


# New Tradeoffs for Decremental Approximate All-Pairs Shortest Paths

Michal Dory 

University of Haifa, Israel

Sebastian Forster 

Department of Computer Science, University of Salzburg, Austria

Yasamin Nazari 

Vrije Universiteit Amsterdam, The Netherlands

Tijn de Vos 

Department of Computer Science, University of Salzburg, Austria

---

## Abstract

We provide new tradeoffs between approximation and running time for the decremental all-pairs shortest paths (APSP) problem. For undirected graphs with  $m$  edges and  $n$  nodes undergoing edge deletions, we provide four new approximate decremental APSP algorithms, two for weighted and two for unweighted graphs. Our first result is  $(2 + \epsilon)$ -APSP with total update time  $\tilde{O}(m^{1/2}n^{3/2})$  (when  $m = n^{1+c}$  for any constant  $0 < c < 1$ ). Prior to our work the fastest algorithm for weighted graphs with approximation at most 3 had total  $\tilde{O}(mn)$  update time for  $(1 + \epsilon)$ -APSP [Bernstein, SICOMP 2016]. Our second result is  $(2 + \epsilon, W_{u,v})$ -APSP with total update time  $\tilde{O}(nm^{3/4})$ , where the second term is an additive stretch with respect to  $W_{u,v}$ , the maximum weight on the shortest path from  $u$  to  $v$ .

Our third result is  $(2 + \epsilon)$ -APSP for unweighted graphs in  $\tilde{O}(m^{7/4})$  update time, which for sparse graphs ( $m = o(n^{8/7})$ ) is the first subquadratic  $(2 + \epsilon)$ -approximation. Our last result for unweighted graphs is  $(1 + \epsilon, 2(k - 1))$ -APSP, for  $k \geq 2$ , with  $\tilde{O}(n^{2-1/k}m^{1/k})$  total update time (when  $m = n^{1+c}$  for any constant  $c > 0$ ). For comparison, in the special case of  $(1 + \epsilon, 2)$ -approximation, this improves over the state-of-the-art algorithm by [Henzinger, Krinninger, Nanongkai, SICOMP 2016] with total update time of  $\tilde{O}(n^{2.5})$ . All of our results are randomized, work against an oblivious adversary, and have constant query time.

**2012 ACM Subject Classification** Theory of computation  $\rightarrow$  Dynamic graph algorithms; Theory of computation  $\rightarrow$  Shortest paths

**Keywords and phrases** Decremental Shortest Path, All-Pairs Shortest Paths

**Digital Object Identifier** 10.4230/LIPIcs.ICALP.2024.58

**Category** Track A: Algorithms, Complexity and Games

**Related Version** *Full Version:* <https://arxiv.org/abs/2211.01152> [34]

**Funding** *Michal Dory:* This work was partially conducted while the author was a postdoc at ETH Zurich. This work was supported in part by funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No. 853109), and the Swiss National Foundation (project grant 200021-184735).

*Sebastian Forster, Yasamin Nazari, Tijn de Vos:* This work is supported by the Austrian Science Fund (FWF): P 32863-N. This project has received funding from the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme (grant agreement No 947702).



© Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos;  
licensed under Creative Commons License CC-BY 4.0

51st International Colloquium on Automata, Languages, and Programming (ICALP 2024).

Editors: Karl Bringmann, Martin Grohe, Gabriele Puppis, and Ola Svensson;

Article No. 58; pp. 58:1–58:19



Leibniz International Proceedings in Informatics

Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



## 1 Introduction

The dynamic algorithms paradigm is becoming increasingly popular for studying algorithmic questions in the presence of gradually changing inputs. A natural goal in this area is to design algorithms that process each change to the input as fast as possible to adapt the algorithm's output (or a data structure for querying the output) to the current state of the input. The time spent after an update to perform these computations is called the *update time* of the algorithm. In many cases, bounds on the update time are obtained in an *amortized* sense as an average over a long enough sequence of updates. Among dynamic graph problems, the question of maintaining exact or approximate shortest paths has received considerable attention in the past two decades. The main focus usually lies on maintaining a distance oracle that answers queries for the distance between a pair of nodes. For this problem, we call an algorithm *fully dynamic* if it supports both insertions and deletions of edges, and *partially dynamic* if it supports only one type of updates; in particular we call it *decremental* if it only supports edge deletions (which is the focus of this paper), and *incremental* if it only supports edge insertions.

The running times of partially dynamic algorithms are usually characterized by their bounds on the *total update time*, which is the accumulated time for processing all updates in a sequence of at most  $m$  deletions (where  $m$  is the maximum number of edges ever contained the graph). A typical design choice, which we also impose in this paper, is small (say polylogarithmic) query time. In particular, our algorithms will have constant query time. While fully dynamic algorithms are more general, the restriction to only one type of updates in partially dynamic algorithms often admits much faster update times. In particular, some partially dynamic algorithms have a total update time that almost matches the running time of the fastest static algorithm, i.e., computing *all* updates does not take significantly more time than processing the graph once.

### Decremental shortest paths

For the decremental single-source shortest paths problem, conditional lower bounds [58, 48] suggest that *exact* decremental algorithms have an  $\Omega(mn)$  bottleneck in their total update time (up to subpolynomial factors). On the other hand, this problem admits a  $(1 + \epsilon)$ -approximation (also called *stretch*) with total update time  $m^{1+o(1)}$  in weighted, undirected graphs [22, 47, 20, 19, 21], which exceeds the running time of the state-of-the-art static algorithm by only a subpolynomial factor. Hence for the single-source shortest paths problem on undirected graphs we can fully characterize for which multiplicative stretches the total update time of the fastest decremental algorithm matches (up to subpolynomial factors) the running time of the fastest static algorithm.

Obtaining a similar characterization for the decremental all-pairs shortest paths (APSP) problem is an intriguing open question. Conditional lower bounds [32, 48] suggest an  $\Omega(mn)$  bottleneck in the total update time of decremental APSP algorithms with (a) any finite stretch on *directed* graphs and (b) with any stretch guarantee with a multiplicative term of  $\alpha \geq 1$  and an additive term of  $\beta \geq 0$  with  $2\alpha + \beta < 4$  on *undirected* graphs. This motivates the study of decremental  $(\alpha, \beta)$ -approximate APSP algorithms such that  $2\alpha + \beta \geq 4$ , i.e., with multiplicative stretch  $\alpha \geq 2$  or additive stretch  $\beta \geq 2$ .

Apart from two notable exceptions [46, 5], all known decremental APSP algorithms fall into one of two categories. They either (a) maintain exact distances or have a relatively small multiplicative stretch of  $(1 + \epsilon)$  or (b) they have a stretch of at least 3. The space in between is largely unexplored in the decremental setting. This stands in sharp contrast to the static

setting where for undirected graphs approximations guarantees different from multiplicative  $(1 + \epsilon)$  or “3 and above” have been the focus of a large body of works [7, 32, 28, 14, 16, 51, 11, 55, 61, 52, 8, 9]. The aforementioned exceptions in dynamic algorithms [46, 5], concern unweighted undirected graphs and maintain  $(1 + \epsilon, 2)$ -approximations that simultaneously have a multiplicative error of  $1 + \epsilon$  and an additive error of 2, which implies a purely multiplicative  $(2 + \epsilon)$ -approximation. The algorithms run in  $\tilde{O}(n^{5/2})^1$  and  $n^{5/2+o(1)}$  total update time respectively. In the next section, we will detail how our algorithms improve over this total update time and generalize to weighted graphs.

Concerning the fully dynamic setting, Bernstein [17] provided an algorithm for  $(2 + \epsilon)$ -APSP which takes  $m^{1+o(1)}$  time per update. Although we are not aware of any lower bounds, it seems to be hard to beat this: no improvements have been made since. The lack of progress in the fully-dynamic setting motivates the study in a partially dynamic setting, where we obtain improvements for the decremental case.

## 1.1 Our Results

In this paper, we provide novel decremental APSP algorithms with approximation guarantees that previously were mostly unexplored in the decremental setting. Our algorithms are randomized and we assume an oblivious adversary. For each pair of nodes, we can not only provide the distance estimate, but we can also report a shortest path  $\pi$  of this length in  $\tilde{O}(|\pi|)$  time using standard techniques, see e.g. [43].

### $(2 + \epsilon)$ -APSP for weighted graphs

Our first contribution is an algorithm for maintaining a  $(2 + \epsilon)$ -approximation in weighted, undirected graphs.

► **Theorem 1.** *Given a weighted graph  $G$  and a constant  $0 < \epsilon < 1$ , there is a decremental data structure that maintains a  $(2 + \epsilon)$ -approximation of APSP. The algorithm has constant query time and the total update time is w.h.p.*

- $\tilde{O}(m^{1/2}n^{3/2}\log^2(nW))$  if  $m = n^{1+\Omega(1)}$  and  $m \leq n^{2-\rho}$  for an arbitrary small constant  $\rho$ ,
- $\tilde{O}(m^{1/2}n^{3/2+\rho}\log^2(nW))$  if  $m \geq n^{2-\rho}$  for any constant  $\rho$ ,
- $\tilde{O}(m^{1/2}n^{3/2+o(1)}\log^2(nW))$  otherwise,

where  $W$  is the ratio between the maximum and minimum weight.

The fastest known algorithm with an approximation ratio at least as good as ours is the  $(1 + \epsilon)$ -approximate decremental APSP algorithm by Bernstein [18] with total update time  $\tilde{O}(mn)$ . With our  $(2 + \epsilon)$ -approximation, we improve upon this total update time when  $m = n^{1+\eta}$ , for any  $\eta > 0$ . Furthermore, the fastest known algorithm with a larger approximation ratio than our algorithm is the  $(3 + \epsilon)$ -approximate decremental distance oracle by Łącki and Nazari [53] with total update time  $\tilde{O}(m\sqrt{n})$ . Our result also has to be compared to the *fully dynamic* algorithm of Bernstein [17] for maintaining a  $(2 + \epsilon)$ -approximation that takes amortized time  $m^{1+o(1)}$  per update. Note that in *unweighted* graphs,  $(2 + \epsilon)$ -approximate decremental APSP algorithm can be maintained with total update time  $\tilde{O}(n^{2.5})$  (which is implied by the results of Abraham and Chechik [5] and Henzinger, Krinninger, and Nanongkai [46]). Our approach improves upon this bound for most densities, and matches it for  $m = n^2$ .

<sup>1</sup> In the introduction, we make two simplifying assumptions: (a)  $\epsilon$  is a constant and (b) the ratio  $W$  between the maximum and minimum weight is polynomial in  $n$ . Unless otherwise noted, the cited algorithms have constant or polylogarithmic query time. Throughout we use  $\tilde{O}(\cdot)$  notation to omit factors that are polylogarithmic in  $n$ .

**$(2 + \epsilon, 1)$ -APSP for unweighted and  $(2 + \epsilon, W_{u,v})$ -APSP for weighted graphs**

Our second contribution is a faster algorithm with an additional additive error term of 1 for unweighted graphs, which in turn can be used to obtain our unweighted  $(2 + \epsilon)$ -APSP result. The corresponding generalization to weighted graphs can be formulated as follows.

► **Theorem 2.** *Given a weighted graph  $G$  and a constant  $0 < \epsilon < 1$ , there is a decremental data structure that maintains a  $(2 + \epsilon, W_{u,v})$ -approximation for APSP, where  $W_{u,v}$  is the maximum weight on a shortest path from  $u$  to  $v$ . The algorithm has constant query time and the total update time is w.h.p.  $\tilde{O}(nm^{3/4} \log^2(nW))$  if  $m = n^{1+\Omega(1)}$ , and  $\tilde{O}(n^{1+o(1)}m^{3/4} \log^2(nW))$  otherwise, where  $W$  is the ratio between the maximum and minimum weight.*

 **$(2 + \epsilon)$ -APSP for unweighted graphs**

We obtain this result by a general reduction from mixed approximations to purely multiplicative approximations, which might be of independent interest beyond the dynamic setting. See Theorem 5 for the statement. We can then combine this with Theorem 2 to obtain a fast algorithm for non-dense unweighted graphs.

► **Theorem 3.** *Given an undirected unweighted graph  $G$ , there is a decremental data structure that maintains a  $(2 + \epsilon)$ -approximation for APSP with constant query time. W.h.p. the total update time is bounded by  $\tilde{O}(m^{7/4})$  if  $m = n^{1+\Omega(1)}$ , and  $\tilde{O}(m^{7/4+o(1)})$  otherwise.*

Note that for  $m = o(n^{8/7})$  this gives the first subquadratic decremental  $(2 + \epsilon)$ -approximate APSP algorithm. For  $m \leq n^{6/5}$ , this approach is faster than our weighted result, Theorem 1, which was already beating the unweighted state-of-the-art [5, 46].

 **$(1 + \epsilon, 2(k - 1))$ -APSP for unweighted graphs**

Our fourth contribution is an algorithm for unweighted, undirected graphs that maintains, for any  $k \geq 2$ , a  $(1 + \epsilon, 2(k - 1))$ -approximation, i.e., a distance estimate that has a multiplicative error of  $1 + \epsilon$  and an additive error of  $2(k - 1)$ .

► **Theorem 4.** *Given an undirected unweighted graph  $G$ , a constant  $0 < \epsilon < 1$  and an integer  $2 \leq k \leq \log n$ , there is a decremental data structure that maintains  $(1 + \epsilon, 2(k - 1))$ -approximation for APSP with constant query time. The expected total update time is bounded by  $\min\{O(n^{2-1/k+o(1)}m^{1/k}), \tilde{O}((n^{2-1/k}m^{1/k})O(1/\epsilon)^{k/\rho})\}$  where  $m = n^{1+\rho}$ .*

Note that for small values of  $k$  and if  $m = n^{1+\rho}$  for a constant  $\rho > 0$ , we get total update time of  $\tilde{O}(n^{2-1/k}m^{1/k})$ , and otherwise we have an extra  $n^{o(1)}$  factor. In addition, in the special case of  $k = \log n$ , we get a near-quadratic update time of  $O(n^{2+o(1)})$ . The state-of-the-art for a purely multiplicative  $(1 + \epsilon)$ -approximation is the algorithm of Roditty and Zwick with total update time  $\tilde{O}(mn)$ .<sup>2</sup> It was shown independently by Abraham and Chechik [5] and by Henzinger, Krinninger, and Nanongkai [46] how to improve upon this total update time bound at the cost of an additional small additive error term: a  $(1 + \epsilon, 2)$ -approximation can be maintained with total update time  $\tilde{O}(n^{2.5})$ . This has been generalized by Henzinger, Krinninger, and Nanongkai [45] to an additive error term of  $2(1 + \frac{2}{\epsilon})^{k-2}$  and total update time  $\tilde{O}(n^{2+1/k}O(\frac{1}{\epsilon})^{k-1})$ . We improve upon this tradeoff in two ways: (1) our additive term is independent of  $1/\epsilon$  and *linear* in  $k$  and (2) our algorithm profits from graphs being sparse.

<sup>2</sup> Note that the algorithms of Roditty and Zwick [59] for unweighted, undirected graphs precedes the more general algorithm of Bernstein [18] for weighted, directed graphs.

## Static follow-up work

Interestingly, our techniques inspired a new algorithm for a static 2-approximate distance oracle [33], giving the first such distance oracle algorithm with subquadratic construction time for sparse graphs. Moreover, for  $m = n$  they match conditional lower bounds [56, 3].

## 1.2 Related Work

### Static algorithms

The baseline for the static APSP problem are the exact textbook algorithms with running times of  $O(n^3)$  and  $\tilde{O}(mn)$ , respectively. There are several works obtaining improvements upon these running times by either shaving subpolynomial factors or by employing fast matrix multiplication, which sometimes comes at the cost of a  $(1 + \epsilon)$ -approximation instead of an exact result. See [67] and [66], and the references therein, for details on these approaches. For the regime of stretch 3 and more in undirected graphs, a multitude of algorithms has been developed with the distance oracle of Thorup and Zwick [64] arguably being the most well-known constructions. In the following, we focus on summarizing the state of affairs for approximate APSP with stretch between  $1 + \epsilon$  and 3.

In *weighted*, undirected graphs, Cohen and Zwick [28] obtained a 2-approximation with running time  $\tilde{O}(m^{1/2}n^{3/2})$ . This running time has been improved to  $\tilde{O}(m\sqrt{n}+n^2)$  by Baswana and Kavitha [14] and, employing fast matrix multiplication, to  $\tilde{O}(n^{2.25})$  by Kavitha [51]. For  $(2 + \epsilon)$ -APSP, Dory et al. [33] provide two algorithms, for dense graphs:  $O(n^{2.214})$  and sparse graphs:  $\tilde{O}(mn^{2/3})$ . In addition, efficient approximation algorithms for stretches of  $\frac{7}{3}$  [28] and  $\frac{5}{2}$  [51] have been obtained. These results have recently been generalized by Akav and Roditty [9] who presented an algorithm with stretch  $2 + \frac{k-2}{k}$  and running time  $\tilde{O}(m^{2/k}n^{2-3/k} + n^2)$  for any  $k \geq 2$ .

In *unweighted*, undirected graphs, a  $(1, 2)$ -approximation algorithm with running time  $\tilde{O}(n^{2.5})$  has been presented by Aingworth, Chekuri, Indyk and Motwani [7]. This has been improved by Dor, Halperin, and Zwick [32] that showed a  $(1, 2)$ -approximation with running time  $\tilde{O}(\min\{n^{3/2}m^{1/2}, n^{7/3}\})$ . They also show a generalized version of the algorithm that gives stretch  $(1, k)$  and running time  $\tilde{O}(\min\{n^{2-2/(k+2)}m^{2/(k+2)}, n^{2+2/(3k-2)}\})$  for every even  $k > 2$ . Recently faster  $(1, 2)$ -approximation algorithms based on fast matrix multiplication techniques were developed [31, 36], the fastest of them runs in  $O(n^{2.260})$  time [36]. This was extended to a  $(1 + \epsilon, 2)$ -approximation in  $O(n^{2.152})$  [33]. In addition, recently Roditty [57] extended the approach of [32] to obtain a combinatorial  $(2, 0)$ -approximation for APSP in  $\tilde{O}(n^{2.25})$  time in unweighted undirected graphs. Using fast matrix multiplication, this running time was improved to  $O(n^{2.032})$  [33, 60].

Berman and Kasiviswanathan [16] showed how to compute a  $(2, 1)$ -approximation in time  $\tilde{O}(n^2)$ . Subsequent works [11, 14, 55, 61, 52] have improved the polylogarithmic factors in the running time and the space requirements for such nearly 2-approximations. Recently, slightly subquadratic algorithms have been given: an algorithm with stretch  $(2(1 + \epsilon), 5)$  by Akav and Roditty [8], and an algorithm with stretch  $(2, 3)$  by Chechik and Zang [25].

### Decremental algorithms

The fastest algorithms for maintaining exact APSP under edge deletions have total update time  $\tilde{O}(n^3)$  [30, 13, 39]. There are several algorithms that are more efficient at the cost of returning only an approximate solution. In particular, a  $(1 + \epsilon)$ -approximation can be maintained in total time  $\tilde{O}(mn)$  [59, 18, 49]. If additionally, an additive error of 2 is tolerable,

then a  $(1 + \epsilon, 2)$ -approximation can be maintained in total time  $\tilde{O}(n^{2.5})$  in unweighted, undirected graphs [46, 5]. Note that such a  $(1 + \epsilon, 2)$ -approximation directly implies a  $(2 + \epsilon)$ -approximation because the only paths of length 1 are edges between neighboring nodes. All of these decremental approximation algorithms are randomized and assume an oblivious adversary. Deterministic algorithms with stretch  $1 + \epsilon$  exist for unweighted, undirected graphs with running time  $\tilde{O}(mn)$  [46], for weighted, undirected graphs with running time  $\tilde{O}(mn^{1+o(1)})$  [21] and for weighted, directed graphs with running time  $\tilde{O}(n^3)$  [50].

Decremental approximate APSP algorithms of larger stretch, namely at least 3, have first been studied by Baswana, Hariharan, and Sen [12]. After a series of improvements [59, 22, 6, 47], the state-of-the-art algorithms of [24, 53] maintain  $(2k - 1)(1 + \epsilon)$ -approximate all-pairs shortest paths for any integer  $k \geq 2$  and  $0 < \epsilon \leq 1$  in total update time  $\tilde{O}((m + n^{1+o(1)})n^{1/k})$  with query time  $O(\log \log(nW))$  and  $O(k)$  respectively. All of these “larger stretch” algorithms are randomized and assume an oblivious adversary.

Recently, deterministic algorithms have been developed by Chuzhoy and Saranurak [27] and by Chuzhoy [26]. One tradeoff in the algorithm of Chuzhoy [26] for example provides total update time  $\tilde{O}(m^{1+\mu})$  for any constant  $\mu$  and polylogarithmic stretch. As observed by Mađry [54], decremental approximate APSP algorithms that are deterministic – or more generally work against an adaptive adversary – can lead to fast static approximation algorithms for the maximum multicommodity flow problem via the Garg-Könemann-Fleischer framework [44, 41]. The above upper bounds for decremental APSP have recently been contrasted by conditional lower bounds [4] stating that constant stretch cannot be achieved with subpolynomial update and query time under certain hardness assumptions on 3SUM or (static) APSP.

### Fully dynamic algorithms

The reference point in fully dynamic APSP with subpolynomial query time is the *exact* algorithm of Demetrescu and Italiano [29] with update time  $\tilde{O}(n^2)$  (with log-factor improvements by Thorup [62]). For undirected graphs, several fully dynamic distance oracles have been developed. In particular, Bernstein [17] developed a distance oracle of stretch  $2 + \epsilon$  (for any given constant  $0 < \epsilon \leq 1$ ) and update time  $O(m^{1+o(1)})$ . In the regime of stretch at least 3, tradeoffs between stretch and update time have been developed by Abraham, Chechik, and Talwar [6], and by Forster, Goranci, and Henzinger [42]. Finally, most fully dynamic algorithm with update time sensitive to the edge density can be combined with a fully dynamic spanner algorithm leading to faster update time at the cost of a multiplicative increase in the stretch, see [15] for the seminal work on fully dynamic spanners.

## 2 High-Level Overview

In this section we provide a high-level overview of our algorithms, for the complete algorithms and their proof we refer to the full version. First we describe our  $(2 + \epsilon)$ -APSP algorithms for weighted graphs, by giving a simpler static version first. Then we describe our  $(2 + \epsilon, W_{u,v})$ -APSP algorithm, where we introduce a notion of *bunch overlap threshold* to overcome the challenge of dynamically maintaining a well-known static adaptive sampling technique [63] (see Section 2.2). We give a reduction from mixed approximations to purely multiplicative approximations, which together with the previous result gives our  $(2 + \epsilon)$ -APSP algorithm for unweighted graphs. Finally, we describe our  $(1 + \epsilon, 2(k - 1))$ -APSP algorithm for any  $k \geq 2$  in unweighted graphs.



## 2.1 $(2 + \epsilon)$ -APSP for Weighted Graphs

We first present a warm-up static algorithm, that we later turn into a dynamic algorithm.

### 2.1.1 Static 2-APSP

We start by reviewing the concepts of bunch and cluster as defined in the seminal distance oracle construction of [64].

Let  $0 < p < 1$  be a parameter and  $A$  a set of nodes sampled with probability  $p$ . For each node  $u$ , the pivot  $p(u)$  is the closest node in  $A$  to  $u$ . The *bunch*  $B(v)$  of a node  $v$  is the set  $B(v) := \{u \in V : d(u, v) < d(v, p(v))\}$  and the *clusters* are  $C(u) := \{v \in V : d(u, v) < d(v, p(v))\} = \{v \in V : u \in B(v)\}$ . Hence bunches and clusters are reverse of each other:  $u \in B(v)$  if and only if  $v \in C(u)$ .

These structures have been widely used in various settings, including the decremental model (e.g. [46, 58, 53, 24]). We can use these existing decremental algorithms to maintain bunches and clusters, but will need to develop new decremental tools to get our desired tradeoffs that we will describe in Section 2.1.2.

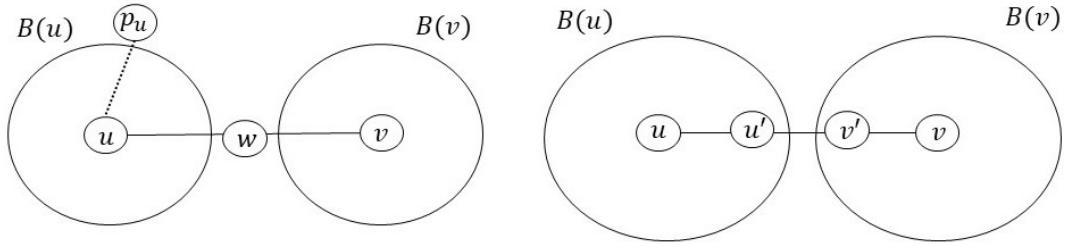
Using the well-known distance oracles of [64], we know that by storing the clusters and bunches and the corresponding distances inside them, we can query 3-approximate distances for any pair. At a high-level, for querying distance between a pair  $u, v$  we either have  $u \in B(v)$  or  $v \in B(u)$  and so we explicitly have stored their distance, or we can get a 3-approximate estimate by computing  $\min\{d(u, p(u)) + d(p(u), v), d(v, p(v)) + d(p(v), u)\}$ . In the following we explain that in some special cases we can use these estimates to obtain a 2-approximation for a pair  $u, v$ , and in other cases by storing more information depending on how  $B(u)$  and  $B(v)$  overlap we can also obtain a 2-approximation to  $d(u, v)$ .

Let us assume that for all nodes  $v$  we have computed  $B(v)$  and have access to all the distances from  $v$  to each node  $u \in B(v)$ . Also assume that we have computed the distances from the set  $A$  to all nodes. We now describe how using this information we can get a 2-approximate estimate of *any* pair of nodes  $u, v$  in one case. In another case we will discuss what other estimates we need to precompute. Let  $\pi$  be the shortest path between  $u$  and  $v$ . We consider two cases depending on how the bunches  $B(u)$  and  $B(v)$  interact with  $\pi$ . A similar case analysis was used in [23, 35] in distributed approximate shortest paths algorithms.

1. There exists  $w \in \pi$  such that  $w \notin B(u) \cup B(v)$  (left case in Figure 1): Since  $w$  is on the shortest path, we either have  $d(w, u) \leq d(u, v)/2$  or  $d(v, w) \leq d(u, v)/2$ . Suppose  $d(w, u) \leq d(u, v)/2$ . We observe that by definition  $d(u, p(u)) \leq d(u, w)$ , hence we obtain  $d(u, p(u)) + d(p(u), v) \leq d(u, p(u)) + d(u, p(u)) + d(u, v) \leq 2d(u, v)$ . The case that  $d(v, w) \leq d(u, v)/2$  is analogous, and hence by computing  $\min\{d(u, p(u)) + d(p(u), v), d(v, p(v)) + d(p(v), u)\}$  we get a 2-approximation.
2. There exists no  $w \in \pi$  such that  $w \notin B(u) \cup B(v)$ . In other words,  $B(u) \cap B(v) \cap \pi = \emptyset$  and there exists at least one edge  $\{u', v'\}$  on  $\pi$  where  $u' \in B(u)$  and  $v' \in B(v)$  (right case in Figure 1<sup>3</sup>): in this case we can find the minimum over estimates obtained through all such pairs  $u', v'$ , i.e. by computing  $d(u, u') + w(u', v') + d(v', v)$ .

If we had access to all the above distances for the bunches and the pivots, we could then use them to query 2-approximate distances between any pair  $u, v \in V$ . As we will see, in our algorithms we only have *approximate bunches and pivots* which lead to  $(2 + \epsilon)$ -approximate queries.

<sup>3</sup> Although the picture implies  $B(u)$  and  $B(v)$  are disjoint, this also includes the case where they overlap, or even where  $u \in B(v)$  or  $v \in B(u)$ .



■ **Figure 1** Possible scenarios for the overlap between the bunches  $B(u)$ ,  $B(v)$  and the shortest path  $\pi$  from  $u$  to  $v$ .

### Running time

We can compute bunches and clusters in  $\tilde{O}(\frac{m}{p})$  time [64]. We need to compute shortest paths from  $A$  for the estimates in Case 1. This takes  $O(|A|m) = \tilde{O}(pnm)$  time using Dijkstra's algorithm. Finally, we consider Case 2. A straight-forward approach is to consider each pair of vertices  $u, v$ , and all  $u' \in B(u)$  and  $v' \in B(v)$ . This takes  $O(\frac{n^2}{p^2})$  time.

We use a sophisticated intermediate step that computes the distance between  $u'$  and  $v$  if there is an edge  $\{u', v'\} \in E$  such that  $v' \in B(v)$ . For each node  $v \in V$ , we consider all  $v' \in B(v)$ , for which we consider all neighbors  $u' \in N(v')$  and compute the estimate  $w(u', v') + d(v', v)$  (and replace current  $u' - v$  minimum if it is smaller). Since  $|B(v)| = \tilde{O}(\frac{1}{p})$ , we can compute these estimates for  $d(u', v)$  in  $\tilde{O}(\frac{n^2}{p})$  time.

Next, for each pair  $u, v \in V$ , we take the minimum over the distances  $d(u, u') + d(u', v)$  for all  $u'$  in the bunch  $B(u)$  using the precomputed distances from the previous step in time  $\tilde{O}(n^2/p)$ , by iterating over all pairs  $u, v \in V$ , and all  $u' \in B(u)$ . So in total the adjacent case takes time  $\tilde{O}(\frac{n^2}{p})$ . Hence the total running time of the algorithm will be  $\tilde{O}(pnm + m/p + n^2/p) = \tilde{O}(pnm + n^2/p)$ . By setting  $p = \sqrt{\frac{n}{m}}$  we obtain the total update time  $\tilde{O}(m^{1/2}n^{3/2})$  as stated in Theorem 1.

There are several subtleties that make it more difficult to maintain these estimates in decremental settings since the bunches and which bunches are adjacent keep on changing over the updates. We next discuss how these can be handled. We obtain a total update time that matches the stated static running time (up to subpolynomial factors).

## 2.1.2 Dynamic Challenges

### Maintaining bunches and pivots

First, we need to dynamically maintain bunches efficiently as nodes may join and leave a bunch throughout the updates using an adaptation of prior work. One option would be maintaining the clusters and bunches using the [59] framework, however using this directly is slow for our purposes. Hence we maintain *approximate* clusters and bunches using hopsets of [53]. This algorithm also maintains approximate pivots, i.e. pivots that are within  $(1 + \epsilon)$ -approximate distance of the true closest sampled node. These estimates let us handle the first case (up to  $(1 + \epsilon)$  approximation). One subtlety is that the type of approximate bunches used in [53] is slightly different with the type of approximate bunches we need for other parts of our algorithm. Specifically we need to bound the number of times nodes in a bunch can change. Roughly speaking, we can show this since the graph is decremental and the estimates obtained from the algorithm of [53] are monotone. Hence we perform *lazy bunch updates*, where we only let a bunch grow if the distance to the pivot grows by at least a factor  $1 + \epsilon$ . This means a bunch grows at most  $O(\log_{1+\epsilon}(nW))$  times. In addition, these approximate bunches need to be taken into account into our stretch analysis.



We note that  $(1 + \epsilon)$ -approximate decremental SSSP takes  $\tilde{O}(m^{1+o(1)})$  total update time. However, we use the multi-source shortest path result from [53], where the  $n^{o(1)}$ -term vanishes when applied with polynomially many sources, giving us a total update time of  $\tilde{O}(|A|m)$ .

## Maintaining estimates for Case 2

When we are in Case 2, we need more tools to keep track of estimates going through these nodes since both the bunches and the distances involved are changing. In particular, we explain how by using heaps in different parts of our algorithm we get efficient update and query times. Moreover, note that we have an additional complication: there are two data structures in this step, where one impacts the other. We need to ensure that an update in the original graph does not lead to many changes in the first data structure, which all need to be processed for the second data structure.

To be precise, we need intermediate data structures  $Q_{u',v}$  that store approximate distances for  $u'$  and  $v$  of the form  $w(u', v') + \delta(v', v)$  for each  $v' \in B(v)$  such that  $\{u', v'\} \in E$ , as explained for the static algorithm<sup>4</sup>. As opposed to the static algorithm, we do not only keep the minimum, but instead maintain a min-heap, from which the minimum is easily extracted.

However, this approach has a problem:  $\delta(v', v)$  might change many times, and for each such change we need to update at most  $|N(v')| \leq n$  heaps. To overcome this problem, we use another notion of *lazy distance update*: we only update a bunch if the estimate of a node changes by at least a factor  $1 + \epsilon$ . Combining this with the fact that, due to the lazy *bunch* update, nodes only join a cluster at most  $O(\log_{1+\epsilon}(nW))$  times, and thus there is only an  $O(\log_{1+\epsilon}(nW) \log(nW))$  overhead in maintaining these min-heaps  $Q_{u',v}$  dynamically due to bunch updates.

Similarly,  $w(u', v')$  might change many times, which has an impact for every  $v \in C(v')$ . Since this can be many nodes, we need to use an additional trick so an adversary cannot increase our update time to  $mn$ . Again the solution is a lazy update scheme: instead of  $w(u', v')$  we use  $\tilde{w}(u', v') = (1 + \epsilon)^{\lceil \log_{1+\epsilon} w(u', v') \rceil}$ , which can only change  $\log_{1+\epsilon} W$  times. This comes at the cost of a  $(1 + \epsilon)$  approximation factor.

We note that our bunches are approximate bunches in three different ways. We have one notion of approximation due to the fact that we are using hopsets (that implicitly maintain bunches on scaled graphs). We have a second notion of approximation due to our lazy bunch update, which only lets nodes join a bunch a bounded number of times. We have a third notion of approximation due to the lazy distance update, which only propagates distance changes  $O(\log(nW))$  times. We need to carefully consider how these different notions of approximate bunch interact with each other and with the stretch of the algorithm.

Next, we want to use the min-heaps  $Q_{u',v}$  to maintain the distance estimates for  $u, v \in V$  (see again the right case in Figure 1). We construct min-heaps  $Q'_{u,v}$ , with for each pair  $u, v$  entries  $\delta(u, u') + \delta(u', v)$  for  $u' \in B(u)$ . Here  $\delta(u', v)$  is the minimum from the intermediate data structure  $Q_{u',v}$ .

An entry  $\delta(u, u') + \delta(u', v)$  in  $Q'_{u,v}$  has to be updated when either  $\delta(u, u')$  or  $\delta(u', v)$  change. We have to make sure we do not have to update these entries too often. For the first part of each entry, we bound updates to the distance estimates  $\delta(u, u')$  by the lazy distance updates. One challenge here is that when the distance estimates in a bunch change we need to update all the impacted heaps i.e. if  $\delta(u, u')$  inside a bunch  $B(u)$  changes, we must update all the heaps  $Q'_{u,v}$  such that there exists  $\{u', v'\} \in E$  with  $v' \in B(v)$ . For this purpose one

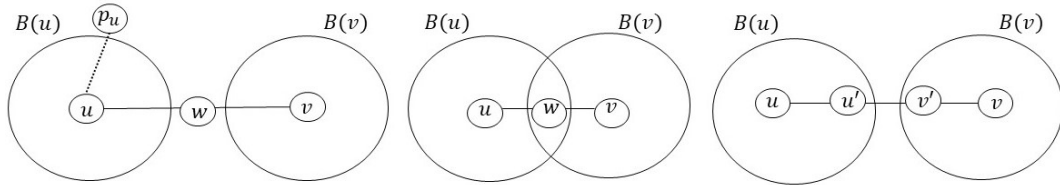
<sup>4</sup> In this section,  $\delta(\cdot, \cdot)$  denotes  $(1 + \epsilon)$ -approximate distances in our dynamic data structures.

approach is the following: for each  $u \in V, u' \in B(u)$  we keep an additional data structure  $\text{Set}_{u,u'}$  that stores for which  $v \in V$  we have  $v' \in B(v)$  with  $\{u', v'\} \in E$ . This means that given a change in  $\delta(u, u')$ , we can update each of the heaps in logarithmic time.

For the second part of the entry, which is the distance estimate  $\delta(u', v)$  from the first data structure, this is more complicated. Another subtlety for our decremental data structures is the fact that  $\delta(u', v)$  may also decrease due to a new node being added to  $B(v)$ . However, our lazy bunch update ensures that  $\delta(u', v)$  can only decrease  $O(\log_{1+\epsilon}(nW))$  times. Further, we use lazy distance increases *in between* such decreases. Hence in total we propagate changes to  $\delta(u', v)$  at most  $O(\log_{1+\epsilon}(nW) \log(nW))$  times, and thus our decremental algorithm has  $\tilde{O}(\log_{1+\epsilon}(nW) \log(nW))$  overhead compared to the static algorithm.

### 2.2 $(2 + \epsilon, W_{u,v})$ -APSP for Weighted Graphs

For a moment, let us go back to the first static algorithm. It turns out, that if instead of computing an estimate for two “adjacent” bunches, we keep an estimate for bunches that overlap in at least one node, we obtain a  $(2, W_{u,v})$ -approximation, where  $W_{u,v}$  is the maximum weight on the shortest path from  $u$  to  $v$ .



■ **Figure 2** Possible scenarios for the overlap between the bunches  $B(u)$  and  $B(v)$  and the shortest path  $\pi$  from  $u$  to  $v$ .

The stretch analysis comes down to the following cases:

- There exists no  $w \in \pi \cap B(u) \cap B(v)$  (the left and right case in Figure 2).
- There exists  $w \in \pi$  such that  $w \in B(u) \cap B(v)$  (the middle case in Figure 2).

The crucial observation here is that the distance estimate via the pivot (Case 1 in the previous section) already gives a  $(2, W_{u,v})$ -approximation for the adjacent case (the right case in Figure 2). So if we maintain an estimate for  $d(u, v)$  if there exists  $w \in \pi$  such that  $w \in B(u) \cap B(v)$  (the “overlap case”), then we obtain a  $(2, W_{u,v})$ -approximation in total.

Here we need to create a min-heap for the overlap case, later we show we can maintain in  $\tilde{O}(nC_{\max}/p)$  time. First we focus on the challenge of bounding  $C_{\max}$ .

#### Efficiency challenge

While the bunches are small, there is no bound on the size of the clusters. To overcome this issue in the static case, Thorup and Zwick developed an alternative way to build the bunches and clusters, that guarantees that the clusters are also small [63]. At a high-level, their approach is to adaptively change (grow)  $A$  by sampling big clusters into smaller clusters using a sampling rate proportional to the cluster size and adding these nodes to the set of pivots  $A$ . In total, in  $\tilde{O}(m/p)$  time we obtain bunches and clusters, *both* of size  $\tilde{O}(1/p)$ . This idea was developed in the context of obtaining compact routing schemes, and later found numerous applications in static algorithms for approximating shortest paths and distances (see e.g., [14, 8, 10, 9]).

Using this trick, the static algorithm has running time  $\tilde{O}(\frac{m}{p} + pm + \frac{n}{p^2}) = \tilde{O}(nm^{3/4} + m^{4/3})$ , for  $p = m^{-1/3}$  or  $\tilde{O}(mn^{1/2} + n^2)$  for  $p = n^{-1/2}$ .

However, the adaptive sampling procedure does not seem to be well-suited to a dynamic setting. The reason is that the sampling procedure is adaptive in such a way that when the bunches/clusters change, the set of sampled nodes can also change. Although (a dynamic adaptation of) the algorithm can guarantee that the set of sampled nodes is small at any particular moment, there is no guarantee on the *monotonicity* of the set  $A$  of sources that we need to maintain distances from. This would add a significant amount of computation necessary to propagate these changes to other parts of the algorithm. A possible solution is to enforce monotonicity by never letting nodes leave  $A$ . This would require us to bound the *total* number of nodes that would ever be sampled, which seems impossible with the current approach. To overcome this, we next suggest a new approach that gives us the monotonicity needed for our dynamic algorithm. For simplicity, we first present this in a static setting.

### Introducing bunch overlap thresholds

Our proposed approach is to divide the nodes into two types depending on the number of bunches they appear in. In particular, we consider a threshold parameter  $\tau$  and we define a node to be a *heavy node* if it is in more than  $\tau$  bunches (equivalently its cluster contains more than  $\tau$  nodes), and otherwise we call it a *light node*. In other words, for the set of light nodes we have  $C_{\max} \leq \tau$ . This threshold introduces a second type of pivot for each node  $u$ , which we denote by  $q(u)$  that is defined to be the closest heavy node to  $u$ . In particular, instead of computing all distance estimates going through nodes  $w \in B(u) \cap B(v)$ , we only compute them for the case that  $w$  is light, and otherwise it is enough to compute the minimum estimates going through the heavy pivots, i.e.  $\min\{d(u, q(u)) + d(q(u), v), d(v, q(v)) + d(q(v), u)\}$ . We emphasize that these heavy pivots are *not a subset* of our original pivots  $p(v), v \in V$ , and they have a different behavior in bounding the running time than the *bunch pivots*. As we will see later, introducing these thresholds is crucial in efficiently obtaining the estimates required for Case 2.2 as we can handle the light and heavy case separately.

### Stretch of the algorithm with bunch overlap thresholds

As discussed above, the stretch analysis was divided into two cases. In the new variant of the algorithm we add a third case, where we look at the distance estimates going through heavy pivots, i.e.  $\min\{d(u, q(u)) + d(q(u), v), d(v, q(v)) + d(q(v), u)\}$ . The main idea is that now we only need to consider the difficult case, Case 2.2, if the relevant node is light, which allows us to implement the algorithm efficiently.

If the relevant node  $w \in B(u) \cap B(v)$  in Case 2 is heavy, we obtain a 2-approximation through the heavy pivot: we get that the  $\min\{d(u, w), d(v, w)\} \leq d(u, v)/2$ , since  $d(u, v) = d(u, w) + d(w, v)$ . W.l.o.g. say  $d(u, w) \leq d(u, v)/2$ . Then  $d(u, q(u)) \leq d(u, w) \leq d(u, v)/2$ , hence  $d(u, q(u)) + d(q(u), v) \leq d(u, q(u)) + d(q(u), u) + d(u, v) \leq 2d(u, v)$ .

### Maintaining estimates through heavy nodes

For the stretch analysis above, we need to maintain a shortest path tree from each heavy node. We ensure a bunch grows at most  $O(\log_{1+\epsilon}(nW))$  times. Hence by a total load argument we get that the *total* number of nodes in all bunches over all updates is at most  $O(\frac{n}{p} \log_{1+\epsilon}(nW))$ . This implies there cannot be too many heavy nodes in total, hence we can enforce monotonicity by keeping in  $V_{\text{heavy}}$  all nodes that were once heavy. As a consequence, we can maintain multi-source approximate distances from all heavy nodes efficiently: we can compute shortest paths from this set to all other nodes in  $O(|V_{\text{heavy}}|m) = O(\frac{nm}{p\tau})$  time.

### Running time of the algorithm with bunch overlap thresholds

Summarizing, we compute bunches and clusters in  $O(m/p)$  time, compute distances from  $A$  to  $V$  in  $O(|A|m) = \tilde{O}(pnm)$  time, and we can compute shortest paths from the heavy nodes in  $O(|V_{\text{heavy}}|m) = O(\frac{nm}{p\tau})$  time.

Finally, we consider Case 2.2. For each node  $u \in V$ , we consider every light node  $w \in B(u)$ , and then all nodes  $v \in C(w)$ . Since  $w$  is light, we have  $|C(w)| \leq \tau$ , so this takes time  $\tilde{O}(n \cdot \frac{1}{p} \cdot \tau)$ .

Hence, in the total update time, we obtain  $\tilde{O}((pnm + \frac{nm}{p\tau} + n\tau/p) \log^2(nW))$ . Setting  $p = m^{-1/4}$  and  $\tau = m^{1/2}$  gives a total update time of  $\tilde{O}(nm^{3/4} \log(nW))$ .

Again for a decremental algorithm, we need to combine this new idea with the subtleties we had in the previous section: the bunches, their overlaps, and the heavy nodes keep on changing over the updates. We obtain a total update time that matches the stated static running time (up to subpolynomial factors) for the algorithm using bunch overlap thresholds.

### 2.2.1 $(2 + \epsilon)$ -APSP for Unweighted Graphs

We observe that in unweighted graphs mixed approximation can always be turned into a multiplicative approximation – at the cost of a blow-up in the number of vertices. More precisely, we prove the following reduction.

► **Theorem 5.** *Let  $\mathcal{A}$  be an algorithm that provides a  $(a + \epsilon, k)$ -approximation for APSP in  $\tau_{\mathcal{A}}(n', m')$  time, for any unweighted  $n'$ -node  $m'$ -edge graph  $G'$ , and any constants  $a, k \in \mathbb{N}_{\geq 1}$  and  $\epsilon \in [0, 1)$ . Given an unweighted graph  $G = (V, E)$  on  $n$  nodes and  $m$  vertices, we can compute  $(a + (k + 2)\epsilon, 0)$ -APSP in  $\tau_{\mathcal{A}}(n + km, (k + 1)m)$  time.*

*If  $\mathcal{A}$  is a dynamic algorithm, then each update takes  $(k + 1) \cdot [\text{update time of } \mathcal{A}]$  time. The query time remains the same (up to a constant factor).*

The result is obtained by splitting every edge into  $k$  edges, by introducing new nodes on the edge. Now a path corresponding to a  $+k$  approximation cannot take a non-trivial detour.

We combine this result with Theorem 2 to obtain a  $(2 + \epsilon)$ -approximation for unweighted graphs in  $\tilde{O}(m^{7/4})$  total update time.

### 2.3 $(1 + \epsilon, 2(k - 1))$ -APSP for Unweighted Graphs

In this section, we describe our near-additive APSP algorithm. Our work is inspired by a classic result of Dor, Halperin, and Zwick [32], that presented a static algorithm that computes purely additive  $+2(k - 1)$  approximation for APSP in  $\tilde{O}(n^{2-1/k}m^{1/k})$  time in unweighted graphs. Our goal is to obtain similar results dynamically. More concretely, we obtain decremental  $(1 + \epsilon, 2(k - 1))$ -approximate APSP in  $O(n^{2-1/k+o(1)}m^{1/k})$  total update time in unweighted graphs.

#### 2.3.1 $(1 + \epsilon, 2)$ -APSP

To explain the high-level idea of the algorithm, we first focus on the special case that  $k = 2$ , and that we only want to approximate the distances between pairs of nodes at distance at most  $d$  from each other, for a parameter  $d$ . In this case, we obtain a  $+2$ -additive approximation in  $\tilde{O}(n^{3/2}m^{1/2}d)$  time. This case already allows to present many of the high-level ideas of the algorithm. Later we explain how to extend the results to the more general case. We start by describing the static algorithm from [32], and then explain the dynamic version. The static version of the data structure is as follows:

- Let  $s_1 = (\frac{m}{n})^{1/2}$ . Let  $V_1$  be the set of *dense nodes*:  $V_1 := \{v \in V : \deg(v) \geq s_1\}$ . Also let  $E_2$  be the set of *sparse edges*, i.e. edges with at least one endpoint with degree less than  $s_1$ .
- **Node set  $D_1$ .** Construct a hitting set  $D_1$  of nodes in  $V_1$ . This means that every node  $v \in V_1$  has a neighbor in  $D_1$ . The size of  $D_1$  is  $O(n \log n / s_1)$ .
- **Edge set  $E^*$ .** Let  $E^*$  be a set of size  $O(n)$  such that for each  $v \in V_1$ , there exists  $u \in D_1 \cap N(v)$  such that  $\{u, v\} \in E^*$ .
- **Computing distances from  $D_1$ .** Store distances  $D_1 \times V$  by running a BFS from each  $u \in D_1$  on the input graph  $G$ .
- **Computing distances from  $V \setminus D_1$ .** For each  $u \in V \setminus D_1$  store a shortest path tree, denoted by  $T_u$  rooted at  $u$  by running Dijkstra on  $(V, E_2 \cup E^* \cup E_u^{D_1})$ , where  $E_u^{D_1}$  is the set of weighted edges corresponding to distances in  $(\{u\} \times D_1)$  computed in the previous step.

### Dynamic data structure

The static algorithm computes distances in two steps. First, it computes distances from  $D_1$  by computing BFS trees in the graph  $G$ . This step can be maintained dynamically by using *Even-Shiloach trees (ES-trees)* [40], a data structure that maintains distances in a decremental graph. Maintaining the distances up to distance  $O(d)$  takes  $O(|D_1|md)$  time. The more challenging part is computing distances from  $V \setminus D_1$ . Here the static algorithm computes distances in the graphs  $H_u = (V, E_2 \cup E^* \cup E_u^{D_1})$ . Note that the graphs  $H_u$  change dynamically in several ways. First, in the static algorithm, the set  $E_2$  is the set of light edges. To keep the correctness of the dynamic algorithm, every time that a node  $v$  no longer has a neighbor in  $D_1$ , we should add all its adjacent edges to  $E_2$ . Second, when an edge in  $E^*$  is deleted from  $G$ , we should replace it by an alternative edge if such an edge exists. Finally, the weights of the edges in  $E_u^{D_1}$  can change over time, when the estimates change. This means that other than deletions of edges, we can also add new edges to the graphs  $H_u$  or change the weights of their edges. Because of these edge insertions we can no longer use the standard ES-tree data structure to maintain the distances, because that only works in a decremental setting.

To overcome this, we use *monotone ES-trees*, a generalization of ES-trees proposed by [46, 45]. In this data structure, to keep the algorithm efficient, when new edges are inserted, the distance estimates do not change. In particular, when edges are inserted, some distances may decrease, however, in such cases the data structure keeps an old larger estimate of the distance. The main challenge in using this data structure is to show that the stretch analysis still holds. In particular, in our case, the stretch analysis of the static algorithm was based on the fact the distances from the nodes  $V \setminus D_1$  are computed in the graphs  $H_u$ , where in the dynamic setting, we do not have this guarantee anymore.

### Stretch analysis

The high-level idea of the stretch analysis uses the special structure of the graphs  $H_u$ . To prove that the distance estimate between  $u$  and  $v$  is at most  $d(u, v) + 2$ , we distinguish between two cases. First, we maintain the property that as long as  $v$  has a neighbor  $x \in D_1$ , then an edge of the form  $\{v, x\}$  is in  $E^*$  for  $x \in D_1 \cap N(v)$ , and since we maintain correctly the distances from  $D_1$ , we can prove that we get an additive stretch of at most 2 in this case.

In the second case, all the edges adjacent to  $v$  are in  $E_2 \subseteq H_u$ , and here we can use an inductive argument on the length of the path to prove that we get an additive stretch of at most 2. Note that the estimate that we get can be larger compared to the distance between  $u$  and  $v$  in the graph  $H_u$ , but we can still show that the additive approximation is at most 2 as needed.

### Update time

Our update time depends on the size of the graphs  $H_u$ . While in the static setting it is easy to bound the number of edges in  $E_2$  and  $E^*$ , in the decremental setting, as new edges are added to these sets, we need a more careful analysis. For example, in the static setting  $|E^*| = O(n)$ , as any node only adds one adjacent edge to the set  $E^*$ , where in the decremental setting, we may need to add many different edges to this set because the previous ones got deleted. However, we can still prove that even if a node needs to add to  $E^*$  edges to all its adjacent neighbors in  $D_1$  during the algorithm, the size of  $H_u$  is small enough as needed. In the full version we show that the total update time is  $\tilde{O}(m^{1/2}n^{3/2}d)$  in expectation, this matches the static complexity up to the factor  $d$ .

### Handling large distances

The algorithm described above is efficient if  $d$  is small, to obtain an efficient algorithm for the general case, we combine our approach with a decremental near-additive APSP algorithm. More concretely, we use an algorithm from [45] that allows to compute  $(1+\epsilon, \beta)$ -approximation for APSP in  $O(n^{2+o(1)})$  total update time, for  $\beta = n^{o(1)}$ . Now we set  $d = \Theta(\beta/\epsilon)$ , and distinguish between two cases. For pairs of nodes at distance at most  $d$  from each other, we get a +2-additive approximation as discussed above, in  $O(m^{1/2}n^{3/2+o(1)})$  time. For pairs of nodes at distance larger than  $d$ , the near-additive approximation is already a  $(1 + O(\epsilon))$ -approximation by the choice of the parameter  $d$ . For such pairs, the additive  $\beta$  term becomes negligible, as their distance is  $\Omega(\beta/\epsilon)$ . Overall, we get a  $(1 + \epsilon, 2)$ -approximation for APSP in  $O(m^{1/2}n^{3/2+o(1)})$  time. In fact, if the graph is dense enough ( $m = n^{1+\Omega(1)}$ ), the  $n^{o(1)}$  term is replaced by a poly-logarithmic term.

### 2.3.2 General $k$

We can extend the algorithm to obtain a  $(1 + \epsilon, 2(k - 1))$ -approximate APSP in  $O(n^{2-1/k+o(1)}m^{1/k})$  total update time, by adapting the general algorithm of [32] to the dynamic setting. At a high-level, instead of having one hitting set  $D_1$ , we have a series of hitting sets  $D_1, D_2, \dots, D_k$ , such that the set  $D_i$  hits nodes of degree  $s_i$ . We compute distances from  $D_i$  in appropriate graphs  $H_u^i$ , that are sparser when the set  $D_i$  is larger. Balancing the parameters of the algorithm leads to the desired total update time.

### 2.3.3 Limitations of previous approaches

We next explain why approaches used in previous decremental algorithms cannot be generalized to obtain our near-additive results. First, the unweighted  $(1 + \epsilon, 2)$ -approximate APSP algorithm that takes  $\tilde{O}(n^{5/2}/\epsilon)$  time [46], is inspired by static algorithms for +2-additive emulators, sparse graphs that preserve the distances up to a +2-additive stretch. The main idea is to maintain a sparse emulator of size  $\tilde{O}(n^{3/2})$ , and exploit its sparsity to obtain a fast algorithm. This construction however is specific for the +2 case, and cannot be generalized to a general  $k$ . Note that the update time of the algorithm crucially depends on the size of the



emulator, and to obtain a better algorithm for a general  $k$ , we need to be able to construct a sparser emulator for a general  $k$ . However, a beautiful lower bound result by Abboud and Bodwin [1] shows that for any constant  $k$ , a purely additive  $+k$  emulator should have  $\Omega(n^{4/3-\epsilon})$  edges. This implies that dynamic algorithms that are based on purely additive emulators seem to require  $\Omega(n^{7/3-\epsilon})$  time, and we cannot get running time arbitrarily close to  $O(n^2)$ . An alternative approach is to build a *near-additive* emulator. This is done in [45], where the authors show a  $(1 + \epsilon, 2(1 + 2/\epsilon)^{k-2})$ -approximation algorithm for decremental APSP with expected total update time of  $\tilde{O}(n^{2+1/k}(37/\epsilon)^{k-1})$ . Here the running time indeed gets closer to  $O(n^2)$ , but this comes at a price of a much worse additive term of  $2(1 + 2/\epsilon)^{k-2}$ . While in the static setting there are also other bounds obtained for near-additive emulators, such as  $O(1 + \epsilon, O(k/\epsilon)^{k-1})$  emulators of size  $O(kn^{1+\frac{1}{2k+1-1}})$  [38, 65, 2, 37], in all of the constructions the additive term depends on  $\epsilon$ , and the dependence on  $\epsilon$  is known to be nearly tight for small  $k$  by lower bounds from [2]. Hence this approach cannot lead to small constant additive terms that do not depend on  $\epsilon$ .

---

## References

- 1 Amir Abboud and Greg Bodwin. The 4/3 additive spanner exponent is tight. *Journal of the ACM (JACM)*, 64(4):1–20, 2017.
- 2 Amir Abboud, Greg Bodwin, and Seth Pettie. A hierarchy of lower bounds for sublinear additive spanners. *SIAM Journal on Computing*, 47(6):2203–2236, 2018.
- 3 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. In *Proc. of the 55th Annual ACM Symposium on Theory of Computing (STOC 2023)*, 2023. doi:10.48550/arXiv.2211.07058.
- 4 Amir Abboud, Karl Bringmann, Seri Khoury, and Or Zamir. Hardness of approximation in P via short cycle removal: cycle detection, distance oracles, and beyond. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2022)*, pages 1487–1500, 2022. doi:10.1145/3519935.3520066.
- 5 Ittai Abraham and Shiri Chechik. Dynamic decremental approximate distance oracles with  $(1 + \epsilon, 2)$  stretch. *CoRR*, abs/1307.1516, 2013. arXiv:1307.1516.
- 6 Ittai Abraham, Shiri Chechik, and Kunal Talwar. Fully dynamic all-pairs shortest paths: Breaking the  $o(n)$  barrier. In *Proceedings of the 17th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX 2014) and the 18th International Workshop on Randomization and Computation (APPROX/RANDOM 2014)*, pages 1–16, 2014. doi:10.4230/LIPIcs.APPROX-RANDOM.2014.1.
- 7 Donald Aingworth, Chandra Chekuri, Piotr Indyk, and Rajeev Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999. Announced at SODA 1996. doi:10.1137/S0097539796303421.
- 8 Maor Akav and Liam Roditty. An almost 2-approximation for all-pairs of shortest paths in subquadratic time. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2020)*, pages 1–11, 2020. doi:10.1137/1.9781611975994.1.
- 9 Maor Akav and Liam Roditty. A unified approach for all pairs approximate shortest paths in weighted undirected graphs. In *Proceedings of the 29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204, pages 4:1–4:18, 2021. doi:10.4230/LIPIcs.ESA.2021.4.
- 10 Arturs Backurs, Liam Roditty, Gilad Segal, Virginia Vassilevska Williams, and Nicole Wein. Toward tight approximation bounds for graph diameter and eccentricities. *SIAM Journal on computing (SICOMP)*, 50(4):1155–1199, 2021.
- 11 Surender Baswana, Vishrut Goyal, and Sandeep Sen. All-pairs nearly 2-approximate shortest paths in  $o(n^2 \text{ polylog } n)$  time. *Theoretical Computer Science*, 410(1):84–93, 2009. Announced at STACS 2005. doi:10.1016/j.tcs.2008.10.018.

- 12 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 394–403, 2003.
- 13 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *Journal of Algorithms*, 62(2):74–92, 2007. Announced at STOC 2002. doi:10.1016/j.jalgor.2004.08.004.
- 14 Surender Baswana and Telikepalli Kavitha. Faster algorithms for all-pairs approximate shortest paths in undirected graphs. *SIAM Journal on Computing*, 39(7):2865–2896, 2010. Announced at FOCS 2006. doi:10.1137/080737174.
- 15 Surender Baswana, Sumeet Khurana, and Soumojit Sarkar. Fully dynamic randomized algorithms for graph spanners. *ACM Transactions on Algorithms*, 8(4):35:1–35:51, 2012. doi:10.1145/2344422.2344425.
- 16 Piotr Berman and Shiva Prasad Kasiviswanathan. Faster approximation of distances in graphs. In Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Norbert Zeh, editors, *Proceedings of the 10th International Workshop on Algorithms and Data Structures (WADS 2007)*, pages 541–552, 2007. doi:10.1007/978-3-540-73951-7\_47.
- 17 Aaron Bernstein. Fully dynamic  $(2 + \epsilon)$  approximate all-pairs shortest paths with fast query and close to linear update time. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009*, pages 693–702. IEEE Computer Society, 2009. doi:10.1109/FOCS.2009.16.
- 18 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM Journal on Computing*, 45(2):548–574, 2016. Announced at STOC 2013. doi:10.1137/130938670.
- 19 Aaron Bernstein. Deterministic partially dynamic single source shortest paths in weighted graphs. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, (ICALP 2017)*, pages 44:1–44:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.44.
- 20 Aaron Bernstein and Shiri Chechik. Deterministic decremental single source shortest paths: beyond the  $O(mn)$  bound. In Daniel Wachs and Yishay Mansour, editors, *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing (STOC 2016)*, pages 389–397, 2016. doi:10.1145/2897518.2897521.
- 21 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *Proceedings of the 62nd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2021)*, pages 1000–1008, 2021. doi:10.1109/FOCS52979.2021.00100.
- 22 Aaron Bernstein and Liam Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011*, pages 1355–1365. SIAM, 2011. doi:10.1137/1.9781611973082.104.
- 23 Keren Censor-Hillel, Michal Dory, Janne H Korhonen, and Dean Leitersdorf. Fast approximate shortest paths in the congested clique. *Distributed Computing*, 34(6):463–487, 2021.
- 24 Shiri Chechik. Near-optimal approximate decremental all pairs shortest paths. In Mikkel Thorup, editor, *Proceedings of the 59th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2018)*, pages 170–181, 2018. doi:10.1109/FOCS.2018.00025.
- 25 Shiri Chechik and Tianyi Zhang. Nearly 2-approximate distance oracles in subquadratic time. In *Proc. of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 551–580. SIAM, 2022. doi:10.1137/1.9781611977073.26.
- 26 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In Samir Khuller and Virginia Vassilevska Williams, editors, *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing (STOC 2021)*, pages 626–639. ACM, 2021. doi:10.1145/3406325.3451025.
- 27 Julia Chuzhoy and Thatchaphol Saranurak. Deterministic algorithms for decremental shortest paths via layered core decomposition. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms (SODA 2021)*, pages 2478–2496, 2021. doi:10.1137/1.9781611976465.147.

- 28 Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38(2):335–353, 2001. Announced at SODA 1997. doi:10.1006/jagm.2000.1117.
- 29 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004. Announced at STOC 2003. doi:10.1145/1039488.1039492.
- 30 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *Journal of Computer and System Sciences*, 72(5):813–837, 2006. Announced at FOCS 2001. doi:10.1016/j.jcss.2005.05.005.
- 31 Mingyang Deng, Yael Kirkpatrick, Victor Rong, Virginia Vassilevska Williams, and Ziqian Zhong. New additive approximations for shortest paths and cycles. In Mikolaj Bojanczyk, Emanuela Merelli, and David P. Woodruff, editors, *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022, July 4-8, 2022, Paris, France*, volume 229 of *LIPICs*, pages 50:1–50:10. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICs.ICALP.2022.50.
- 32 Dorit Dor, Shay Halperin, and Uri Zwick. All-pairs almost shortest paths. *SIAM Journal on Computing*, 29(5):1740–1759, 2000. Announced at FOCS 1996. doi:10.1137/S0097539797327908.
- 33 Michal Dory, Sebastian Forster, Yael Kirkpatrick, Yasamin Nazari, Virginia Vassilevska Williams, and Tijn de Vos. Fast 2-approximate all-pairs shortest paths. In *Proc. of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4728–4757. SIAM, 2024. arXiv:2307.09258.
- 34 Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. New tradeoffs for decremental approximate all-pairs shortest paths. *CoRR*, abs/2211.01152, 2022. doi:10.48550/arXiv.2211.01152.
- 35 Michal Dory and Merav Parter. Exponentially faster shortest paths in the congested clique. *ACM Journal of the ACM (JACM)*, 69(4):1–42, 2022.
- 36 Anita Dürr. Improved bounds for rectangular monotone min-plus product and applications. *Information Processing Letters*, page 106358, 2023.
- 37 Michael Elkin and Ofer Neiman. Near-additive spanners and near-exact hopsets, a unified view. *Bulletin of EATCS*, 1(130), 2020.
- 38 Michael Elkin and David Peleg.  $(1 + \epsilon, \beta)$ -spanner constructions for general graphs. *SIAM Journal on Computing*, 33(3):608–631, 2004.
- 39 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in unweighted digraphs versus an adaptive adversary. In *Proc. of the 48th International Colloquium on Automata, Languages, and Programming, (ICALP 2021)*, pages 64:1–64:20, 2021. doi:10.4230/LIPICs.ICALP.2021.64.
- 40 Shimon Even and Yossi Shiloach. An on-line edge-deletion problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.
- 41 Lisa Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal of Discrete Mathematics*, 13(4):505–520, 2000. Announced at FOCS 1999. doi:10.1137/S0895480199355754.
- 42 Sebastian Forster, Gramoz Goranci, and Monika Henzinger. Dynamic maintenance of low-stretch probabilistic tree embeddings with applications. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, (SODA 2021)*, pages 1226–1245, 2021. doi:10.1137/1.9781611976465.75.
- 43 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles. In *Proceedings of the 31th Annual European Symposium on Algorithms (ESA 2023)*, 2023.
- 44 Naveen Garg and Jochen Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007. Announced at FOCS 1998. doi:10.1137/S0097539704446232.

- 45 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A subquadratic-time algorithm for decremental single-source shortest paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014*, pages 1053–1072. SIAM, 2014. doi:10.1137/1.9781611973402.79.
- 46 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the  $o(mn)$  barrier and derandomization. *SIAM Journal on Computing*, 45(3):947–1006, 2016. Announced at FOCS 2013.
- 47 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Decremental single-source shortest paths on undirected graphs in near-linear total update time. *Journal of the ACM*, 65(6):36:1–36:40, 2018. Announced at FOCS 2014. doi:10.1145/3218657.
- 48 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing (STOC 2015)*, pages 21–30, 2015. doi:10.1145/2746539.2746609.
- 49 Adam Karczmarz and Jakub Łącki. Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs. In *Proceedings of the 27th Annual European Symposium on Algorithms (ESA 2019)*, pages 65:1–65:15, 2019. doi:10.4230/LIPIcs.ESA.2019.65.
- 50 Adam Karczmarz and Jakub Łącki. Simple label-correcting algorithms for partially dynamic approximate shortest paths in directed graphs. In *Proceedings of the 3rd Symposium on Simplicity in Algorithms (SOSA 2020)*, pages 106–120, 2020. doi:10.1137/1.9781611976014.15.
- 51 Telikepalli Kavitha. Faster algorithms for all-pairs small stretch distances in weighted graphs. *Algorithmica*, 63(1-2):224–245, 2012. Announced at FSTTCS 2007. doi:10.1007/s00453-011-9529-y.
- 52 Mathias Bæk Tejs Knudsen. Additive spanners and distance oracles in quadratic time. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming, (ICALP 2017)*, pages 64:1–64:12, 2017. doi:10.4230/LIPIcs.ICALP.2017.64.
- 53 Jakub Łącki and Yasamin Nazari. Near-Optimal Decremental Hopsets with Applications. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.
- 54 Aleksander Mądry. Faster approximation schemes for fractional multicommodity flow problems via dynamic graph algorithms. In *Proceedings of the 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pages 121–130, 2010. doi:10.1145/1806689.1806708.
- 55 Mihai Patrascu and Liam Roditty. Distance oracles beyond the thorup-zwick bound. *SIAM Journal on Computing*, 43(1):300–311, 2014. Announced at FOCS 2010. doi:10.1137/11084128X.
- 56 Mihai Patrascu, Liam Roditty, and Mikkel Thorup. A new infinity of distance oracles for sparse graphs. In *Proc. of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012)*, pages 738–747. IEEE Computer Society, 2012. doi:10.1109/FOCS.2012.44.
- 57 Liam Roditty. New algorithms for all pairs approximate shortest paths. In Barna Saha and Rocco A. Servedio, editors, *Proceedings of the 55th Annual ACM Symposium on Theory of Computing, STOC 2023, Orlando, FL, USA, June 20-23, 2023*, pages 309–320. ACM, 2023. doi:10.1145/3564246.3585197.
- 58 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. Announced at ESA 2004. doi:10.1007/s00453-010-9401-5.
- 59 Liam Roditty and Uri Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. Announced at FOCS 2004. doi:10.1137/090776573.
- 60 Barna Saha and Christopher Ye. Faster approximate all pairs shortest paths. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4758–4827. SIAM, 2024. arXiv:2309.13225.

- 61 Christian Sommer. All-pairs approximate shortest paths and distance oracle preprocessing. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming, (ICALP 2016)*, volume 55, pages 55:1–55:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.55.
- 62 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 2004)*, pages 384–396, 2004. doi:10.1007/978-3-540-27810-8\_33.
- 63 Mikkel Thorup and Uri Zwick. Compact routing schemes. In Arnold L. Rosenberg, editor, *Proceedings of the Thirteenth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA 2001*, pages 1–10. ACM, 2001. doi:10.1145/378580.378581.
- 64 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.
- 65 Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 802–809, 2006.
- 66 R. Ryan Williams. Faster all-pairs shortest paths via circuit complexity. *SIAM Journal on Computing*, 47(5):1965–1985, 2018. Announced at STOC 2014. doi:10.1137/15M1024524.
- 67 Uri Zwick. Exact and approximate distances in graphs – A survey. In *Proceedings of the 9th Annual European Symposium on Algorithms (ESA 2001)*, volume 2161, pages 33–48, 2001. doi:10.1007/3-540-44676-1\_3.