4 could be modified to support batch queries to $M^{-1}_{I,J}$ with $|I| = n^{\delta_1}$, $|J| = n^{\delta_2}$ in $\tilde{O}(n^{s+\omega(\delta_1,\nu,\delta_2})$, which is the result we stated in Section II Figure 1, when choosing $\nu = \min(\mu, 0.5285)$.

The high level idea of Lemma 4.3 is to express the changes to the matrix M as rank-1 updates of the form $M + uv^\top$. For such updates the new inverse of $(M+uv^\top)^{-1}$ is given via the Sherman-Morrison identity.

Lemma 4.5 (Sherman-Morrison)**.** *Let M be an $n \times n$ matrix and u, v be n-dimensional vectors, then*

$$(M + uv^\top)^{-1} = M^{-1} - M^{-1}u(1 + v^\top M^{-1} u)^{-1} v^\top M^{-1}.$$

A dynamic algorithm receives several updates in online sequence. For this purpose one could use the Sherman-Morrison-Woodbury identity, however, given the online nature of the updates (i.e. the updates are given one-by-one) the following incremental variant of Sherman-Morrison is more useful:

Lemma 4.6. *Let M be an $n \times n$ matrix and $u_{(1)}, ..., u_{(k)}$, $v_{(1)}, ..., v_{(k)}$ be n-dimensional vectors. Define $M_{(t)} := M + \sum_{i=1}^{t} u_{(i)} v_{(i)}^\top$ for $t = 0, ..., k$, so $M_{(t)} = M_{(t-1)} + u_{(t)} v_{(t)}^\top$. Further define $\hat{u}_{(t)} := M_{(t-1)}^{-1} u_{(t)}$ and $\hat{v}_{(t)} := (1 + v_{(t)}^\top \hat{u}_{(t)})^{-1} v_{(t)}^\top M_{(t-1)}^{-1}$ for $t = 1, ..., k$.*

---

[16] [7, Theorem 3] maintains the adjoint modulo X^{n+1}, but as stated in the proof of [7, Theorem 6] the algorithm can also be used modulo X^{n^s} in which case the complexity is as stated in Lemma 4.4. In [7] Sankowski considered maintaining the adjoint for the case, where the input matrix M has degree 1, but the algorithm can also be used to maintain the inverse for matrices of any degree bounded by n^s, as proven in [52, Appendix C1, C2].

Then for all for $t = 0, ..., k$

$$M_{(t)}^{-1} = M^{-1} - \sum_{i=1}^{t} \hat{u}_{(i)} \hat{v}_{(i)}^\top.$$

Proof: Via Lemma 4.5 we know

$$\begin{aligned} M_{(t)}^{-1} &= (M_{(t-1)} + u_{(t)} v_{(t)}^\top)^{-1} \\ &= M_{(t-1)}^{-1} \\ &\qquad - M_{(t-1)}^{-1} u_{(t)} (1 + v_{(t)}^\top M_{(t-1)}^{-1} u_{(t)})^{-1} v_{(t)}^\top M_{(t-1)}^{-1} \\ &= M_{(t-1)}^{-1} - \hat{u}_{(t)} (1 + v_{(t)}^\top \hat{u}_{(t)})^{-1} v_{(t)}^\top M_{(t-1)}^{-1} \\ &= M_{(t-1)}^{-1} - \hat{u}_{(t)} \hat{v}_{(t)}^\top \end{aligned}$$

so by induction $M_{(t)}^{-1} = M^{-1} - \sum_{i=1}^{t} \hat{u}_{(i)} \hat{v}_{(i)}^\top$, because $M_{(0)}^{-1} = M^{-1}$. ■

We now have all tools available to prove Lemma 4.3.

Proof of Lemma 4.3:

We will first give the high-level idea: Let M be the input matrix during the pre-processing and $M_{(k)}$ be the matrix after k updates. We will express the element updates to M via rank-1 updates, so for every update we receive a pair of vectors u, v, i.e. adding $s \in F$ to entry (i, j) of $M_{(k)}$ is the same as adding the outer-product $uv^\top$ for $u = f \cdot e_i$, $v = g \cdot e_j$ and some $f, g \in \mathbb{F}[X]/\langle X^{n^s} \rangle$.

This means after k updates, we are tasked with maintaining the inverse of $M_{(k)} = M + \sum_{i=1}^{k} u_{(i)} v_{(i)}^\top$, where $u_{(i)}, v_{(i)}$ is the pair of vectors that specify the ith update, and each vector $u_{(i)}, v_{(i)}$ has only one non-zero entry.

Thanks to Lemma 4.6 we know

$$(M + \sum_{i=1}^{k} u_{(i)} v_{(i)}^\top)^{-1} = M^{-1} - \sum_{i=1}^{k} \hat{u}_{(i)} \hat{v}_{(i)}^\top.$$

The high-level idea is to compute the vectors $\hat{u}_{(i)}, \hat{v}_{(i)}$ after every update. These vectors are useful for the following reason: Let $\hat{U}, \hat{V}$ be the $k \times n$ matrices, where the ith column is $\hat{u}_{(i)}$ and $\hat{v}_{(i)}$ respectively. Then $\sum_{i=1}^{k} \hat{u}_{(i)} \hat{v}_{(i)}^\top = \hat{U} \hat{V}^\top$ and thus

$$((M + \sum_{i=1}^{k} u_{(i)} v_{(i)}^\top)^{-1})^{[d]} = (M^{-1})^{[d]} - (\hat{U} \hat{V}^\top)^{[d]}.$$

So by applying Lemma 2.6 to the product $\hat{U} \hat{V}^\top$, we can easily answer the queries.

To make sure the matrices $\hat{U}, \hat{V}$ do not become too large, we will reset the algorithm after n^μ updates.

Pre-processing: We initialize Lemma 4.4 for matrix M and we compute $(M^{-1})^{[k]}$ for every $k \in S$. This is done by computing M^{-1} in $\tilde{O}(n^{s+\omega})$ operations.

Update: Assume we handle the kth update, i.e. we receive the pair $u_{(k)}, v_{(k)}$ and have $M_{(k)} = u(k) v_{(k)}^\top$. The update routine consists of the following steps:

1) Compute $\hat{u}_{(k)} := M_{(k-1)}^{-1} u_{(k)}$ and $\hat{v}_{(k)} := (1 + v_{(k)}^\top \hat{u}_{(k)})^{-1} v_{(k)}^\top M_{(t-1)}^{-1}$ as defined in Lemma 4.6.
2) Let $\hat{U}, \hat{V}$ be the $n \times k$ matrices, where the ith columns is $\hat{u}_{(i)}, \hat{v}_{(i)}$ respectively.
3) Update the algorithm of Lemma 4.4.

Note that the data-structure of Lemma 4.4 is always updated at the end in step 3. Thus at the start of the kth update, we can query rows and columns of $M_{(k-1)}^{-1}$ via the data-structure of Lemma 4.4.

This allows us to compute $\hat{u}_{(k)} := M_{(k-1)}^{-1} u_{(k)}$ in $\tilde{O}(n^{1+s} + u(n^s, n)) = \tilde{O}(u(n^s, n))$ operations as follows: The vector $u_{(k)}$ has only one non-zero element, so $M_{(k-1)}^{-1} u_{(k)}$ is just one column of $M_{(k-1)}^{-1}$ scaled by the non-zero entry of $u_{(k)}$. The column can be queried via Lemma 4.4 in $O(u(n^s, n))$ operations, and multiplying each of the n entries of that column by the non-zero entry of $u_{(k)}$ needs $\tilde{O}(n^s)$ operations. Thus a total of $\tilde{O}(n^{1+s} + u(n^s, n))$ operations is required, which can be bounded by $\tilde{O}(u(n^s, n))$.

Likewise, $\hat{v}_{(k)} := (1 + v_{(k)}^\top \hat{u}_{(k)})^{-1} v_{(k)}^\top M_{(t-1)}^{-1}$ can be computed in $\tilde{O}(u(n^s, n))$ operations: The vector $v_{(k)}$ has just one non-zero entry, so $v_{(k)}^\top M_{(t-1)}^{-1}$ is just one row of $M_{(t-1)}^{-1}$, scaled by the non-zero entry of $v_{(k)}$. This can be computed in the same way as $M_{(k-1)}^{-1} u_{(k)}$, except that, instead of a columns, we now query a row of $M_{(t-1)}^{-1}$ in $O(u(n^s, n))$ operations via Lemma 4.4. The polynomial $(1 + v_{(k)}^\top \hat{u}_{(k)})^{-1}$ can be computed in $\tilde{O}(n^{1+s})$ as we have an inner product of two n dimensional vectors of degree n^s, and inverting the resulting degree d polynomial needs only $\tilde{O}(n^s)$ operations. Note that the inverse $(1 + v_{(k)}^\top \hat{u}_{(k)})^{-1}$ exists, because we can assume, without loss of generality, that $v_{(k)}$ is of the form $X \cdot v$ for some $v \in (\mathbb{F}[X]/\langle X^{n^s} \rangle)^n$, so $v_{(k)}$ has no constant terms. This is because, by assumption of Lemma 4.3, element updates to $M = \mathbb{I} - X \cdot A$ are actually element updates to A, which is multiplied with X. Hence $(1 + v_{(k)}^\top \hat{u}_{(k)}) = 1 + X \cdot p(X)$, for some polynomial $p(X)$, and it is thus invertible (see preliminaries about inverting polynomials).

For step 2, note that $\hat{u}_{(i)}, \hat{v}_{(i)}$ for $i < k$ were already computed during the previous updates. So for each update, the matrices $\hat{U}, \hat{V}$ change by just adding one column, given by $\hat{u}_{(k)}$ and $\hat{v}_{(k)}$ respectively. This means step 2 requires only $O(n^{1+s})$ operations.

In summary, an update requires $\tilde{O}(u(n^s, n))$ operations, because the $\tilde{O}(n^{1+s})$ term is subsumed.

Query: When we want to compute the submatrix $((M + \sum_{(i=1}^{k} u_{(i)} v_{(i)}^\top)^{-1})_{I,J}^{[d]}$ for some $d \in S$ and $I, J \subset [n]$, then we need to compute $(\hat{U}_{I,[k]} \hat{V}_{[k],J}^\top))^{[d]}$ and subtract it from $(M^{-1})_{I,J}^{[d]}$. The latter is known because of the pre-processing and the former can be computed in $\tilde{O}(n^{\omega(\delta_1, s+\mu, \delta_2)})$ operations via Lemma 2.6, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.

Reset: After upto n^μ updates, we reset the algorithm. For this we must compute $((M + \sum_{i=1}^{k} u_{(i)} v_{(i)}^\top))^{-1})^{[d]}$ for all $d \in S$. Afterward we set $M \leftarrow M + \sum_{i=1}^{k} u_{(i)} v_{(i)}^{\top}$.

The matrices $((M+\sum_{i=1}^{k} u_{(i)} v_{(i)}^{\top})^{-1})^{[d]}$ can be computed the same way as in the query phase for $I = J = [n]$ in $\tilde{O}(|S| n^{\omega(1,s+\mu,1)})$ operations. This leads to an amortized update complexity of $\tilde{O}(|S| n^{\omega(1,s+\mu,1)-\mu} + u(n^s, n))$ operations per update. This can be made worst-case via the standard-technique of running two copies of this algorithm in parallel and spreading out the reset computation over several updates. While one copy of the algorithm is performing the reset, the other copy is able to answer the queries. For a formal proof of this standard-technique see the full version of the paper.

■

Now that Lemma 4.4 is proven, we can finally apply the reduction from Lemma 2.3 to obtain data-structures for distance problems.

Corollary 4.7. *Let $0 \leq \mu, s$ and let G be an n-node graph with positive integer weights. Let $S \subset [n^s]$ be any subset. Let $u(d, n)$ be a bound on update and query time of Lemma 4.4 for an $n \times n$ matrix modulo X^d.*

Then there exists a Monte Carlo dynamic algorithm with $\tilde{O}(sn^{s+\omega})$ pre-processing and $\tilde{O}(|S| s n^{\omega(1,s+\mu,1)-\mu} + su(n^s, n))$ worst-case update time for each edge update.

For any $I, J \subset V$ and $k \in S$, we can query for the pairs $I \times J$ the boolean matrix, which answers for all $s \in I$, $t \in J$, if the distance from s to t is at most k. Each such query requires $\tilde{O}(sn^{\omega(\delta_1,\mu+s,\delta_2)})$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$.

Proof:

Let G be the given graph and $A(G)$ be the matrix as defined in Lemma 2.3, then for $M := \mathbb{I} - A(G)$ we have that $(M^{-1})_{u,v}^{[k]} \neq 0$ if and only if $\operatorname{dist}(u, v) \geq k$. Thus we simply maintain M^{-1} via Lemma 4.4, where each edge update to G corresponds to an element update to $A(G)$.

The extra factor s in the complexity of Corollary 4.7 compared to Lemma 4.4 comes from the fact that we now measure the time instead of arithmetic operations. For Lemma 2.3 to hold with high probability, we must use $\mathbb{F} = \mathbb{Z}_p$ where the prime p has bit-length $\Theta(s \log n)$, so one arithmetic operation requires $\tilde{O}(s)$ time in the standard model.

■

Corollary 4.7 works for some arbitrary set $S \subset [n^s]$. To prove Theorem 4.2, we are only left with specifying the correct set S for Corollary 4.7.

Proof of Theorem 4.2: Theorem 4.2 is directly implied by Corollary 4.7 by letting $S = \{\lfloor (1+\varepsilon)^k \rfloor \mid 0 \leq k \leq \lceil \log_{(1+\varepsilon)} n^s \rceil\}$.

We simply run Corollary 4.7 and whenever we ask for the distances of some pairs $I \times J \subset V \times V$, we query for every $k \in S$, if the distance is at most k. This way we obtain $(1+\varepsilon)$-approximate distances, of distance upto n^s.

The complexity of Theorem 4.2 is the same as Corollary 4.7. We have $|S| = O(s/\varepsilon \log n)$, so the update time is $\tilde{O}(s^2 n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + su(n^s, n))$ and the query time is $\tilde{O}(s^2 n^{\omega(\delta_1,\mu+s,\delta_2)}/\varepsilon)$.

■

B. Approximate Distances for Real Weights

In the previous subsection we handled the case of graphs with positive integer edge weights. We now extend the results to the case of real edge weights. The technique is based on the integer rounding trick used in [34, Lemma 8.1, Theorem 8.2].

Theorem 4.8. *Let $0 < \varepsilon, 0 \leq \delta$. Given a dynamic algorithm that maintains $(1+\delta)$-approximate distances upto $3h/\varepsilon$ on graphs with integer weights, then there exists a dynamic algorithm for $(1+\delta)(1+\varepsilon)$-approximate h-hop distances on graphs with real weights from $[1, W]$. The update, query and pre-processing complexity all increase by a factor of $O(\log(Wn))$.*

Before proving Theorem 4.8, we observe that it immediately implies Theorem 4.1.

Proof of Theorem 4.1: We can use Theorem 4.2 to maintain $(1+\varepsilon)$-approximate distances upto $3n^s/\varepsilon$. Via Theorem 4.8 we then obtain $(1+\varepsilon)^2$-approximate n^s-hop distances for real weighted graphs. By choosing a slightly smaller approximation factor, we can also obtain $(1+\varepsilon)$-approximate n^s-hop distances.

Compared to Theorem 4.2 the complexities increase by a factor of $O(\varepsilon^{-1} \log \varepsilon^{-1} \log(nW))$. Here the factor $O(\varepsilon^{-1} \log \varepsilon^{-1})$ comes from maintaining the distance upto $O(n^s/\varepsilon)$ and the factor $O(\log(nW))$ comes from Theorem 4.8. ■

Lemma 4.9. *Let $G = (V, E)$ be a graph with n nodes and real edge weights from $[1, W]$. For any $0 < A, B$ define $G' = (V, E')$ to be the graph with $E' = \{(u, v) \in E \mid c_{u,v} \leq B\}$ and integer edge weights $c'_{u,v} = \lceil A c_{u,v} / B \rceil$.*

Then for any path from s to t in G of length $\operatorname{dist}_G(s,t) \leq B$ let h be the number of hops. We have $\operatorname{dist}_G(s,t) \leq (B/A) \operatorname{dist}_{G'}(s,t) \leq \operatorname{dist}_G(s,t) + (B/A)h$.

Proof: Since $\operatorname{dist}_G(s,t) \leq B$, all edges used by the path in G also exist in G'. Because of the rounding, we have $\operatorname{dist}_G(s,t) \leq (B/A) \operatorname{dist}_{G'}(s,t)$ and each used edge can cause an error of at most B/A, so $(B/A) \operatorname{dist}_{G'}(s,t) \leq \operatorname{dist}_G(s,t) + (B/A)h$. ■

Lemma 4.10. *Let $0 \leq s$, $\varepsilon > 0$ and let $G = (V, E)$ be a graph with n nodes and real edge-weights from $[1, W]$. Define $\lceil \log_2 nW \rceil$ graphs G_i as in Lemma 4.9 for $B_i = 2^i$, $A = 2n^s 1/\varepsilon$ for $i = 1, ..., \lceil \log_2 nW \rceil$.*

Then for any pair $s, t \in V$ we have $\operatorname{dist}_G(s,t) \leq \min_i (B_i/A) \operatorname{dist}_{G_i}(s,t)$ and if the shortest st-path uses at *most n^s hops, then we also have*

$$\min_i (B_i/A) \operatorname{dist}_{G_i}(s,t) \leq (1+\varepsilon) \operatorname{dist}_G(s,t).$$

Proof: The first inequality

$$\operatorname{dist}_G(s,t) \leq \min_i (B_i/A) \operatorname{dist}_{G_i}(s,t)$$

follows directly from Lemma 4.9. The second inequality follows from the following observation: Let i be such that $2^{i-1} \leq \operatorname{dist}_G(s,t) \leq 2^i$ and let h be the number of hops for the shortest st-path. Then this path in G also exists in G_i and $(B_i/A) \operatorname{dist}_{G_i}(s,t) \leq \operatorname{dist}_G(s,t) + (B_i/A)h \leq \operatorname{dist}_G(s,t) + \varepsilon \operatorname{dist}_G(s,t) n^{-s} h$. So if the number of hops h is at most n^s, then we obtain the promised $(1+\varepsilon)$-approximation.

■

Proof of Theorem 4.8: Consider the graphs G_i for $i = 1, ..., \lceil \log nW \rceil$ from Lemma 4.10. The largest n^s-hop distance in any G_i is bounded by $n^s A = 3n^s/\varepsilon$. So when the dynamic algorithm for integer weights can maintain the distance upto $3n^s/\varepsilon$, then it can maintain n^s-hop distances for all graphs G_i for $i = 1, ..., \lceil \log nW \rceil$.

Thus we let the given algorithm run on all G_i and for any distance query, we return $\min_i (B_i/A) \operatorname{dist}_{G_i}(s,t)$ as in Lemma 4.10. This yields $(1+\varepsilon)(1+\delta)$-approximate n^s-hop distances, because the distances in each G_i are only maintained $(1+\delta)$-approximately. ■

Non-oblivious adversaries: Note that all the graph algorithm of Section IV work against non-oblivious adversaries, i.e. updates are allowed to depend on the query results. The only random choices are the random field elements for the non-zero entries of the matrix in Corollary 4.7. The correct return values of Corollary 4.7 are uniquely determined by the input graph and the query-input (I, J, k). The actual values, returned by Corollary 4.7 , are correct with high probability, so with high probability the returned values for any query do not leak any information about the random choices.

V. Results for All-Pairs-Distances

In this section we will prove the result of Theorem 1.2, as presented in the introduction. The result is split into two theorems, one for directed, weighted graphs and one for undirected graphs with small integer weights.

For directed, weighted graphs we will prove the following:

Theorem 5.1 (Approximate APSP, Real weights, Directed, Queries)**.** *Let G be a directed graph with n nodes and real weights from $[1, W]$. Then for any $0 \leq s \leq 1$ and $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G in $\tilde{O}((n^{1.5286+s}/\varepsilon + n^{\omega(1,1,1-s)+1-2s}/\varepsilon^2 + n^{\omega(1-s,1-s,1)}/\varepsilon^2) \log W)$ update time. We can query for any $I, J \subset V$ the distances of the pairs $I \times J$ in $\tilde{O}(n^{\omega(\delta_1, 1-s, \delta_2)}/\varepsilon^2 \log W)$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$. The pre-processing requires $\tilde{O}(n^{s+\omega}/\varepsilon \log W)$ time.*

For current ω and $s \approx 0.334, \mu \approx 0.316$ the update time is $\tilde{O}(n^{1.863}/\varepsilon^2 \log W)$, with query time for a single pair is $\tilde{O}(n^{0.666}/\varepsilon^2 \log W)$. The pre-processing time is $\tilde{O}(n^{2.708} \log W)$.

For undirected graphs we will show the following result:

Theorem 5.2 (Approximate APSP, Unweighted (or with W), Undirected, Queries)**.** *Let G be a directed graph with n nodes and integer weights from $\{1, ..., W\}$ where $W = n^\ell$. Then for any $0 \leq s \leq 1$ and $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G in $\tilde{O}(n^{\omega(1,1,s+\mu+\ell)+\mu}/\varepsilon + u(n^{s+\ell}, n) + n^{\omega(1-s,s+\mu+\ell,1)}/\varepsilon^2 + n^{(1-s)\omega}/\varepsilon^{1+\omega}))$ update time. We can query for any $I, J \subset V$ the distances of the pairs $I \times J$ in $\tilde{O}(n^{\omega(\delta_1, s+\mu+\ell, \delta_2)}/\varepsilon)$ time, where δ_1, δ_2 are such that $|I| = n^{\delta_1}, |J| = n^{\delta_2}$. The pre-processing requires $\tilde{O}(W n^{s+\omega}/\varepsilon)$ time.*

For current ω the pre-processing and update time for an unweighted graph are $\tilde{O}(n^{2.621}/\varepsilon)$ and $\tilde{O}(n^{1.823}/\varepsilon^{3.373})$ respectively with $s \approx 0.248$ and $\mu \approx 0.202$. The query time for a single pair is $O(n^{0.45}/\varepsilon)$.

The high-level idea of all our algorithms is to maintain distances for short hop paths using Theorem 4.1 (and Theorem 4.2 for integer edge weights), and use hitting set arguments to compute distances for large hop paths.

Hitting set arguments, as introduced in [58], allow us to decompose paths with many hops into segments with fewer hops, specifically the following lemma will be an important tool:

Lemma 5.3. *Let G be a graph with n nodes and let $H \subset V$ be a random subset of size $c\frac{n}{d} \ln n$.*

With probability at least $1 - n^{2-c}$ we have that: For every $s, t \in V$, where the shortest st-path uses at least d hops, this shortest st-path can be decomposed into segments $s \rightarrow h_1 \rightarrow h_2 \rightarrow ... \rightarrow h_k \rightarrow t$, where $h_i \in H$ for every $i = 1, ..., k$ and each segment uses at most d hops.

This lemma implies, that to compute some shortest st-path with many hops, we only need to know the distances of paths with few hops between the pairs $(\{s\} \cup H) \times (H \cup \{t\})$. The idea of these hitting-set arguments goes back to [58]. The proof for Lemma 5.3 can be found in the full version.

The other important ingredient for this section are the following two results by [34], which allows us to compute approximate $(\min, +)$-products and approximate distances.

Lemma 5.4 ([34, Theorem 8.2])**.** *Let G be a directed graph with n nodes and real edge weights in $[1, W]$, then we can compute all-pairs-distances of G in $\tilde{O}(n^\omega/\varepsilon \log W)$ time.*

Or in other words, we can compute for any $n \times n$ matrix D with entries in $[1, W]$, a $(1+\varepsilon)$-approximation of the nth power of D using the $(\min, +)$-product.

The original result [34, Theorem 8.1] for the approximate $(\min,+)$-products is stated for square matrices, but it can also be used for rectangular matrices:

Lemma 5.5 ([34, Theorem 8.1])**.** *Let A be an $n^a \times n^b$ and B be an $n^b \times n^c$ matrix, each with real entries from $[1, W]$, then we can compute a $(1+\varepsilon)$-approximate $(\min,+)$-product $A \star B$ in $\tilde{O}(n^{\omega(a,b,c)}/\varepsilon \log W)$ time.*

A. Weighted Approximate Distances (Proof of Theorem 5.1)

We will start with the results for weighted all-pairs-distances. The following lemma allows us to extend the short hop distances of Theorem 4.1 to large hop distances.

Lemma 5.6. *Let G be a directed graph with n nodes and real weights from $[1, W]$. Assume we are already given an $(1+\delta)$-approximate h-hop distance matrix D. Let $H \subset V$ be a random subset of size $c(n/h)\ln n$. Define*

$$\hat{D} := D_{V,H} \star D_{H,H}^{(\star|H|)} \star D_{H,V}, \tag{1}$$

where $\star$ is a $(\min,+)$-product and $(\star|H|)$ is the $|H|$th power using $(\min,+)$-products.

Then for any $u, v \in V$ we have $\operatorname{dist}_G(u,v) \leq \hat{D}_{u,v}$. Further, with probability at least $1 - n^{2-c}$, we have for any $u, v \in V$, where the shortest uv-path has at least h hops, that $\hat{D}_{u,v} \leq (1+\delta)\operatorname{dist}_G(u,v)$. Consequently, we can obtain approximate all-pairs-distances for any number of hops, by taking the entry-wise minimum of D and $\hat{D}$.

Proof: According to Lemma 5.3 we have that (w.h.p) every shortest path using at least h hops can be decomposed into segments $s \to h_1 \to h_2 \to \ldots \to t$ where each $h_i \in H$ and each segment uses at most n^s hops. Thus we obtain the distances of all pairs (u, v) where the shortest uv-path uses at least h hops, by computing the $(\min,+)$-product

$$D_{V,H} \star D_{H,H}^{(\star|V|)} \star D_{H,V}$$

where $(\star|V|)$ refers to the $|V|$th power using the $(\min,+)$-product. Note that the power $D_{H,H}^{(\star|V|)}$ can be reduced to $D_{H,H}^{(\star|H|)}$, because the matrix $D_{H,H}$ can be considered an edge weight matrix of a matrix on node set H and then $D_{H,H}^{(\star|V|)}$ is the all-pairs-distance matrix. Since every shortest path in that graph can use at most $|H|$ hops, the $|H|$th power is enough.

■

Note, for constant c and $h = n^s$ (so $|H| = \tilde{O}(n^{1-s})$) we can compute $D_{H,H}^{(\star|H|)}$ approximately via Lemma 5.4 in $\tilde{O}(n^{(1-s)\omega}/\varepsilon \log W)$ time. The remaining products of (1) can be computed approximately in $\tilde{O}(n^{\omega(1,1-s,1)}/\varepsilon \log W)$ time via Lemma 5.5.

Proof of Theorem 5.1: We maintain the $(1+\varepsilon)$-approximate n^s-hop distances via Theorem 4.1. We want to compute (1) of Lemma 5.6, but split the computation of the $(\min,+)$-product into two parts. After every update we sample a random set of nodes $H \subset V$ of size $\tilde{O}(n^{1-s})$ as in Lemma 5.6, and query the distances for the pairs $H \times V$ and $V \times H$. Let $D_{H,V}, D_{V,H}$ be the obtained distance matrices, then we compute a $(1+\varepsilon)$-approximation of the $(\min,+)$-product $D_{V,H} \star D_{H,H}^{(\star|H|)}$ via Lemma 5.5. The update time is thus

$$\tilde{O}(\underbrace{n^{\omega(1,s+\mu,1)-\mu}/\varepsilon^2 + u(n^s/\varepsilon, n)) \log W}_{\text{Theorem 4.1}}$$
$$+ \underbrace{n^{\omega(1,\mu+s,1-s)}/\varepsilon^2 \log W}_{\text{query } D_{V,H}, D_{H,V}} + \underbrace{n^{\omega(1-s,1-s,1)}/\varepsilon \log W}_{\text{compute } D_{H,H}^{(\star|H|)} \star D_{H,V}}).$$

Queries: When there is some query for the distances of the pairs $I \times J$, then we compute $\hat{D} := D_{I,H} \star D_{H,H}^{(\star|H|)} \star D_{H,J} = (D_{V,H} \star D_{H,H}^{(\star|H|)})_{I,H} \star D_{H,J}$, where $D_{V,H} \star D_{H,H}^{(\star|H|)}$ was already computed during the last update. Each entry yields the distance of the pair $(u, v) \in I \times J$, if the shortest uv-path uses at least n^s nodes (see Lemma 5.6). Thus we also query the n^s-hop distances for the pairs $I \times J$ via Theorem 4.1 and return for each pair $(u, v) \in I \times J$ the minimum of the two distances $\hat{D}_{u,v}$ and $D_{u,v}$. The query time is thus

$$\tilde{O}(\underbrace{n^{\omega(\delta_1,s+\mu,\delta 2)}/\varepsilon^2 \log W}_{\text{query } D_{I,J}} + \underbrace{n^{\omega(\delta_1,1-s,\delta_2)}/\varepsilon \log W}_{\text{compute } \hat{D}}),$$

where δ_1, δ_2 are such that $n^{\delta_1} = |I|$ and $n^{\delta_2} = |J|$.

Technically this maintains a $(1+\varepsilon)^2$ approximation, but we can simply choose a slightly smaller approximation for Theorem 4.1 and Lemma 5.5 to obtain a $(1+\varepsilon)$ approximation. Further, for $1 - s = \mu + s$ we obtain the update/query complexties as stated in Theorem 5.1.

Note that Theorem 4.1 works against non-oblivious adversaries, so Theorem 5.1 works against non-oblivious adversaries as well, because we sample a new random hitting-set H after every update.

■

We can maintain all-pairs-distances explicitly, by querying all distance after every single update. Thus we obtain the following result:

Theorem 5.7 (Approximate APSP, Real weights, Directed, Almost-n^2)**.** *Let G be a directed graph with n nodes and real weights from $[1, W]$. Then for any $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G in $\tilde{O}(n^{\omega(1,1,0.5)}/\varepsilon^2 \log W)$ update time. The pre-processing requires $\tilde{O}(n^{1/2+\omega}/\varepsilon \log W)$ time.*

For current ω the update time is $\tilde{O}(n^{2.045}/\varepsilon^2 \log W)$[61].

Proof: Theorem 5.7 is implied by Theorem 5.1, by performing a query to $I = J = V$ after every update. In that case the update and query time are balanced for $\mu = 0, s = 0.5$, which yields $\tilde{O}(n^{\omega(1,0.5,1)}/\varepsilon^2 \log W) = \tilde{O}(n^{2.044183}/\varepsilon^2 \log W)$ update time. ■

To obtain an algorithm for dynamic single-source distances, we could just use Theorem 5.1 to query the distances from the source-node. However, the update time can be slightly improved as follows:

Theorem 5.8 (Approximate SSSP, Real weights, Directed)**.** *Let G be a directed graph with n nodes and real weights from $[1, W]$. Then for any $0 \leq s, \mu \leq 1$ and $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G in $\tilde{O}((n^{1.5286+s} + n^{\omega(1,1,s+\mu)-\mu} + n^{\omega(1-s,\mu+s,1)})/\varepsilon^2 \log W)$ update time. The pre-processing requires $\tilde{O}(n^{s+\omega}/\varepsilon \log W)$ time.*

For current ω and $s \approx 0.248, \mu \approx 0.202$ the update time is $\tilde{O}(n^{1.823}/\varepsilon^2 \log W)$.

Proof of Theorem 5.8: Let $v \in V$ be the source node for which we want to maintain single source distances. We maintain distances upto n^s hops via Theorem 4.1.

After every update we sample a hitting-set H as in Lemma 5.6. We query the approximate h-hop distances $D_{H\cup\{v\},V}$ via Theorem 4.1 and construct a graph G_H with node set V and edges between every $(u, w) \in (H \cup \{v\}) \times V$ with cost $D_{u,w}$.

Via Lemma 5.3 we know that w.h.p. the single-source distances rooted at v in G_H are the same as in G, so we can compute them in $O(|V||H|) = \tilde{O}(n^{2-s})$ time using Dijkstra. The time for constructing G_H and running Dijkstra is subsumed by querying $D_{H\cup\{v\},V}$, so the update time for the single-source algorithm is $\tilde{O}((n^{\omega(1,s+\mu,1)-\mu}/\varepsilon^2 + u(n^s/\varepsilon, n) + n^{\omega(1-s,s+\mu,1)}/\varepsilon^2) \log W)$.

As Theorem 4.1 works against non-oblivious adversaries, and we sample a new hitting-set H after every update, Theorem 5.8 works against non-oblivious adversaries as well. ■

B. Unweighted Undirected Approximate Distances (Proof of Theorem 5.2)

For undirected graphs we exploit the idea from [5]. The high-level idea is as follows: We have a hitting set H and want to compute the st-distance, while assuming the shortest path uses at least n^s hops. The path can be split into segments $s \to h_1 \to ... \to h_k \to t$ but to find this path, we would usually need to figure out which element of H is h_1. In the case of unweighted, undirected graphs, this can be solved approximately by choosing any $x, y \in H$ with distance $\operatorname{dist}(s, x), \operatorname{dist}(y, t) \leq n^s/2\varepsilon$. Then we know $\operatorname{dist}(x, y) \leq \operatorname{dist}(s, t) + n^s\varepsilon$, because we can find a xy-path via the segments $x \to s \to t \to y$, and also have $\operatorname{dist}(s, t) \leq \operatorname{dist}(x, y) + n^s\varepsilon$, because we can fine a st-path via the segments $s \to x \to y \to t$. Thus we have $(1-\varepsilon)\operatorname{dist}(x, y) \leq \operatorname{dist}(s, t) \leq (1+\varepsilon)\operatorname{dist}(x, y)$ for unweighted undirected graphs where the shortest st-path uses at least n^s hops.

Lemma 5.9. *Let G be an undirected graph with n nodes and integer edge weights in $\{1, 2, ..., W\}$. Let $H \subset V$ be a random subset of size $2cn/(h\varepsilon)\ln h$ for some constant $c > 0$. Assume we are given $(1+\varepsilon)$-approximate distances $D_{H\times V}$, where all pairs u, v with $\operatorname{dist}_G(u, v) > Wh$ are allowed to have $D_{u,v} > (1+\varepsilon)Wh$.*

Then we can construct a distance oracle $\hat{D}$ in $\tilde{O}((n/h)^\omega/\varepsilon^{1+\omega})$ time, such that each query $\hat{D}(u, v)$ for any $u, v \in V$ requires only $O(1)$ time, and with probability at least n^{2-c}:

- $\operatorname{dist}_G(u, v) \leq \hat{D}_{u,v}$
- $\hat{D}_{u,v} \leq (1+\varepsilon)^3 \operatorname{dist}_G(u, v)$ *if* $\operatorname{dist}_G(u, v) \geq Wh$.

Proof: According to Lemma 5.3, with probability at least n^{2-c}, every shortest path using at least $0.25h\varepsilon$ hops can be decomposed into segments $s \to h_1 \to h_2 \to ... \to t$ where each $h_i \in H$ and each segment uses at most $0.25h\varepsilon$ hops

Further, any h-hop path can have cost at most Wh, thus the given $(1+\varepsilon)$-approximate distances upto Wh are also $(1+\varepsilon)$-approximate h-hop distances. This means can compute $(1+\varepsilon)^2$-approximate distances for the pairs $H \times H$ at extra cost $\tilde{O}((n/h)^\omega/\varepsilon^{\omega+1})$ by computing $\Delta := D^{(\star|H|)}_{H,H}$.

Next, we assign each node $v \in V$ some node $x_v \in H$ with $D_{v,x_v} \leq 0.25Wh\varepsilon$. This requires only $O(|H|n) = O(n^2/(h\varepsilon))$ time for all $v \in V$ together. (We will later explain what happens to nodes, where no such $x_v \in H$ exists.)

Note that this also implies $\operatorname{dist}(u, x_u), \operatorname{dist}(v, x_v) \leq 0.25Wh\varepsilon$ and since the graph is undirected, this leads to

$$\begin{aligned}\operatorname{dist}(x_u, x_v) &\leq \operatorname{dist}(x_u, u) + \operatorname{dist}(u, v) + \operatorname{dist}(v, x_v)\\ &\leq \operatorname{dist}(u, v) + 0.5Wh\varepsilon,\\ \operatorname{dist}(u, v) &\leq \operatorname{dist}(u, x_u) + \operatorname{dist}(x_u, x_v) + \operatorname{dist}(x_v, v)\\ &\leq \operatorname{dist}(x_u, x_v) + 0.5Wh\varepsilon.\end{aligned}$$

Thus for any pair u, v with $\operatorname{dist}(u, v) \geq Wh$ we have:

$$\operatorname{dist}(u, v) \leq \Delta_{x_u,x_v} + 0.5Wh\varepsilon \leq \operatorname{dist}(u, v)(1+\varepsilon)^3.$$

If some node $u \in V$ has no $x_u \in H$, then there also is no $v \in V$ with $Wh \leq \operatorname{dist}(u, v) < \infty$, as otherwise there should be some $h \in H$ with $\operatorname{dist}(u, h) \leq Wh\varepsilon$ along the path from u to v. Thus for some distance query for a pair u, v, we either return $\Delta_{x_u,x_v} + 0.5Wh\varepsilon$ or ∞ if there is no x_u or x_v. ■

Proof of Theorem 5.2: Let $0 \leq s \leq 1$ be some parameter. We maintain the $(1+\varepsilon)$-approximate distances D upto Wn^s via Theorem 4.2. After every update we also construct the distance oracle $\hat{D}$ from Lemma 5.9 for a new random hitting-set H of size $\tilde{O}(n^{1-s}/\varepsilon)$. To construct this oracle, we need to query the distances $D_{H,V}$ for the pairs $V \times H$, so for $W = n^\ell$ the update time becomes $\tilde{O}(n^{\omega(1,1,s+\mu+\ell)+\mu}/\varepsilon + u(n^{s+\ell}, n) + n^{\omega(1-s,s+\mu+\ell,1)}/\varepsilon^2 + n^{(1-s)\omega}/\varepsilon^{1+\omega})$,

When answering some query for all pairs in some set $I \times J$, we compute $D_{I,J}$ and take the entry-wise minimum with

$\hat{D}_{I,J}$. This way we obtain $(1+\varepsilon)^3$ approximate distances in $\tilde{O}(n^{\omega(\delta_1, s+\mu+\ell, \delta_2)})$ time. By choosing a slightly smaller approximation factor in Theorem 4.2 and lemma 5.9 this can be made $(1+\varepsilon)$-approximate.

Note that Theorem 4.2 works against non-oblivious adversaries, so Theorem 5.2 works against non-oblivious adversaries as well, because we sample a new random hitting-set H after every update. ■

Theorem 5.10 (Approx APSP, Unweighted (or with W), Undirected, n^2)**.** *Let G be an undirected graph with n nodes and integer weights from $\{1, ..., W\}$. Then for any $0 < \varepsilon$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate all-pairs-distances of G in $\tilde{O}(Wn^{1.844}/\varepsilon + n^2/\varepsilon^{1+\varepsilon})$ update time. The pre-processing requires $\tilde{O}(Wn^{2.53}/\varepsilon)$ time.*

Proof: Theorem 5.10 follows from Theorem 5.2 by querying all distances for $I = J = V$ after every update. For $\mu = 0$ and maximum edge weight $W = n^\ell$ the update time is

$$\tilde{O}(n^{\omega(1,1,s+\ell)}/\varepsilon + n^{(1-s)\omega}/\varepsilon^{1+\omega}).$$

By setting $s = 1 - 2/\omega \approx 0.157$ we have $n^{(1-s)\omega} = n^2$. For small $W = O(n^{0.156})$, the update time can be bounded by $\tilde{O}(n^2/\varepsilon^{1+\omega})$, because $\omega(1,1,s+\ell) = 2$. For $W = \Omega(n^{0.156})$ the $n^{\omega(1,1,0.157+\ell)}/\varepsilon$ term can be bounded by $\tilde{O}(n^{\omega(1,1,0.157+\ell)}/\varepsilon) = \tilde{O}(n^{2+\ell-0.156}/\varepsilon) = \tilde{O}(Wn^{1.844}/\varepsilon)$. Thus for any W, we have update time $\tilde{O}(Wn^{1.844}/\varepsilon + n^2/\varepsilon^{1+\varepsilon})$.

The pre-processing requires $O(Wn^{\omega+s}) = O(Wn^{2.53})$ time. ■

VI. RESULTS FOR DIAMETER, RADIUS AND ECCENTRICITIES

In this section we will prove several results for dynamic diameter. The main result is the following nearly $(1.5+\varepsilon)$-approximate algorithm for diameter and radius.

Theorem 6.1 (Nearly $(1.5+\epsilon)$-Approx Diameter/Radius, Unweighted (only), Directed, Sub-n^2)**.** *Let G be an unweighted directed graph with n nodes. Then for any $0 \leq s, \mu \leq 1$ and $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains a nearly $(1.5+\varepsilon)$-approximation $\tilde{D}$ of the diameter of G, such that*

$$\left(\frac{2}{3} - \varepsilon\right) \operatorname{diam}(G) - 1/3 \leq \tilde{D} \leq (1+\varepsilon) \operatorname{diam}(G).$$

The update time of the algorithm can be bounded by

$$\tilde{O}((n^{\omega(1,1,s+\mu)-\mu} + n^{\omega(0.5,s+\mu,1)})/\varepsilon + n^{1.5286+s} + n^{\omega(1-s,\mu+s,1-s)}/\varepsilon^2 + n^{(1-s)\omega}/\varepsilon^{1+\omega}).$$

The pre-processing requires $\tilde{O}(n^{s+\omega}/\varepsilon)$ time. If the graph is undirected, then we can also maintain a nearly $(1.5+\varepsilon)$-approximate radius $\tilde{R}$, such that

$$\operatorname{radius}(G)/(1+\varepsilon) \leq \tilde{R} \leq (1.5+\varepsilon) \operatorname{radius}(G) + 2/3,$$

For current ω and $\mu \approx 0.4s \approx 0.25$ the update time is $O(n^{1.779}/\varepsilon^{1+\omega})$.

We will prove this result in Section VI-B. There we will also present results for dynamic eccentricities. Before proving Theorem 6.1 in Section VI-B, we will prove some results for large diameter graphs in Section VI-A, which can also be used to maintain a $(1+\varepsilon)$-approximation.

Many of the results we prove and use in this section hold only for strongly connected graphs, which is why we require the following lemma:

Lemma 6.2 ([52, Corollary C.17],[6, Theorem 3])**.** *For every $\mu > 0$ there exists a Monte Carlo dynamic algorithm that can detect if a graph is strongly connected. The update time per edge update is $O(n^{\omega(1,1,\mu)-\mu} + n^{1+\mu}) = O(n^{1.5286})$ and the pre-processing requires $O(n^\omega)$ time.*[17]

By running this dynamic algorithm in parallel, we can detect if the graph is no longer strongly connected, in which case our dynamic diameter result will simply return ∞.

A. Unweighted Approximate Diameter (Proof of Theorems 6.1 and 6.4)

The high-level idea for unweighted, $(1+\varepsilon)$-approximate diameters is very simple: For diameter less than n^s, we can find the diameter by computing all-pairs-distances via Theorem 4.2. If the diameter is larger than n^s, then for a random hitting set the longest shortest path can be decomposed into segments $s \to h_1 \to ... \to h_k \to t$, where $h_i \in H$ are hitting-set nodes and each segment has length at most $n^s\varepsilon$. Thus we can find an approximate diameter by looking only for the longest path between the nodes H.

Lemma 6.3. *Let G be an n-node graph with integer edge weights from $\{1, 2, ..., W\}$.*

Let $H = \{h_1, h_2, ...\} \subset V$ be a random subset of size $c2n/(d\varepsilon^2)\ln n$. Define G_H to be the graph on node set H and edge set $\{(u,v) \mid \operatorname{dist}_G(u,v) \leq Wd\}$ with cost $c_{u,v} = \operatorname{dist}_G(u,v)$.

Then with probability at least $1 - n^{2-c}$ we have:

$$\begin{aligned} \operatorname{diam}(G) &\leq \operatorname{diam}(G_H) + Wd\varepsilon \\ &\leq (1+\varepsilon)\operatorname{diam}(G) \text{ if } \operatorname{diam}(G) \geq Wd. \end{aligned}$$

If the graph is undirected, we additionally also have

$$\begin{aligned} \operatorname{radius}(G) &\leq \operatorname{radius}(G_H) + 0.5Wd\varepsilon \\ &\leq (1+\varepsilon)\operatorname{radius}(G) \text{ if } \operatorname{radius}(G) \geq Wd \end{aligned}$$

We will prove Lemma 6.3 later in this section. For now we use it to show how to maintain a $(1+\varepsilon)$-approximation of the diameter.

---

[17] Note that the update complexity is subsumed by Lemma 4.4. All algorithms in Section IV use Lemma 4.4, so running Lemma 6.2 in parallel does not further affect their complexity.

Theorem 6.4 (Approximate Diameter, Unweighted (or with W), Directed, n^2)**.** *Let G be a directed graph with n nodes and integer weights from $\{1, ..., W\}$ and let ℓ be such that $W = n^\ell$. Then for any $\varepsilon > 0$ there exists a Monte Carlo dynamic algorithm that maintains $(1+\varepsilon)$-approximate diameter of G in $\tilde{O}(Wn^{1.844}/\varepsilon + n^2/\varepsilon^{1+\omega})$ update time. The pre-processing requires $\tilde{O}(Wn^{2.53}/\varepsilon)$ time.*

Proof: Let $0 \leq s \leq 1$ be some parameter. We maintain the $(1+\varepsilon)$-approximate n^s-hops distances via Theorem 4.2 (by maintaining the distances upto $Wn^s = n^{s+\ell}$ for $n^\ell = W$). After every update we query all distances via $I = J = V$. Thus every update requires $\tilde{O}(n^{\omega(1,1,s+\ell)}/\varepsilon)$ time per update by choosing $\mu = 0$.

We also obtain a pair $P \subset V \times V$ of pairs with distance greater than Wn^s (i.e. all pairs where the returned distance is ∞). If this set is empty, then we can get the diameter by looking at the longest distance we computed. If the set is non-empty, then we know the diameter must be larger than Wn^s.

We also run Lemma 6.2 in parallel to check if the graph is strongly connected. If it is not strongly connected, then we return ∞ as diameter. Otherwise, if the graph is strongly connected, then we can use Lemma 6.3 to compute an approximation of the diameter. For this we sample a random set $H \subset V$ of size $\tilde{O}(n^{1-s}/\varepsilon)$ and construct the graph G_H as in Lemma 6.3 in $\tilde{O}(n^{2-2s}/\varepsilon^2)$ time.

We can compute the approximate diameter of G_H by computing $(1+\varepsilon)$-approximate all-pairs-distances in $\tilde{O}(n^{(1-s)\omega}/\varepsilon^{1+\omega})$ time using Lemma 5.3. (Note that we only have approximate distances of G when constructing G_H, so technically we obtain a $(1+\varepsilon)^3$-approximation of $\operatorname{diam}(G)$, but we can just choose ε to be small enough.)

The update time is thus $\tilde{O}(n^{\omega(1,1,s+\ell)}/\varepsilon + n^{(1-s)\omega}/\varepsilon^{1+\omega})$, which can be bounded by $\tilde{O}(Wn^{1.844}/\varepsilon + n^2/\varepsilon^{\omega+1})$. (For details, see the proof of Theorem 5.10, where the same term was bounded.)

Note that the algorithm works against non-oblivious adversaries, because (w.h.p.) the result from Theorem 4.2 does not leak any information about the random choices and we can sample a new hitting set in Lemma 6.3 for every update. ■

We are left with proving Lemma 6.3.

Proof of Lemma 6.3:

The main idea is to decompose paths in G via Lemma 5.3 and then map them to paths in G_H and bound their length.

Let H be the random set of size $c2n/(d\varepsilon^2)$, then with probability at least $1 - n^{2-c}$, every shortest path (using at least d hops) can be decomposed into segments $s \to h_1 \to h_2 \to ... \to h_k \to t$ where each $h_i \in H$ and the segments have at most $\varepsilon d/2$ hops.

Thus for $u, v \in H$ we have $\operatorname{dist}_G(s,t) = \operatorname{dist}_{G_H}(u,v)$ and for every $s, t \in V$ with $\operatorname{dist}_G(s,t) \geq Wd$, there exist $u, v \in H$ such that $\operatorname{dist}_{G_H}(u,v) \leq \operatorname{dist}_G(s,t) \leq \operatorname{dist}_G(u,v) + Wd\varepsilon = \operatorname{dist}_{G_H}(u,v) + Wd\varepsilon$.

Approximating the Diameter: Let $s, t \in V$ be the pairs such that $\operatorname{dist}_G(s,t) = \operatorname{diam}(G)$ and the shortest st-path uses at least d hops, and let $s_H, t_H \in H$ be the first and last node from H along the shortest st-path. Then $\operatorname{dist}_G(s_H, t_H) \leq \operatorname{dist}_G(s,t) \leq \operatorname{dist}_G(s_H, t_H) + Wd\varepsilon$ and

$$\begin{aligned}
\operatorname{diam}(G) &\leq \operatorname{dist}_G(s_H, t_H) + Wd\varepsilon \\
&= \operatorname{dist}_{G_H}(s_H, t_H) + Wd\varepsilon \leq \operatorname{diam}(G_H), \\
\operatorname{diam}(G_H) &= \max_{u,v \in H} \operatorname{dist}_{G_H}(u,v) = \max_{u,v \in H} \operatorname{dist}_G(u,v) \\
&\leq \max_{u,v \in V} \operatorname{dist}_G(u,v) = \operatorname{diam}(G).
\end{aligned}$$

So if $\operatorname{diam}(G) \geq Wd$, then $\operatorname{diam}(G_H) + Wd\varepsilon$ is a $(1+\varepsilon)$-approximation of $\operatorname{diam}(G)$.

Approximating the Radius: For the radius we can use the same arguments as for the diameter:

$$\begin{aligned}
\operatorname{radius}(G) &= \min_{s \in V} \max_{t \in V} \operatorname{dist}_G(s,t) \leq \min_{s \in H} \max_{t \in V} \operatorname{dist}_G(s,t) \\
&\leq \min_{s \in H} \max_{t \in H} \operatorname{dist}_G(s,t) + 0.5Wd\varepsilon \\
&= \min_{s \in H} \max_{t \in H} \operatorname{dist}_{G_H}(s,t) + 0.5Wd\varepsilon \\
&= \operatorname{radius}(G_H) + 0.5Wd\varepsilon
\end{aligned}$$

If the graph is undirected we can also get a bound from the other direction as follows: Let $s \in V$ be the node such that $\operatorname{radius}(G) = \max_{v \in V} \operatorname{dist}_G(s,v)$ and assume $\operatorname{radius}(G) \geq Wd$, then w.h.p. there is a $s_H \in H$ with $\operatorname{dist}_G(s_h, s) \leq 0.5Wd\varepsilon$ and thus $\max_{v \in V} \operatorname{dist}_G(s_H, v) \leq \max_{v \in V} \operatorname{dist}_G(s,v) + 0.5Wd\varepsilon$. This leads to the following bound:

$$\begin{aligned}
\operatorname{radius}(G_H) &= \min_{s \in H} \max_{t \in H} \operatorname{dist}_{G_H}(s,t) \\
&= \min_{s \in H} \max_{t \in H} \operatorname{dist}_G(s,t) \\
&\leq \min_{s \in H} \max_{t \in V} \operatorname{dist}_G(s,t) \\
&\leq \min_{s \in V} \max_{t \in V} \operatorname{dist}_G(s,t) + 0.5Wd\varepsilon \\
&= \operatorname{radius}(G) + 0.5Wd\varepsilon
\end{aligned}$$

Thus for undirected graphs with $\operatorname{radius}(G) \geq Wd$ we have $\operatorname{radius}(G) \leq \operatorname{radius}(G_H) + 0.5Wd\varepsilon \leq \operatorname{radius}(G) + Wd\varepsilon \leq (1+\varepsilon)\operatorname{radius}(G)$, so $\operatorname{radius}(G_H) + 0.5Wd\varepsilon$ is a $(1+\varepsilon)$-approximation of the radius. ■

B. Unweighted 1.5 Approximate Diameter

The following result is from [39, Lemma 4], where Roditty and V. Williams present a static algorithm to compute a nearly 1.5-approximation of the diameter in unweighted graphs in $\tilde{O}(m\sqrt{n})$ time. The high level idea is to compute BFS trees originating at $\tilde{O}(\sqrt{n})$ uniformly at random chosen nodes $S \subset V$, then find the node w furthest away from the nodes S and compute another BFS tree for this distant node v and its $\sqrt{n}$ nearest neighbors. The largest computed distance of all BFS trees then yields a nearly 1.5-approximation of the diameter.

We simply simulate their algorithm by querying the single-source (and single-sink) distances via Theorem 4.2 instead of performing the breadth-first-searches.

The original proof of [39, Lemma 4] assumes exact distances (i.e. the distances obtained by running BFS), but the result also holds when we are given $(1+\varepsilon)$-approximate distances instead, in which case we obtain a nearly $(1.5+\varepsilon)$-approximate diameter. The proof of the following Theorem 6.5 can be found in the full version of the paper.

Theorem 6.5 (Based on [39, Lemma 4])**.** *Let G be a directed unweighted graph with diameter $D = 3h+z$, where $h \geq 0$ and $z \in \{0,1,2\}$. Let $0 < \varepsilon \leq 1$ and let $\tilde{D}$ be the approximate diameter returned by Algorithm 6.6.*

Then we have w.h.p. $\min\{(2-3\varepsilon)h+z, 2h+1\} \leq \tilde{D} \leq (1+\varepsilon)D$, which can also be written as $(\frac{2}{3}-\varepsilon)D - \frac{1}{3} \leq \tilde{D} \leq (1+\varepsilon)D$.

Algorithm 6.6. *Assume we are given a $(1+\varepsilon)$-approximate distance matrix $\tilde{d}$. This matrix does not need to be given explicitly, it is enough if we can query rows/columns, so $\tilde{d}$ could be some distance oracle.*[18]

1) *Let $S \subset V$ be a random set of vertices of size $\tilde{\Theta}(\sqrt{n})$.*
2) *Query $(1+\varepsilon)$-approximate distances from/to the nodes in S, so $\tilde{d}(s,u)$ for all $s \in S, u \in V$.*
3) *Let $w \in V$ be the node with largest (approximate) distance to S, i.e. for all $u \in V$ we have $\min_{s\in S}\tilde{d}(w,s) \geq \min_{s\in S}\tilde{d}(u,s)$.*
4) *Query the distance from/to w, i.e. we query $\tilde{d}(u,w)$ and $\tilde{d}(w,u)$ for every $u \in V$.*
5) *Let $W \subset V$ be a set of size $\sqrt{n}$ such that $\tilde{d}(w,u) \leq \tilde{d}(w,v)$ for all $u \in W$ and $v \in V \setminus W$.*
   Ties are broken by node index. (The set W are the $\sqrt{n}$ approximately closest nodes to w.)
6) *$\tilde{D} := \max_{v\in V, u\in S\cup W\cup\{w\}}\{\tilde{d}(u,v), \tilde{d}(v,u)\}$, i.e. the largest of all so far computed distances.*

Proof of Theorem 6.1: Let $0 \leq s$ be some parameter. We run Theorem 4.2, which allows us to query $(1+\varepsilon)$-approximate distances upto n^s, and we also run Lemma 6.2 in parallel to check, if the graph is strongly connected.

If it is not strongly connected, then we know the diameter is ∞. Otherwise we proceed by running Theorem 6.5 as described in the next paragraph.

Case 1, Simulating [39]: We use the approximate distances of Theorem 4.2 to run Theorem 6.5 (Algorithm 6.6). Note that Theorem 6.5 performs the following queries: (i) $\tilde{O}(\sqrt{n})$ sources/sinks (set S) (ii) one source/sink (node w) (iii) $\sqrt{n}$ sources/sinks (set W). The time required for all queries (i), (ii), (iii) together is $\tilde{O}(n^{\omega(0.5,\mu+s,1)})$.

Note that Theorem 4.2 only maintains distances upto n^s. If some distance of a queried pair is larger than n^s, then we can detect that, but we do not receive the actual distance. In such a case we know that the diameter must be larger than n^s, so we abort Algorithm 6.6 and instead compute the diameter differently.

Case 2, $\operatorname{diam}(G) \geq n^s$: If Algorithm 6.6 fails, because some computed distance is larger than n^s, then we also know that $\operatorname{diam}(G) \geq n^s$. This means we can now use Lemma 6.3 instead. Let H be the random set of nodes that Lemma 6.3 uses, then querying the distances of the pairs $H \times H$ requires $\tilde{O}(n^{\omega(1-s,\mu+s,1-s)}/\varepsilon^2)$ time, as $|H| = \tilde{O}(n^{1-s}/\varepsilon)$.

Hence the total time for running Lemmas 6.3 and 5.4 to approximate the diameter becomes $\tilde{O}(n^{\omega(1-s,\mu+s,1-s)}/\varepsilon^2 + n^{(1-s)\omega}/\varepsilon^{1+\omega})$ time.

Update time: The update time complexity for approximating the diameter is:

$$\tilde{O}(\underbrace{n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + u(n^s,n)}_{\text{Theorem 4.2}} + \underbrace{n^{\omega(0.5,\mu+s,1)}/\varepsilon)}_{\text{Algorithm 6.6}}$$
$$+\underbrace{n^{\omega(1-s,\mu+s,1-s)}/\varepsilon^2}_{\text{pairs } H\times H} + \underbrace{n^{(1-s)\omega}/\varepsilon^{1+\omega}}_{\text{Lemmas 6.3 and 5.4}})$$

Radius: The algorithm works almost identically, if we want to maintain the diameter. If during the last step of Algorithm 6.6 we set $\tilde{R} := \min_{v\in S\cup W\cup\{w\}}\max_{u\in V}\{\tilde{d}(u,v)\}$, then $\tilde{R}$ is a nearly $(1.5+\varepsilon)$-approximation of the radius (see the full version for a proof of this claim).

We again have to consider the two cases $\operatorname{radius}(G) > n^s$ and $\operatorname{radius}(G) \leq n^s$.

- By running Lemma 6.2, we can detect if $\operatorname{radius}(G) = \infty$, because then the graph is not connected. If the graph is connected, we proceed to the next case:
- Let r be the node with the property $\operatorname{radius}(G) = \max_{v\in V}\operatorname{dist}(r,v)$, then for any $u,v \in V$ we have $\operatorname{dist}(u,v) \leq \operatorname{dist}(u,r) + \operatorname{dist}(r,v) \leq 2\operatorname{radius}(G)$. Hence it is enough to run Theorem 4.2 for distances upto $2n^s$ and compute $\tilde{R}$ via the adaption to Algorithm 6.6. If during some distance query to Theorem 4.2 we realize that the distance is larger than $2n^s$, then we know the radius is larger than n^s. We then cancel Algorithm 6.6 and go to the next case.
- $n^s \leq \operatorname{radius}(G) < \infty$: This case is identical to the case where the diameter is more than n^s. We simply use Lemmas 6.3 and 5.4 to obtain an approximation of the radius.

Note that Theorem 4.2 works against non-oblivious adversaries, so Theorem 6.1 works against non-oblivious adversaries as well, because we sample a new random hitting-set S in Algorithm 6.6 after every update. ■

We can also maintain nearly $(5/3+\varepsilon)$-approximate eccentricities, by dynamically maintaining $(1+\varepsilon)$-approximate

---

[18] The result of the queries must be fixed, i.e. it is not allowed to depend on the order in which we query the results. More formally, we require that for every k_1, k_2 and $u_1, v_1, ..., u_{k_1}, v_{k_1}, u'_1, v'_1, ..., u'_{k_2}, v'_{k_2}, s, t \in V$, the two sequences $\tilde{d}(u_1,v_1), ..., \tilde{d}(u_{k_1},v_{k_1}), \tilde{d}(s,t)$ and $\tilde{d}(u'_1,v'_1), ..., \tilde{d}(u'_{k_2},v'_{k_2}), \tilde{d}(s,t)$ both return the same result for $\tilde{d}(s,t)$.

distances and simulating a variant of the static algorithm for approximating eccentricities of [40]. The static algorithm would usually perform BFS searches, but instead we query the distances via our dynamic algorithm.

Theorem 6.7. *Let G be an unweighted, undirected n-node graph. Then for any $0 < \varepsilon$, $0 \le \mu \le 1$, $0 \le s < 0.5$, there exists a Monte Carlo dynamic algorithm that maintains nearly $(3/5+\varepsilon)$-approximate eccentricities $\widetilde{ecc}(\cdot)$, such that for every v:*

$$\left(\frac{3}{5}-\varepsilon\right) ecc(v) - 4/7 \le \widetilde{ecc}(v) \le ecc(v)$$

The update time is

$$\tilde{O}(n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + u(n^s,n))/\varepsilon \\ + n^{\omega(1,\mu+s,1-s)}/\varepsilon^2 + n^{(1-s)\cdot\omega}/\varepsilon^{\omega+1}),$$

and the pre-processing requires $\tilde{O}(n^{\omega+s})$ time (both complexities are the same as in Theorem 5.2). For current ω the pre-processing and update time are $\tilde{O}(n^{2.621})$ and $\tilde{O}(n^{1.823}/\varepsilon^{3.373})$ respectively with $s \approx 0.248$ and $\mu \approx 0.202$.

Proof: We simulate a variant of the static algorithm for approximating eccentricities of [40] as defined in the full version of our paper. For that we need to query the distances of $\tilde{O}(\sqrt{n})$ source nodes to every other node. As the graph is undirected this can be done using the dynamic algorithm of Theorem 5.2. The query time for this many sources is $\tilde{O}(n^{\omega(0.5,s+\mu,1)})$, which together with the update time of Theorem 5.2 results in our dynamic eccentricities algorithm having the update time

$$\tilde{O}(n^{\omega(1,s+\mu,1)-\mu}/\varepsilon + u(n^s,n))/\varepsilon \\ +n^{\omega(1,\mu+s,1-s)}/\varepsilon^2 + n^{(1-s)\cdot\omega}/\varepsilon^{\omega+1} + n^{\omega(0.5,s+\mu,1)}).$$

Note that $u(n^s,n)) > n^{1.5+s}$, even if $\omega = 2$, hence we can only obtain a subquadratic algorithm for $s < 0.5$. With this observation the $n^{\omega(0.5,s+\mu,1)}$ term is subsumed by $n^{\omega(1,\mu+s,1-s)}/\varepsilon^2$. So the update time of Theorem 6.7 is the same as Theorem 5.2.

Note that Theorem 5.2 works against non-oblivious adversaries, so Theorem 6.7 works against non-oblivious adversaries as well. ■

VII. OPEN PROBLEMS

An obvious open problem is to improve our bounds and proving matching conditional lower bounds. Improving our bounds with amortization should already be interesting (except for APSP). Another intriguing question is whether fast matrix multiplication is really needed to get the bounds we achieve here. Note that a conditional lower bound of Abboud and V. Williams [22] suggests that this is the case for SSSP on directed graphs[19]. We are not aware of similar lower bounds for other problems.

Whether the bounds similar to ours hold without the ϵ term remains open (e.g. *exact* SSSP and APSP). A subquadratic update time for exact unweighted SSSP and exact weighted st-shortest paths will be very interesting. A subquadratic update time for exact weighted SSSP will be surprising, since such bound has already been ruled out for algorithms with $n^{3-\epsilon}$ proprocessing time [22]. Of course, showing $O(n^2)$ worst-case update time for maintaining APSP exactly remains a major open problem.

Finally, note that our algorithms can only maintain distances but cannot report the corresponding paths. Supporting such operation is interesting, especially doing for APSP in the same time complexity as Demetrescu and Italiano's algorithm [8].

ACKNOWLEDGMENT

This project has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme under grant agreement No 715672. Danupon Nanongkai was also partially supported by the Swedish Research Council (Reg. No. 2015-04659).

REFERENCES

[1] J. van den Brand and D. Nanongkai, "Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time," *CoRR*, vol. abs/1909.10850, 2019.

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

[3] M. Henzinger, S. Krinninger, and D. Nanongkai, "Dynamic approximate all-pairs shortest paths: Breaking the O(mn) barrier and derandomization," *SIAM J. Comput.*, vol. 45, no. 3, pp. 947–1006, 2016, announced at FOCS'13.

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

[5] L. Roditty and U. Zwick, "Dynamic approximate all-pairs shortest paths in undirected graphs," *SIAM J. Comput.*, vol. 41, no. 3, pp. 670–683, 2012, announced at FOCS'04.

[6] P. Sankowski, "Dynamic transitive closure via dynamic matrix inverse (extended abstract)," in *FOCS*. IEEE Computer Society, 2004, pp. 509–517.

[7] ——, "Subquadratic algorithm for dynamic shortest distances," in *COCOON*, ser. Lecture Notes in Computer Science, vol. 3595. Springer, 2005, pp. 461–470.

---

[19] In particular, under the Boolean Matrix Multiplication (BMM) conjecture, there is no "combinatorial" algorithm with $n^{3-\epsilon}$ preprocessing time and $n^{2-\epsilon}$ update time even for st-Reachability (maintaining the reachability between two nodes)

[8] C. Demetrescu and G. F. Italiano, "A new approach to dynamic all pairs shortest paths," *J. ACM*, vol. 51, no. 6, pp. 968–992, 2004, announced at STOC'03.

[9] M. Thorup, "Worst-case update times for fully-dynamic all-pairs shortest paths," in *STOC*. ACM, 2005, pp. 112–119.

[10] I. Abraham, S. Chechik, and S. Krinninger, "Fully dynamic all-pairs shortest paths with worst-case update-time revisited," in *SODA*. SIAM, 2017, pp. 440–452.

[11] F. L. Gall, "Powers of tensors and fast matrix multiplication," in *ISSAC*. ACM, 2014, pp. 296–303.

[12] V. V. Williams, "Multiplying matrices faster than coppersmith-winograd," in *STOC*. ACM, 2012, pp. 887–898.

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

[14] M. Henzinger, S. Krinninger, and D. Nanongkai, "Decremental single-source shortest paths on undirected graphs in near-linear total update time," *J. ACM*, vol. 65, no. 6, pp. 36:1–36:40, 2018, announced at FOCS'14 and ICALP'15.

[15] ——, "Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs," in *STOC*. ACM, 2014, pp. 674–683.

[16] ——, "A subquadratic-time algorithm for decremental single-source shortest paths," in *SODA*. SIAM, 2014, pp. 1053–1072.

[17] ——, "Sublinear-time maintenance of breadth-first spanning tree in partially dynamic networks," in *ICALP (2)*, ser. Lecture Notes in Computer Science, vol. 7966. Springer, 2013, pp. 607–619.

[18] A. Bernstein and S. Chechik, "Deterministic decremental single source shortest paths: beyond the o(mn) bound," in *STOC*. ACM, 2016, pp. 389–397.

[19] A. Bernstein, "Deterministic partially dynamic single source shortest paths in weighted graphs," in *ICALP*, ser. LIPIcs, vol. 80. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017, pp. 44:1–44:14.

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

[21] L. Roditty and U. Zwick, "On dynamic shortest paths problems," *Algorithmica*, vol. 61, no. 2, pp. 389–401, 2011, announced at ESA'04.

[22] A. Abboud and V. V. Williams, "Popular conjectures imply strong lower bounds for dynamic problems," in *FOCS*. IEEE Computer Society, 2014, pp. 434–443.

[23] A. Bernstein, S. Forster, and M. Henzinger, "A deamortization approach for dynamic spanner and dynamic maximal matching," in *SODA*. SIAM, 2019, pp. 1899–1918.

[24] D. Nanongkai, T. Saranurak, and C. Wulff-Nilsen, "Dynamic minimum spanning forest with subpolynomial worst-case update time," in *FOCS*. IEEE Computer Society, 2017, pp. 950–961.

[25] D. Nanongkai and T. Saranurak, "Dynamic spanning forest with worst-case update time: adaptive, las vegas, and o($n^{1/2 - \epsilon}$)-time," in *STOC*. ACM, 2017, pp. 1122–1129.

[26] C. Wulff-Nilsen, "Fully-dynamic minimum spanning forest with improved worst-case update time," in *STOC*. ACM, 2017, pp. 1130–1143.

[27] S. Bhattacharya, M. Henzinger, and D. Nanongkai, "Fully dynamic approximate maximum matching and minimum vertex cover in $O(\log^3 n)$ worst case update time," in *SODA*. SIAM, 2017, pp. 470–489.

[28] B. M. Kapron, V. King, and B. Mountjoy, "Dynamic graph connectivity in polylogarithmic worst case time," in *SODA*. SIAM, 2013, pp. 1131–1142.

[29] M. Charikar and S. Solomon, "Fully dynamic almost-maximal matching: Breaking the polynomial worst-case time barrier," in *ICALP*, ser. LIPIcs, vol. 107. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, pp. 33:1–33:14.

[30] T. Kopelowitz, R. Krauthgamer, E. Porat, and S. Solomon, "Orienting fully dynamic graphs with worst-case time bounds," in *ICALP (2)*, ser. Lecture Notes in Computer Science, vol. 8573. Springer, 2014, pp. 532–543.

[31] M. Arar, S. Chechik, S. Cohen, C. Stein, and D. Wajc, "Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms," in *ICALP*, ser. LIPIcs, vol. 107. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018, pp. 7:1–7:16.

[32] G. Bodwin and S. Krinninger, "Fully dynamic spanners with worst-case update time," in *ESA*, ser. LIPIcs, vol. 57. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016, pp. 17:1–17:18.

[33] C. Wulff-Nilsen and M. Probst, "Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds," *Manuscript*, 2019, accepted to SODA'20.

[34] U. Zwick, "All pairs shortest paths using bridging sets and rectangular matrix multiplication," *J. ACM*, vol. 49, no. 3, pp. 289–317, 2002, announced at FOCS'98.

[35] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, no. 1, pp. 1–13, 1977.

[36] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[37] B. Ancona, M. Henzinger, L. Roditty, V. V. Williams, and N. Wein, "Algorithms and hardness for diameter in dynamic graphs," *CoRR*, vol. abs/1811.12527, 2018.

[38] M. Henzinger, S. Krinninger, D. Nanongkai, and T. Saranurak, "Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture," in *STOC*. ACM, 2015, pp. 21–30.

[39] L. Roditty and V. V. Williams, “Fast approximation algorithms for the diameter and radius of sparse graphs,” in *STOC*. ACM, 2013, pp. 515–524.

[40] S. Chechik, D. H. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. V. Williams, “Better approximation algorithms for the graph diameter,” in *SODA*. SIAM, 2014, pp. 1041–1052.

[41] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani, “Fast estimation of diameter and shortest paths (without matrix multiplication),” *SIAM J. Comput.*, vol. 28, no. 4, pp. 1167–1181, 1999, announced at SODA’96.

[42] M. Cairo, R. Grossi, and R. Rizzi, “New bounds for approximating extremal distances in undirected graphs,” in *SODA*. SIAM, 2016, pp. 363–376.

[43] A. Backurs, L. Roditty, G. Segal, V. V. Williams, and N. Wein, “Towards tight approximation bounds for graph diameter and eccentricities,” in *STOC*. ACM, 2018, pp. 267–280.

[44] K. Choudhary and O. Gold, “Diameter spanner, eccentricity spanner, and approximating extremal graph distances: Static, dynamic, and fault tolerant,” *CoRR*, vol. abs/1812.01602, 2018.

[45] A. Shoshan and U. Zwick, “All pairs shortest paths in undirected graphs with integer weights,” in *FOCS*. IEEE Computer Society, 1999, pp. 605–615.

[46] M. Cygan, H. N. Gabow, and P. Sankowski, “Algorithmic applications of baur-strassen’s theorem: Shortest cycles, diameter, and matchings,” *J. ACM*, vol. 62, no. 4, pp. 28:1–28:30, 2015, announced at FOCS’12.

[47] D. Eppstein and J. Wang, “Fast approximation of centrality,” *J. Graph Algorithms Appl.*, vol. 8, pp. 39–45, 2004, announced at SODA’01.

[48] C. Demetrescu and G. F. Italiano, “Trade-offs for fully dynamic transitive closure on dags: breaking through the o(n^2 barrier,” *J. ACM*, vol. 52, no. 2, pp. 147–156, 2005, announced at FOCS’00.

[49] V. V. Williams, “Faster replacement paths,” in *SODA*. SIAM, 2011, pp. 1337–1346.

[50] F. Grandoni and V. V. Williams, “Improved distance sensitivity oracles via fast single-source replacement paths,” in *FOCS*. IEEE Computer Society, 2012, pp. 748–757.

[51] O. Weimann and R. Yuster, “Replacement paths and distance sensitivity oracles via fast matrix multiplication,” *ACM Trans. Algorithms*, vol. 9, no. 2, pp. 14:1–14:13, 2013.

[52] J. van den Brand, D. Nanongkai, and T. Saranurak, “Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds,” *FOCS*, 2019.

[53] J. van den Brand and T. Saranurak, “Sensitive distance and reachability oracles for large batch updates,” *FOCS*, 2019.

[54] A. Bernstein, “Maintaining shortest paths under deletions in weighted directed graphs,” *SIAM J. Comput.*, vol. 45, no. 2, pp. 548–574, 2016, announced at STOC’13.

[55] A. Madry, “Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms,” in *STOC*. ACM, 2010, pp. 121–130.

[56] P. Raghavan and C. D. Thompson, “Provably good routing in graphs: Regular arrays,” in *STOC*. ACM, 1985, pp. 79–87.

[57] D. Nanongkai, “Distributed approximation algorithms for weighted shortest paths,” in *STOC*. ACM, 2014, pp. 565–573.

[58] J. D. Ullman and M. Yannakakis, “High-probability parallel transitive-closure algorithms,” *SIAM J. Comput.*, vol. 20, no. 1, pp. 100–125, 1991, announced at SPAA’90.

[59] J. Sherman and W. J. Morrison, “Adjustment of an inverse matrix corresponding to a change in one element of a given matrix,” *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 124–127, 1950.

[60] M. A. Woodbury, “Inverting modified matrices,” *Memorandum report*, vol. 42, no. 106, p. 336, 1950.

[61] F. L. Gall and F. Urrutia, “Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor,” in *SODA*. SIAM, 2018, pp. 1029–1046.