Bootstrapping Dynamic Distance Oracles

Sebastian Forster
Department of Computer Science, University of Salzburg, Austria

Gramoz Goranci
Faculty of Computer Science, University of Vienna, Austria

Yasamin Nazari[1]
Department of Computer Science, VU Amsterdam, The Netherlands

Antonis Skarlatos
Department of Computer Science, University of Salzburg, Austria

Abstract

Designing approximate all-pairs distance oracles in the fully dynamic setting is one of the central problems in dynamic graph algorithms. Despite extensive research on this topic, the first result breaking the $O(\sqrt{n})$ barrier on the update time for any non-trivial approximation was introduced only recently by Forster, Goranci and Henzinger [SODA'21] who achieved $m^{1/\rho+o(1)}$ amortized update time with a $O(\log n)^{3\rho-2}$ factor in the approximation ratio, for any parameter $\rho \geq 1$.

In this paper, we give the first *constant-stretch* fully dynamic distance oracle with small polynomial update and query time. Prior work required either at least a poly-logarithmic approximation or much larger update time. Our result gives a more fine-grained trade-off between stretch and update time, for instance we can achieve constant stretch of $O(\frac{1}{\rho^2})^{4/\rho}$ in amortized update time $\tilde{O}(n^\rho)$, and query time $\tilde{O}(n^{\rho/8})$ for any constant parameter $0 < \rho < 1$. Our algorithm is randomized and assumes an oblivious adversary.

A core technical idea underlying our construction is to design a black-box reduction from decremental approximate hub-labeling schemes to fully dynamic distance oracles, which may be of independent interest. We then apply this reduction repeatedly to an existing decremental algorithm to bootstrap our fully dynamic solution.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases Dynamic graph algorithms, Distance Oracles, Shortest Paths

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.50

Related Version *Full Version*: `https://arxiv.org/abs/2303.06102`

Funding 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).

1 Introduction

The All-Pairs Shortest Paths (APSP) problem is one of the cornerstone graph problems in combinatorial optimization. It has a wide range of applications, for instance in route planning, navigation systems, and routing in networks, and it has been extensively studied from both practical and theoretical perspectives. In theoretical computer science, this problem enjoys much popularity due to its historic contributions to the development of fundamental algorithmic tools and definitions as well as being used as a subroutine for solving other problems.

[1] This work was conducted when this author was a postdoc at University of Salzburg.

31st Annual European Symposium on Algorithms (ESA 2023).
Editors: Inge Li Gørtz, Martin Farach-Colton, Simon J. Puglisi, and Grzegorz Herman; Article No. 50; pp. 50:1–50:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The APSP problem has also been studied extensively in *dynamic* settings. Here, the underlying graph undergoes edge insertions and deletions (referred to as edge *updates*), and the goal is to quickly report an approximation to the shortest paths between *any* source-target vertex pair. The dynamic setting is perhaps even more realistic for some of the applications of the APSP problem, e.g., in navigation systems, as link statistics of road networks are prone to changes because of evolving traffic conditions. A naive (but rather expensive) solution to handle these updates is achieved by running an exact static algorithm after each update. However, at an intuitive level, one would expect to somehow exploit the fact that a single update is small compared to the size of the network, and thus come up with much faster update times.

Much of the research literature in dynamic APSP has focused on the *partially* dynamic setting. In contrast to the *fully* dynamic counterpart, this weaker model restricts the types of updates to edge insertions or deletions only. Some reasons for studying partially dynamic algorithms include their application as a subroutine in speeding up static algorithms (e.g., flow problems [36]), or their utilization as a stepping stone for designing fully-dynamic algorithms, something that we will also exploit in this work. The popularity of the partially dynamic setting can also attributed to the fact that dealing with only one type of update usually leads to better algorithmic guarantees. In fact, the fully dynamic APSP problem admits strong conditional lower bounds in the *low approximation* regimes: under plausible hardness assumptions, Abboud and Vassilevska Williams [3], and later Henzinger, Krinninger, Nanongkai, and Saranurak [32] show that there are no dynamic APSP algorithms achieving a $(3-\epsilon)$ approximation with sublinear query time and the update time being a small polynomial.

From an upper bounds perspective, there are only two works that achieve sublinear update time for fully dynamic APSP. Abraham, Chechik, and Talwar [5] showed that there is an algorithm that achieves constant approximation and sublinear update time. However, their algorithm cannot break the $O(\sqrt{n})$ barrier on the update time. Forster, Goranci, and Henzinger [25] gave different trade-offs between approximation and update time. In particular, in $n^{o(1)}$ amortized update time and polylogarithmic query time they achieve $n^{o(1)}$ approximation. These two works suffer from either a large approximation guarantee or update time, leaving open the following key question:

Is there a fully dynamic APSP algorithm that achieves constant *approximation with a very small polynomial update and query time?*

1.1 Our result

In this paper, we answer the question of achieving constant approximation with a very small polynomial update time for the fully dynamic APSP in the affirmative, also known as the *fully dynamic distance oracle* problem. More generally, we obtain a trade-off between approximation, update time, and query time as follows:

▶ **Theorem 1.** *Given a weighted undirected graph $G=(V,E,w)$ with polynomial weights*[2]*, and a constant parameter $0<\rho<1$, there is a randomized fully dynamic distance oracle with constant stretch $(\frac{256}{\rho^2})^{4/\rho}$ that w.h.p. achieves $\tilde{O}(n^\rho)$ amortized update time and $\tilde{O}(n^{\rho/8})$ query time. These guarantees hold against an oblivious adversary.*

[2] In this paper, we assume for ease of notation that the edge weights are integers in the range from 1 to W, where W is polynomial in n. Using a standard approach (see e.g., [8]) this extends to rational edge weights in some range from the minimum weight $W_{\min}$ to the maximum weight $W_{\max}$, where $W_{\max}/W_{\min}$ is polynomial in n.

Our distance oracle can also be extended to report the actual (approximate) shortest path when answering queries (see the full version [26] for a sketch). In addition to the constant stretch regime, we obtain several interesting tradeoffs, as shown in Theorem 5. For example, our algorithm achieves $O(\log \log n)$ stretch with a much faster query time of $n^{o(1)}$ and very small polynomial update time (see Corollary 6).

Our result brings the algorithmic guarantees on fully dynamic distance oracles closer to the recent conditional hardness result by Abboud, Bringmann, Khoury, and Zamir [2] (and the subsequent refinement in [1]), who showed that there is no fully dynamic algorithm that simultaneously achieves constant approximation and $n^{o(1)}$ update and query time. We also remark that our results are consistent with their lower bound since if we insist on constant approximation, the above trade-off shows that the update time cannot be made as efficient as $n^{o(1)}$.

On the technical side, our result follows the widespread "high-level" approach of extending decremental algorithms to the fully dynamic setting (see e.g. [33, 38, 39, 40, 10, 30, 5, 25]) and it is inspired by recent developments on the dynamic distance oracle literature that rely on vertex sparsification [25, 18, 27]. Specifically, we design a reduction that turns a decremental hub-labeling scheme with some specific properties into a fully dynamic distance oracle, which may be of independent interest. Our key observation is that an existing state-of-the-art decremental distance oracle that works against an oblivious adversary can serve as such hub-labeling scheme. The fully dynamic distance oracle is then obtained by repeatedly applying the reduction whilst carefully tuning various parameters across levels in the hierarchy.

More generally, our reduction does not make any assumptions on the adversary and is based on properties that are quite natural. At a high-level, we consider decremental approximate hub-labeling schemes with the following properties. (1) For every vertex $v \in V$, maintain a set $S(v)$, called a *hub set*, that has bounded size. (2) For every vertex $v \in V$, maintain distance estimates $\delta(v, u)$ for each $u \in S(v)$, with bounded *recourse*, which is defined as the number of times such distance estimates are affected during the execution of the algorithm. (3) Return the final estimate between a pair of vertices $s, t \in V$, by minimizing estimates over elements in $S(s) \cap S(t)$.

Many known distance oracles (e.g. variants of the well-known distance oracle of [44]) have a query mechanism that satisfies the first and third properties, while efficient dynamic distance oracles are often based on bounded recourse structures satisfying the second property.

Hence we hope that this reduction can be further utilized in the future by characterizing deterministic decremental distance oracles or the ones with different stretch/time tradeoffs as such hub-labeling schemes. Similar reductions have been previously proposed in [5] and then refined in [25] in slightly different contexts. In this work, in addition to refining this approach for obtaining a constant stretch distance oracle, we aim to keep the reduction as modular as possible to facilitate potential future applications.

1.2 Related Work

In the following, we give an overview of existing works on fully dynamic all-pairs distance oracles by dividing them into several categories based on their stretch guarantee. Unless noted otherwise, all algorithms cited in the following are randomized and have amortized update time. We report running time bounds for constant accuracy parameter ϵ and assume that we are dealing with graphs with positive integer edge weights that are polynomial in the number of vertices. We would also like to point out that all "combinatorial" algorithms discussed in the following (i.e., algorithms that do not rely on "algebraic" techniques like dynamic matrix inverse) are internally employing decremental algorithms. Decremental algorithms have also been studied on their own with various tradeoffs [40, 10, 31, 16, 35, 24], and competitive deterministic algorithms have been devised, e.g., [30, 11, 19].

Exact. After earlier attempts on the problem [34, 23], Demetrescu and Italiano [22] presented their seminal work on exact distance maintenance achieving $\tilde{O}(n^2)$ update time (with log-factor improvements by Thorup [42]) and constant query time for weighted directed graphs.

Subsequently, researchers have developed algorithms with subcubic worst-case update time and constant query time [43, 4] with some of them being deterministic [28, 17]. Note that one can construct a simple update sequence for which any fully dynamic algorithm maintaining the distance matrix or the shortest path matrix explicitly needs to perform $\Omega(n^2)$ changes to this matrix per update.

Algorithms breaking the n^2 barrier at the cost of large query time have been obtained in unweighted directed graphs by Roditty and Zwick [39] (update time $\tilde{O}(mn^2/t^2)$ and query time $O(t)$ for any $\sqrt{n} \leq t \leq n^{3/4}$), Sankowski [41] (worst-case update time $O(n^{1.897})$ and query time $O(n^{1.265})$), and van den Brand, Nanongkai, and Saranurak [15] (worst-case update time $O(n^{1.724})$ and query time $O(n^{1.724})$). The latter two approaches are algebraic and their running time bounds depend on the matrix multiplication coefficient ω.

$(1 + \epsilon)$-approximation. In addition to exact algorithms, combinatorial and algebraic algorithms have also been developed for the low stretch regime of $(1 + \epsilon)$-approximation. In particular, Roditty and Zwick [40] obtained the following trade-off with a combinatorial algorithm: update time $\tilde{O}(mn/t)$ and query time $O(t)$ for any $\delta > 0$ and $t \leq m^{1/2-\delta}$. Subsequently, for $t \leq \sqrt{n}$, a deterministic variant was developed [30] and it was generalized to weighted directed graphs [10]. Furthermore, by a standard reduction (see e.g. [12]) using a decremental approximate single-source shortest paths algorithm [31, 11], one obtains a combinatorial, deterministic algorithm with update time $O(nm^{1+o(1)}/t)$ and query time $O(t)$ for any $t \leq n$, for the fully dynamic all-pairs problem in weighted undirected graphs. Conditional lower bounds [37, 3, 32] suggest that the update and the query time cannot be both small polynomials in n. For example, no algorithm can maintain a $(5/3 - \epsilon)$-approximation with update time $O(m^{1/2-\delta})$ and query time $O(m^{1-\delta})$ for any $\delta > 0$, unless the OMv conjecture fails [32].

Algebraic approaches can achieve subquadratic update time and sublinear query time, namely worst-case update time $O(n^{1.863})$ and query time $O(n^{0.666})$ in weighted directed graphs [14], or worst-case update time $O(n^{1.788})$ and query time $O(n^{0.45})$ in unweighted undirected graphs [13]. As the conditional lower bound by Abboud and Vassilevska Williams [3] shows, algebraic approaches seem to be necessary in this regime: unless one is able to multiply two $n \times n$ Boolean matrices in $O(n^{3-\delta})$ time for some constant $\delta > 0$, no fully dynamic algorithm for st reachability in directed graphs can have $O(n^{2-\delta'})$ update and query time and $O(n^{3-\delta'})$ preprocessing time (for some constant $\delta' > 0$). While not explicitly stated in the paper, the same conditional lower bound extends to fully dynamic $(1+\epsilon)$-approximate st distances on undirected unweighted graphs for a small enough constant ϵ.

$(2 + \epsilon)$-approximation. Apart from earlier work [34], the only relevant algorithm in the $(2+\epsilon)$-approximation regime is by Bernstein [9] and achieves update time $m^{1+o(1)}$ and query time $O(\log\log\log n)$ in weighted undirected graphs. It can be made deterministic using the deterministic approximate single-source shortest path algorithm by Bernstein, Probst Gutenberg, and Saranurak [11]. The only conditional lower bound in this regime that we are aware of states that no algorithm can maintain a $(3 - \epsilon)$-approximation with update time $O(n^{1/2-\delta})$ and query time $O(n^{1-\delta})$ for any $\delta > 0$, unless the OMv conjecture fails [32].

Larger approximation. In the regime of stretch at least 3, the following trade-offs between stretch and update time have been developed: Abraham, Chechik, and Talwar [5] designed an algorithm for unweighted undirected graphs with stretch $2^{O(\rho k)}$, update time $\tilde{O}(m^{1/2}n^{1/k})$, and query time $O(k^2\rho^2)$, where $k \geq 1$ is a freely chosen parameter and $\rho = 1 + \lceil \log n^{1-1/k} / \log(m/n^{1-1/k}) \rceil$. Forster, Goranci, and Henzinger [25] designed an algorithm for weighted undirected graphs with stretch $O(\log n)^{3k-2}$, update time $m^{1/k+o(1)} \cdot O(\log n)^{4k-2}$, and query time $O(k(\log n)^2)$, where $k \geq 2$ is an arbitrary integer parameter. Very recently, Chuzhoy and Zhang [20] independently obtained a deterministic algorithm for weighted undirected graphs with stretch $(\log \log n)^{2^{1/\rho^3}}$, update time $\tilde{O}(n^{O(\rho)})$, and query time $\tilde{O}(2^{O(1/\rho)})$ for any choice of $\frac{2}{(\log n)^{1/200}} < \rho < \frac{1}{400}$. Similar to our work, they also achieve sublogarithmic stretch but their guarantee cannot be reduced all the way to a constant. While our algorithm has the advantage of achieving constant stretch, their algorithm is deterministic, and thus works against an adaptive adversary. Finally, note that any algorithm whose update time depends on the sparsity of the graph (possibly also a static one) can be run on a spanner of the input graph maintained by a fully dynamic spanner algorithm [7]. These upper bounds are complemented by the following conditional lower bound: for any integer constant $k \geq 2$, there is no dynamic approximate distance oracle with stretch $2k - 1$, update time $O(m^u)$ and query time $O(m^q)$ with $ku + (k + 1)q < 1$, unless the 3-SUM conjecture fails [1].

2 Preliminaries

We consider weighted undirected graphs $G = (V, E, w)$ with positive integer edge weights. We denote by $n = |V|$ the number of vertices, by $m = |E|$ the number of edges, and by W the maximum weight of an edge. For every pair of vertices $u, v \in V$, the *distance* $\mathrm{dist}_G(u, v)$ between u and v in G is the length of a shortest path from u to v in G. For a path P, we denote by $w_G(P)$ the length of P in G, by $E(P)$ the edges of P, and by $|P| = |E(P)|$ the number of edges of P. Also for a graph H, we denote by $V(H)$ and $E(H)$ the vertex and the edge set of H respectively.

In dynamic graph algorithms, the graph is subject to updates and the algorithm has to process these updates by spending as little time as possible. In this paper, we consider updates that insert a single edge to the graph or delete a single edge from the graph. Moreover, observe that an update that changes the weight of an edge can be simulated by two updates, where the first update deletes the corresponding edge and the second update re-inserts the edge with the new weight. Let $G^{(0)}$ be the initial graph, and $G^{(\tau)}$ be the graph at time τ which is the time after τ updates have been performed to the graph.

In this paper we are interested in designing *fully dynamic* algorithms which can process edge insertions and edge deletions, and thus, weight changes as well. A *decremental* algorithm can process only edge deletions and weight increases. We assume that the updates to the graph are performed by an *oblivious adversary* who fixes the sequence of updates before the algorithm starts. Namely, the adversary cannot adapt the updates based on the choices of the algorithm during the execution. We say that an algorithm has *amortized update time* $u(n, m)$ if its total time spent for processing any sequence of ℓ updates is bounded by $\ell \cdot u(n, m)$, when it starts from an empty graph with n vertices and during all the updates has at most m edges (the time needed to initialize the algorithm on the empty graph before the first update is also included). An algorithm is *path reporting* if after a query can also return the corresponding path explicitly.

In our analysis we use $\tilde{O}(1)$ to hide factors polylogarithmic in nW. Namely, we write $\tilde{O}(1)^d$ to represent the term $O(\log^{cd} nW)$, for a constant c and a parameter d.

3 Fully Dynamic Distance Oracle

The technical details of our distance oracle are divided into three parts. Initially in Section 3.1, we give the definition of a hub-labeling scheme together with other useful definitions. Afterwards, we provide a reduction for extending a decremental approximate hub-labeling scheme with some properties to a fully dynamic distance oracle. Then in Section 3.2, we explain how an existing decremental algorithm gives us an approximate hub-labeling scheme that we can use in this reduction, and finally in Section 3.3 we put everything together by applying our reduction repeatedly, in order to get a family of fully dynamic distance oracles.

3.1 From decremental hub-labeling scheme to fully dynamic distance oracle via reduction

We start by defining approximate hub-labeling schemes, and then explain how they are used in our reduction. Hub-labeling schemes were formally defined by [6] (and were previously introduced under the name 2-hop cover[3] in [21]). We are slightly modifying the definition for our purpose, for instance by considering an approximate variant.

▶ **Definition 2** (Approximate Hub-Labeling Scheme). *Given a graph $G = (V, E)$, a* hub-labeling scheme $\mathcal{L}$ *of* stretch α *consists of*

1. *for every vertex $v \in V$, a* hub set $S(v) \subseteq V$ *and*
2. *for every pair of vertices $u, v \in V$, a distance estimate $\delta(v, u)$ such that* $\mathrm{dist}_G(v, u) \leq \delta(v, u) < \infty$ *if $u \in S(v)$ and $\delta(v, u) = \infty$ otherwise.*

and for every pair of vertices s and t guarantees that

$$\delta_{\mathcal{L}}(s, t) := \min_{v \in S(s) \cap S(t)} (\delta(s, v) + \delta(t, v)) \leq \alpha \cdot \mathrm{dist}_G(s, t)\,.$$

The *distance label* of a vertex v consists of the hub set $S(v)$ and the corresponding distance estimates $\delta(v, u)$, for all $u \in S(v)$.

Note that the definition implies $\delta_{\mathcal{L}}(s, t) \geq \mathrm{dist}_G(s, t)$ for every pair of vertices s and t. Furthermore, a hub-labeling scheme of stretch α directly implements a distance oracle of stretch α with query time $O(\max_{v \in V} |S(v)|)$ that consists of the collection of distance labels for all vertices $v \in V$. We also remark that the entries of value ∞ in the distance estimate $\delta(\cdot, \cdot)$ do not need to be stored explicitly if the hub sets are stored explicitly and that the distance estimate $\delta(\cdot, \cdot)$ is not necessarily symmetric.

In the following we consider *decremental* algorithms for maintaining approximate hub-labeling schemes, that is, *decremental approximate hub-labeling schemes* which process each edge deletion in the graph by first updating their internal data structures and then outputting the changes made to the hub sets and the distance estimates $\delta(\cdot, \cdot)$. Namely for a vertex $v \in V$, vertices may leave or join $S(v)$, or the distance estimates of vertices belonging to $S(v)$ may change, since the decremental algorithm has to update this information for maintaining correctness at query time.

Denote by $S^{(\tau)}(v)$ the hub set of a vertex $v \in V$ after τ updates have been processed by the decremental approximate hub-labeling scheme (we may omit the superscript τ whenever time is fixed), where $\tau \geq 1$ is an integer parameter. Then for a pair of vertices $u, v \in V$, the distance estimate $\delta(v, u)$ after τ updates is defined based on Definition 2 and $S^{(\tau)}(v)$. Namely, if u is inside the hub set of v after τ updates (i.e., $u \in S^{(\tau)}(v)$) then $\mathrm{dist}_{G^{(\tau)}}(v, u) \leq \delta(v, u) < \infty$, otherwise $\delta(v, u) = \infty$.

[3] The concept of 2-hop cover or hub-labeling should not be confused with the (related) concept of a hopset that we will later see in Section 3.2.

After τ edge deletions have been processed by the decremental approximate hub-labeling scheme, there are three possible types of changes to the distance estimates $\delta(v,\cdot)$ that correspond to a vertex $v \in V$, due to the last edge deletion. (1) The distance estimate $\delta(v,u)$ changes for a vertex $u \in S^{(\tau-1)}(v) \cap S^{(\tau)}(v)$ that remains inside the hub set of v. (2) The distance estimate $\delta(v,u)$ becomes ∞ because a vertex $u \in S^{(\tau-1)}(v) \setminus S^{(\tau)}(v)$ leaves the hub set of v. (3) The distance estimate $\delta(v,u)$ receives a finite value because a vertex $u \in S^{(\tau)}(v) \setminus S^{(\tau-1)}(v)$ enters the hub set of v. Let $\chi^{(\tau)}(v)$ be the number of all these changes to $\delta(v,\cdot)$ corresponding to v at time τ. In other words, for a fixed vertex $v \in V$, the value of $\chi^{(\tau)}(v)$ is equal to the number of vertices u whose corresponding value of $\delta(v,u)$ changes due to the last edge deletion. Moreover, let $X(v) = \sum_\tau \chi^{(\tau)}(v)$ be the total number of such changes to $\delta(v,\cdot)$ corresponding to v over the course of the algorithm.

In the following lemma, we present a reduction from a decremental approximate hub-labeling scheme to a fully dynamic distance oracle.

▶ **Lemma 3.** *Consider a decremental hub-labeling scheme $\mathcal{A}$ of stretch α with total update time $T_{\mathcal{A}}(n,m,W)$ and query time $Q_{\mathcal{A}}(n,m,W)$, with the following properties:*

1. *$\forall v \in V$ and $\forall \tau : |S^{(\tau)}(v)| \leq \gamma$. In other words, the size of the hub set of any vertex is bounded by γ at any moment of the algorithm.*
2. *$\forall v \in V : X(v) \leq \zeta$. In other words, for every vertex $v \in V$ the total number of changes to $\delta(v,\cdot)$ is at most ζ over the course of the algorithm. Moreover the algorithm detects and reports these changes explicitly.*

Then given $\mathcal{A}$ and a fully dynamic distance oracle $\mathcal{B}$ of stretch β with amortized update time $t_{\mathcal{B}}(n,m,W)$ and query time $Q_{\mathcal{B}}(n,m,W)$, for any integer $\ell \geq 1$, there is a fully dynamic distance oracle $\mathcal{C}$ of stretch $\alpha\beta$ with amortized update time $t_{\mathcal{C}}(n,m,W) = T_{\mathcal{A}}(n,m,W)/\ell + t_{\mathcal{B}}(\min(\ell(2+2\mu),n), \ell(1+2\mu), nW) \cdot (2+4\mu)$ and query time $Q_{\mathcal{C}}(n,m,W) = Q_{\mathcal{A}}(n,m,W) + \gamma^2 \cdot Q_{\mathcal{B}}(\min(\ell(2+2\mu),n), \ell(1+2\mu), nW)$, where $\mu = \gamma + \zeta$.

Proof. We organize the proof in three parts. The first part gives the reduction from $\mathcal{A}$ and $\mathcal{B}$ to $\mathcal{C}$, and the second and third part concern the correctness and the running times respectively.

Reduction. The fully dynamic distance oracle $\mathcal{C}$ proceeds in phases of length ℓ. For each phase, we denote by τ the number of updates processed by $\mathcal{A}$ during the phase. At the beginning of the first phase (which is also the beginning of the algorithm), $\mathcal{C}$ initializes the fully dynamic distance oracle $\mathcal{B}$ on the initially empty graph G consisting of 2ℓ vertices[4], and sets an update counter to 0. Whenever an update to G occurs in the first phase, the update is directly processed by $\mathcal{B}$.[5] As soon as the number of updates is more than ℓ, the second phase is started. We define several sets and the graph H that the fully dynamic distance oracle $\mathcal{C}$ maintains during each subsequent phase:

- Let F be the set of edges present in G at the beginning of the phase, E be the current set of edges in G, and D be the set of edges deleted from G during the phase.
- Let $I = E \setminus (F \setminus D)$ be the set of edges inserted to G since the beginning of the phase without subsequently having been deleted during the phase, and $U = \{v \in V \mid \exists e \in I : v \in e\}$ be the set of endpoints of edges in I.

[4] This minor technical detail makes sure that $\mathcal{B}$ does not have to deal with vertex insertions.
[5] The special treatment of the first ℓ updates is just a technical necessity for a rigorous amortization argument in the running time analysis.

- Let H be the auxiliary graph that consists of all edges $(u,v) \in I$, together with their hub sets $S^{(\tau)}(u)$ and $S^{(\tau)}(v)$ after τ edge deletions have been processed by $\mathcal{A}$. Specifically, $V(H) = \{v \in V \mid v \in U \text{ or } (u \in U \text{ and } v \in S^{(\tau)}(u))\}$ and $E(H) = \{(u,v) \mid (u,v) \in I \text{ or } (v \in U \text{ and } u \in S^{(\tau)}(v))\}$. Note that at any fixed moment, the size of $V(H)$ is at most $\ell \cdot (2+2\gamma)$ and the size of $E(H)$ is at most $\ell \cdot (1+2\gamma)$.

At the beginning of each subsequent phase, $\mathcal{C}$ stores the sets F, E, D, I, U, and the auxiliary graph H, and sets an update counter to 0. Furthermore, $\mathcal{C}$ initializes the decremental approximate hub-labeling scheme $\mathcal{A}$ on the current graph G, and the fully dynamic distance oracle $\mathcal{B}$ on H which is initially an empty "sketch" graph on $\ell \cdot (2+2\mu)$ vertices. The graph H can be thought of as responsible for maintaining estimates for paths that use inserted edges.

Whenever an update to G occurs, $\mathcal{C}$ first checks via the update counter whether the number of updates since the beginning of the phase is more than ℓ. If this is the case, then $\mathcal{C}$ starts a new phase. Otherwise, after an update the fully dynamic distance oracle $\mathcal{C}$ does the following. On the insertion of an edge (u,v) to G, $\mathcal{C}$ adds (u,v) to I, adds u and v to U, and adds the edge (u,v) to H, together with the edges (u,p) for every $p \in S^{(\tau)}(u)$ and (v,p) for every $p \in S^{(\tau)}(v)$. Any time an edge (u,v) is added to H, its weight is set to:

$$w_H(u,v) = \min(w_G(u,v), \delta(u,v), \delta(v,u)).$$

Whenever the first edge incident to some vertex v is added to H, the algorithm finds a "fresh" vertex (of degree 0) in H and henceforth identifies it as v. This is always possible, since by the two properties of $\mathcal{A}$, the number of such vertices in a phase of length ℓ is at most $\ell \cdot (2+2\mu)$.

On the deletion of an edge $(u,v) \in E$ from G, there are two cases to consider.

1. If the edge (u,v) was not present at the beginning of the current phase, or has been deleted and re-inserted (i.e., $(u,v) \in I$), then $\mathcal{C}$ removes (u,v) from I, adds (u,v) to D, and updates the set U and the graph H accordingly. In particular, if $u \in U$ and $v \in S^{(\tau)}(u)$, or $v \in U$ and $u \in S^{(\tau)}(v)$, $\mathcal{C}$ updates the weight of the edge (u,v) in H to $w_H(u,v) = \min(\delta(u,v), \delta(v,u))$ (as $w_G(u,v) = \infty$ after the deletion), otherwise $\mathcal{C}$ removes (u,v) from H. Also, for all the vertices v that left U and all the edges $(v,p) \in E(H)$ such that $p \in S^{(\tau)}(v)$, if $p \in U$ and $v \in S^{(\tau)}(p)$, then $\mathcal{C}$ updates the weight of (v,p) in H to $w_H(v,p) = \delta(p,v)$ (as $v \notin U$ after the deletion), and otherwise $\mathcal{C}$ removes (v,p) from H.
2. If the edge (u,v) was present at the beginning of the current phase and has not been deleted yet (i.e., $(u,v) \in F \setminus D$), then $\mathcal{C}$ adds (u,v) to D and the deletion is processed by $\mathcal{A}$. Whenever $\mathcal{A}$ changes some distance estimates $\delta(v,\cdot)$ that correspond to a vertex $v \in U$ (i.e, v is a vertex of H and an endpoint of an edge in I) and its hub set, $\mathcal{C}$ updates the graph H accordingly. In particular, there are three possible scenarios at time τ of $\mathcal{A}$.[6] (1) Whenever the value of $\delta(v,u)$ changes for a vertex $u \in S^{(\tau-1)}(v) \cap S^{(\tau)}(v)$ that remains inside the hub set of v, $\mathcal{C}$ updates the weight of the edge (v,u) in H to $w_H(v,u) = \min(w_G(v,u), \delta(v,u), \delta(u,v))$. (2) Whenever a vertex $u \in S^{(\tau-1)}(v) \setminus S^{(\tau)}(v)$ leaves the hub set of v, then if $(v,u) \in I$ or $u \in U$ and $v \in S^{(\tau)}(u)$, $\mathcal{C}$ updates the weight of the edge (v,u) in H to $w_H(v,u) = \min(w_G(v,u), \delta(u,v))$ (as $\delta(v,u) = \infty$ after the deletion), otherwise $\mathcal{C}$ removes (v,u) from H. (3) Whenever a vertex $u \in S^{(\tau)}(v) \setminus S^{(\tau-1)}(v)$ enters the hub set of v, $\mathcal{C}$ adds the edge (v,u) to H (unless it exists already) and updates its weight to $w_H(v,u) = \min(w_G(v,u), \delta(v,u), \delta(u,v))$. Note that the number of these

[6] Note that τ is the number of updates processed only by $\mathcal{A}$ during the phase.

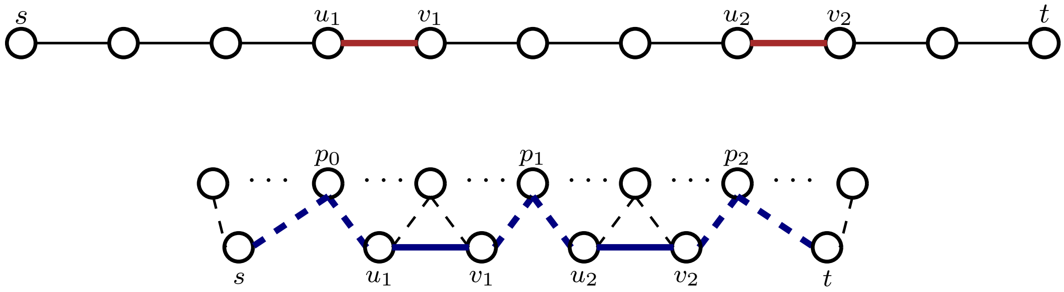

Figure 1 Illustration of an s-t shortest path. The brown thick edges have been inserted since the beginning of the phase. The corresponding subgraph of the auxiliary graph H is also depicted (note that the vertices s and t are not necessarily part of H). The blue thick edges are the ones that participate in the correctness analysis of the query. Dashed edges depict edges inside the hub sets.

changes at time τ of $\mathcal{A}$ is equal to $\chi^{(\tau)}(v)$ for a vertex $v \in V$. Observe also that based on the two properties of $\mathcal{A}$, the number of vertices that participate in H during a phase of length ℓ is at most $\ell \cdot (2 + 2\mu)$. Thus we can always find a "fresh" vertex (of degree 0) in H.

Finally, all the changes performed to H are processed by the fully dynamic distance oracle $\mathcal{B}$ running on H, where edge weight changes are simulated by a deletion followed by a re-insertion.

Now a query for the approximate distance between any pair of vertices s and t is answered by returning:

$$\delta_{\mathcal{C}}(s,t) = \min\left(\min_{p \in S^{(\tau)}(s) \cap V(H), q \in S^{(\tau)}(t) \cap V(H)} \left(\delta(s,p) + \delta_{\mathcal{B}}(p,q) + \delta(t,q)\right), \delta_{\mathcal{A}}(s,t)\right).$$

Whenever $S^{(\tau)}(s) \cap V(H) = \emptyset$ or $S^{(\tau)}(t) \cap V(H) = \emptyset$, we let the inner term $\min(\cdot)$ to be ∞.

Correctness. To prove the correctness of this algorithm, we need to show that $\mathrm{dist}_G(s,t) \leq \delta_{\mathcal{C}}(s,t) \leq \alpha\beta \cdot \mathrm{dist}_G(s,t)$. The lower bound $\mathrm{dist}_G(s,t) \leq \delta_{\mathcal{C}}(s,t)$ is immediate, since for each approximate distance returned by $\mathcal{C}$, the corresponding path uses edges from G or distance estimates from the decremental approximate hub-labeling scheme which are never an underestimation of the real distance. To prove the upper bound, consider a shortest path π from s to t in G, and let $G_{\mathcal{A}}$ be the graph maintained by $\mathcal{A}$ (i.e., the edge set of $G_{\mathcal{A}}$ is $E(G_{\mathcal{A}}) = F \setminus D$). If the path π contains only edges from the set $F \setminus D$, then $\delta_{\mathcal{C}}(s,t) \leq \delta_{\mathcal{A}}(s,t) \leq \alpha \cdot \mathrm{dist}_{G_{\mathcal{A}}}(s,t) = \alpha \cdot \mathrm{dist}_G(s,t)$, and the claim follows. Otherwise, let $(u_1, v_1), \ldots, (u_j, v_j) \in I$ denote the edges of π that have been inserted since the beginning of the current phase in order of appearance on π. Furthermore, let $p_0 \in S^{(\tau)}(s) \cap S^{(\tau)}(u_1)$ be the vertex that "certifies" $\delta_{\mathcal{A}}(s, u_1)$, that is, $\delta_{\mathcal{A}}(s, u_1) = \delta(s, p_0) + \delta(u_1, p_0)$. Similarly, let $p_j \in S^{(\tau)}(v_j) \cap S^{(\tau)}(t)$ be the vertex that "certifies" $\delta_{\mathcal{A}}(v_j, t)$, and for every $1 \leq i \leq j-1$, let $p_i \in S^{(\tau)}(v_i) \cap S^{(\tau)}(u_{i+1})$ be the vertex that "certifies" $\delta_{\mathcal{A}}(v_i, u_{i+1})$ (see Figure 1). These vertices must exist by the definition of an approximate hub-labeling scheme. Furthermore, by the construction of H, the edges (u_1, p_0) and (v_j, p_j) have been inserted to H, because $u_1 \in U$ and $p_0 \in S^{(\tau)}(u_1)$, and $v_j \in U$ and $p_j \in S^{(\tau)}(v_j)$ respectively. Hence, the vertices p_0 and p_j belong to $V(H)$, and the sum $\delta(s, p_0) + \delta_{\mathcal{B}}(p_0, p_j) + \delta(t, p_j)$ participates in the inner term $\min(\cdot)$. Therefore to analyze the claimed upper-bound on the stretch, we proceed as follows:

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\le \delta(s,p_0) + \delta_{\mathcal{B}}(p_0,p_j) + \delta(t,p_j) \\
&\text{(stretch guarantee of } \mathcal{B}\text{)} \\
&\le \delta(s,p_0) + \beta \cdot \mathrm{dist}_H(p_0,p_j) + \delta(t,p_j) \\
&\text{(triangle inequality)} \\
&\le \delta(s,p_0) + \beta \cdot \mathrm{dist}_H(p_0,u_1) \\
&\quad + \sum_{1\le i\le j-1} \beta \cdot (\mathrm{dist}_H(u_i,v_i) + \mathrm{dist}_H(v_i,p_i) + \mathrm{dist}_H(p_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_H(u_j,v_j) + \mathrm{dist}_H(v_j,p_j)) + \delta(t,p_j) \\
&(\mathrm{dist}_H \le w_H) \\
&\le \delta(s,p_0) + \beta \cdot w_H(p_0,u_1) + \sum_{1\le i\le j-1} \beta \cdot (w_H(u_i,v_i) + w_H(v_i,p_i) + w_H(p_i,u_{i+1})) \\
&\quad + \beta \cdot (w_H(u_j,v_j) + w_H(v_j,p_j)) + \delta(t,p_j)
\end{aligned}
$$

By the construction of H, the edges (u_i,v_i) of π and the corresponding edges (p_{i-1},u_i) and (v_i,p_i) have been inserted to H[7], because $(u_i,v_i) \in I$, $u_i \in U$ and $p_{i-1} \in S^{(\tau)}(u_i)$, and $v_i \in U$ and $p_i \in S^{(\tau)}(v_i)$ respectively. Hence by the definition of $w_H(\cdot)$, we can replace $w_H(u_i,v_i)$ with $w_G(u_i,v_i)$, $w_H(p_{i-1},u_i)$ with $\delta(u_i,p_{i-1})$ and $w_H(v_i,p_i)$ with $\delta(v_i,p_i)$. As a result, we have that (where $\alpha \ge 1$ and $\beta \ge 1$):

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\le \delta(s,p_0) + \beta \cdot \delta(u_1,p_0) + \sum_{1\le i\le j-1} \beta \cdot (w_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (w_G(u_j,v_j) + \delta(v_j,p_j)) + \delta(t,p_j) \\
&(\pi \text{ is a shortest path}) \\
&= \delta(s,p_0) + \beta \cdot \delta(u_1,p_0) + \sum_{1\le i\le j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta(v_j,p_j)) + \delta(t,p_j) \\
&\le \beta \cdot (\delta(s,p_0) + \delta(u_1,p_0)) + \sum_{1\le i\le j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta(v_i,p_i) + \delta(u_{i+1},p_i)) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta(v_j,p_j) + \delta(t,p_j)) \\
&\text{(definition of approximate hub-labeling scheme)} \\
&= \beta \cdot \delta_{\mathcal{A}}(s,u_1) + \sum_{1\le i\le j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \delta_{\mathcal{A}}(v_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \delta_{\mathcal{A}}(v_j,t))
\end{aligned}
$$

From the stretch guarantee of $\mathcal{A}$, it holds that $\delta_{\mathcal{A}}(u,v) \le \alpha \cdot d_{G_{\mathcal{A}}}(u,v)$ for any pair of vertices $u,v \in V$. For any two vertices v_i, u_{i+1} from the previous sum, the subpath of π from v_i to u_{i+1} uses edges only from the set $F \setminus D$, implying that $d_{G_{\mathcal{A}}}(v_i,u_{i+1}) = d_G(v_i,u_{i+1})$. The same argument holds for the pairs s,u_1 and v_j,t, thus it follows that:

$$
\begin{aligned}
\delta_{\mathcal{C}}(s,t) &\le \alpha\beta \cdot \mathrm{dist}_G(s,u_1) + \sum_{1\le i\le j-1} \beta \cdot (\mathrm{dist}_G(u_i,v_i) + \alpha \cdot \mathrm{dist}_G(v_i,u_{i+1})) \\
&\quad + \beta \cdot (\mathrm{dist}_G(u_j,v_j) + \alpha \cdot \mathrm{dist}_G(v_j,t))
\end{aligned}
$$

[7] If $v_i = u_{i+1}$ then $p_i = v_i$, and so $w_H(v_i,p_i) = w_H(p_i,u_{i+1}) = 0$.

$$\leq \alpha\beta \cdot \text{dist}_G(s, u_1) + \sum_{1 \leq i \leq j-1} \alpha\beta \cdot (\text{dist}_G(u_i, v_i) + \text{dist}_G(v_i, u_{i+1}))$$
$$+ \alpha\beta \cdot (\text{dist}_G(u_j, v_j) + \text{dist}_G(v_j, t)) = \alpha\beta \cdot \text{dist}_G(s, t).$$

Update and Query time. To analyze the running times, consider a fixed phase of length ℓ. During the first phase, the query time is $Q_{\mathcal{B}}(2\ell, \ell, W)$ and the amortized update is $t_{\mathcal{B}}(2\ell, \ell, W)$, as the initially empty graph G can have at most 2ℓ vertices and ℓ edges after ℓ updates. For the subsequent phases we proceed as follows. By the construction of H and the two properties of $\mathcal{A}$, the graph H has at most $\min(\ell(2+2\mu), n)$ vertices and $\ell(1+2\mu)$ edges during the phase, and the maximum edge weight in H is nW (the maximum distance in G).[8] Moreover by the first property we have that $|S^{(\tau)}(s) \cap V(H)| \leq \gamma$ and $|S^{(\tau)}(t) \cap V(H)| \leq \gamma$. Therefore the query time is equal to:

$$Q_{\mathcal{C}}(n, m, W) = Q_{\mathcal{A}}(n, m, W) + \gamma^2 \cdot Q_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW).$$

Let us now analyze the amortized update time. Since the total update time of $\mathcal{A}$ is $T_{\mathcal{A}}(n, m, W)$ and the amortized update time of $\mathcal{B}$ is $t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW)$ during the phase, it remains to bound the total number of updates to H per phase. Whenever an edge $e = (u, v)$ is inserted to G, we add to H the two endpoints u and v together with their hub sets $S^{(\tau)}(u)$ and $S^{(\tau)}(v)$, and at most $1 + 2\gamma$ updates can occur to H. Until (u, v) is deleted from H, every update to H between u, v and their hub sets modifies an entry of the distance estimate $\delta(u, \cdot)$ or $\delta(v, \cdot)$. By the definition of $\chi^{(\tau)}(\cdot)$, the number of entries of the distance estimates $\delta(u, \cdot)$ and $\delta(v, \cdot)$ that are modified at time τ of $\mathcal{A}$ is equal to $\chi^{(\tau)}(u) + \chi^{(\tau)}(v)$. Hence until (u, v) is deleted from H, the total number of updates to H between u, v and their hub sets is equal to $2 \cdot (\sum_\tau \chi^{(\tau)}(u) + \sum_\tau \chi^{(\tau)}(v)) = 2 \cdot (X(u) + X(v))$,[9] which is at most 4ζ based on the second property of Lemma 3. Moreover, when the edge e is deleted from G, at most $1 + 2\gamma$ updates can occur to H. Therefore, the total number of updates to H that correspond to an inserted edge in G, is at most $2 + 4\gamma + 4\zeta = 2 + 4\mu$ per phase. Since there can be at most ℓ inserted edges per phase, the total number of updates to H during a phase is at most $\ell(2+4\mu)$. This implies that the total time for processing all updates during a phase is $T_{\mathcal{A}}(n, m, W) + t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW) \cdot \ell(2+4\mu)$, which (when amortized over the ℓ updates of the previous phase) amounts to an amortized update time of:

$$T_{\mathcal{C}}(n, m, W) = \frac{T_{\mathcal{A}}(n, m, W)}{\ell} + t_{\mathcal{B}}(\min(\ell(2+2\mu), n), \ell(1+2\mu), nW) \cdot (2+4\mu). \blacktriangleleft$$

3.2 Decremental approximate hub-labeling scheme

In this section, we argue that an existing decremental distance oracle from [35] also provides an approximate hub-labeling scheme whose properties make the reduction of Lemma 3 quite efficient. This decremental algorithm is based on the well-known static Thorup-Zwick (TZ) distance oracle [44].

Thorup-Zwick distance oracle. Given a graph $G = (V, E)$, the construction starts by defining a non-increasing sequence of sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_k = \emptyset$, where for each $1 \leq i < k$, the set A_i is obtained by subsampling each element of A_{i-1} independently with probability $n^{-1/k}$.

[8] We can assume that $\delta(\cdot, \cdot)$ is upper bounded by nW whenever it has a finite value, since the maximum distance in G is at most nW. Likewise, we can use the value $nW + 1$ instead of ∞.

[9] We multiply by 2 because edge weight changes are simulated by a deletion followed by a re-insertion.

For every vertex $v \in V$ and $1 \leq i < k$, let $\delta(v, A_i) = \min_{u \in A_i} \mathrm{dist}_G(v, u)$ be the minimum distance from v to a vertex in A_i. As $A_k = \emptyset$, we let $\delta(v, A_k) = \infty$. Moreover, let $p_i(v) \in A_i$ be a vertex in A_i closest to v, that is, $\mathrm{dist}_G(v, p_i(v)) = \delta(v, A_i)$. Then, the bunch $B(v) \subseteq V$ of each $v \in V$ is defined as:

$$B(v) = \bigcup_{i=0}^{k-1} B_i(v) \ , \quad \text{where} \quad B_i(v) = \{u \in A_i \setminus A_{i+1} : \mathrm{dist}_G(v, u) < \mathrm{dist}_G(v, A_{i+1})\}.$$

The cluster of a vertex $u \in A_i \setminus A_{i+1}$ is defined as $C(u) = \{v \in V : \mathrm{dist}_G(v, u) < \mathrm{dist}_G(v, A_{i+1})\}$. Observe that $u \in B(v)$ if and only if $v \in C(u)$, for any $u, v \in V$.

As noted in [44], this construction is a hub-labeling scheme of stretch $2k-1$ (see Definition 2), where the hub set $S(v)$ of a vertex $v \in V$ is $S(v) = B(v) \cup (\bigcup_{i=0}^{k-1} \{p_i(v)\})$. In other words, bunches and pivots of all the k levels form a hub set for v. For obtaining the distance estimates $\delta(v, \cdot)$ for all $v \in V$ as in Definition 2, we need the associated distances $\delta(v, u) = \mathrm{dist}_G(v, u)$ for all $u \in S(v)$. It can be shown that with a simple modification of the stretch argument (e.g. see [29]), it is enough to only use the bunches as the hub sets, and explicit access to pivots is not necessary. Hence for simplifying the presentation in this section we assume that the hub sets are equivalent with the bunches. As shown in [44], the size of the bunch of any vertex is w.h.p. bounded by $\tilde{O}(n^{1/k})$. Recall that the maximum hub set size is one of the parameters governing the efficiency of our reduction.

In the next lemma we present the decremental algorithm of [35] which has good properties for the reduction of Lemma 3. For a more detailed explanation of the lemma see the full version [26].

▶ **Lemma 4** (Implicit in [35]). *Given a weighted undirected graph $G = (V, E)$ and $k > 1, 0 < \epsilon < 1$, there is a decremental hub-labeling scheme of stretch $(2k-1)(1+\epsilon)$ and w.h.p. the total update time is $\tilde{O}(mn^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$. Moreover, w.h.p. we have the following two properties:*

1. *$\forall v \in V$ and $\forall \tau : |S^{(\tau)}(v)| \leq \tilde{O}(n^{1/k})$. In other words, the size of the bunch of any vertex is bounded by $\tilde{O}(n^{1/k})$ at any moment of the algorithm.*
2. *$\forall v \in V : X(v) \leq \tilde{O}(n^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$. In other words, for every vertex $v \in V$ the total number of changes to $\delta(v, \cdot)$ is at most $\tilde{O}(n^{1/k}) \cdot O(\log nW/\epsilon)^{2k+1}$ over the course of the algorithm. Moreover the algorithm detects and reports these changes explicitly.*

3.3 Putting it together

In this section we explain how to obtain our final fully dynamic distance oracle by using the decremental algorithm of Section 3.2 in our reduction of Lemma 3.

▶ **Theorem 5.** *For any integer parameters $i \geq 0, k > 1$, there is a fully dynamic distance oracle $\mathcal{B}_i$ with stretch $(4k)^i$ and w.h.p. the amortized update time is $t_{\mathcal{B}_i}(n, m, W) = \tilde{O}(1)^{ki} \cdot m^{3/(3i+1)} \cdot n^{4i/k}$ and the query time $Q_{\mathcal{B}_i}(n, m, W) = \tilde{O}(1)^i \cdot n^{2i/k}$.*

Proof. The proof is by induction on the parameter i. For the base case $i = 0$, let $\mathcal{B}_0$ be the trivial fully dynamic distance oracle that achieves stretch 1, amortized update time $t_{\mathcal{B}_0}(n, m, W) = O(n^3)$, and query time $Q_{\mathcal{B}_0}(n, m, W) = O(1)$, by recomputing all-pairs shortest paths from scratch after each update (e.g., with the Floyd–Warshall algorithm).

For the induction step, let $\mathcal{A}$ denote the decremental approximate hub-labeling scheme from Lemma 4 with stretch $\alpha = 4k$ and w.h.p. total update time $T_{\mathcal{A}}(n, m, W) = \tilde{O}(1)^k \cdot mn^{1/k}$ and query time $Q_{\mathcal{A}}(n, m, W) = \tilde{O}(1) \cdot n^{1/k}$, where ϵ has been replaced with any value strictly smaller than $\frac{1}{2}$. By inductive hypothesis, we have that $\mathcal{B}_i$ (with $i \geq 0$) is a fully dynamic distance oracle of stretch $\beta_i = (4k)^i$ with amortized update time $\tilde{O}(1)^{ki} \cdot m^{3/(3i+1)} \cdot n^{4i/k}$ and query time $\tilde{O}(1)^i \cdot n^{2i/k}$. Based on Lemma 4, w.h.p. the decremental approximate hub-labeling scheme $\mathcal{A}$ satisfies the properties of Lemma 3 with $\gamma = \tilde{O}(1) \cdot n^{1/k}$ and $\zeta = \tilde{O}(1)^k \cdot n^{1/k}$. By applying then Lemma 3 to $\mathcal{A}$ and $\mathcal{B}_i$ with $\ell = m^{(3i+1)/(3i+4)}$, the resulting fully dynamic distance oracle $\mathcal{B}_{i+1}$ has stretch $(4k)^{i+1}$, and amortized update time: $t_{\mathcal{B}_{i+1}}(n, m, W) = T_{\mathcal{A}}(n, m, W)/\ell + t_{\mathcal{B}_i}(n, \ell(1 + 2\mu), nW) \cdot (2 + 4\mu)$. The first term is equal to: $\tilde{O}(1)^k \cdot mn^{1/k}/\ell = \tilde{O}(1)^k \cdot m^{3/(3i+4)} \cdot n^{1/k} = \tilde{O}(1)^k \cdot m^{3/(3(i+1)+1)} \cdot n^{1/k}$, and the second term is equal to (where $\mu = \tilde{O}(1)^k \cdot n^{1/k}$):

$$
\begin{aligned}
&t_{\mathcal{B}_i}(n, \ell(1 + 2\mu), nW) \cdot (2 + 4\mu) = \tilde{O}(1)^{ki} \cdot (\ell \cdot \tilde{O}(1)^k \cdot n^{1/k})^{3/(3i+1)} \cdot n^{4i/k} \cdot \tilde{O}(1)^k \cdot n^{1/k} \\
&\text{(Replace } \ell \text{ with } m^{(3i+1)/(3i+4)}\text{)} \\
&= \tilde{O}(1)^{ki} \cdot (m^{(3i+1)/(3i+4)} \cdot \tilde{O}(1)^k \cdot n^{1/k})^{3/(3i+1)} \cdot n^{4i/k} \cdot \tilde{O}(1)^k \cdot n^{1/k} \\
&\text{(Replace } n^{3/(3i+1)k} \text{ with } n^{3/k} \text{ and } \tilde{O}(1)^{3k/(3i+1)} \text{ with } \tilde{O}(1)^{3k}\text{)} \\
&= \tilde{O}(1)^{ki} \cdot m^{3/(3i+4)} \cdot \tilde{O}(1)^{3k} \cdot n^{3/k} \cdot n^{4i/k} \cdot \tilde{O}(1)^k \cdot n^{1/k} \\
&= \tilde{O}(1)^{ki+k} \cdot m^{3/(3i+4)} \cdot n^{(4i+4)/k} = \tilde{O}(1)^{k(i+1)} \cdot m^{3/(3(i+1)+1)} \cdot n^{4(i+1)/k}.
\end{aligned}
$$

Therefore the amortized update time of $\mathcal{B}_{i+1}$ is:

$$t_{\mathcal{B}_{i+1}}(n, m, W) = \tilde{O}(1)^{k(i+1)} \cdot m^{3/(3(i+1)+1)} \cdot n^{4(i+1)/k}.$$

Finally the query time of $\mathcal{B}_{i+1}$ is (where $\gamma^2 = \tilde{O}(1)^2 \cdot n^{2/k}$):

$$
\begin{aligned}
Q_{\mathcal{B}_{i+1}}(n, m, W) &= Q_{\mathcal{A}}(n, m, W) + \gamma^2 \cdot Q_{\mathcal{B}_i}(n, \ell(1 + 2\mu), nW) \\
&= \tilde{O}(1) \cdot n^{1/k} + \tilde{O}(1)^2 \cdot n^{2/k} \cdot \tilde{O}(1)^i \cdot n^{2i/k} = \tilde{O}(1)^{i+1} \cdot n^{2(i+1)/k}.
\end{aligned}
$$

and so the distance oracle $\mathcal{B}_{i+1}$ has the desired guarantees. ◀

Proof of Theorem 1. By Theorem 5, for any $i \geq 1, k > 1$, there is a fully dynamic distance oracle $\mathcal{B}_i$ of stretch $(4k)^i$ that w.h.p. achieves $\tilde{O}(1)^{ki} \cdot m^{1/i} \cdot n^{4i/k}$ amortized update time and $\tilde{O}(1)^i \cdot n^{2i/k}$ query time. Since $m \leq n^2$, by setting $i = \frac{4}{\rho}$ and $k = \frac{64}{\rho^2}$ the claim follows. ◀

In Theorem 5, we can set i to be a constant and set $k = O(\log \log n)^{1/i}$ to obtain another tradeoff, which is summarized in the following corollary.

▶ **Corollary 6.** *Given a weighted undirected graph $G = (V, E)$, there is a fully dynamic distance oracle with stretch $O(\log \log n)$ that w.h.p. achieves $n^{o(1)}$ query time and $\tilde{O}(n^\rho)$ amortized update time, for an arbitrarily small constant ρ.*

Finally note that we can also obtain similar tradeoffs as [25] where all three of stretch, amortized update time and query time are $n^{o(1)}$, by setting $k = O(\log \log n)^2$ and $i = O(\log \log n)$ in Theorem 5.

References

1 Amir Abboud, Karl Bringmann, and Nick Fischer. Stronger 3-sum lower bounds for approximate distance oracles via additive combinatorics. *CoRR*, abs/2211.07058, 2022.

2 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.

3 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS 2014)*, pages 434–443, 2014. doi:10.1109/FOCS.2014.53.

4 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proc. of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2017)*, pages 440–452, 2017. doi:10.1137/1.9781611974782.28.

5 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.

6 Ittai Abraham, Daniel Delling, Andrew V Goldberg, and Renato F Werneck. Hierarchical hub labelings for shortest paths. In *Algorithms–ESA 2012: 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings 20*, pages 24–35. Springer, 2012.

7 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.

8 Ruben Becker, Sebastian Forster, Andreas Karrenbauer, and Christoph Lenzen. Near-optimal approximate shortest paths and transshipment in distributed and streaming models. *arXiv preprint arXiv:1607.05127*, 2016.

9 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.

10 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.

11 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.

12 Aaron Bernstein, Maximilian Probst, and Christian Wulff-Nilsen. Decremental strongly-connected components and single-source reachability in near-linear time. In *Proc. of the 51st Annual ACM SIGACT Symposium on Theory of Computing (STOC 2019)*, pages 365–376, 2019. doi:10.1145/3313276.3316335.

13 Jan Van Den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *2022 IEEE 63rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1011–1022, 2022. doi:10.1109/FOCS54457.2022.00099.

14 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *FOCS*, pages 436–455. IEEE Computer Society, 2019.

15 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *FOCS*, pages 456–480. IEEE Computer Society, 2019.

16 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.

17 Shiri Chechik and Tianyi Zhang. Faster deterministic worst-case fully dynamic all-pairs shortest paths via decremental hop-restricted shortest paths. In Nikhil Bansal and Viswanath Nagarajan, editors, *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 87–99. SIAM, 2023.

18 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1135–1146. IEEE, 2020. `doi:10.1109/FOCS46700.2020.00109`.

19 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`.

20 Julia Chuzhoy and Ruimin Zhang. A new deterministic algorithm for fully dynamic all-pairs shortest paths. In *Proceedings of the 55th ACM Annual Symposium on Theory of Computing (STOC 2023)*, 2023. `arXiv:2304.09321`.

21 Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. *SIAM Journal on Computing*, 32(5):1338–1355, 2003.

22 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`.

23 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`.

24 Michal Dory, Sebastian Forster, Yasamin Nazari, and Tijn de Vos. New tradeoffs for decremental approximate all-pairs shortest paths. *arXiv preprint arXiv:2211.01152*, 2022.

25 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`.

26 Sebastian Forster, Gramoz Goranci, Yasamin Nazari, and Antonis Skarlatos. Bootstrapping dynamic distance oracles, 2023. `arXiv:2303.06102`.

27 Sebastian Forster, Yasamin Nazari, and Maximilian Probst Gutenberg. Deterministic incremental APSP with polylogarithmic update time and stretch. *CoRR*, abs/2211.04217, 2022. `doi:10.48550/arXiv.2211.04217`.

28 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proc. of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2562–2574, 2020. `doi:10.1137/1.9781611975994.156`.

29 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, Portland, Oregon, USA, January 5-7, 2014*, pages 1053–1072. SIAM, 2014.

30 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic approximate all-pairs shortest paths: Breaking the $o(mn)$ barrier and derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. Announced at FOCS 2013. `doi:10.1137/140957299`.

31 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`.

32 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC 2015)*, pages 21–30, 2015. `doi:10.1145/2746539.2746609`.

33 Monika Rauch Henzinger and Valerie King. Fully dynamic biconnectivity and transitive closure. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, pages 664–672. IEEE, 1995.

34 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

35 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.

36 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.

37 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, (STOC 2010)*, pages 603–610, 2010. doi:10.1145/1806689.1806772.

38 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM Journal on Computing*, 37(5):1455–1471, 2008. Announced at FOCS 2002. doi:10.1137/060650271.

39 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.

40 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.

41 Piotr Sankowski. Subquadratic algorithm for dynamic shortest distances. In *COCOON*, volume 3595 of *Lecture Notes in Computer Science*, pages 461–470. Springer, 2005.

42 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.

43 Mikkel Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of the 37th Annual ACM Symposium on Theory of Computing (STOC 2005)*, pages 112–119, 2005. doi:10.1145/1060590.1060607.

44 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM (JACM)*, 52(1):1–24, 2005.