2013 IEEE 54th Annual Symposium on Foundations of Computer Science

Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization

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

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

Danupon Nanongkai
Division of Mathematical Sciences
Nanyang Technological University
Singapore

***Abstract*—We study dynamic $(1+\epsilon)$-approximation algorithms for the all-pairs shortest paths problem in unweighted undirected n-node m-edge graphs under edge deletions. The fastest algorithm for this problem is a randomized algorithm with a total update time of $\tilde{O}(mn)$ and constant query time by Roditty and Zwick [34] (FOCS 2004). The fastest deterministic algorithm is from a 1981 paper by Even and Shiloach [23]; it has a total update time of $O(mn^2)$ and constant query time. We improve these results as follows:**

(1) We present an algorithm with a total update time of $\tilde{O}(n^{5/2})$ and constant query time that has an additive error of two in addition to the $1+\epsilon$ multiplicative error. This beats the previous $\tilde{O}(mn)$ time when $m = \Omega(n^{3/2})$. Note that the additive error is *unavoidable* since, even in the *static* case, an $O(n^{3-\delta})$-time (a so-called *truly subcubic*) combinatorial algorithm with $1+\epsilon$ multiplicative error cannot have an additive error less than $2-\epsilon$, unless we make a major breakthrough for Boolean matrix multiplication [19] and many other long-standing problems [40].
The algorithm can also be turned into a $(2+\epsilon)$-approximation algorithm (without an additive error) with the same time guarantees, improving the recent $(3+\epsilon)$-approximation algorithm with $\tilde{O}(n^{5/2+O(1/\sqrt{\log n})})$ running time of Bernstein and Roditty [12] (SODA 2011) in terms of both approximation and time guarantees.

(2) We present a deterministic algorithm with a total update time of $\tilde{O}(mn)$ and a query time of $O(\log\log n)$. The algorithm has a multiplicative error of $1+\epsilon$ and gives the first improved deterministic algorithm since 1981. It also answers an open question raised by Bernstein in his STOC 2013 paper [11].

In order to achieve our results, we introduce two new techniques: (1) A *monotone Even-Shiloach tree* algorithm which maintains a bounded-distance shortest-paths tree on a certain type of emulator called *locally persevering emulator*. (2) A derandomization technique based on *moving Even-Shiloach trees* as a way to derandomize the standard random set argument. These techniques might be of independent interest.

***Keywords*-dynamic graph algorithms; all-pairs shortest paths; derandomization; emulator;**

I. Introduction

Dynamic graph algorithms is one of the classic areas in theoretical computer science with a countless number of applications. It concerns maintaining properties of dynamically changing graphs. The objective of a dynamic graph algorithm is to efficiently process an online sequence of update operations, such as edge insertions and deletions, and query operations on a certain graph property. It has to quickly maintain the graph property despite an *adversarial* order of edge deletions and additions. Dynamic graph problems are usually classified according to the types of updates allowed: *decremental* problems allow only deletions, *incremental* problems allow only insertions, and *fully dynamic* problems allow both.

A. The Problem

We consider the *decremental all-pairs shortest paths* (APSP) problem where we wish to maintain the distances in an undirected unweighted graph under a sequence of the following delete and distance query operations:

- DELETE(u, v): delete edge (u, v) from the graph, and
- DISTANCE(x, y): return the distance between vertex x and vertex y in the current graph G, denoted by $d_G(x, y)$.

We use the term *single-source shortest paths* (SSSP) to refer to the special case where the distance query can be done only when $x = s$, for a pre-specified *source node* s. The efficiency is judged by two parameters: *query time* denoting the time needed to answer *each* distance query, and *total update time* denoting the time needed to process *all* edge deletions. The running time will be in terms of n, the number of nodes in the graph, and m, the number of edges *before* any deletion. We use $\tilde{O}$-notation to hide an $O(\mathrm{poly}\log n)$ term. When it is clear from the context, we use "time" instead of "total

The full version of this paper is available as [26] at http://arxiv.org/abs/1308.0776.

M. Henzinger and S. Krinninger are supported by the Austrian Science Fund (FWF): P23499-N23, the Vienna Science and Technology Fund (WWTF) grant ICT10-002, the University of Vienna (IK I049-N), and a Google Faculty Research Award. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 317532.

D. Nanongkai is supported in part by the following research grants: Nanyang Technological University grant M58110000, Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 2 grant MOE2010-T2-2-082, and Singapore MOE AcRF Tier 1 grant MOE2012-T1-001-094. Part of this work was done while D. Nanongkai was at the University of Vienna.

0272-5428/13 $26.00 © 2013 IEEE
DOI 10.1109/FOCS.2013.64

update time", and, unless stated otherwise, the query time is $O(1)$. One of the main focuses of this problem in the literature, which is also the goal in this paper, is to *optimize the total update time* while keeping the query time and *approximation guarantees* small. We say that an algorithm provides an (α, β)-*approximation* if the distance query on nodes x and y on the current graph G returns $\delta(x, y)$ such that $d_G(x, y) \leq \delta(x, y) \leq \alpha d_G(x, y) + \beta$. We call α and β *multiplicative* and *additive errors*, respectively. We are particularly interested in the case where $\alpha = 1 + \epsilon$, for an arbitrarily small constant $\epsilon > 0$, β is a small constant, and the query time is constant or near-constant.

Previous Results: Prior to our work, the best total update time for *deterministic* algorithms was $\tilde{O}(mn^2)$ by one of the earliest papers in the area from 1981 by Even and Shiloach [23]. The fastest *exact randomized* algorithms are the $\tilde{O}(n^3)$-time algorithms by Demetrescu and Italiano [18] and Baswana, Hariharan, and Sen [7]. The fastest *approximation* algorithm is the $\tilde{O}(mn)$-time $(1 + \epsilon, 0)$-approximation algorithm by Roditty and Zwick [34]. If we insist on an $O(n^{3-\delta})$ running time, for some constant $\delta > 0$, Bernstein and Roditty [12] obtain an $\tilde{O}(n^{2+1/k+O(1/\sqrt{\log n})})$-time $(2k-1+\epsilon, 0)$-approximation algorithm, for any integer $k \geq 2$, which gives, e.g., a $(3+\epsilon, 0)$-approximation guarantee in $\tilde{O}(n^{5/2+O(1/\sqrt{\log n})})$ time. All these algorithms have an $O(1)$ worst case query time. See Section II for more detail and other related results.

B. Our Results

We present improved randomized and deterministic algorithms. Our deterministic algorithm provides a $(1 + \epsilon, 0)$-approximation and runs in $\tilde{O}(mn/\epsilon)$ total update time. Our randomized algorithm runs in $\tilde{O}(n^{5/2}/\epsilon^2)$ time and can guarantee both $(1+\epsilon, 2)$- and $(2+\epsilon, 0)$-approximations. Table I compares our results with previous results. In short, we make the following improvements over previous algorithms (further discussions follow).

- The total running time of deterministic algorithms is improved from Even and Shiloach's $\tilde{O}(mn^2)$ to $\tilde{O}(mn)$ (at the cost of $(1+\epsilon, 0)$-approximation and $O(\log \log n)$ query time). This is the first improvement since 1981.
- For $m = \omega(n^{3/2})$, the total running time is improved from Roditty and Zwick's $\tilde{O}(mn)$ to $\tilde{O}(n^{5/2})$, at the cost of an additive error of two, which appears only when the distance is $O(1/\epsilon)$ (since otherwise it could be treated as a multiplicative error of $O(\epsilon)$) and is *unavoidable* (as discussed below).
- Our $(2 + \epsilon, 0)$-approximation algorithm improves the algorithm of Bernstein and Roditty in terms of both total update time and approximation guarantee. The multiplicative error of $2 + \epsilon$ is essentially the best we can hope for, if we do not want any additive error.

To obtain these algorithms, we present two novel techniques, called *Moving Even-Shiloach Tree* and *Monotone Even-Shiloach Tree*, based on a classic technique of Even and Shiloach [23]. These techniques are reviewed in Section III.

Improved Deterministic Algorithm: In 1981, Even and Shiloach [23] presented a deterministic decremental SSSP algorithm for undirected, unweighted graphs with a total update time of $O(mn)$ over all deletions. This was the first dynamic problem studied in theoretical computer science [12]. By running this algorithm from n different nodes, we get an $O(mn^2)$-time decremental algorithm for APSP. No progress on deterministic decremental APSP has been made since then. Our algorithm achieves the first improvement over this algorithm, at the cost of a $(1+\epsilon, 0)$-approximation guarantee and $O(\log \log n)$ query time. (Note that our algorithm is also faster than the current fastest randomized algorithm [34] by a $\log n$ factor.)

Theorem 1 (Deterministic $O((mn \log n)/\epsilon)$ total update time)**.** *Given an unweighted undirected graph and $0 < \epsilon \leq 1$, there is a deterministic decremental algorithm for maintaining $(1 + \epsilon)$-approximate shortest paths with a total update time of $O((mn \log n)/\epsilon)$ and a query time of $O(\log \log n)$.*

Our deterministic algorithm also answers a question recently raised by Bernstein [11] which asks for a deterministic algorithm with a total update time of $\tilde{O}(mn/\epsilon)$. As pointed out in [11] and several other places, this question is important due to the fact that deterministic algorithms can deal with an *adaptive offline adversary* (the strongest adversary model in online computation [13], [9]) while the randomized algorithms developed so far assume an *oblivious adversary* (the weakest adversary model) where the order of edge deletions must be fixed before an algorithm makes random choices. Our deterministic algorithm answers exactly this question.

Improved Randomized Algorithm: Our aim is to improve the $\tilde{O}(mn)$ running time of Roditty and Zwick [34] to so-called *truly subcubic time*, i.e., $O(n^{3-\delta})$ time for some constant $\delta > 0$, a running time that is highly sought of in many problems (e.g., [40], [39], [32][1]). Note, however, that this improvement has to come at the cost of worse approximation:

Fact 2 ([19], [40])**.** *For any $\alpha \geq 1$ and $\beta \geq 0$ such that $2\alpha+\beta < 4$, there is **no** combinatorial (α, β)-approximation algorithm, not even a static one, for APSP on unweighted undirected graphs that is truly subcubic, unless we make a major breakthrough on many long-standing open problems, such as a combinatorial Boolean matrix multiplication and triangle detection.*

This fact is due to the reductions of Dor, Halperin, and Zwick [19] and Vassilevska Williams and Williams [40] (see

[1]Bernstein [11] also recently mentioned a few future directions towards this running time in the decremental setting.

Reference	Total Running Time	Approximation	Deterministic?
Even and Shiloach [23]	$\tilde{O}(mn^2)$	Exact	Yes
This paper	$\tilde{O}(mn/\epsilon)$	$(1+\epsilon, 0)$	Yes
Demetrescu and Italiano [18] and Baswana, Hariharan, and Sen [7]	$\tilde{O}(n^3)$	Exact	No
Roditty and Zwick [34]	$\tilde{O}(mn/\epsilon)$	$(1+\epsilon, 0)$	No
This paper	$\tilde{O}(n^{5/2}/\epsilon^2)$	$(1+\epsilon, 2)$	No
Bernstein and Roditty [12]	$\tilde{O}(n^{5/2+\sqrt{\log(6/\epsilon)/\sqrt{\log n}}})$	$(3+\epsilon, 0)$	No
This paper	$\tilde{O}(n^{5/2}/\epsilon^2)$	$(2+\epsilon, 0)$	No

Table I: Comparisons between our and previous algorithms that are closely related. For details of these and other results see Section II. All algorithms, except our deterministic algorithm, have $O(1)$ query time. Our deterministic algorithm has $O(\log \log n)$ query time.

the full version for a proof sketch). (Roditty and Zwick [33] also showed a similar fact for decremental SSSP.) Thus, the best approximation guarantee we can expect from combinatorial truly subcubic algorithms is, e.g., a multiplicative or additive error of at least two. Our algorithms achieve essentially these *best approximation guarantees*[2]: in $\tilde{O}(n^{5/2})$ time, we get a $(1+\epsilon, 2)$-approximation.

Theorem 3 (Randomized $(1+\epsilon, 2)$-approximation with truly-subcubic total update time)**.** *Given an unweighted undirected graph and $0 < \epsilon \leq 1$, there is a $(1+\epsilon, 2)$-approximate decremental APSP algorithm with constant query time and a total update time of $O(n^{5/2}(\log^2 n)/\epsilon^2 + n^{5/2}(\log^4 n)/\epsilon)$. The approximation guarantee holds with high probability.*

Additionally, if we do not want any additive error, we can get a $(2+\epsilon, 0)$-approximation.

Corollary 4 (Randomized $(2+\epsilon, 0)$-approximation with truly-subcubic total update time)**.** *Given an unweighted undirected graph and $0 < \epsilon \leq 1$, there is a $(2+\epsilon, 0)$-approximate decremental APSP algorithm with constant query time and a total update time of $O(n^{5/2}(\log^2 n)/\epsilon^2 + n^{5/2}(\log^4 n)/\epsilon)$. The approximation guarantee holds with high probability.*

This result easily follows from the observation that if the distance between two nodes is 1, then we can answer queries for their distance exactly by checking whether they are connected by an edge. We note that, prior to our work, Bernstein and Roditty's algorithm [12] can achieve, e.g., a $(3+\epsilon, 0)$-approximation guarantee in $\tilde{O}(n^{5/2+O(\sqrt{1/\log n})})$ time. This result is improved by our $(2+\epsilon, 0)$-approximation algorithm in terms of both time and approximation guarantee, and is far worse than our $(1+\epsilon, 2)$-approximation guarantee, especially when the distance is large. Also note that the running time of our $(1+\epsilon, 2)$-approximation algorithm improves the $\tilde{O}(mn)$ one of Roditty and Zwick [34] when $m = \omega(n^{3/2})$, except that our algorithm gives an additive error of two which is unavoidable and appears only when the distance is $O(1/\epsilon)$ (since otherwise it could be counted as a multiplicative error of $O(\epsilon)$).

[2]We note that there is still some room to eliminate the ϵ-terms. But nothing beyond this is likely to be possible.

II. Related Work

Dynamic APSP has a long history, with the first papers dating back to 1967 [30], [31][3]. It also has a tight connection with its *static* counterpart (where the graph does not change), which is one of the most fundamental problems in computer science: On the one hand, we wish to devise a dynamic algorithm that beats the naive algorithm where we recompute shortest paths *from scratch* using static algorithms after every deletion. On the other hand, the best we can hope for is to match the total update time of decremental algorithms to the best running time of static algorithms. To understand the whole picture, let us first recall the current situation in the static setting. We will focus on combinatorial algorithms[4] since our and most previous decremental algorithms are combinatorial. Static APSP on unweighted undirected graphs can be solved in $O(mn)$ time by simply constructing a breadth-first search tree from every node. Interestingly, this algorithm is the fastest combinatorial algorithm for APSP (despite other fast non-combinatorial algorithms based on matrix multiplication). In fact, a faster combinatorial algorithm will be a *major breakthrough*, not just because computing shortest paths is a long-standing problem by itself, but also because it will imply faster algorithms for other long-standing problems, as stated in Fact 2.

The fact that the best static algorithm takes $O(mn)$ time means two things: First, the naive algorithm will take $O(m^2 n)$ total update time. Second, the best total update time we can hope for is $O(mn)$. A result that is perhaps the first that beats the naive $O(m^2 n)$-time algorithm is from 1981 by Even and Shiloach [23], for the special case of SSSP. They showed an $O(mn)$ total update time with $O(1)$ query time; this implies a total update time of $O(mn^2)$ for APSP. Roditty and Zwick [33] later provided evidence that the $O(mn)$-time decremental unweighted SSSP algorithm of Even and Shiloach is the fastest possible by showing that this is at least as hard as several natural static problems such as Boolean matrix multiplication and the problem of finding all edges of

[3]The early papers [30], [31], however, were not able to beat the naive algorithm where we compute APSP from scratch after every change.

[4]The vague term "combinatorial algorithm" is usually used to refer to algorithms that do not use algebraic operations such as matrix multiplication.

a graph that are contained in triangles. For the incremental setting, Ausiello, Italiano, Marchetti-Spaccamela, and Nanni [3] presented an $\tilde{O}(n^3)$-time APSP algorithm on unweighted directed graphs. (An extension of this algorithm for graphs with small integer edge weights is given in [4].) After that, many efficient fully-dynamic algorithms have been proposed (e.g., [27], [29], [24], [18], [16]). Subsequently, Demetrescu and Italiano [17] achieved a major breakthrough for the fully dynamic case: they obtained a fully dynamic deterministic algorithm for the directed APSP problem with an amortized time of $\tilde{O}(n^2)$ *per update*, implying a total update time of $\tilde{O}(mn^2)$ over all deletions in the decremental setting, the same running time as the algorithm of Even and Shiloach. (Thorup [35] presented an improvement of this result.) An amortized update time of $\tilde{O}(n^2)$ is essentially optimal if the distance matrix is to be explicitly maintained, as done by the algorithm of [17], since each update operation may change $\Omega(n^2)$ entries in the matrix. Even for unweighted, undirected graphs, no faster algorithm is known. Thus, the $O(mn^2)$ total update time of Even and Shiloach *remains the best* for deterministic decremental algorithms, even on undirected unweighted graphs and if approximation is allowed.

For the case of randomized algorithms, Demetrescu and Italiano [18] obtained an exact decremental algorithm on weighted directed graphs with $\tilde{O}(n^3)$ total update time[5] (if weight increments are not considered). Baswana, Hariharan, and Sen [7] obtained an exact decremental algorithm on unweighted directed graphs with $\tilde{O}(n^3)$ total update time. They also obtained a $(1+\epsilon, 0)$-approximation algorithm with $\tilde{O}(m^{1/2}n^2)$ total update time. In [6], they improved the running time further on undirected unweighted graphs, at the cost of a worse approximation guarantee: they obtained approximation guarantees of $(3, 0)$, $(5, 0)$, $(7, 0)$ in $\tilde{O}(mn^{10/9})$, $\tilde{O}(mn^{14/13})$, and $\tilde{O}(mn^{28/27})$ time, respectively. Roditty and Zwick [34] presented two improved algorithms for unweighted, undirected graphs. The first was a $(1+\epsilon, 0)$-approximate decremental APSP algorithm with constant query time and a total update time of $\tilde{O}(mn)$. This algorithm remains the current fastest. The second algorithm achieved a worse approximation bound of $(2k-1, 0)$, for any $2 \leq k \leq \log n$, which has the advantage of requiring less space ($\tilde{O}(mn + n^{1/k})$). By modifying the second algorithm to work on an emulator, Bernstein and Roditty [12] presented the first truly subcubic algorithm which gives a $(2k-1+\epsilon, 0)$-approximation and has a total update time of $\tilde{O}(n^{1+1/k+O(1/\sqrt{\log n})})$. They also presented a $(1+\epsilon, 0)$-approximation $\tilde{O}(n^{2+O(1/\sqrt{\log n})})$-time algorithm for SSSP, which is the first improvement since the algorithm of Even and Shiloach. Very recently, Bernstein [11] presented a $(1+\epsilon, 0)$-approximation $\tilde{O}(mn \log W)$-time algorithm for the directed weighted case, where W is the ratio of the largest edge weight ever seen in the graph to the smallest such weight.

We note that the $(1+\epsilon, 0)$-approximation $\tilde{O}(mn)$-time algorithm of Roditty and Zwick matches the state of the art in the static setting; thus, it is essentially tight. However, by allowing additive error, this running time was improved in the static setting. For example, Dor, Halperin, and Zwick [19], extending the approach of Aingworth et al. [1], presented a $(1, 2)$-approximation for APSP in unweighted undirected graphs with a running time of $O(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$. Elkin [20] presented an algorithm for unweighted undirected graphs with a running time of $O(mn^{\rho} + n^2\zeta)$ that approximates the distances with a multiplicative error of $1+\epsilon$ and an additive error that is a function of ζ, ρ and ϵ. There is no decremental algorithm with additive error prior to our algorithm.

III. Overview of Algorithms and Analyses

Our results build on two previous algorithms. The first algorithm is the classic SSSP algorithm of Even and Shiloach [23] (with the more general analysis of King [29]), which we will refer to as *Even-Shiloach tree*. The second algorithm is the $(1+\epsilon, 0)$-approximation APSP algorithm of Roditty and Zwick [34]. We actually view the algorithm of Roditty and Zwick as a *framework* which runs several Even-Shiloach trees and maintains some properties while edges are deleted. We wish to alter the Roditty-Zwick framework but doing so usually makes it hard to bound the cost of maintaining Even-Shiloach trees (as we will discuss later). Our main technical contribution is the development of new variations of the Even-Shiloach tree, called *moving Even-Shiloach tree* and *monotone Even-Shiloach tree*, which are suitable for our modified Roditty-Zwick frameworks. Since there are many other algorithms that run Even-Shiloach trees as subroutines, it might be possible that other algorithms will benefit from our new Even-Shiloach trees as well.

Review of Even-Shiloach Tree: The Even-Shiloach tree has two parameters: a root (or source) node s and the range (or depth) R. It maintains a breadth-first search tree rooted at s and the distances between s and all other nodes in the dynamic graph, up to distance R (if the distance is more than R, it will be set to ∞). It has a query time of $O(1)$ and a total update time of $O(mR)$ over all deletions. The total update time crucially relies on the fact that the distance between s and any node v changes at most R times before it exceeds R (i.e., from 1 to R). This property heavily relies on the "decrementality" of the model, i.e., the distance between two nodes never decreases, and is easily destroyed when we try to use the Even-Shiloach tree in a more general setting (e.g., when we want to allow edge insertions or alter the Roditty-Zwick framework). Most of our effort in constructing both

[5]This algorithm actually works in a much more general setting where each edge weight can assume S different values. Note that the amortized time per update of this algorithm is $\tilde{O}(Sn)$, but this holds only when there are $\Omega(n^2)$ updates (see [18, Theorem 10]). Also note that the algorithm is randomized with one-sided error.

randomized and deterministic algorithms will be spent on recovering the destroyed decrementality.

A. Monotone Even-Shiloach Tree for Improved Randomized Algorithms

The high-level idea of our randomized algorithm is to run an existing decremental algorithm of Roditty and Zwick [34] on an *emulator*, a sparser *weighted* graph that approximates the distances in the original graph (see the full version for more detail). This approach is commonly used in the static setting (e.g., [1], [19], [20], [22], [5], [14], [15], [36], [41]), and it was recently used for the first time in the decremental setting by Bernstein and Roditty [12]. As pointed out by Bernstein and Roditty, while it is a simple task to run an existing APSP algorithm on an emulator in the static setting, doing so in the decremental setting is not easy since it will destroy the "decrementality" of the setting: when an edge in the original graph is deleted, we might have to *insert* an edge into the emulator. Thus, we cannot run decremental algorithms on an arbitrary emulator, because from the perspective of this emulator, we are not in a decremental setting.

Bernstein and Roditty manage to get around this problem by constructing an emulator with a special property[6]. Roughly speaking, they show that their emulator guarantees that *the distance between any two nodes changes* $\tilde{O}(n)$ *times*. Based on this simple property, they show that the $(2k-1,0)$-approximation algorithm of Roditty and Zwick [34] can be run on their emulator with a small running time. However, they *cannot* run the $(1+\epsilon,0)$-approximation algorithm of Roditty and Zwick on their emulator. The main reason is that this algorithm relies on a more general property of a graph under deletions: for any R between 1 and n, the distance between any two nodes changes at most R times *before it exceeds* R (i.e., it changes from 1 to R). They suggested to find an emulator with this more general property as a future research direction.

In our algorithm, we manage to run the $(1+\epsilon,0)$-approximation algorithm of Roditty and Zwick on our emulator, but in a *conceptually different* way from Bernstein and Roditty. In particular, we do not construct the emulator asked for by Bernstein and Roditty; rather, we show that there is a type of emulators such that, while edge insertions can occur often, their effect can be *ignored*. We then modify the algorithm of Roditty and Zwick to incorporate this ignorance. More precisely, the algorithm of Roditty and Zwick relies on the classic Even-Shiloach tree. We develop a simple variant of this classic algorithm called *monotone Even-Shiloach tree* that can handle restricted kinds of insertions and use it to replace the classic Even-Shiloach tree in the algorithm of Roditty and Zwick.

Our modification to the Even-Shiloach tree is as follows. Recall that the Even-Shiloach tree can maintain the distances between a specific node s and all other nodes, up to R, in $O(mR)$ total update time under edge deletions. This is because, for any node v, it has to do work $O(\deg(v))$ (the degree of v) only when the distance between s and v changes, which will happen at most R times (from 1 to R) in the decremental model. Thus, the total work on each node v will be $O(R\deg(v))$ which sums to $O(mR)$ in total. This algorithm does not perform well when there are edge insertions: one edge insertion could cause a *decrease* in the distance between s and v by as much as $\Omega(R)$, causing an additional $\Omega(R)$ distance changes. The idea of our monotone Even-Shiloach tree is extremely simple: *ignore distance decreases*! It is easy to show that the total update time of our algorithm remains the same $O(mR)$ as the classic one. The hard part is proving that it gives a good approximation when run on an emulator. This is because it does not maintain the *exact* distances on an emulator anymore. So, even when the emulator gives a good approximate distance on the original graph, our monotone Even-Shiloach tree might not. Our monotone Even-Shiloach tree does not give any guarantee for the distances in the emulator, but we can show that it still approximates the distances in the original graph. Of course, this will *not* work on any emulator; but we can show that it works on a specific type of emulators that we call *locally persevering emulators*.[7] Roughly speaking, a locally persevering emulator is an emulator where, for any "nearby"[8] nodes u and v in the original graph, either

(1) there is a shortest u-v path in the original graph that appears in the emulator, or
(2) there is a path in the emulator that approximates the distance in the original graph and *behaves in a persevering way*, in the sense that all edges of this path are in the emulator since before the first deletion and their weights never decrease. We call the latter path a *persevering path*.

Once we have the right definition of a locally persevering emulator, proving that our monotone Even-Shiloach tree gives a good distance estimate is conceptually simple (we sketch the proof idea below). Our last step is to show that such an emulator exists and can be efficiently maintained under edge deletions. We show (roughly) that we can maintain an emulator, which $(1+\epsilon,2)$-approximates the distances

[6]In fact, their emulator is basically identical to one used earlier by Bernstein [10], which is in turn a modification of a spanner developed by Thorup and Zwick [36], [37]. However, the properties they proved are entirely new.

[7]We remark that there are other emulators that can be maintained in the decremental setting, e.g., [36], [37], [34], [10], [12], [2], [21], [8]. We are the first to introduce the notion of locally persevering emulators and show that there is an emulator that has this property.

[8]Note that the word "nearby" will be parameterized by a parameter τ in the formal definition. So, formally, we must use the term (α,β,τ)-locally persevering emulator where α and β are multiplicative and additive approximation factors, respectively. See the full version for detail.

and has $\tilde{O}(n^{3/2})$ edges, in $\tilde{O}(n^{5/2})$ total update time under edge deletions. By running the $\tilde{O}(mn)$-time algorithm of Roditty and Zwick on this emulator, replacing the classic Even-Shiloach tree by our monotone version, we have the desired $\tilde{O}(n^{5/2})$-time $(1+\epsilon, 2)$-approximation algorithm. To turn this algorithm into a $(2+\epsilon, 0)$-approximation, we observe that we can check if two nodes are of distance one easily; thus, we only have to use our $(1+\epsilon, 2)$-approximation algorithm to answer a distance query when the distance between two nodes is at least two. In this case, the additive error of two can be treated as a multiplicative factor.

Proving the Approximation Guarantee of the Monotone Even-Shiloach Tree: To illustrate why our monotone Even-Shiloach tree gives a good approximation when run on a locally persevering emulator, we sketch a result that is weaker and simpler than our main results; we show how to $(3, 0)$-approximate distances from a particular node s to other nodes. This fact easily leads to a $(3+\epsilon, 0)$-approximation $\tilde{O}(n^{5/2})$-time algorithm, which gives the same guarantee as the algorithm of Bernstein and Roditty [12], and is slightly faster and reasonably simpler. To achieve this, we modify the emulator of Dor et al. [19]: Randomly select $\tilde{O}(\sqrt{n})$ nodes. At any time, the emulator consists of all edges incident to nodes of degree at most $\sqrt{n}$ and edges from each random node c to every node v of distance at most 2 from c with weight equal to their distance. It can be shown that this emulator can be maintained in $\tilde{O}(mn^{1/2}) = \tilde{O}(n^{5/2})$ time under edge deletions. Moreover, it is a $(3, 0)$-emulator with high probability: for every edge (u, v), either

(i) (u, v) is in the emulator, or
(ii) there is a path $\langle u, c, v\rangle$ of length at most three, where c is a random node.

Observe further that if (ii) happens, then the path $\langle u, c, v\rangle$ is *persevering* (as in item (2) above):

(ii') $\langle u, c, v\rangle$ must be in this emulator since before the first deletion, and the weights of the edges (u, c) and (c, v) never decrease.

It follows that this emulator is locally persevering.[9] Now we show that when we run the monotone Even-Shiloach tree on the above emulator, it gives $(3, 0)$-approximate distances between s and all other nodes. Recall that the monotone Even-Shiloach tree maintains a distance estimate, say $\ell(v)$, between s and every node v in the emulator[10]. For every node v, the value of $\ell(v)$ is regularly updated, except that when the degree of a node drops to $\sqrt{n}$ and the resulting insertion of an edge, say (u, v), decreases the distance between v and s in the emulator; in particular, $\ell(v) > \ell(u) + w(u, v)$ where $w(u, v)$ is the weight of edge (u, v). A usual way to modify the Even-Shiloach tree for dealing with such an insertion [12] is to decrease the value of $\ell(v)$ to $\ell(u)+w(u, v)$. Our monotone Even-Shiloach tree will *not* do this and keeps $\ell(v)$ unchanged. In this case, we say that the node v and the edge (u, v) *become stretched*. In general, an edge (u, v) is *stretched* if $\ell(v) > \ell(u) + w(u, v)$ or $\ell(u) > \ell(v) + w(u, v)$, and a node is stretched if it is incident to a stretched edge. Two observations that we will use are

(O1) as long as a node v is stretched, it will not change $\ell(v)$, and
(O2) a stretched edge must be an inserted edge.

We will argue that $\ell(v)$ of every node v is at most three times its real distance to s in the original graph. To prove this for a stretched node v, we simply use the fact that this is true before v becomes stretched (by induction), and $\ell(v)$ has not changed since then (by (O1)). If v is not stretched, we consider a shortest v-s path $\langle v, u_1, u_2, \ldots, s\rangle$ in the original graph. We will prove that

$$\ell(v) \leq \ell(u_1) + 3;$$

thus, assuming that $\ell(u_1)$ satisfies the claim (by induction), $\ell(v)$ will satisfy the claim as well. To prove this, observe that if the edge (v, u_1) is contained in the emulator then we know that $\ell(v) \leq \ell(u_1)+1$ (since v is not stretched), and we are done. Otherwise, by the fact that this emulator is locally persevering, we know that there is a path $P = \langle v, c, u_1\rangle$ of length at most three in the emulator, and it is persevering (see (ii')). By (O2), *edges in P are not stretched*. It follows that

$$\ell(v) \leq \ell(c)+w(v, c) \leq \ell(u_1)+w(v, c)+w(c, u_1) \leq \ell(u_1)+3,$$

where $w(v, c)$ and $w(c, u_1)$ are the current weights of edges (v, c) and (c, u_1), respectively, in the emulator. The claim follows.

In the full version, we show how to refine the above argument to obtain a $(1+\epsilon, 2)$-approximation guarantee. The first refinement, which is simple, is extending the emulator above to a $(1+\epsilon, 2)$-emulator. This is done by adding edges from every random node c to all nodes of distance at most $1/\epsilon$ from c. The next refinement, which is the main one, is the formal definition of (α, β, τ)-locally persevering emulator for some parameters α, β, and τ (see below), and extending the proof outlined above to show that the monotone Even-Shiloach tree on such an emulator will give an $(\alpha + 1/\tau, \beta)$-approximate distance. We finally show that our simple $(1+\epsilon, 2)$-emulator is a $(1+\epsilon, 2, 1/\epsilon)$-locally persevering emulator.

We end this section with the formal definition of locally-persevering emulator. In the definitions below, we view a dynamic graph as a sequence of graphs $H_0, H_1, \ldots$ respectively with edge weights $w_0, w_1, \ldots$, written as $\mathcal{H} = (H_i, w_i)_{0\leq i\leq k}$. We first define the notion of *persevering path*.

[9] We note that we are being vague here. To be formal, we define the notion of (α, β, τ)-locally persevering emulator in the full version, and the emulator we just define will be $(3, 0, 1)$-locally persevering.

[10] Here ℓ stands for "level" as $\ell(v)$ is the level of v in the breadth-first search tree rooted at s.

Definition 5 (Persevering path)**.** *Let $\mathcal{H} = (H_i, w_i)_{0\leq i\leq k}$ be a dynamic weighted graph. We say that a path P is* persevering up to time t *(where $t \leq k$) if for every edge (u, v) on P and, for all $0 \leq i \leq t$, $(u, v) \in E(H_i)$ and, for all $0 \leq i < t$, $w_i(u, v) \leq w_{i+1}(u, v)$. In other words, edges in P always exist in $\mathcal{H}$ up to time t and their weights never decrease.*

To motivate the definition of locally persevering emulator, we note that an (α, β)-emulator of a dynamic graph $\mathcal{G} = (G_i)_{0\leq i\leq k}$ is usually used to refer to another dynamic weighted graph $\mathcal{H} = (H_i, w_i)_{0\leq i\leq k}$ over the same set of nodes that preserves the distance of the original dynamic graph, i.e., for all $i \leq k$ and all nodes x and y, there is a path P in H_i such that $d_{G_i}(x, y) \leq w_i(P) \leq \alpha d_{G_i}(x, y)+\beta$. The notion of a *locally persevering* emulator puts an additional restriction on such paths. In particular, this concerns nodes x and y such that $d_{G_i}(x, y) \leq \tau$ for some parameter τ. We demand that the path P must be either a shortest path in G_i or a persevering path.

Definition 6 (Locally persevering emulator)**.** *Consider parameters $\alpha \geq 1$, $\beta \geq 0$ and $\tau \geq 1$, a dynamic graph $\mathcal{G} = (G_i)_{0\leq i\leq k}$, and a dynamic weighted graph $\mathcal{H} = (H_i, w_i)_{0\leq i\leq k}$. For every $i \leq k$, we say that a path P in G_i is* contained *in (H_i, w_i) if every edge of P is contained in H_i and has weight 1. We say that $\mathcal{H}$ is an* (α, β, τ)-locally persevering emulator of $\mathcal{G}$ *if for every $0 \leq i \leq k$ and for all nodes x and y we have $d_{G_i}(x, y) \leq d_{H_i,w_i}(x, y)$ and if $d_{G_i}(x, y) \leq \tau$, then additionally one of the following holds: either a shortest path P from x to y in G_i is contained in (H_i, w_i), or there is a path P' from x to y in $\mathcal{H}$ that is persevering up to time i and satisfies $w_i(P') \leq \alpha d_{G_i}(x, y) + \beta$.*

B. Moving Even-Shiloach Tree for Improved Deterministic Algorithms

Many distance-related algorithms in both dynamic and static settings use the following *randomized argument* as an important technique: if we select $\tilde{O}(h)$ nodes, called *centers*, uniformly at random, then every node will be at distance at most n/h from one of the centers with high probability [38], [34]. This even holds in the decremental setting (assuming an oblivious adversary). Like other algorithms, the Roditty-Zwick framework also heavily relies on this argument, which is the only reason why it is randomized. Our goal is to derandomize this argument. Specifically, for several different values of h, the Roditty-Zwick framework selects $\tilde{O}(h)$ centers and uses the randomized argument above to argue that every node in a connected component of size at least n/h is *covered* by a center in the sense that it will always be within distance at most n/h from at least one center; we call this set of centers a *center cover*. It also maintains an Even-Shiloach tree of depth $R = O(n/h)$ from these h centers, which takes a total update time of $\tilde{O}(mR)$ for each tree and thus $\tilde{O}(hmR) = \tilde{O}(mn)$ over all trees. To derandomize the above process, we have two constraints:

(1) the center cover must be maintained (i.e., every node in a component of size at least n/h has a center nearby), and
(2) the number of centers (and thus Even-Shiloach trees maintained) must be $\tilde{O}(h)$ in total.

Maintaining these constraints in the *static* setting is fairly simple, as in the following algorithm.

Algorithm 7. *As long as there is a node v in a "big" connected component (i.e., of size at least n/h) that is not covered by any center, make v a new center.*

Algorithm 7 clearly guarantees the first constraint. The second constraint follows from the fact that the distance between any two centers is more than n/h. Since understanding the proof for guaranteeing the second constraint is important for understanding our *charging argument* later, we sketch it here. Let us label the centers by numbers $j = 1, 2, \ldots, h$. For each center number j, we let B^j be a "ball" of radius $n/2h$; i.e., B^j is a set of nodes of distance at most $n/2h$ from center number j. Observe that B^j and $B^{j'}$ are disjoint for distinct centers j and j' since the distance between these centers is more than n/h Moreover, $|B^j| \geq n/2h$ since every center is in a big connected component. So, the number of balls (thus the number of centers) is at most $n/(n/2h) = 2h$. This guarantees the second constraint. Thus, we can guarantee both constraints in the static setting.

However, *after* edge deletions, some nodes in big components might not be covered anymore and, if we keep repeating Algorithm 7, we might have to keep creating new centers to the point that the second constraint is violated. The key idea that we introduce to avoid this problem is to allow a center and the Even-Shiloach tree rooted at it to *move*. We call this a *moving Even-Shiloach tree* or *moving center* data structure. Specifically, in the moving Even-Shiloach tree, we view a root (center) s *not* as a node, but as a *token* that can be placed on any node, and the task of the moving Even-Shiloach tree is to maintain the distance between the node that the root is placed on and all other nodes, up to distance R. We allow a *move operation* where we can move the root to a new node and the corresponding Even-Shiloach tree must be adjusted accordingly. To illustrate the power of the move operation, consider the following simple modification of Algorithm 7. (Later, we also have to modify this algorithm due to other problems that we will discuss next.)

Algorithm 8. *As long as there is a node v in a big connected component that is not covered by any center, we make it a center as follows. If there is a center in a small connected component, we move this center to v; otherwise, we open a new center at v.*

Algorithm 8 reuses centers and Even-Shiloach trees in small connected components[11] without violating the first constraint since nodes in small connected components do not need to be covered. The second constraint can also be guaranteed by showing that $|B^j| \geq n/2h$ for all j when we open a new center. Thus, by using moving Even-Shiloach trees, we can guarantee the two constraints above. We are, however, *not done yet*. This is because our new move operation also incurs a cost! *The most nontrivial idea in our algorithm is a charging argument to bound this cost.* There are two types of cost. First, the *relocation cost* which is the cost of constructing a new breadth-first search tree rooted at a new location of the center. This cost can be bounded by $O(m)$ since we can construct a breadth-first search tree by running the static $O(m)$-time algorithm. Thus, it will be enough to guarantee that we do not move Even-Shiloach trees more than $O(n)$ times. In fact, this is already guaranteed in Algorithm 8 since we will *never* move an Even-Shiloach tree back to the same node. The second cost, which is *much harder* to bound, is the *additional maintenance cost*. Recall that we can bound the total update time of an Even-Shiloach tree by $O(mR)$ because of the fact that the distance between its root (center) and each other node changes at most R times before exceeding R, by increasing from 1 to R. However, when we move the root from, say, a node u to its neighbor v, the distance between the new root v and some node, say x, might be smaller than the previous distance from u to x. In other words, *the decrementality property is destroyed*. Fortunately, observe that the distance change will be *at most one* per node when we move a tree to a neighboring node. Using a standard argument, we can then conclude that *moving a tree between neighboring nodes costs an additional distance maintenance cost of* $O(m)$. This motivates us to define the notion of *moving distance* to measure how far we move the Even-Shiloach trees in total. We will be able to bound the maintenance cost by $O(mn)$ if we can show that the total moving distance (summing over all moving Even-Shiloach trees) is $O(n)$. Bounding the total moving distance by $O(n)$ while having only $O(h)$ Even-Shiloach trees is the most challenging part in obtaining our deterministic algorithm. We do it by using a careful charging argument. We sketch this argument here. For more intuition and detail, see the full version.

Charging Argument for Bounding the Total Moving Distance: Recall that we denote the centers by numbers $j = 1, 2, \ldots, h$. We make a few modifications to Algorithm 8. The most important change is the introduction of the set C^j for each center j (which is the root of a moving Even-Shiloach tree). This will lead to a few other changes. The importance of C^j is that we will "charge" the moving cost to nodes in C^j; in particular, we bound the total moving distance to be $O(n)$ by showing that the moving distance of center j can be bounded by $|C^j|$, and C^j and $C^{j'}$ are disjoint for distinct centers j and j'. The other important changes are the definitions of "ball" and "small connected component" which will now depend on C^j.

- We change the definition of B^j from a ball of radius $n/2h$ to a ball of radius $(n/2h) - |C^j|$.
- We redefine the notion of "small connected component" as follows: we say that a center j is in a small connected component if the connected component containing it has less than $(n/2h) - |C^j|$ nodes (instead of n/h nodes).

These new definitions might not be intuitive, but they are crucial for the charging argument. We also have to modify Algorithm 8 in a counter-intuitive way: the most important modification is that we have to give up the nice property that the distance between any two center is more than $n/2h$ as in Algorithms 7 and 8. In fact, we will *always* move a center out of a small connected component, and we will move it *as little as possible*, even though the new location could be near other centers. In particular, consider the deletion of an edge (u, v). It can be shown that there is *at most one* center j that is in a small connected component (according to the new definition), and this center j must be in the same connected component as u or v. Suppose that such a center j exists, and it is in the same connected component as u, say X. Then, we will move center j to v, which is just enough to move j out of component X (it is easy to see that v is the node outside of X that is nearest to j before the deletion). We will also update C^j by adding all nodes of X to C^j. This finishes the moving step, and it can be shown that there is no center in a small connected component now. Next, we make sure that every node is covered by opening a new center at nodes that are not covered, as in Algorithm 7. To conclude, our algorithm is as follows.

Algorithm 9. *Consider the deletion of an edge (u, v). Check if there is a center j that is in a "small" connected component X (of size less than $(n/2h) - |C^j|$). If there is such a j (there will be at most one such j), move it out of X to a new node which is the unique node in $\{u, v\} \setminus X$. After moving, execute the static algorithm as in Algorithm 7.*

To see that the total moving distance is $O(n)$, observe that when we move a center j out of component X in Algorithm 9, we incur a moving distance of at most $|X|$ (since we can move j along a path in X). Thus, we can always bound the total moving distance of center j by $|C^j|$. We additionally show that C^j and $C^{j'}$ are disjoint for different centers j and j'. So, the total moving distance over all centers is at most $\sum_j |C^j| \leq n$. We also have to bound the number of centers. Since we give up the nice

[11]We note the detail that we need a deterministic dynamic connectivity data structure [25], [28] to implement Algorithm 8. The additional cost incurred is negligible.

property that centers are far apart, we cannot use the same argument to show that the sets B^j are disjoint and big (i.e., $|B^j| \geq n/2h$), as in Algorithm 8 and Algorithm 9. However, using C^j, we can still show something very similar: We can show that $B^j \cup C^j$ and $B^{j'} \cup C^{j'}$ are disjoint for distinct j and j', and that $|B^j \cup C^{j'}| \geq n/2h$. Thus, we can still bound the number of centers by $O(h)$ as before.

IV. Conclusion

We obtained two new algorithms for solving the decremental approximate APSP algorithm in unweighted undirected graphs. The first algorithm provides a $(1+\epsilon, 2)$-approximation and has a total update time of $\tilde{O}(n^{5/2}/\epsilon^2)$ and constant query time. The main idea behind this algorithm is to run an algorithm of Roditty and Zwick [34] on a sparse dynamic emulator. In particular, we modify the central shortest paths tree data structure of Even and Shiloach [23], [29] to deal with edge insertions in a monotone manner. Our approach is conceptually different from the approach of Bernstein and Roditty [12] who also maintain an ES-tree in a sparse dynamic emulator. The second algorithm provides a $(1+\epsilon, 0)$-approximation and has a *deterministic* total update time of $\tilde{O}(mn \log n/\epsilon)$ and constant query time. We obtain it by derandomizing the algorithm of [34] using a new amortization argument based on the idea of relocating Even-Shiloach trees.

Our results directly motivate the following directions for further research. It would be interesting to extend our derandomization technique to other randomized algorithms. In particular, we ask whether it is possible to derandomize the *exact* decremental APSP algorithm of Baswana, Hariharan and Sen [7] (total update time $\tilde{O}(n^3)$).

Another interesting direction is to check whether our monotone ES-tree approach also works for other dynamic emulators, also for weighted graphs. One of the tools that we used was a dynamic $(1+\epsilon, 2)$-emulator for unweighted undirected graphs. Is it also possible to obtain *purely additive* dynamic emulators or spanners with small additive error?

The sparsification techniques used here and at other places only work for undirected graphs. Can we also get subcubic algorithms for *directed* graphs (without relying on these sparsification techniques)? In fact, not even decremental single-source reachability in directed graphs can currently be done faster than with a total time of $O(mn)$ for a sequence of $\Omega(m)$ deletions.

Maybe the most important open problem in this field is a faster APSP algorithm for the fully dynamic setting. The fully dynamic algorithm of Demetrescu and Italiano [17] provides exact distances and takes time $\tilde{O}(n^2)$ per update, which is essentially optimal. Is it possible to get a faster fully dynamic algorithm that still provides a good approximation, for example a $(1+\epsilon)$-approximation?

References

[1] 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, 1996.

[2] G. Ausiello, P. G. Franciosa, and G. F. Italiano, "Small Stretch Spanners on Dynamic Graphs," *Journal of Graph Algorithms and Applications*, vol. 10, no. 2, pp. 365–385, 2006, announced at ESA, 2005.

[3] G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, and U. Nanni, "Incremental algorithms for minimal length paths," *J. Algorithms*, vol. 12, no. 4, pp. 615–638, 1991, announced at SODA, 1990.

[4] ——, "On-line computation of minimal and maximal length paths," *Theor. Comput. Sci.*, vol. 95, no. 2, pp. 245–261, 1992.

[5] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg, "Near-linear time construction of sparse neighborhood covers," *SIAM J. Comput.*, vol. 28, no. 1, pp. 263–277, 1998, announced at FOCS, 1993.

[6] S. Baswana, R. Hariharan, and S. Sen, "Maintaining all-pairs approximate shortest paths under deletion of edges," in *SODA*, 2003, pp. 394–403.

[7] ——, "Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths," *Journal of Algorithms*, vol. 62, no. 2, pp. 74–92, 2007, announced at STOC, 2002.

[8] S. Baswana, S. Khurana, and S. Sarkar, "Fully Dynamic Randomized Algorithms for Graph Spanners," *ACM Transactions on Algorithms*, vol. 8, no. 4, pp. 35:1–35:51, 2012, announced at ESA, 2004, and SODA, 2008.

[9] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, "On the power of randomization in on-line algorithms," *Algorithmica*, vol. 11, no. 1, pp. 2–14, 1994.

[10] A. Bernstein, "Fully Dynamic $(2+\epsilon)$ Approximate All-Pairs Shortest Paths with Fast Query and Close to Linear Update Time," in *FOCS*, 2009, pp. 693–702.

[11] ——, "Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs," in *STOC*, 2013, p. to appear.

[12] A. Bernstein and L. Roditty, "Improved Dynamic Algorithms for Maintaining Approximate Shortest Paths Under Deletions," in *SODA*, 2011, pp. 1355–1365.

[13] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[14] E. Cohen, "Fast algorithms for constructing t-spanners and paths with stretch t," *SIAM J. Comput.*, vol. 28, no. 1, pp. 210–236, 1998, announced at FOCS, 1993.

[15] E. Cohen and U. Zwick, "All-pairs small-stretch paths," *J. Algorithms*, vol. 38, no. 2, pp. 335–353, 2001, announced at SODA, 1997.

[16] C. Demetrescu and G. F. Italiano, "Improved bounds and new trade-offs for dynamic all pairs shortest paths," in *ICALP*, 2002, pp. 633–643.

[17] ——, "A New Approach to Dynamic All Pairs Shortest Paths," *Journal of the ACM*, vol. 51, no. 6, pp. 968–992, 2004, announced at STOC, 2003.

[18] ——, "Fully dynamic all pairs shortest paths with real edge weights," *Journal of Computer and System Sciences*, vol. 72, no. 5, pp. 813–837, 2006, announced at FOCS, 2001.

[19] D. Dor, S. Halperin, and U. Zwick, "All-Pairs Almost Shortest Paths," *SIAM Journal on Computing*, vol. 29, no. 5, pp. 1740–1759, 2000, announced at FOCS, 1996.

[20] M. Elkin, "Computing almost shortest paths," *ACM Transactions on Algorithms*, vol. 1, no. 2, pp. 283–323, 2005, announced at PODC, 2001.

[21] ——, "Streaming and Fully Dynamic Centralized Algorithms for Constructing and Maintaining Sparse Spanners," *ACM Transactions on Algorithms*, vol. 7, no. 2, pp. 20:1–20:17, 2011, announced at ICALP, 2007.

[22] M. Elkin and D. Peleg, "$(1+\epsilon, \beta)$-spanner constructions for general graphs," *SIAM J. Comput.*, vol. 33, no. 3, pp. 608–631, 2004, announced at STOC, 2001.

[23] S. Even and Y. Shiloach, "An On-Line Edge-Deletion Problem," *Journal of the ACM*, vol. 28, no. 1, pp. 1–4, 1981.

[24] J. Fakcharoenphol and S. Rao, "Planar graphs, negative weight edges, shortest paths, and near linear time," *J. Comput. Syst. Sci.*, vol. 72, no. 5, pp. 868–889, 2006, announced at FOCS, 2001.

[25] M. Henzinger and V. King, "Maintaining minimum spanning forests in dynamic graphs," *SIAM Journal on Computing*, vol. 31, no. 2, p. 364374, 2001, announced at ICALP, 1997.

[26] M. Henzinger, S. Krinninger, and D. Nanongkai, "Dynamic approximate all-pairs shortest paths: Breaking the $O(mn)$ barrier and derandomization," *CoRR*, vol. abs/1308.0776, 2013. [Online]. Available: http://arxiv.org/abs/1308.0776

[27] M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian, "Faster shortest-path algorithms for planar graphs," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 3–23, 1997, announced at STOC, 1994.

[28] J. Holm, K. de Lichtenberg, and M. Thorup, "Poly-Logarithmic Deterministic Fully-Dynamic Algorithms for Connectivity, Minimum Spanning Tree, 2-Edge, and Biconnectivity," *Journal of the ACM*, vol. 48, no. 4, pp. 723–760, 2001, announced at STOC, 1998.

[29] V. King, "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs," in *STOC*, 1999, pp. 81–91.

[30] P. Loubal and B. A. T. S. Commission, *A Network Evaluation Procedure*. Bay Area Transportation Study Commission, 1967.

[31] J. D. Murchland, "The effect of increasing or decreasing the length of a single arc on all shortest distances in a graph," London Business School, Transport Network Theory Unit, Tech. Rep. LBS-TNT-26, 1967.

[32] L. Roditty and R. Tov, "Approximating the girth," *ACM Trans. Algorithms*, vol. 9, no. 2, pp. 15:1–15:13, Mar. 2013, announced at SODA, 2011.

[33] L. Roditty and U. Zwick, "On Dynamic Shortest Paths Problems," *Algorithmica*, vol. 61, no. 2, pp. 389–401, 2011, announced at ESA, 2004.

[34] ——, "Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs," *SIAM Journal on Computing*, vol. 41, no. 3, pp. 670–683, 2012, announced at FOCS, 2004.

[35] M. Thorup, "Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles," in *SWAT*, 2004, pp. 384–396.

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

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

[38] J. D. Ullman and M. Yannakakis, "High-probability parallel transitive-closure algorithms," *SIAM J. Comput.*, vol. 20, no. 1, pp. 100–125, 1991.

[39] V. Vassilevska, R. Williams, and R. Yuster, "All pairs bottleneck paths and max-min matrix products in truly subcubic time," *Theory of Computing*, vol. 5, no. 1, pp. 173–189, 2009, announced at STOC, 2007.

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

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