Fully Dynamic All-Pairs Shortest Paths: Breaking the $O(n)$ Barrier

Ittai Abraham, Shiri Chechik, and Kunal Talwar

Microsoft Research Silicon Valley, Mountain View CA, USA
{ittaia,schechik,kunal}@microsoft.com

Abstract

A fully dynamic approximate distance oracle is a distance reporting data structure that supports dynamic insert edge and delete edge operations. In this paper we break a longstanding barrier in the design of fully dynamic all-pairs approximate distance oracles. All previous results for this model incurred an amortized cost of at least $\Omega(n)$ per operation. We present the first construction that provides constant stretch and $o(m)$ amortized update time. For graphs that are not too dense (where $|E| = O(|V|^{2-\delta})$ for some $\delta > 0$) we break the $O(n)$ barrier and provide the first construction with constant stretch and $o(n)$ amortized cost.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Shortest Paths, Dynamic Algorithms

Digital Object Identifier 10.4230/LIPIcs.APPROX-RANDOM.2014.1

1 Introduction

A *dynamic distance oracle* (DDO), also known as the *dynamic all pairs shortest path (APSP)*, is a data structure that is capable of efficiently processing an adversarial sequence of delete, insert and distance query operations. A *delete* operation deletes a single edge from the graph. An *insert* operation adds a single edge to the graph. A *query* operation receives a pair of nodes and returns a distance estimation. A *dynamic graph* is some initial graph G and a sequence of delete and insert operations. We say that a dynamic algorithm is only *decremental* if it handles only delete operations, only *incremental* if it handles only insert operations, and *fully dynamic* if it handles both. A dynamic algorithm is only *non-contracting* if it handles both delete and insert but only under the promise that the distances between any two points never get shorter.

A dynamic *approximate distance oracle* has *stretch* k if the returned distance estimate for every pair of nodes is at least the actual distance between them and at most k times their actual distance. A *single-source* dynamic distance oracle (SSDDO) has a fixed source s and all distance queries must involve the source s.

Even for single-source decremental dynamic distance oracles we do not know of any non-trivial bounds on worst-case operation costs. So it is natural to consider amortized costs as the next best measure. The *amortized cost* of a fully dynamic distance oracle is the average cost given a sequence of m operations taken over all possible adversarial sequences and all possible graphs with n vertices that start out with m edges. For a decremental only DDO we can measure the *total cost* as the total cost given an arbitrary sequence of m delete operations taken over all possible graphs with n vertices that start out with m edges. For a non-contracting DDO and a parameter m we define the *total cost* as the total cost given an arbitrary sequence of insert and delete operations such that the initial number of edges plus the total number of insert operations is at most m, taken over all possible graphs with n vertices.

17th Int'l Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX'14) / 18th Int'l Workshop on Randomization and Computation (RANDOM'14).
Editors: Klaus Jansen, José Rolim, Nikhil Devanur, and Cristopher Moore; pp. 1–16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we consider fully dynamic DDO for undirected unweighted graphs. For exact distances the best known bound is achieved by Henzinger *et al.* [12] who obtain amortized cost[1] of $O((n^{1.8+o(1)} + m^{1+o(1)})/m)$. When $2 + \epsilon$ approximate distances are allowed then Bernstein [5] obtained $\hat{O}(m)$ amortized cost[2]. For larger stretch Baswana, Khurana, and Sarkar [4] obtain, for any k, stretch $4k$ and $\hat{O}(n^{1+1/k})$ amortized update cost.

We note that all previous constructions suffer from an inherent amortized cost of $O(n)$ due to the potential need to run an exact single source shortest path algorithm (e.g. Dijkstra) on the graph G itself or on a sparser subgraph H. Indeed it seems that $O(n)$ amortized cost is a natural barrier for all existing approaches, even when allowing super constant stretch.

The main result of this paper is a new construction that circumvents this barrier and obtains an amortized cost of $o(m)$ and constant stretch. In fact our construction obtains $o(n)$ amortized update and constant stretch for any graph that is not super-dense (formally when $|E| = o(|V|^2)$).

▶ **Theorem 1.** *For any integer k, there exists a fully dynamic DO with amortized expected update time $\tilde{O}(m^{1/2} \cdot n^{1/k})$, query time $O(k^2\rho^2)$, and $2^{O(k\rho)}$ stretch, where $\rho = 1 + \lceil \frac{\log n^{1-1/k}}{\log (m/n^{1-1/k})} \rceil$.*

Note that $\rho \leq k$ and that if $m = n^{1+\epsilon}$ for any constant $\epsilon > 0$ then $\rho = O(1)$. For any graph with $m = n^{2-\delta}$, our algorithm breaks the $O(n)$ barrier with only $O(1)$ stretch.

For any large constant stretch and non-super dense graph our result dominates all previous fully dynamic results. At the extreme, for spare graphs ($|E| = O(|V|)$) and stretch $O(\log n)$ our amortized cost is $o(n^{\frac{1}{2}+\delta})$ for any $\delta > 0$, while [4] require amortized cost $\Omega(n)$.

Our result is obtained by combining two new ingredients. The first is an extremely time-efficient decremental only DDO scheme.

▶ **Theorem 2.** *For any positive integer k, one can maintain a decremental all-pairs shortest paths algorithm for a graph $G = (V, E)$ with stretch $2^{O(\rho k)}$ in total update time $\tilde{O}(mn^{1/k})$ and with $O(k\rho)$ time per query, where $\rho = (1 + \lceil \frac{\log n^{1-1/k}}{\log (m/n^{1-1/k})} \rceil)$.*

This improves on the best known decremental-only dynamic distance oracle of Bernstein and Roddity [6] (that get total update time $\hat{O}(n^{2+\frac{2}{1+stretch}})$) whenever m is $O(n^{2-\delta})$. In particular for sparse graphs with $m = n$, we get total update time $\tilde{O}(n^{1+\frac{1}{k}})$, whereas all previously known results had at least $\Omega(n^2)$ total update time.

The second is a transformation from a decremental only DDO to a fully dynamic DDO that avoids the $\Omega(n)$ worst case insert costs that are present in all previous results. Our reduction is fairly general, and can use other decremental only DDOs. We note that the idea of transforming a decremental algorithm into a fully dynamic algorithm was first suggested by Henzinger and King [11]. In this paper we use this extension in a non-trivial way.

1.1 Related Work

Even and Shiloach, in 1981, presented a decremental SSDDO for undirected, unweighted graphs with $O(n)$ amortized cost and $O(1)$ query time with stretch 1 (exact distances). A similar scheme was independently found by Dinitz [10]. Additional generalizations, optimizations and reductions were studied in [13, 14, 21].

[1] As usual, n (respectively, m) is the number of nodes (resp., edges) in the graph.
[2] Throughout, the $\tilde{O}$ notation suppresses polylogarithmic factors and the $\hat{O}$ notation suppresses $n^{O(1/\sqrt{\log n})}$ factors

Ausiello *et al.* [1] presented an incremental DDO for weighted directed graphs with amortized cost $O(n^3 \log n/m)$ and $O(1)$ query time. Henzinger and King showed a decremental DDO for weighted directed graphs with amortized cost $\tilde{O}((n^2/t)+n)$ and $O(t)$ query time.

King [13] presented a fully DDO for unweighted graphs with amortized cost $\tilde{O}(n^{2.5})$ and $O(1)$ query time. Demetrescu and Italiano [9] presented a fully DDO for directed weighted graphs with amortized cost $\tilde{O}(n^{2.5}\sqrt{S})$, where S is the possible number of different weight values in the graph.

Demetrescu and Italiano [8], in a major breakthrough devised a fully dynamic exact DDO for directed general graphs with non negative edge weights, with amortized cost $\tilde{O}(n^2)$. Thorup [15] extended the result to negative edge weights and [16] obtained worst case update time $\tilde{O}(n^{2.75})$.

The dynamic distance oracle problem was also studied when approximated distances are allowed. Much work was on the incremental-only and decremental-only e.g., [2, 3, 19, 6, 20]. Recently, Henzinger *et al.* [12] improved the amortized cost of decremental single source shortest paths (SSSP) for unweighted undirected graphs to $O((n^{1.8+o(1)}+m^{1+o(1)})/m)$.

Fully Dynamic Approximate DDOs for General Graphs: King [13] presented a fully DDO with amortized cost $\tilde{O}(n^2)$, $O(1)$ query time and $(1+\epsilon)$ stretch. Roditty and Zwick [19, 20] presented a fully DDO for any fixed $\epsilon, \delta > 0$ and every $t \leq m^{1/2-\delta}$, with expected amortized cost of $\tilde{O}(mn/t)$ and worst case query time of $O(t)$ and $(1+\epsilon)$ stretch. Note that as $t \leq m^{1/2-\delta}$, the best amortized cost that can be achieved using this algorithm is $\Omega(m^{1/2+\delta}n) > \Omega(m)$. Later, Bernstein [5] presented fully DDO with $O(\log\log\log n)$ query time, $2+\epsilon$ stretch and $\hat{O}(m)$ amortized cost, where $\hat{O}(f(n)) = f(n)n^{O(1/\sqrt{\log n})}$.

Recently, Baswana, Khurana, and Sarkar [4] presented a fully DDO for undirected unweighted graphs breaking the $O(n^2)$ barrier for dense graphs. For an integer parameter k, the construction of [4] has stretch $4k$, $\hat{O}(n^{1+1/k})$ amortized update cost, and $O(\log\log\log n)$ query time.

2 Preliminaries and Notation

Our algorithm is randomized and we assume an oblivious adversary (the sequence of insert and delete operations is determined before the random coins). For simplicity, we describe how to retrieve an estimation on the distances. Our algorithms can also be easily augmented to also report paths. For a graph H, let $V(H)$ be the nodes in H and let $E(H)$ be the edges of H. For an edge $(x,y) \in E(H)$, let $\omega(x,y,H)$ be the weight of the edge (x,y) in the graph H. For a graph H and nodes u and v, $\mathbf{dist}(u,v,H)$ is the distance between u and v in the graph H. Similarly, $\mathbf{dist}(u,S,H)$ for a graph H, node u and a set of nodes S is the minimum distance in H from u to a node in S. For a node v, distance ρ and graph H, let $B(u,\rho,H)$ be the set of nodes at distance at most ρ from u in H. When $H=G$, we sometimes omit it and write simply $\mathbf{dist}(u,v)$ instead of $\mathbf{dist}(u,v,G)$, or $B(u,\rho)$ instead of $B(u,\rho,G)$.

As alluded to earlier, we will often consider distance oracles that work for integer edge-weighted graphs under both edge insertion and deletion, but only under the promise that for every pair of vertices, the distance only increases over time. Thus if we ever insert an edge (u,v), its length will be at least as large as the current shortest path length. We call this the non-contracting dynamic setting. Dynamic distance oracles for this setting will be useful subroutine in our algorithms.

2.1 Existing Decremental SSSP Algorithms

2.1.1 The Decremental SSSP Algorithm of King [13]

Our algorithm uses the decremental SSSP algorithm of King [13] as an ingredient. The properties of King's algorithm are summarized in the following theorem.

▶ **Theorem 3.** *[13] Given a directed graph with positive integer edge weights, a source node s and a distance d, one can decrementally maintains a shortest path tree T from s up to distance d in total time $O(md)$. Moreover, given a node v, one can extract in $O(1)$ time* $\mathbf{dist}(v, s)$ *in case $v \in T$ or determine that $v \notin T$.*

King's algorithm starts by constructing a shortest path tree T rooted at s. Each time an edge $e = (x, y)$ is deleted, where x is in the same connected component as s in $T \setminus e$, an attempt is made to find a substitute edge to y that does not increase the distance from s to y. If such an edge is found then the recovery phase is over. Note that in this case the distances from s to y and to all nodes in y's subtree are unchanged. In case no such edge found, the best edge is chosen, i.e., the edge that connect y on the shortest path possible. The process is continued recursively on all y's children. The crucial property of this algorithm is that it explores the edges of a node v only when the distance from s to v increases. This gives a total running time of $O(md)$ as the distance from s to a node v may increase at most d times before exceeding d.

This analysis of the decremental SSSP algorithm of King [13] works in the decremental only setting but breaks down if we allow edge insertions. Indeed the analysis relies on the fact that the distance from a node to s can change at most d times before exceeding d, which is not true if we allow arbitrary edge insertions. However, if the edges have integer weights and insertions are guaranteed to ensure that the distances do not decrease over time, it is easy to verify that the analysis of King [13] works as it is. Thus Theorem 3 also holds for the non-contracting dynamic setting.

2.1.2 The Decremental Algorithm of Roditty and Zwick [20]

Another ingredient in our algorithm is the approximate decremental APSP algorithm of Roditty and Zwick [20]. Roditty and Zwick [20] showed how to construct a decremental all pairs shortest path data structure DO_{RZ} up to depth d for a given graph H and integer k such that one can answer any distance query in $O(k)$ time within stretch $2k - 1$, and the total update time is $\tilde{O}(mn^{1/k}d)$. More precisely, the dynamic data structure of Roditty and Zwick can be easily tweaked to either return an estimate within $2k - 1$ stretch or determine that $\mathbf{dist}(s, t, H) > d$, and return infinity in this case. In addition, as in the King's algorithm [13], the entire analysis relies on the fact that distances never get shorter and thus also works in the non-contracting setting.

3 Techniques

In this section we outline the high level ideas of our construction for the fully dynamic APSP algorithm. Our algorithm consists of two main parts. The first part is a new decremental APSP algorithm with total update time that can get arbitrarily close to $\tilde{O}(m)$, while paying in the stretch. More precisely, we show for any positive integer k, a decremental APSP algorithm with total update time of $\tilde{O}(mn^{1/k})$ and with stretch $2^{O(k^2)}$ (and $2^{O(k)}$ when $m = n^{1+\epsilon}$). The second part takes the decremental APSP and augments it to accommodate insertion operations. We provide a general framework for obtaining fully dynamic APSP which can be potentially used with other decremental APSP.

Usually, the hard part in decremental APSP algorithms is in handling long distances. We introduce a new approach that allows efficient handling of all distances. Loosely speaking, we maintain k non-contracting dynamic graphs G_i. The graph G_0 is simply G and we use [20] upto depth $n^{1/k}$ on G_0 to answer distances upto to $n^{1/k}$. We then construct a dynamic non-contracting graph G_1, that dynamically maintains the property that every two nodes whose distance in G is $n^{1/k}$ have a 2-hop path between them in G_1. This implies that distances upto $n^{2/k}$ in G have a path of length $O(n^{1/k})$ in G_1. Hence we use [20] upto depth $n^{1/k}$ on G_1 to answer distances upto to $n^{1/k}$ in G_1 which correspond to distances in G between $n^{1/k}$ to $n^{2/k}$.

In the same way, we iteratively construct a dynamic non-contracting graph G_i, that dynamically maintains the property that every two nodes whose distance in G is $n^{i/k}$ have a 2-hop path between them in G_i. This implies that distances upto $n^{i/k}$ in G have a path of length $O(n^{1/k})$ in G_i. Hence we use [20] upto depth $n^{1/k}$ on G_i to answer distances upto to $n^{1/k}$ in G_i which correspond to distances in G between $n^{i/k}$ to $n^{(i+1)/k}$.

The core difficulty is dynamically maintaining the property that every two nodes whose distance in G is $n^{i/k}$ have a 2-hop path between them in G_i while keeping G_i sparse. For example, consider two nodes that start by having two $n^{i/k}$ long paths between them in G and the 2-hop path maintained in G_i is induced by the first path. Due to edge removals in G of the first path, we now have to discover and maintain in G_i a new 2-hop path that is induced by the second $n^{i/k}$ long path between them in G. Observe that this implies that deleting edges in G may force insert operations of new edges in G_i. In order to control the total cost of this addition we must do two things (1) guarantee that these edges insertions are non-contracting (2) bound the total number of edge insertions.

Suppose we want to maintain distances in G_1 between every two nodes whose distance in G is $\alpha = n^{1/k}$. For simplicity, first assume $k = 2$. Our solution is roughly as follows: we sample $\tilde{O}(\sqrt{n})$ *pivots* A and build and maintain a decremental tree $T(u)$ of radius 3α around each pivot u. We build in G_1 an edge u, v of length 1 between any pivot u and $v \in T(u)$ (in this part edges may be dynamically deleted but none are dynamically inserted). Consider a node w. There are two cases. If $\mathbf{dist}(w, A) \leq 2\alpha$ then there is a nearby pivot that provides the desired 2-hop property for pairs involving w. Otherwise with high probability $|B(w, 2\alpha)| < \sqrt{n}$ and hence for all $x \in B(w, \alpha)$, we have $|B(x, \alpha)| < \sqrt{n}$. So in this case, we *activate* w by building and maintaining a decremental tree $T(w)$ of radius α around w. The main observation is that due to the sparseness condition for activation, for any node y, in the entire decremental sequence on G, y will belong to at most $\sqrt{n}$ trees induced from activated nodes. This implies that we can bound the total number of edges added to G_1 by $\tilde{O}(n^{1+1/2})$. We still have the problem of guaranteeing that these edges insertions are non-contracting in G_1. The solution is to add in G_1 an edge w, v of length 2 between any activated node w and $v \in T(w)$. Just before activating w it must be that (w, v) have a 2-hop path where each hop has length 1 in G_1, and hence choosing length 2 is adequate.

More generally, we maintain k dynamic non-contracting graphs. For each G_ℓ we maintain k subsets $A_1, \ldots, A_k$. The set A_i is of size $\tilde{O}(n^{i/k})$. For each $w \in A_i$, we *activate* w and build a decremental tree of radius $3^{k-i}\alpha$ around w roughly when w is "far enough" from all A_j for all $j < i$. This allows us to bound the total number of edges added to G_ℓ for the set A_i by $\tilde{O}(n^{1+1/k})$. To maintain the non-contracting property we add in G_ℓ an edge w, v of length 2^i between any activated node $w \in A_i$ and $v \in T(w)$. See section 4 for details.

Our approach for the second part is to take our decremental DDO and augment it to accommodate insertion operations. We provide a general framework for obtaining fully dynamic DO which can be potentially used with other decremental DDOs. At a high level, our approach is to maintain two sub-components: (1) for delete operations we maintain a decremental DDO (2) for insert operations we use a *sketch-graph* data structure that maintains an approximate distance oracle over all the newly inserted edges. Once sub-component (2) becomes too large we re-build the two components from scratch and hence obtaining good amortized guarantees. The key advantage is that our costs are proportional to the number of newly inserted edges, not the total number of edges in the graph. During a query we use both components and combine their results to find a low stretch estimation. We also need to update the sketch-graph appropriately during each delete operation. In order to get a smaller update time we exploit some additional properties in the construction of Roditty and Zwick [20].

4 New Decremental Shortest Paths

4.1 The Main Building Block

4.1.1 Properties

In this section we present an algorithm that takes as input three integers k, α, and m_r and a dynamic graph G_r where the graph G_r is guaranteed to satisfy the following properties.

(a) All edge weights in G_r are in $\{1, 2, \ldots, 2^{k-1}\}$ (both the initial edges and the edges that may be dynamically added).

(b) The graph G_r has the property that distances never decrease over time (but it may happen that edges are both added and removed over time).

(c) The initial number of edges in G_r plus the total number of edge insertion operation on G_r is at most m_r.

The algorithm outputs a non-contracting dynamic data structure that produces a dynamic graph G_{r+1} with the following properties:

1. The edge weights in G_{r+1} are in $\{1, 2, \ldots, 2^{k-1}\}$ (both the initial edges and the edges that may be dynamically added).
2. Every two nodes at distance at most α in G_r have a two hop path between them in G_{r+1} of length at most 2^{k-1}.
3. At any point, the non-contracting data structure maintains:

$$\frac{\mathbf{dist}(u, v, G_r)}{(3^{k-1}\alpha)} \leq \mathbf{dist}(u, v, G_{r+1}) \leq 2^{k-1} \cdot \left(\frac{\mathbf{dist}(s, t, G_r)}{(\alpha - 2^{k-1})} + 1 \right)$$

4. The dynamic non-contracting data structure for G_{r+1} incurs a total cost of $\tilde{O}(k3^k n^{1/k} \alpha m_r)$.
5. The non-contracting data structure maintains that the graph G_{r+1} has the property that distances never decrease over time (but it may happen that edges are both added and removed over time).
6. The initial number of edges in G_{r+1} plus the total number of edge insertion operation on G_{r+1} is at most $\tilde{O}(kn^{1+1/k})$.

4.1.2 Constructing the Decremental Distance Oracle

We define a sequence of sets $A_1, \ldots, A_k$ as follows: The set A_k is simply V. For $1 < i \leq k-1$ set A_i is a sample of V independently at random with probability $c \log n / n^{1-i/k}$ (for some small constant c). Thus the set A_i contains in expectation $\tilde{O}(n^{i/k})$ nodes.

For each $v \in A_i$ we will define a *condition* under which we *activate* v and hereafter build and maintain a decremental SSSP tree $T(v)$ for depth $3^{k-i}\alpha$ on graph G_r. Loosely speaking, for every node v in A_i for every $i > 1$, the algorithm constructs a decremental SSSP tree $T(v)$ once all the distances $\mathbf{dist}(v, A_j, G_r)$ for every $1 \leq j < i$ are "sufficiently" large.

We say that a node $v \in A_i$ is *i-active* if the tree $T(v)$ was already constructed, otherwise it is *i-inactive*. All nodes in A_1 are initially active, namely, for every node v in A_1 maintain a decremental SSSP tree $T(v)$ up to depth $3^{k-1}\alpha$ on the graph G_r. For every index $1 \leq i \leq k-1$ and a node u, the algorithm maintains in a heap H_i^u the distances $\mathbf{dist}(v, u, G_r)$ for every $v \in A_i$ such that v is i-active and $u \in T(v)$. Let $H_i^u.min$ be the minimum in the heap. For every $i > 1$ and i-inactive node $v \in A_i$: if the condition $H_j^u.min > (3^{k-j} - 3^{k-i})\alpha$ for every $j < i$ holds, then the algorithm constructs and maintains a decremental SSSP tree $T(v)$ up to depth $3^{k-i}\alpha$ on the graph G_r. Each time the distance $\mathbf{dist}(u, v, T(v))$ for some $u \in T(v)$ changes, the algorithm updates the relevant heap and checks if u should be activated.

The graph G_{r+1} is constructed and maintained as follows. For every node $v \in A_i$ and $u \in T(v)$, add an edge between u and v of weight 2^{i-1}. Once a node u is removed from $T(v)$, remove the corresponding edge from G_{r+1}. Similarly, once v is i-activated and $T(v)$ for $v \in A_i$ is constructed add all edges (u, v) for every $u \in T(v)$ with weight 2^{i-1}. In each change of G_r, the algorithm first updates all the trees $T(v)$, then adds the relevant edges to G_{r+1}, and then removes the relevant edges from G_{r+1}. Adding the edges before the deletions ensures that distances are never decrease in G_{r+1}. This concludes the construction.

Observe that one change in G_r may lead multiple changes in G_{r+1} (a removal of an edge may increase some distances in trees $T(v)$ that may lead to the construction of new trees).

4.1.3 Analysis

The next claim bounds the number of trees $T(v)$ a node u may belong to in the entire run of the algorithm.

▶ Claim 4. W.h.p. every node u belongs to at most $\tilde{O}(k \cdot n^{1/k})$ trees $T(v)$ in the entire run of the algorithm.

Proof. There are $\tilde{O}(n^{1/k})$ nodes in A_1 in expectation, therefore u may belong to $\tilde{O}(n^{1/k})$ trees $T(v)$ such that $v \in A_1$.

We claim that u belongs to a tree $T(v)$ for $v \in A_i$ and $i > 1$, only if $\mathbf{dist}(u, A_{i-1}, G_r) > 3^{k-i}\alpha$. To see this, assume that $\mathbf{dist}(u, A_{i-1}, G_r) \leq 3^{k-i}\alpha$. Assume, towards contradiction, that $u \in T(v)$ for some $v \in A_i$. Recall that the depth of $T(v)$ is $3^{k-i}\alpha$. We get that $\mathbf{dist}(v, A_{i-1}, G_r) \leq \mathbf{dist}(v, u, G_r) + \mathbf{dist}(u, A_{i-1}, G_r) \leq (3^{k-i} + 3^{k-i})\alpha \leq 2 \cdot 3^{k-i}\alpha$. Hence there is a node $w \in A_{i-1}$ such that $\mathbf{dist}(v, w, G_r) \leq 2 \cdot 3^{k-i}\alpha$.

We need to consider two cases. The first case is when w is $(i-1)$-active and the second case is when w is $(i-1)$-inactive. Consider the case where w is $(i-1)$-active. The tree $T(w)$ is constructed up to depth $3^{k-(i-1)}\alpha = 3^{k-i+1}\alpha$. As $\mathbf{dist}(v, w, G_r) \leq (2 \cdot 3^{k-i})\alpha < 3^{k-i+1}\alpha$ we have $v \in T(w)$. Note that in this case,

$$\begin{aligned} H_{i-1}^v.min &\leq \mathbf{dist}(v, w, G_r) \\ &\leq 2 \cdot 3^{k-i}\alpha \\ &= (3^{k-i+1} - 3^{k-i})\alpha \\ &= (3^{k-(i-1)} - 3^{k-i})\alpha. \end{aligned}$$

Hence, by definition, the tree $T(v)$ was not supposed to be constructed yet, which is a contradiction.

Consider now the second case, where w is $(i-1)$-inactive. The node w is $(i-1)$-inactive only if there is a j-active node $z \in A_j$ for some $j < i-1$ such that $w \in T(z)$ and $\mathbf{dist}(w,z,G_r) \leq (3^{k-j} - 3^{k-(i-1)})\alpha$.

It follows that

$$\begin{aligned}\mathbf{dist}(v,z,G_r) &\leq \mathbf{dist}(v,w,G_r) + \mathbf{dist}(w,z,G_r)\\ &\leq (2\cdot 3^{k-i})\alpha + (3^{k-j} - 3^{k-(i-1)})\alpha\\ &= (2\cdot 3^{k-i} + 3^{k-j} - 3^{k-i+1})\alpha\\ &= (2\cdot 3^{k-i} + 3^{k-j} - 3\cdot 3^{k-i})\alpha\\ &= (3^{k-j} - 3^{k-i})\alpha.\end{aligned}$$

Note that $v \in T(z)$. It follows that, $H^v_j.min \leq \mathbf{dist}(v,z,G_r) \leq (3^{k-j} - 3^{k-i})\alpha$. Hence, by definition, the tree $T(v)$ was not supposed to be constructed yet, which is a contradiction.

It follows that, $\mathbf{dist}(u, A_{i-1}, G_r) > 3^{k-i}\alpha$. Hence, by applying Chernoff's bound [3] we get that w.h.p. $|B(u, 3^{k-i}\alpha)| \leq n^{1-(i-1)/k}$. The set A_i contains every node independently at random with probability $c\log n/n^{1-i/k}$. Hence in expectation we have $|B(u,3^{k-i}\alpha) \cap A_i| \leq c\log n/n^{1-i/k} \cdot n^{1-(i-1)/k} = \tilde{O}(n^{1/k})$, as required. ◀

We next show that G_{r+1} satisfies the desired properties.

▶ **Lemma 5.** *Suppose that the input dynamic graph G_r satisfies properties (a)-(c). Then the graph G_{r+1} satisfies properties 1–6.*

Proof. It is not hard to verify property (1), that all edges in G_{r+1} are of weight in $\{1, 2, \ldots, 2^{k-1}\}$.

We now prove property (2), that is, for every two nodes u and v such that $\mathbf{dist}(u,v,G_r) \leq \alpha$, there is a path between u and v in G_{r+1} of at most two hop and of length at most 2^{k-1}. To see this, recall that $A_k = V$. If v is k-active then the tree $T(v)$ is constructed up to depth α and as $\mathbf{dist}(u,v,G_r) \leq \alpha$ we have $u \in T(v)$. Therefore, by construction the graph G_{r+1} contains the edge (u,v) of weight 2^{k-1}, as required. So assume now v is k-inactive. By construction if v is k-inactive then there is an index $j < k$ and a j-active node $w \in A_j$ such that $v \in T(w)$ and $\mathbf{dist}(v,w,G_r) \leq (3^{k-j} - 1)\alpha$. Note that $\mathbf{dist}(u,w,G_r) \leq \mathbf{dist}(u,v,G_r) + \mathbf{dist}(v,w,G_r) \leq \alpha + (3^{k-j}-1)\alpha \leq 3^{k-j}\alpha$. As the tree $T(w)$ contains all nodes at distance in G_r at most $3^{k-j}\alpha$ from w, we get that $u \in T(w)$. By construction, the graph G_{r+1} contains both edges (w,v) and (w,u), both of weight 2^{j-1}. Hence G_{r+1} has a two hop path between u and v of length $2^j \leq 2^{k-1}$, as required.

To see property (3) consider two nodes u and v. We need to show $\mathbf{dist}(u,v,G_r)/(3^{k-1}\alpha) \leq \mathbf{dist}(u,v,G_{r+1}) \leq 2^{k-1}(\mathbf{dist}(s,t,G_r)/\alpha + 1)$.

Let us first show the first inequality, that is, $\mathbf{dist}(u,v,G_r)/(3^{k-1}\alpha) \leq \mathbf{dist}(u,v,G_{r+1})$. Let $P(u,v,G_{r+1})$ be the shortest path between u and v in G_{r+1}. Let (x,y) be an edge on $P(u,v,G_{r+1})$. Recall that by construction as (x,y) is an edge in G_{r+1} then there is an index i such that $x \in A_i$, and $y \in T(x)$, or $y \in A_i$, and $x \in T(y)$. Assume w.l.o.g. that $x \in A_i$ and $y \in T(x)$.

[3] Note that the claim needs to hold w.h.p. for every considered graph during the entire running of the algorithm, note however that as there are at most $m_r \leq n^2$ deletions and therefore at most n^2 considered graphs. Hence by setting the constant c to be large enough we can show that the claim holds w.h.p. for every considered graph.

Note that $\mathbf{dist}(x, y, G_{r+1}) = 2^{i-1}$ and that $\mathbf{dist}(x, y, G_r) \le 3^{k-i}\alpha$. We get

$$\begin{aligned}\mathbf{dist}(x, y, G_r) &\le 3^{k-i}\alpha \\ &= 3^{k-i}\alpha/2^{i-1} \cdot 2^{i-1} \\ &= 3^{k-i}\alpha/2^{i-1} \cdot \mathbf{dist}(x, y, G_{r+1}) \\ &\le 3^{k-1}\alpha \cdot \mathbf{dist}(x, y, G_{r+1}).\end{aligned}$$

Hence, $\mathbf{dist}(x, y, G_r)/(3^{k-1}\alpha) \le \mathbf{dist}(x, y, G_{r+1})$. It follows that

$$\begin{aligned}\mathbf{dist}(u, v, G_{r+1}) &= \sum_{(x,y)\in P(u,v,G_{r+1})} \omega(x, y, G_{r+1}) \\ &\ge \sum_{(x,y)\in P(u,v,G_{r+1})} \mathbf{dist}(x, y, G_r)/(3^{k-1}\alpha) \\ &\ge \mathbf{dist}(u, v, G_r)/(3^{k-1}\alpha),\end{aligned}$$

as required.

We now turn to prove the second inequality, namely, $\mathbf{dist}(u, v, G_{r+1}) \le 2^{k-1}(\mathbf{dist}(u, v, G_r)/(\alpha-2^{k-1})+1)$. Consider the shortest path $P(u, v, G_r)$ between u and v in G_r. Let $x_0 = u$ and let x_i be the furthest away node on $P(u, v, G_r)$ from x_{i-1} such that $\mathbf{dist}(u, v, G_r) \le \alpha$. As the maximum edge weight in G_r is 2^{k-1} it is not hard to verify that $\mathbf{dist}(x_{i-1}, x_i, G_r) < \alpha-2^{k-1}$. In addition, by property (2) we have that x_{i-1} and x_i are connected by a two hop path of length at most 2^{k-1}.

Hence, u and v are connected by a path in G_{r+1} of length at most $\lceil \mathbf{dist}(u, v, G_r)/(\alpha - 2^{k-1})\rceil \cdot 2^{k-1}$. Therefore,

$$\begin{aligned}\mathbf{dist}(u, v, G_{r+1}) &\le \lceil \mathbf{dist}(u, v, G_r)/(\alpha - 2^{k-1})\rceil \cdot 2^{k-1} \\ &\le (\mathbf{dist}(u, v, G_r)/(\alpha - 2^{k-1}) + 1) \cdot 2^{k-1}.\end{aligned}$$

We get that, $\mathbf{dist}(u, v, G_r)/(3^{k-1}\alpha) \le \mathbf{dist}(u, v, G_{r+1}) \le (\mathbf{dist}(s, t, G_r)/(\alpha-2^{k-1})+1)\cdot 2^{k-1}$.

Let us turn to prove property (4). By claim 4 all nodes belong to $\tilde{O}(kn^{1/k})$ trees. Each tree is maintained up to depth $3^{k-1}\alpha$ (or less). Let $\deg_r(u)$ be the degree of u in G_r. Consider a tree $T(v)$, the total time for maintaining $T(v)$ is $\sum_{u\in T(v)} 3^{k-1}\alpha \deg_r(u)$. Thus for all trees $\sum_{v\in V, u\in T(v)} 3^{k-1}\alpha \deg_r(u) = \tilde{O}(k3^{k-1}n^{1/k}\alpha m_r)$.

We now turn to property (5), that is, edges may be added to G_{r+1} but distances are only increasing. We need to show that when we add an edge (u, v) with weight $\omega(u, v)$ then $\mathbf{dist}(u, v, G_{r+1}) \le \omega(u, v)$. Recall that an edge (u, v) is added to G_{r+1} when the tree $T(v)$ for some $v \in A_i$ is constructed. The tree $T(v)$ is constructed once the distances $H^u_j.min > (3^{k-j} - 3^{k-i})\alpha$ for every $j < i$. Namely, just before this change we had $H^u_j.min \le (3^{k-j} - 3^{k-i})\alpha$ for some $j < i$. It follows that there exists a node $w \in A_j$ such that $\mathbf{dist}(v, w, G_r) = (3^{k-j} - 3^{k-i})\alpha$. Note that $\mathbf{dist}(v, u, G_r) \le 3^{k-i}\alpha$. Hence $\mathbf{dist}(u, w, G_r) \le \mathbf{dist}(u, v, G_r) + \mathbf{dist}(v, w, G_r) \le 3^{k-i}\alpha + (3^{k-j} - 3^{k-i})\alpha = 3^{k-j}\alpha$. It follows that $u \in T(w)$. We get that the edges (u, w) and (w, v) existed in G_{r+1} just before this change. In addition, $\omega(u, w) = 2^{j-1} \le 2^{i-1}$ and $\omega(w, v) = 2^{j-1} \le 2^{i-1}$. The algorithm adds an edge (u, v) with $\omega(u, v) = 2^i$. It is not hard to see that the distance (u, v) did not decrease by adding the edge (u, v).

Property (6), i.e., that the total number of edges in G_{r+1} is $\tilde{O}(kn^{1+1/k})$ easily follows from the fact that every node belong to $\tilde{O}(kn^{1/k})$ trees. ◀

4.2 The Main Decremental Construction

In this section we show our decremental algorithm.

Construction. Let us start with describing the construction. Set $G_0 = G$. Construct G_1 using the data structure from the previous section on G_0 with $\alpha_1 = n^{1/k}$ and k. For every $1 < i \leq \rho$, construct and maintain G_i using the data structure from the previous section on G_{i-1} with $\alpha_i = 2^k \lceil \frac{m}{n^{1-1/k}} \rceil$ and k. (The reason for considering different α's for the different levels if to achieve a better stretch for not super sparse graphs). For every $0 \leq i \leq \rho$ construct the data structure DO^i_{RZ} of Roditty and Zwick [20] up to distances α_i on the graph G_i. The concludes the construction.

Answering a Distance Query. The algorithm for answering distance query given two nodes s and t is as follows. Find the first i such that DO^i_{RZ} returns an estimation on $\mathbf{dist}(s,t)$, i.e., the first i such that s and t are at distance at most $2\alpha_i$ from one another in G_i. Let $\mu_0 = 1$ and $\mu_i = \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)}$ for $i > 0$. Return $\mu_i DO^i_{RZ}(s,t)$.

It is not hard to see that the query time is $O(k \cdot \rho)$ as the algorithm invokes the Roditty and Zwick [20] query algorithm at most ρ times, where each distance query of Roditty and Zwick [20] takes $O(k)$ time.

Analysis. The next auxiliary claim shows that every two nodes whose distance in G is at most $\alpha_1(\alpha_2/2^k)^{i-1}$ are connected by a path of at most two hop in G_i.

▶ Claim 6. Every two nodes u and v such that $\mathbf{dist}(u,v,G) \leq \alpha_1(\alpha_2/2^k)^{i-1}$ are connected in G_i by a path of at most 2-hop.

Proof. We prove it by induction on i. For $i = 1$, the proof follows by property (2) on G_1. Assume correctness for every j such that $1 \leq j < i$ and consider i. Consider two nodes u and v such that $\mathbf{dist}(u,v,G) \leq \alpha_1(\alpha_2/2^k)^{i-1}$.

Consider the shortest path $P(u,v)$ between u and v in G. Let $u = x_0$ and $x_r = v$ for $r = \lceil \mathbf{dist}(u,v,G)/(\alpha_1(\alpha_2/2^k)^{i-2}) \rceil$. For $1 \leq \ell < r$, let x_ℓ be the next node on the path $P(u,v)$ (closer to v) such that $\mathbf{dist}(x_{\ell-1}, x_\ell) = \alpha_1(\alpha_2/2^k)^{i-2}$.

By the induction hypothesis x_ℓ and $x_{\ell+1}$ are connected by a two-hop path in G_{i-1}. As the weights in G_{i-1} are at most 2^{k-1}. We get that the length of the path between u and v in G_{i-1} is at most

$$\begin{aligned}
\mathbf{dist}(u,v,G_{i-1}) &\leq 2^k \lceil \mathbf{dist}(u,v,G)/(\alpha_1(\alpha_2/2^k)^{i-2}) \rceil \\
&\leq 2^k \lceil \alpha_1(\alpha_2/2^k)^{i-1}/(\alpha_1(\alpha_2/2^k)^{i-2}) \rceil \\
&\leq 2^k \lceil 2^k m/n^{1-1/k}/2^k \rceil \\
&= 2^k m/n^{1-1/k} \\
&= \alpha_2
\end{aligned}$$

By property (2) we have that u and v are connected by a two-hop path in G_i. ◀

▶ Claim 7. Let i be the minimal index such that DO^i_{RZ} returns an estimation on $\mathbf{dist}(s,t)$. If $i > 0$, then $\mathbf{dist}(s,t,G) \geq \alpha_1(\alpha_2/2^k)^{i-1}$.

Proof. It is not hard to see by Claim 6 that if $\mathbf{dist}(s,t,G) \leq \alpha_1(\alpha_2/2^k)^{i-1}$ then there is $2\alpha_2/2^k$ hop paths from s to t in G_{i-1}, namely, $\mathbf{dist}(s,t,G_{i-1}) \leq \alpha_2$. Therefore by construction DO^{i-1}_{RZ} is supposed to return an estimate on $\mathbf{dist}(s,t)$. It follows that $\mathbf{dist}(s,t,G) \geq \alpha_1(\alpha_2/2^k)^{i-1}$. ◀

The next lemma bounds the stretch of the algorithm.

▶ **Lemma 8.** *The distance* $\hat{\mathbf{dist}}(s,t)$ *returned by the algorithm satisfies* $\mathbf{dist}(s,t) \le \hat{\mathbf{dist}}(s,t) \le k(2k-1)6^{\rho k}\mathbf{dist}(s,t)$.

Proof. Let i be the minimal index such that DO^i_{RZ} returns an estimation on $\mathbf{dist}(s,t)$. If $i = 0$, then $\hat{\mathbf{dist}}(s,t) = DO^0_{RZ}(s,t)$. Note that $\mathbf{dist}(s,t,G) \le DO^0_{RZ}(s,t) \le (2k-1)\mathbf{dist}(s,t,G)$ and we are done. So assume $i > 0$.

Let us first prove the first direction, that is, $\mathbf{dist}(s,t) \le \hat{\mathbf{dist}}(s,t)$. By applying property (3) recursively we have $\mathbf{dist}(s,t,G_i) \ge \mathbf{dist}(s,t,G_0)/(3^{i\cdot(k-1)}\alpha_1\alpha_2^{i-1})$. We get,

$$\begin{aligned}
\hat{\mathbf{dist}}(s,t) &= \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot DO^i_{RZ}(s,t)\\
&\ge \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot \mathbf{dist}(s,t,G_i)\\
&\ge \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot \mathbf{dist}(u,v,G_0)/(3^{i\cdot(k-1)}\alpha_1\alpha_2^{i-1})\\
&= \mathbf{dist}(u,v,G_0).
\end{aligned}$$

We are left to show the other direction, that is, $\hat{\mathbf{dist}}(s,t) \le k(2k-1)6^{\rho k}\mathbf{dist}(s,t,G_0)$.

Recall that $\alpha_1, \alpha_2 > 2^k$. By property (3) and straightforward calculations we have the following.

$$\begin{aligned}
\mathbf{dist}(s,t,G_i) &\le 2^{k-1}(\mathbf{dist}(s,t,G_{i-1})/(\alpha_2 - 2^{k-1}) + 1)\\
&= 2^{k-1}\mathbf{dist}(s,t,G_{i-1})/(\alpha_2 - 2^{k-1}) + 2^{k-1}\\
&\le 2^{i(k-1)}\mathbf{dist}(s,t,G_0)/((\alpha_2 - 2^{k-1})^{i-1} \cdot (\alpha_1 - 2^{k-1})) + i \cdot 2^{k-1}\\
&\le 2^{i(k-1)}\mathbf{dist}(s,t,G_0)/((\alpha_2/2)^{i-1} \cdot (\alpha_1/2)) + i \cdot 2^{k-1}\\
&\le 2^{ik}\mathbf{dist}(s,t,G_0)/((\alpha_2)^{i-1} \cdot (\alpha_1)) + i \cdot 2^{k-1}.
\end{aligned}$$

By Claim 7 we have $\mathbf{dist}(s,t,G) \ge \alpha_1(\alpha_2/2^k)^{i-1}$. Hence,

$$\begin{aligned}
\hat{\mathbf{dist}}(s,t) &= \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot DO^i_{RZ}(s,t)\\
&\le \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot (2k-1)\mathbf{dist}(s,t,G_i)\\
&\le \alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot (2k-1)(2^{ik}\mathbf{dist}(s,t,G_0)/((\alpha_2)^{i-1} \cdot (\alpha_1)) + i \cdot 2^{k-1})\\
&\le (2k-1)6^{ik}\mathbf{dist}(s,t,G_0) + i\alpha_1 \cdot \alpha_2^{i-1} \cdot 3^{i(k-1)} \cdot (2k-1) \cdot 2^{k-1}\\
&\le (2k-1)6^{ik}\mathbf{dist}(s,t,G_0) + i \cdot 3^{i(k-1)} \cdot (2k-1) \cdot 2^{k-1} \cdot \mathbf{dist}(s,t,G_0)\\
&\le k(2k-1)6^{ik}\mathbf{dist}(s,t,G_0).
\end{aligned}$$

as required. ◀

We conclude the following.

▶ **Theorem 9.** *For any positive integer* k*, one can maintain a decremental all-pairs shortest paths algorithm for a given graph* $G = (V,E)$ *with stretch* $2^{O(\rho k)}$ *in total update time* $\tilde{O}(mn^{1/k})$ *and with* $O(k\rho)$ *time per query, where* $\rho = (1 + \lceil \frac{\log n^{1-1/k}}{\log(m/n^{1-1/k})} \rceil)$

5 Fully Dynamic Approximate All-Pairs Shortest Paths

Our fully dynamic data structure will maintain the decremental data structure from the last section, along with an auxiliary data structure to “remember” the insertions. To prevent this second data structure from becoming too large, we will periodically rebuilid the complete data structure (i.e. both the decremental part and the auxiliary part). Throughout, $G = (V, E)$ is the updated graph, namely, the graph after all the updates that have occurred so far. Let D be the set of deletions and U be the set of insertions since the data structure was last rebuilt. Let $\hat{G}$ be the graph when the data structure was last rebuilt. Let G_D be the graph obtained by deleting the set of edges D from $\hat{G}$.

The main idea of our fully dynamic algorithm is the following: We maintain two data structures, the first is a decremental distance oracle **Dec** and the second is an *auxiliary distance oracle* M whose goal is to handle edge insertions. More specifically, the decremental distance oracle is capable of answering approximate distance queries in the graph G_D, and the sketch distance oracle is capable of answering approximate distance queries in the graph G, but only between nodes in $V(U)$, where $V(U)$ is the set of all nodes incident to some edge in U.

In addition, for every node $v \in V$, we ensure a simple access to some “close” nodes in $V(U)$, called the *pivots* of v. To answer distance queries between a pair of nodes s and t, the algorithm computes for every two pivots $p(s)$ of s and $p(t)$ of t the distance $\mathbf{Dec}(s, p(s)) + M(p(s), p(t)) + \mathbf{Dec}(p(t), t)$ and returns the minimum over all pairs of pivots, where $\mathbf{Dec}(u, v)$ is the distance returned by invoking the query algorithm of **Dec** on (u, v). Let $P(s, t, G)$ be the shortest path from s to t in G. If the path $P(s, t, G)$ does not contain edges that were inserted since the data structure was last constructed then $\mathbf{Dec}(s, t)$ gives a good estimation on $\mathbf{dist}(s, t, G)$. Otherwise, we show that there must be nodes $p(s), p(t) \in V(U)$ such that $\mathbf{Dec}(s, p(s)) + M(p(s), p(t)) + \mathbf{Dec}(p(t), t)$ is a good estimation on $\mathbf{dist}(s, t, G)$.

When the sketch distance oracle becomes too “large” we simply construct the data structure from scratch.

An important advantage of our scheme is that it is quite general. One can plug in it any decremental and sketch distance oracles.

5.1 Fully Dynamic APSP Beyond $O(n)$

We show a fully dynamic APSP algorithm with amortized update time $\tilde{O}(m^{1/2} \cdot n^{1/k})$ with $2^{O(k\rho)}$ stretch. Note that $\rho \leq k$ and that if $m = n^{1+\epsilon}$ for constant ϵ then $\rho = O(1)$. For any graph with $m = n^{2-\delta}$ for some constant $\delta > 0$, our algorithm can break the $O(n)$ barrier with only $O(1)$ stretch.

We use the decremental distance oracle **Dec** from Section 4 to get our fully dynamic APSP. In order to get a fast query time we do not treat **Dec** as a black box, but rather exploit some additional properties in the construction of Roditty and Zwick [19, 20]. As mentioned earlier in our decremental distance oracle **Dec** on each graph G_i our algorithm invokes the Roditty and Zwick [19, 20] decremental algorithm. Roditty and Zwick showed how to maintain the Thorup-Zwick distance oracle decrementally for distances smaller than $\overline{d}$ with total update time $O(\overline{d}mn^{1/k})$. Let us first outline the construction of Thorup-Zwick distance oracle [18]. We will then see how to exploit some properties in the Thorup-Zwick distance oracle [18] in order to get our efficient fully dynamic algorithm.

5.1.1 The Static Distance Oracle of Thorup and Zwick

We outline the construction of the Thorup-Zwick distance oracle [18]. For a given positive integer k, construct the sets $V = A_0 \supseteq A_1 \supseteq \cdots \supseteq A_{k-1}$ as follows: The i-th level A_i is constructed by sampling the vertices of A_{i-1} independently at random with probability $n^{-1/k}$ for $1 \leq i \leq k-1$. The set A_k is set to be the empty set. Next, for every vertex v, define the bunch of v as follows $B(v) = \bigcup_{i=0}^{k-1} B_i(v)$, where $B_i(v) = \{u \in A_i \setminus A_{i+1} \mid \mathbf{dist}(v,u) < \mathbf{dist}(v, A_{i+1})\} \cup \{p_i(v)\}$. The pivot $p_i(v)$ is the closest vertex to v in A_i (break ties arbitrarily).

The Thorup-Zwick Data Structure: For every vertex v store $B(v)$ and for every vertex $w \in B(v)$ store $\mathbf{dist}(w,v)$. In addition, for every vertex v and every index i where $1 \leq i \leq k-1$ store $p_i(v)$ and $\mathbf{dist}(v, p_i(v))$.

The Thorup-Zwick Query Algorithm: Let i be the first index such that either $p_i(s) \in B(t)$ or $p_i(t) \in B(s)$. Assume w.l.o.g. that $p_i(s) \in B(t)$. Return $\mathbf{dist}(s, p_i(s)) + \mathbf{dist}(p_i(s), t)$.

It was shown in [18] that for every $j \leq i$, $\mathbf{dist}(s, p_j(s)) \leq (j-1)\mathbf{dist}(s,t)$ and $\mathbf{dist}(t, p_j(s)) \leq j\mathbf{dist}(s,t)$. Similarly for every $j \leq i$, $\mathbf{dist}(t, p_j(t)) \leq (j-1)\mathbf{dist}(s,t)$ and $\mathbf{dist}(s, p_j(t)) \leq j\mathbf{dist}(s,t)$. Combining it with the fact that $i \leq k-1$, we get the $2k-1$ stretch.

In our case, in order to get fast query we will sometime have only access to the pivots of s. It was shown in [17] that even if we check only the pivots of s and take the first i such that $p_i(s) \in B(t)$ then $\mathbf{dist}(s, p_i(s)) + \mathbf{dist}(p_i(s), t) \leq (4k-3)\mathbf{dist}(s,t)$.

In addition, Thorup and Zwick showed that the expected size of the bunch is $O(k \cdot n^{1/k})$. Finally, Thorup and Zwick showed that in the static case constructing their data structure can be done in time $\tilde{O}(m \cdot n^{1/k})$ time.

5.1.2 Our Fully Dynamic All-Pairs Shortest Paths

As mentioned earlier, we use the decremental distance oracle **Dec** from Section 4. In our decremental distance oracle **Dec** on each graph G_i our algorithm invokes the Roditty and Zwick [19, 20] decremental distance oracle. For each i and a node v, let $B^i(v)$ be the bunch of v in DO^i_{RZ} and let $p^i_j(v)$ be the j'th pivot of v in DO^i_{RZ}. Recall that our decremental algorithm **Dec** returns $\mu_i \cdot DO^i_{RZ}(s,t)$ for the first i such that DO^i_{RZ} returns an estimation on $\mathbf{dist}(s,t)$. In addition, we have that either $DO^i_{RZ}(s,t) = DO^i_{RZ}(s, p^i_{i_1}(s)) + DO^i_{RZ}(p^i_{i_1}(s), t)$ for some $1 \leq i_1 \leq k-1$ or $DO^i_{RZ}(s,t) = DO^i_{RZ}(s, p^i_{i_2}(t)) + DO^i_{RZ}(p^i_{i_2}(t), t)$ for some $1 \leq i_2 \leq k-1$. Moreover, for nodes x, y such that $x \in B^i(y)$ we have $DO^i_{RZ}(x,y) = \mathbf{dist}(x, y, G_i)$.

After $m^{1/2}$ insertions we reconstruct the data structure from scratch. Our data structure contains two main parts **Dec** and an auxiliary distance oracle M. In addition, we maintain the set U containing all the edges that were added since the last time the data structure was constructed.

(Re)Construction: Construct **Dec** as described in Section 4 and an empty set U.

Deletion Operation for an Edge e: We do the following two steps.

1. Invoke the deletion operation of **Dec** for the edge e, and
2. Renew the data structure M as follows: Construct a graph H, initially set to be empty. Add the set of nodes $V(U)$ to H. For every node $v \in V(U)$ and $1 \leq i \leq \rho$, add $B^i(v)$ to H. Add edges from v to every node x in $B^i(v)$, set the weight of the edge to be $\mu_i DO^i_{RZ}(x,v) = \mu_i \mathbf{dist}(x, v, G_i)$. In addition add the set of edges U to H. Compute the Thorup-Zwick distance oracle $DO_{TZ,H}$ on H with parameter k. Store $DO_{TZ,H}$.

Insertion Operation for an Edge $\mathbf{e} = (\mathbf{x}, \mathbf{y})$: Add the edge to U. If $|U| \geq m^{1/2}$ then reconstruct the data structure. Otherwise, renew the data structure M (as explained in the deletion operation).

Query Operation Given Two Nodes s and t: Let $DO_{TZ,H}(x,y)$ be the distance estimate returned by the Thprup-Zwick query algorithm on x and y, if either $x \notin V(H)$ or $y \notin V(H)$ then we just set $DO_{TZ,H}(x,y) = \infty$.

Return the minimum distance between $\mathbf{Dec}(s,t)$ and the minimum

$$\min\{\, \mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) + DO_{TZ,H}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(s)) + \mu_{i_2} DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) \mid 1 \leq j_1, j_2 \leq k-1, 1 \leq i_1, i_2 \leq \rho \}.$$

Analysis: We now turn to the analysis. The key lemma that we present next, bounds the stretch of our dynamic distance oracle.

▶ **Lemma 10.** *Consider two nodes s and t, the distance $\hat{\mathbf{dist}}(s,t)$ returned by the query algorithm satisfies, $\mathbf{dist}(s,t,G) \leq \hat{\mathbf{dist}}(s,t) \leq 2^{O(k\rho)}\mathbf{dist}(s,t,G)$.*

Proof. By the same analysis as in Lemma 8 we can show that all edges $e = (w,v) \in E(H)$, satisfy, $\omega(w,v,H) \geq \mathbf{dist}(w,v,G)$ (where $\omega(x,y,H')$ for nodes x,y and subgraphs H' is the weight of the edge (x,y) in H'). Similarly, $\mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) \geq \mathbf{dist}(s, p^{i_1}_{j_1}(s))$ and $\mu_{i_2} DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) \geq \mathbf{dist}(t, p^{i_2}_{j_2}(t))$ for every $1 \leq j_1, j_2 \leq k-1, 1 \leq i_1, i_2 \leq \rho$. It thus easily follows that $\hat{\mathbf{dist}}(s,t) \geq \mathbf{dist}(s,t,G)$. It remains to prove that $\hat{\mathbf{dist}}(s,t) \leq 2^{O(k\rho)}\mathbf{dist}(s,t,G)$.

We consider two cases. First suppose that the shortest path $P(s,t,G)$ does not contain any edge in U. Note that in this case $\mathbf{dist}(s,t,G) = \mathbf{dist}(s,t,G_D)$, note also that $\mathbf{Dec}(s,t) \leq 2^{O(k\rho)}\mathbf{dist}(s,t,G)$ and we are done.

The second case is when $P(s,t,G)$ contains at least one edge in U. Let $\{e_i = (x_i, y_i) \mid 1 \leq i \leq r\}$ be the edges in U that appear in $P(s,t)$, where the edge e_{i-1} appears before the edge e_i (namely, the edge e_{i-1} is closer to s in $P(s,t)$ than e_i). In addition, assume also that x_i appears before y_i in $P(s,t)$ (namely, x_i is closer to s in $P(s,t,G)$ than y_i) for every $1 \leq i \leq r$. Note also that $\mathbf{dist}(y_i, x_{i+1}, G) = \mathbf{dist}(y_i, x_{i+1}, G_D)$.

Let ℓ be the first index such that DO^{ℓ}_{RZ} returns an estimate of $\mathbf{dist}(y_i, x_{i+1})$. Recall that either $DO^{\ell}_{RZ} = \mathbf{dist}(y_i, p^{\ell}_{j_1}(y_i), G_\ell) + \mathbf{dist}(p^{\ell}_{j_1}(y_i), x_{i+1}, G_\ell)$ for some j_1 or $DO^{\ell}_{RZ} = \mathbf{dist}(y_i, p^{\ell}_{j_2}(x_{i+1}), G_\ell) + \mathbf{dist}(p^{\ell}_{j_2}(x_{i+1}), x_{i+1}, G_\ell)$ for some j_2. Assume w.l.o.g. that $DO^{\ell}_{RZ}(y_i, x_{i+1}) = \mathbf{dist}(y_i, p^{\ell}_{j_1}(y_i), G_\ell) + \mathbf{dist}(p^{\ell}_{j_1}(y_i), x_{i+1}, G_\ell)$ for some j_1. Recall that by the Thorup-Zwick analysis we have $p^{\ell}_{j_1}(y_i) \in B(x_{i+1})$. Thus by construction the edges $(y_i, p^{\ell}_{j_1}(y_i)), (p^{\ell}_{j_1}(y_i), x_{i+1})$ appear in H, with weights $\mu_\ell \mathbf{dist}(y_i, p^{\ell}_{j_1}(y_i), G_\ell)$ and $\mu_\ell \mathbf{dist}(p^{\ell}_{j_1}(y_i), x_{i+1}, G_\ell)$. Thus the graph H contains a path from y_i to x_{i+1} of length $\mu_\ell \mathbf{dist}(y_i, p^{\ell}_{j_1}(y_i), G_\ell) + \mu_i \mathbf{dist}(p^{\ell}_{j_1}(y_i), x_{i+1}, G_\ell) = \mu_\ell DO^{\ell}_{RZ}(y_i, x_{i+1})$. By the analysis of the decremental algorithm, we have $\mathbf{dist}(y_i, x_{i+1}, H) \leq \mu_\ell DO^{\ell}_{RZ}(y_i, x_{i+1}) \leq 2^{O(k\rho)}\mathbf{dist}(y_i, x_{i+1}, G)$. It follows that $\mathbf{dist}(x_1, y_r, H) \leq 2^{O(k\rho)}\mathbf{dist}(x_1, y_r, G)$.

Similarly, let i_1 be the first index such that $DO^{i_1}_{RZ}$ returns an estimate of $\mathbf{dist}(s, x_1)$. Let j_1 be the first index such that $p^{i_1}_{j_1}(s) \in B^{i_1}(x_1)$. Recall that by the analysis of the Thorup-Zwick we have, $\mathbf{dist}(s, p^{i_1}_{j_1}(s), G_{i_1}) + \mathbf{dist}(p^{i_1}_{j_1}(s), x_1, G_{i_1}) \leq (4k-3)\mathbf{dist}(s, x_1, G_{i_1})$. By similar analysis of Lemma 8 we have $\mu_{i_1} DO^{i_1}_{RZ}(s, x_1) = \mu_{i_1}(\mathbf{dist}(s, p^{i_1}_{j_1}(s), G_{i_1}) + \mathbf{dist}(p^{i_1}_{j_1}(s), x_1, G_{i_1})) = \mu_{i_1}(DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) + DO^{i_1}_{RZ}(p^{i_1}_{j_1}(s), x_1)) \leq 2^{O(k\rho)}\mathbf{dist}(s, x_1, G)$. Note also that the edge $(p^{i_1}_{j_1}(s), x_1)$ exists in H of weight $\mu_{i_1} DO^{i_1}_{RZ}(p^{i_1}_{j_1}(s), x_1)$.

Similarly, we have indices i_2 and j_2 such that $\mu_{i_2} DO^{i_2}_{RZ}(y_r, t) = \mu_{i_2}(DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) + DO^{i_2}_{RZ}(p^{i_2}_{j_2}(t), y_r))$. Note also that the edge $(p^{i_2}_{j_2}(t), y_r)$ exists in H of weight $\mu_{i_2} DO^{i_2}_{RZ}(p^{i_2}_{j_2}(t), y_r)$.

It is not hard to verify now that by concatenating all these paths together with the edges $e_i = (x_i, y_i)$, we have, $\mathbf{dist}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(t), H) \leq 2^{O(k\rho)}(\mathbf{dist}(p^{i_1}_{j_1}(s), x_1, G) + \mathbf{dist}(x_1, y_r, G) + \mathbf{dist}(y_r, p^{i_2}_{j_2}(t))$. It follows,

$$\begin{aligned} DO_{TZ,H}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(t)) &\leq (2k-1)\mathbf{dist}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(t), H) \\ &\leq 2^{O(k\rho)}\mathbf{dist}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(t), G). \end{aligned}$$

Hence, we get that,

$$\begin{aligned} \hat{\mathbf{dist}}(s,t) &= \min\{\mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) + DO_{TZ,H}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(s)) + \mu_{i_2} DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) \mid \\ &\quad 1 \leq j_1, j_2 \leq k-1, 1 \leq i_1, i_2 \leq \rho\} \\ &\leq \mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) + (2k-1)\mathbf{dist}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(s), H) + \mu_{i_2} DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) \\ &\leq 2^{O(k\rho)}\mathbf{dist}(s,t,G) \end{aligned}$$

◀

The next lemma bounds the update time.

▶ **Lemma 11.** *The amortized time for a single update is $\tilde{O}(m^{1/2} \cdot n^{2/k})$.*

Proof. The graph H contains $\tilde{O}(|U| \cdot n^{1/k})$ edges and can be constructed in time $\tilde{O}(|U| \cdot n^{1/k})$ given the DO_{RZ} data structures. Constructing $DO_{TZ,H}$ on H takes $\tilde{O}(|U| \cdot n^{1/k} \cdot n^{1/k})$. Recall that after $m^{1/2}$ updates the data structure is being reconstructed so that $|U| \leq m^{1/2}$. Thus the time to construct $DO_{TZ,H}$ in each update step is $\tilde{O}(m^{1/2} \cdot n^{2/k})$.

The total update time of constructing the decremental data structure **Dec** is $\tilde{O}(mn^{1/k})$. This should be amortized over the $m^{1/2}$ updates until **Dec** is being reconstructed again. We get that the amortized per update for maintaining **Dec** is $\tilde{O}(m^{1/2}n^{1/k})$. It is not hard to see now that the amortized update time is $\tilde{O}(m^{1/2} \cdot n^{2/k})$.

The lemma follows. ◀

The next lemma bounds the query time.

▶ **Lemma 12.** *The query time is $O((k\rho)^2)$.*

Proof. Recall that the query algorithm returns the minimum distance between $\mathbf{Dec}(s,t)$ and the minimum

$$\min\{\mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) + DO_{TZ,H}(p^{i_1}_{j_1}(s), p^{i_2}_{j_2}(s)) + \mu_{i_2} DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(t)) \mid \\ 1 \leq j_1, j_2 \leq k-1, 1 \leq i_1, i_2 \leq \rho\}.$$

Computing $\mathbf{Dec}(s,t)$ takes $O(k\rho)$ time. Computing $\mu_{i_1} DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s))$ for $1 \leq j_1 \leq k-1$ and $1 \leq i_1 \leq \rho$ takes $O(1)$ (the algorithm stores $DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s)) = \mathbf{dist}(s, p^{i_1}_{j_1}(s), G_{i_1})$). Thus computing all $DO^{i_1}_{RZ}(s, p^{i_1}_{j_1}(s))$ for $1 \leq j_1 \leq k-1$ and $1 \leq i_1 \leq \rho$ takes $O(k\rho)$ time. Similarly, computing all $DO^{i_2}_{RZ}(t, p^{i_2}_{j_2}(2))$ for $1 \leq j_2 \leq k-1$ and $1 \leq i_2 \leq \rho$ takes $O(k\rho)$ time.

In addition, Thorup and Zwick query algorithm is $O(k)$. It was shown in [7] how to reduce the query time to constant (while keeping the rest of the parameters the same). Thus, computing $DO_{TZ,H}(x,y)$ for two nodes x, y can be done in $O(1)$ time.

It is not hard to see now that finding the minimum takes $O(k^2\rho^2)$ time. ◀

By maintaining this data structure for setting $k' = \lfloor k/2 \rfloor$ we can get update time $\tilde{O}(m^{1/2} \cdot n^{1/k})$. This will increase the logarithm of the stretch by only an $O(1)$ factor, and the query time goes up by $O(1)$. This proves Theorem 1.

References

1 G. Ausiello, G. F. Italiano, A. Marchetti-Spaccamela, and U. Nanni. Incremental algorithms for minimal length paths. *J. Algorithms, 12(4)*, 615–638, 1991.

2 S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In *34th ACM Symp. on Theory of Computing (STOC)*, 117–123, 2002.

3 S. Baswana, R. Hariharan, and S. Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *14th ACM Symp. on Discrete Algorithms (SODA)*, 394–403, 2003.

4 S. Baswana, S. Khurana, and S. Sarkar. Fully dynamic randomized algorithms for graph spanners. In *ACM Transactions on Algorithms*, 8(4):35, 2012.

5 A. Bernstein. Fully dynamic approximate all-pairs shortest paths with query and close to linear update time. In *50th IEEE Symp. on Foundations of Computer Science (FOCS)*, 50–60, 2009.

6 A. Bernstein, L. Roditty. Improved dynamic algorithms for maintaining approximate shortest paths under deletions. In *22nd ACM Symp. on Discrete Algorithms (SODA)*, 1355–1365, 2011.

7 S.Chechik. Approximate Distance Oracles with Constant Query Time. To appear in *Proc. 46th ACM Symp. on Theory of Computing (STOC)*, 2014.

8 C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *J. of the ACM, 51*, 2004.

9 C. Demetrescu and G. F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *J. of Computer and System Sciences, 72(5)*, 813–837, 2006. Special issue of *FOCS*'01.

10 Y. Dinitz. Dinitz' algorithm: The original version and Even's version. In *Essays in Memory of Shimon Even*, 218–240, 2006

11 M. R. Henzinger and V. King. Maintaining Minimum Spanning Forests in Dynamic Graphs. *SIAM J. Computing, 31(2)*, 364–374, 2001.

12 M. R. Henzinger, S. Krinninger and D. Nanongkai. A Subquadratic-Time Algorithm for Decremental Single-Source Shortest Paths. In *25th ACM Symp. on Discrete Algorithms (SODA)*, 1053–1072, 2014.

13 V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th IEEE Symp. on Foundations of Computer Science (FOCS)*, 81–91, 1999.

14 V. King and M. Thorup. A space saving trick for directed dynamic transitive closure and shortest path algorithms. In Jie Wang, editor, COCOON, volume 2108 of *Lecture Notes in Computer Science*, 268–277, 2001.

15 M. Thorup. Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. In *9th Scandinavian Workshop on Algorithm Theory (SWAT)*, 384–396, 2004.

16 M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *37th ACM Symp. on Theory of Computing (STOC)*, 112–119, 2005.

17 M. Thorup and U. Zwick. Compact routing schemes. In *13th ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, 1–10, 2001.

18 M. Thorup and U. Zwick. Approximate distance oracles. In *Journal of the ACM*, pages 1–24, 2005.

19 L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *45th IEEE Symp. on Foundations of Computer Science (FOCS)*, 499–508, 2004.

20 L. Roditty and U. Zwick. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM J. Comput.* 41(3): 670–683, 2012.

21 L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. 18th European Symposium on Algorithms (ESA)*, 580–591, 2004.