Improved Dynamic Algorithms for Maintaining Approximate Shortest Paths Under Deletions

Aaron Bernstein* Liam Roditty†

Abstract

We present the first dynamic shortest paths algorithms that make any progress beyond a long-standing $O(n)$ update time barrier (while maintaining a reasonable query time), although it is only progress for not-too-sparse graphs. In particular, we obtain new decremental algorithms for two approximate shortest-path problems in unweighted, undirected graphs. Both algorithms are randomized (Las Vegas).

- Given a source s, we present an algorithm that maintains $(1+\epsilon)$-approximate shortest paths from s with an expected *total* update time of $\widetilde{O}(n^{2+O(1/\sqrt{\log n})})$ over all deletions (so the amortized time is about $\widetilde{O}(n^2/m)$). The worst-case query time is constant. The best previous result goes back *three* decades to Even and Shiloach [16] and Dinitz [12]. They show how to decrementally maintain an *exact* shortest path tree with a total update time of $O(mn)$ (amortized update time $O(n)$). Roditty and Zwick [22] have shown that $O(mn)$ is actually optimal for *exact* paths (barring a better combinatorial algorithm for boolean matrix multiplication), unless we are willing to settle for a $\Omega(n)$ query time. In fact, until now, even *approximate* dynamic algorithms were not able to go beyond $O(mn)$.

- For any fixed integer $k \geq 2$, we present an algorithm that decrementally maintains a distance oracle (for *all pairs* shortest distances) with a total expected update time of $\widetilde{O}(n^{2+1/k+O(1/\sqrt{\log n})})$ (amortized update time about $\widetilde{O}(n^{2+1/k}/m)$. The space requirement is only $O(m+n^{1+1/k})$, the stretch of the returned distances is at most $2k-1+\epsilon$, and the worst-case query time is $O(1)$. The best previous result of Roditty and Zwick [21] had a total update time of $\widetilde{O}(mn)$ and a stretch of $2k-1$. Note that our algorithm implicitly solves the decremental all-pairs shortest path problem with the same bounds; the best previous approximation algorithm of Roditty and Zwick [21] returned $(1+\epsilon)$ approximate distances, but used $O(n^2)$ space, and required $\widetilde{O}(mn)$ total update time. As with the previous problem, our algorithm is the first to make progress beyond the $O(mn)$ total update time barrier while maintaining a small query time.

We present a general framework for accelerating decremental algorithms. In particular, our main idea is to run existing decremental algorithms on a sparse subgraph (such as a spanner or emulator) of the graph rather than on the original graph G. Although this is a common approach for *static* approximate shortest-path problems, it has never been used in a decremental setting because maintaining the subgraph H as edges are being deleted from G might require *inserting* edges into H, thus ruining the "decrementality" of the setting. We overcome this by presenting an emulator whose maintenance only requires a limited number of well-behaved insertions.

In other words, we present a general technique for running decremental algorithms on a sparse subgraph of the graph. Once our framework is in place, applying it to any particular decremental algorithm only requires trivial modifications; most of the work consists of showing that these algorithms *as they are* still work in our restricted fully dynamic setting, where we encounter not just arbitrary deletions (as in the original setting), but also restricted insertions.

*Funded by NSF GRFP Fellowship. Department of Computer Science, Columbia University; New York, NY, 10027; email: bernstei@gmail.com

†Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel, email: liamr@macs.biu.ac.il

1 Introduction

1.1 The Problem Dynamic algorithms are used to model settings that change over time. We study the problem of maintaining shortest path information as edges are being inserted and deleted from the graph. The objective of a dynamic single source shortest paths (SSSP) algorithm is to efficiently process an online sequence of delete, insert, and query operations. Each delete operation removes a single edge from the underlying graph, while an insert operation adds an edge. A query operation can ask for the shortest distance in the current graph between the source and any other vertex. In dynamic all pairs shortest paths (APSP), a query can ask for the shortest distance between *any* pair of vertices. A dynamic algorithm is said to be *decremental* if it can only process delete operations, *incremental* if it can only process insert operations, and *fully dynamic* if it can process both. Note that the naive approach to all these problems is to recompute shortest paths from scratch after each update. This yields an update time of $\widetilde{O}(m)$ for dynamic SSSP, and $\widetilde{O}(mn)$ for APSP.

The problem of maintaining single source shortest paths under deletions was the first dynamic problem studied in theoretical computer science. Not only it is interesting in and of itself, but it arises as a subproblem in many other decremental and fully dynamic algorithms ([18, 6, 21, 22, 7]). In 1981, Even and Shiloach [16] presented a decremental SSSP algorithm for *undirected*, unweighted graphs with $O(1)$ query time and a *total* update time of $O(mn)$ over all deletions (so the amortized update time is $O(n)$). Around the same time, Dinitz [12] independently presented a similar result, but it was written in Russian and not known in the west. King [18] later extended the $O(mn)$ bound to directed graphs, and King and Thorup [19] presented a technique that allows us to implement this algorithm using less space.

Achieving an $o(mn)$ total update time (and reasonable query time) for decremental SSSP has been a long-standing open problem, but no progress has been made in the last *three* decades. Roditty and Zwick [22] provide an explanation for this by showing that the incremental and decremental *unweighted* SSSP problems are at least as hard as several natural static problems such as Boolean matrix multiplication and the problem of finding all edges of a graph that are contained in triangles. Obtaining a combinatorial Boolean matrix multiplication algorithm whose running time is $O((mn)^{1-\epsilon} + n^2)$, or $O(n^{3-\epsilon})$, for some $\epsilon > 0$, is a major open problem.

In contrast, a rich body of the STOC/FOCS/SODA literature has considered the problem of dynamic *all-pairs* shortest paths. Ausiello *et al.* [2] obtained an incremental APSP algorithm for unweighted directed graphs with a total running time of $O(n^3 \log n)$. Henzinger and King [17] presented a decremental APSP algorithm for directed unweighted graphs whose total running time, over all deletions, is $\widetilde{O}(\frac{mn^2}{t} + mn)$, but whose query time is $O(t)$. King [18] presented two *fully dynamic* APSP algorithms for unweighted graphs. The first achieves an amortized update time of $\widetilde{O}(n^{2.5})$, while the second is a $1+\epsilon$ approximation algorithm with amortized update time $\widetilde{O}(n^2)$ (query times are constant). Demetrescu and Italiano [11] presented a fully dynamic algorithm for a directed graph G where each edge can have at most S different real values; the amortized update time is $\widetilde{O}(n^{2.5}\sqrt{S})$.

The big breakthrough came in STOC 2003, where Demetrescu and Italiano [10] presented a fully dynamic APSP algorithm for directed graphs with arbitrary non-negative real edge weights with an amortized update time of $\widetilde{O}(n^2)$. Thorup [23] slightly improved on the update time, and extended the algorithm to work for negative weights. Thorup [24] also used this algorithm together with other ideas to achieve a *worst-case* update time of $\widetilde{O}(n^{2.75})$. Baswana *et al.* [4] presented a *decremental* APSP algorithm for unweighted directed graphs with a total update time of $\widetilde{O}(n^3)$ (amortized update time $\widetilde{O}(n^3/m)$).

This leaves us with several solid barriers. Roditty and Zwick [22] show that we are unlikely beat the $O(mn)$ total update time for *exact* decremental SSSP. Beating the $O(n^2)$ update time for fully dynamic APSP also seems difficult because any algorithm that explicitly maintains the distance matrix requires at least $\Omega(n^2)$ time per update. Finally, although this barrier seems less solid, no one has been able to achieve an $o(n^3)$ algorithm for decremental APSP. However, there exist several approximation algorithms that break through these barriers.

Baswana *et al.* [4] presented a $(1+\epsilon)$ approximation algorithm for decremental APSP in unweighted, directed graphs with a total update time of $\widetilde{O}(n^2\sqrt{m})$. In [5], the same authors presented several decremental algorithms for *undirected* graphs, such as a 3-approximate algorithm with amortized time $\widetilde{O}(n^{10/9})$, and a 7-approximate algorithm with amortized update time $\widetilde{O}(n^{28/27})$. Roditty and Zwick [21] presented two improved algorithms for unweighted, undirected graphs. The first was a $(1+\epsilon)$-approximate decremental APSP algorithm with a constant query time and a *total* update time of only $\widetilde{O}(mn)$ (amortized update time $\widetilde{O}(n)$). The

second algorithm achieved a worse approximation bound of $2k-1$ ($2 \leq k \leq \log(n)$), but had the added advantage of only using $\widetilde{O}(m + n^{1+1/k})$ space. Recently, Bernstein [7] presented a $(2+\epsilon)$-approximate *fully dynamic* APSP algorithm for weighted, undirected graphs with a $O(\log\log\log(n))$ query time and a close to linear update time (almost, but not quite $\widetilde{O}(m)$).

1.2 Our Contributions Although approximation has been used to develop several faster dynamic algorithms, none have been able to beat the long-standing $O(mn)$ total update time barrier for decremental SSSP. We present the first decremental algorithms which show that improvements are possible beyond $O(mn)$. Our first result is a decremental $(1+\epsilon)$ approximate algorithm for decremental SSSP with $O(1)$ query time and a *total* update time close to $\widetilde{O}(n^2)$ (amortized update time $\widetilde{O}(n^2/m)$). Technically, it is actually $O(n^{2+O(1/\sqrt{\log(n)})})$. Our second result is a $(2k-1+\epsilon)$ approximate decremental *APSP* algorithm with $O(k)$ query and total update time $O(n^{2+1/k+O(1/\sqrt{\log(n)})})$ ($2 \leq k \leq \log(n)$ is a parameter of our choosing). The space requirement is only $\widetilde{O}(m + n^{1+1/k})$. Both results apply to unweighted undirected graphs.

Our approach relies on extending a common tool used in static algorithms to the dynamic setting. Many approximate shortest paths algorithms in undirected graphs start by constructing a sparse *emulator* (or spanner) of the graph (*i.e* another graph on the same vertex set with a similar shortest distance structure – see Definition 2.1 for a formal description), and then computing shortest paths in the *sparse* emulator rather than in the original graph (see [3, 8, 1, 13, 9, 15, 14, 27, 25]). However, this approach has never been used in a decremental setting. The reason for this is that as edges in G are being deleted, we also have to change the edges in the emulator H, and a deletion in G can lead to *insertions* into H. Thus, we cannot run decremental algorithms on our emulator, because from the perspective of H, we are not in a decremental setting.

Our main contribution is an emulator that possesses several novel properties relating to how it changes as edges in G are being deleted. The emulator itself is basically identical to one used by Bernstein [7], which is in turn a modification of a spanner developed by Thorup and Zwick [26]. However, the properties we prove are entirely *new to this paper*. Intuitively, we show that insertions into H are relatively rare and well behaved, so although from the perspective of H we are no longer in a purely decremental setting, we are in a *restricted* fully dynamic setting where we allow arbitrary deletions and restricted insertions.

We then show that that many existing decremental algorithms can run in this restricted fully dynamic setting. In other words, this paper presents a general technique for running decremental algorithms on an emulator. We apply it to the two state of the art decremental algorithms, but it could potentially be applied to others as they come up. In fact, once our framework is in place, the modifications to specific algorithms are trivial: it is only the analysis that requires work. In particular, we show that these decremental algorithms only rely on a few specific properties of the decremental setting – properties which are preserved in our restricted fully dynamic setting.

The rest of this paper is organized as follows. In the next section we introduce preliminaries. In Section 3 we present our emulator and analyze its special properties. We then show in Section 4 how our techniques can be used to obtain a faster decremental SSSP algorithm. In Section 5 we present our improved decremental distance oracle.

2 Preliminaries

Let $G=(V,E)$ be our unweighted, undirected graph. For any pair $x, y \in V$, let $\pi(x,y)$ be the shortest $x-y$ path, and let $\delta(x,y)$ be the weight of $\pi(x,y)$. We say that an algorithm outputs an α approximation if given any query input x, y it outputs a value in $[\delta(x,y), \alpha\delta(x,y)]$. Throughout the paper, ϵ refers to an arbitrary positive constant < 1.

DEFINITION 2.1. *An emulator H of G is a graph with the same vertex set as G, but with different (possibly weighted) edges. We let $w_H(x,y)$ be the weight of edge (x,y) in H, we let $\pi_H(x,y)$ be the shortest $x-y$ path in H, and we let $\delta_H(x,y)$ be the length of this path. We say that H is an (α,β)-emulator if for any pair of vertices x, y we have $\delta(x,y) \leq \delta_H(x,y) \leq \alpha\delta(x,y)+\beta$. We say that H is an α-emulator if it is an $(\alpha,0)$-emulator. (Remark: Many approximate graph algorithms use spanners, which are emulators whose edge set must be a subset of the original edges E. However, we stick to the more general definition of emulators.)*

2.1 An Existing Decremental SSSP Algorithm We rely on an existing algorithm of King [18], which given a directed graph G with positive *integer* edge weights, and a source s, decrementally maintains a shortest path tree *up to distance* d in a total of $O(md)$ time (see Section 2.1 of [18]). The main property of King's algorithm is that as edges in G are being deleted, it only explores the edges incident on a vertex v when the distance from s to v changes.

Moreover, it only explores the edges of v a constant number of times per distance change. This implies a total update time of $O(md)$ because the distance to a vertex v can increase at most d times before it exceeds d. (Note that in unweighted graphs $d \leq n$, so $O(md) = O(mn)$)

2.2 The Techniques of Thorup and Zwick Both of our results rely on techniques used in the approximate distance oracle of Thorup and Zwick [25], which we now review.

DEFINITION 2.2. *Let $V = A_0 \supseteq A_1 \supseteq ... \supseteq A_{k-1} \supseteq A_k = \emptyset$ be sets of vertices ($2 \leq k \leq \log(n)$ is a parameter of our choosing). In particular, we start with $A_0 = V$, and every vertex in A_i is independently sampled and put into A_{i+1} with probability $1/n^{1/k}$. We refer to the indices $1, 2, ..., k$ as vertex priorities, where the priority of v is i if and only if $v \in A_i \setminus A_{i+1}$.*

DEFINITION 2.3. *Define the i-witness of v, or $p_i(v)$, to be vertex in A_i that is nearest to v: $p_i(v) = argmin_{w \in A_i}(\delta(v, w))$. To break ties, we pick the $p_i(v)$ that survives to the set A_j of largest index (equivalently, it is contained in the most sets A_j). Define $\delta(v, A_i)$ to be $\delta(v, p_i(v))$.*

DEFINITION 2.4. *Given a vertex $v \in A_i - A_{i+1}$, we define the cluster of v to be $C(v) = \{w \in V \mid \delta(w, v) < \delta(w, A_{i+1})\}$. We define the bunch of v to be $B(v) = \{w \in V \mid v \in C(w)\}$*

LEMMA 2.1. *[25] With high probability, the size of every bunch is $\widetilde{O}(kn^{1/k}) = \widetilde{O}(n^{1/k})$. Thus, the size of all the bunches (or all the clusters) is $\widetilde{O}(kn^{1+1/k})$.*

Since the above lemma is true with high probability, we assume for the rest of this paper that it always holds.

THEOREM 2.1. *[25] Consider the oracle that for every vertex v stores $C(v)$ and all witnesses $p_i(v)(i \leq k-1)$. With high probability, this oracle requires $O(mn^{1/k})$ construction time and $O(n^{1+1/k})$ space (recall: k, the number of vertex priorities, is a parameter of our choosing). Given any query (x, y) it can return a $(2k-1)$ approximation to $\delta(x, y)$ in $O(k)$ time.*

Roditty and Zwick [21] showed how to efficiently maintain this oracle in a decremental setting. Note that because the set hierarchy $A_0 \supseteq A_1 \supseteq A_2 \supseteq ... \supseteq A_k$ is picked in an oblivious manner, without ever looking at the graph $G = (V, E)$, we can use the same set hierarchy for all the versions of the graph. That is, although the clusters might change as G changes, the sets A_i themselves remain the same.

THEOREM 2.2. *[21] Let G be an undirected graph with positive edge weights. Given any distance d, we can decrementally maintain all clusters and witnesses up to distance d in an expected total time of $O(mdn^{1/k})$ over all deletions in G. By "up to distance d" we mean that we do not maintain witnesses $p_i(v)$ or cluster members $w \in C(v)$ whose shortest distance from v is greater than d.*

3 Using an Emulator

A common approach to solving approximate shortest path problems on dense graphs is to avoid working with the original graph G, and work with a sparse α-emulator H instead. We follow this basic guideline, but it is much more difficult in a dynamic setting because as we delete edges in G, we also need change the edges in H in order for it to remain an α-emulator.

As mentioned in the introduction, the main issue we run into as that even though we only *delete* edges in G, maintaining H during these deletions sometimes involves *inserting* edges into H. So it is not enough to just present a suitable emulator and then run existing decremental algorithms on H; when working with H, we are no longer in a purely decremental setting. We overcome this by proving that insertions into H are relatively rare and well-behaved, and then showing that many existing decremental algorithms can efficiently handle a small number of such well-behaved insertions.

The emulator we use is basically identical to an emulator used by Bernstein [7], which is in turn a modification of one developed by Thorup and Zwick [26]. Both are based upon the techniques covered in Section 2.2. Note that although the emulator itself is not new, we prove several new properties relating to how it changes as edges in G are deleted.

3.1 The Emulator We start with an emulator of Thorup and Zwick [26] (they actually used a spanner, but for simplicity, we express it as an emulator).

THEOREM 3.1. *[26] Let H be the following undirected emulator: for every vertex v, and every $w \in C(v)$, H contains an edge of weight $\delta(v, w)$ from v to w. Also, for every vertex v and every vertex priority i, H contains an edge of weight $\delta(v, p_i(v))$ from v to $p_i(v)$. Then, H contains $O(kn^{1+1/k})$ edges and is a $((1+\epsilon/2), \zeta)$ emulator, where $\zeta = O((6/\epsilon)^k)$. (Technical note: Lemma 2.3 of Thorup and Zwick works for* any *ϵ, so plugging $\epsilon/2$ into their lemma we get our claim. In particular, note that $2 + 2/(\epsilon/2)$ – the term in their lemma – is $\leq 6/\epsilon$ because $\epsilon < 1$).*

COROLLARY 3.1. *If $\delta(x, y) \geq (2/\epsilon)\zeta$ (recall: $\zeta = O((6/\epsilon)^k)$) then $\delta_H(x, y) \leq (1+\epsilon)\delta(x, y)$*

Proof. $\delta_H(x,y) \leq (1+\epsilon/2)\delta(x,y)+\zeta \leq (1+\epsilon/2)\delta(x,y)+(\epsilon/2)\delta(x,y) = (1+\epsilon)\delta(x,y)$. ∎

Bernstein [7] showed that the properties of H still hold if we remove all heavy edges. We take advantage of this, although for different reasons.

DEFINITION 3.1. *We define γ to be $(24/\epsilon)\zeta$ (recall: $\zeta = O((6/\epsilon)^k)$ is the additive error of H).*

THEOREM 3.2. *[7] Let H be the same emulator as in Theorem 3.1, except with all edges of weight $\geq \gamma$ removed. Then, H has $O(kn^{1+1/k})$ edges and is a $((1+\epsilon), O((6/\epsilon)^k))$ emulator. Even better, If $\delta(x,y) \geq \gamma/2$ then $\delta_H(x,y) \leq (1+\epsilon)\delta(x,y)$ (no additive factor).*

Proof. The proof is almost identical to one in [7], and is given here for the sake of completeness. To bound the approximation error of H, we let H' be the original emulator without heavy edges removed. We now break our proof into two possible cases. If $\delta(x,y) \leq \gamma/3$, then we know that $\delta_{H'}(x,y) \leq (1+\epsilon)\gamma/3 + \zeta \leq \gamma$, so the path from x to y in H' never uses edges of length greater than γ, so this path must also exist in H.

If $\delta(x,y) > \gamma/3$ then we split $\pi(x,y)$ into paths of length $\lceil\gamma/12\rceil$. That is, we let $y_1 = x$, we let y_2 be the vertex on $\pi(x,y)$ that is at distance $\lceil\gamma/12\rceil$ from x, we let y_3 be the vertex at distance $\lceil\gamma/12\rceil$ from y_2 and so on up to some y_r. We define $y_{r+1} = y$. We then let π_i be the subpath of $\pi(x,y)$ from y_i to y_{i+1}.

(Technical note: we choose y_r in such a way that $\lceil\gamma/12\rceil \leq w(\pi_r) \leq \lceil\gamma/6\rceil + 1$. This can always be done because if we break $\pi(x,y)$ into paths of length $\lceil\gamma/12\rceil$, then the remainder left over will have length $\leq \lceil\gamma/12\rceil$, so we can just concatenate this remainder to the last path of length $\lceil\gamma/12\rceil$, thus yielding a path of length $\leq 2\lceil\gamma/12\rceil \leq \lceil\gamma/6\rceil + 1$.)

Note that for any i we have $w(\pi_i) \geq \gamma/12 \geq (2/\epsilon)\zeta$ (see Definition 3.1 for ζ), so by the corollary of Theorem 3.1 there exists a $(1+\epsilon)$ approximate path p_i in H' (see Figure 1). But all of the p_i have length less than $(1+\epsilon)(\lceil\gamma/6\rceil + 1) < \gamma/3$, so they must also be in H (not just H'). Thus, we consider the path $p = p_1 \circ p_2 \circ ... \circ p_r$; this is an $x-y$ path in H of length $\leq (1+\epsilon)\delta(x,y)$, which completes the proof. See Figure 1. ∎

Recall that the whole purpose of constructing a sparse emulator H was to run all of our algorithms on H instead of G. But as we delete edges from G we also need to modify H so that it remains a $((1+\epsilon), O((6/\epsilon)^k))$ emulator. In particular, the clusters and witnesses in G change as we delete edges, so we need to maintain all of the $C(v)$ and $p_i(v)$.

LEMMA 3.1. *We can decrementally maintain the emulator in Theorem 3.2 in a total of $O(m\gamma n^{1/k})$ time over all deletions in G. This stems from Theorem 2.2.*

3.2 Dynamic Properties of H Although we use an already existing emulator, we use it in an entirely novel way; all of the analysis in this section is *new to this paper*. Given a graph G, our approach is to maintain a sparse emulator H, and run existing decremental algorithms on H instead of on G. The problem is that from the perspective of H, we are *not* in a decremental setting; changes to G can cause edge *insertions* into H. In particular, if a deletion in G causes some $\delta(w, A_i)$ to increase, then this might cause w to join some cluster $C(v)$, which inserts edge (v,w) into H.

Thus, if we are to run existing decremental algorithms on H, we must extend these algorithms to also handle insertions into H. The main difficulty is that many decremental algorithms rely on the fact that if we only allow edge deletions in an integer-weighted graph, then the shortest distance between any two vertices can change at most d times (over all deletions to G) before this distance exceeds d. But this is no longer true if we allow insertions because the distance can slowly increase from 1 to d, and then decrease back to 1 with just a single insertion.

As such, we do not know how to extend decremental algorithms to work in a general fully dynamic setting. But we show that in our specific case, even though edges can be inserted into H, it is nonetheless possible to give a good bound on the number of times the distance between two vertices can change. This allows us to extend many decremental algorithms to work in our partially incremental setting.

LEMMA 3.2. *Letting H be the emulator in Theorem 3.2, then the number of edges inserted into H over all deletions in G is $\widetilde{O}(k\gamma n^{1+1/k})$ (with high probability).*

Proof. By the definition of clusters, the only way a deletion in G can cause an edge (v,w) to be inserted into H is if the deletion causes $\delta(w, A_i)$ to increase for some i, which in turn causes w to join $C(v)$ (here, v must have a priority $(i-1)$). Also, since all edges in H have weight less than γ, $\delta(w, A_i)$ must have been less than γ before the deletion in G.

But note that since all edges in G have weight 1, $\delta(w, A_i)$ can increase at most γ times before it exceeds γ. Moreover, every time $\delta(w, A_i)$ increases, at most $\widetilde{O}(kn^{1/k})$ edges (v,w) are inserted into H; this is because (v,w) can only exist in H if $w \in C(v)$, and at any time, w is contained in $\widetilde{O}(kn^{1/k})$ clusters – see Lemma 2.1. Thus, for any vertex w and any vertex priority i, the total number of edges inserted into H on account of $\delta(w, A_i)$ increasing is $\widetilde{O}(\gamma kn^{1/k})$. But there are only $O(kn)$ pairs (w,i),

which leads to an $\widetilde{O}(k^2\gamma n^{1+1/k})$ upper bound on the total number of edges inserted into H. A more careful analysis reduces this to $\widetilde{O}(k\gamma n^{1+1/k})$, but we omit the details. ∎

LEMMA 3.3. *If an edge (x,y) is inserted into H, then before the insertion $\delta_H(x,y) \leq 3\gamma$. As such, the insertion of any edge (x,y) into H can cause $\delta_H(x,y)$ to decrease by at most 3γ.*

Proof. We know that all edges in H have weight at most γ, and we know that the weight of any edge (x,y) in H is $\delta(x,y)$, so after the update $\delta(x,y) = w_H(x,y) \leq \gamma$. But recall that even though $\delta_H(x,y)$ can decrease, $\delta(x,y)$ can only increase (edges are only removed from the original graph G), so *before* the update $\delta(x,y) \leq w_H(x,y) \leq \gamma$. But since H is a $((1+\epsilon), O((6/\epsilon)^k))$ emulator, this implies that before the update $\delta_H(x,y) \leq (1+\epsilon)w_H(x,y) + O((6/\epsilon)^k) \leq (1+\epsilon)\gamma + \epsilon\gamma < 3\gamma$. ∎

LEMMA 3.4. *Given any two vertices x, y in H, the number of times that $\delta_H(x,y)$ changes over all deletions in G is $\widetilde{O}(k\gamma^2 n^{1+1/k})$ (with high probability).*

Proof. Let Δ^{DEC} refer to the *total* sum of distance changes to $\delta_H(x,y)$ caused by decremental updates, and let Δ^{INC} refer to the total sum of distance changes caused by incremental updates. So if $\delta_H(x,y)$ changes from 4 to 9 to 2 to 5 to 1 then $\Delta^{\text{INC}} = 7+4 = 11$ and $\Delta^{\text{DEC}} = 5+3 = 8$. It is not hard to see that $\delta_H(x,y)$ changes at most $\Delta^{\text{DEC}} + \Delta^{\text{INC}}$ times.

Intuitively, Δ^{INC} and Δ^{DEC} must be about the same because $\delta_H(x,y)$ cannot decrease by too much without increasing at some point. In particular, $\Delta^{\text{INC}} - n \leq \Delta^{\text{DEC}} \leq \Delta^{\text{INC}} + n$ because $\delta_H(x,y)$ can never be smaller than 1 or larger than n. But we know that $\Delta^{\text{INC}} = \widetilde{O}(k\gamma^2 n^{1+1/k})$ because Lemma 3.2 tells us that a total of $\widetilde{O}(k\gamma n^{1+1/k})$ edges are inserted into H, and Lemma 3.3 tells us that each of these edges causes $\delta_H(x,y)$ to decrease by at most 3γ. So $\Delta^{\text{INC}} = \widetilde{O}(k\gamma^2 n^{1+1/k})$, and $\Delta^{\text{DEC}} \leq O(\Delta^{\text{INC}} + n) = \widetilde{O}(k\gamma^2 n^{1+1/k})$, so $\Delta^{\text{INC}} + \Delta^{\text{DEC}} = \widetilde{O}(k\gamma^2 n^{1+1/k})$, as desired. ∎

We described our emulator H with respect to a parameter k of our choosing (k is the number of vertex priorities). We now set k to be $\sqrt{\log(n)}/\sqrt{\log(6/\epsilon)}$, which yields the following:

DEFINITION 3.2. *Define $\alpha = \sqrt{\log(n)} n^{\sqrt{\log(6/\epsilon)}/\sqrt{\log(n)}}$. Define $\beta = n^{\sqrt{\log(6/\epsilon)}/\sqrt{\log(n)}}$. Note that α, β, $\alpha\beta$ and $\alpha\beta^2$ are all $\widetilde{O}(n^{O(1/\sqrt{\log(n)})})$, which is very small.*

THEOREM 3.3. *We can construct an emulator H with the following properties. Note that H changes as edges in G are deleted.*

1. *H always has $\widetilde{O}(\alpha n)$ edges.*
2. *If $\delta(x,y) = \Omega(\beta)$ then $\delta_H(x,y) \leq (1+\epsilon)\delta(x,y)$.*
3. *Every edge in H has weight at most $O(\beta)$.*
4. *At most $\widetilde{O}(\alpha\beta n)$ edges are inserted into H over all deletions in G.*
5. *If a deletion in G causes edge (x,y) to be inserted into H, then before the insertion, $\delta_H(x,y) = O(\beta)$.*
6. *Given any two vertices x, y, $\delta_H(x,y)$ changes at most $\widetilde{O}(\alpha\beta^2 n)$ times over all deletions in G.*
7. *H can be maintained in $\widetilde{O}(\alpha\beta m)$ time over all deletions in G.*

4 Decremental Single Source Shortest Paths

We now present a $(1+\epsilon)$-approximate algorithm for decremental single source shortest paths ($SSSP$) that has a total update time of $O(\alpha^2\beta^3 n^2) = \widetilde{O}(n^2 \cdot n^{O(1/\sqrt{\log(n)})})$. Our basic approach is to construct our sparse emulator H, and then run King's decremental SSSP algorithm [18] on H (see Section 2.1). The only catch is that we have to show that King's algorithm can efficiently handle edge insertions into H. Also, H only serves as a $(1+\epsilon)$ emulator for distances larger than β (see property 2 of Theorem 3.3), so we use a different (much simpler) approach for small distances.

Recall that the key property of King's decremental algorithm is that we only explore the edges incident on some vertex v when $\delta(s,v)$ changes (s is the source). We show that it is easy to preserve this property under insertions (in addition to deletions) by mimicking the original procedure for deletions. The basic idea is that when we insert an edge (u,v), if $\delta(s,v)$ does not change then we stop. Otherwise, we compute a shortest path tree from v, but we truncate the procedure at any vertex whose shortest distance from s remains unchanged. This intuition leads to the following lemma, which can be viewed also as a simplified version of the output sensitive algorithm of Ramalingam and Reps [20].

LEMMA 4.1. *Given a source s, we can construct a fully dynamic SSSP algorithm with the property that we only explore the edges incident on a vertex v when some update changes the shortest distance from s to v (either by increasing or decreasing it). Every such update causes us to explore the edges of v only a constant number of times.*

Proof. In order to extend King's algorithm [18] to handle insertions, we precisely state what information the decremental version of the algorithm stores,

and then show how to maintain this information under insertions while satisfying the property of Lemma 4.1 (*i.e.* that the edges of a vertex are only scanned if the distance to that vertex changes). Even though our paper focuses on undirected graphs, we show how to extend King's algorithm for directed graphs with positive weights, because this is no more complicated.

Fix some source s. Given any vertex v, let $\delta(v) = \delta(s, v)$. Let $In(v) = \{u \mid (u, v) \in E\}$, and let $Out(v) = \{u \mid (v, u) \in E\}$. Naturally, King's decremental algorithm stores the shortest path tree itself, along with parent/children pointers, and shortest distances. Also, for every vertex v it stores a min-heap $pred(v)$ that contains all vertices in $In(v)$ except for the parent of v: the key of a vertex u in $pred(v)$ is $\delta(u) + w(u, v)$. (Technical note: using a min-heap for $pred(v)$ yields a total update time of $O(md \log(n))$ rather than $O(md)$. King shows how to reduce this to $O(md)$, but for simplicity, we stick to a min-heap).

We now show how to maintain all this information during an Insert(u, v). Note that our algorithm for insertions never actually relies on $pred(v)$, but we must nonetheless show how to maintain $pred(v)$ under insertions so that future deletions can run smoothly.

The algorithm is very simple: we basically compute a shortest path tree from v, except that we stop at any vertex whose distance from s has not decreased. Note that every vertex w starts with a value $\delta(w)$, the shortest distance before the insertion, and that our algorithm modifies $\delta(w)$ if the insertion causes the shortest distance to decrease. Throughout the algorithm we let H refer to a min-heap which is initially empty (H corresponds to the min-heap in Dijkstra's algorithm). See pseudocode below.
The above algorithm maintains all the necessary information, and it is clear that we only explore the edges incident on a vertex y if $\delta(y)$ has decreased. This completes the proof. ∎

Assuming an unweighted graph, this lemma yields a total update time of $O(mn)$ in a decremental setting because the distance from s to any v can increase at most n times. But the lemma yields no good bounds in a fully dynamic setting because the distance from s to v can potentially change many, many times. However, in our specific setting, $\delta_H(x, v)$ changes a total of $\widetilde{O}(\alpha\beta^2 n)$ times (property 6 of Theorem 3.3). Thus, *the edges of any vertex v are only scanned a total of $\widetilde{O}(\alpha\beta^2 n)$ times* (over all deletions in G). The following lemma completes our analysis:

LEMMA 4.2. *If the edges of any vertex v (in H) are scanned at most T times over all deletions in G (in our case $T = \widetilde{O}(\alpha\beta^2 n)$), then the <u>total</u> update time over all vertices is $\widetilde{O}(\alpha\beta n T)$.*

Figure 1: Insertions in King's Algorithm

Insert(u, v):
$H \leftarrow \emptyset$
relax(u, v)
While $H \neq \emptyset$
  $x \leftarrow$ delete_min(H)
  For All $y \in Out(x)$
    relax(x, y)

relax(x, y):
If $\delta(x) + w(x, y) < \delta(y)$
  $\delta(y) \leftarrow \delta(x) + w(x, y)$
  Make x the parent of y in the tree
  Add the previous parent of y to $pred(y)$
  H.insert$(y, \delta(y))$
Else
  Update key of x in $pred(y)$

Proof. In a static setting, the running time would be $\Sigma_v \deg(v) T$, which is easy to bound because $\Sigma_v \deg(v)$ is at most twice the number of edges in the graph. The problem is that in our dynamic setting, the degree of a vertex in H changes over time, and it is hard to bound the sum of different degrees at different times. To overcome this, let $\deg^{\text{MAX}}(v)$ be the total number of edges that are ever incident on v (in H) during all deletions from G (that is, include any edge (u, v) that exists *at some time*). Our total update time is $O(T\Sigma_v \deg^{\text{MAX}}(v))$. But $\Sigma_v \deg^{\text{MAX}}(v) = \widetilde{O}(\alpha\beta n)$ because H starts with $\widetilde{O}(\alpha n)$ edges, and at most $\widetilde{O}(\alpha\beta n)$ new edges are inserted (properties 1 and 4 of Theorem 3.3). This completes the proof. ∎

As mentioned earlier, we maintain two different shortest path trees: one for small distances, and one for larger ones. Firstly, we use King's original algorithm to maintain a decremental shortest path tree in G (from s) up to distance $O(\beta)$ – this takes $O(m\beta)$ time. Secondly, we use our extension of King's algorithm to maintain a shortest path tree from s in H (for all distances). To compute an approximation to some $\delta(s, v)$, we query in both trees, and return the smaller of the two distances. If $\delta(s, v) = O(\beta)$ then the first tree returns an exact distance, and if $\delta(x, v) = \Omega(\beta)$ then by property 2 of Theorem 3.3, the second tree returns a $(1+\epsilon)$ approximation. This yields

THEOREM 4.1. *Given an unweighted, undirected graph and a source s, we can construct a data struc-*

ture that decrementally maintains $(1+\epsilon)$ *approximate shortest distances from* s *while achieving the following bounds: the worst case query time is constant, and the total update time is (with high probability)* $O(\beta m + \alpha^2\beta^3 n^2) = \widetilde{O}(n^2 \cdot n^{O(1/\sqrt{\log(n)})})$

5 Decremental distance oracle

Roditty and Zwick [21] presented a decremental distance oracle up to distance d with a total update time of $O(dmn^{1/k})$ (Theorem 2.2). In this section, we apply our general framework and present the following improvement:

THEOREM 5.1. *Let* G *be an unweighted, undirected graph that undergoes a sequence of edge deletions, and let* $k \geq 2$ *be a fixed integer. We can construct a data structure that decrementally maintains all-pairs* $(2k-1+\epsilon)$ *shortest distances while maintaining the following bounds: the worst case query time is* $O(1)$*, and with high probability, the space requirement is* $O(m + n^{1+1/k})$*, and the total update time is* $\widetilde{O}(\alpha^2\beta^3 n^{2+1/k} + mn^{1/k}\beta) = \widetilde{O}(n^{2+1/k} \cdot n^{O(1/\sqrt{\log(n)})})$.

Our construction is based on the emulator introduced in Section 3. We barely modify the original algorithm of Roditty and Zwick [21], but we must extend the analysis to work in our non-decremental setting. Our presentation is rather technical because the original analysis itself is quite subtle, even in the purely decremental setting. In adapting their proof to our new setting, we face similar technical obstacles.

Roughly speaking, we would like to do the following. We maintain distances smaller than β directly with the decremental distance oracle of Roditty and Zwick (total update time $\widetilde{O}(mn^{1/k}\beta)$), while maintaining distances larger than β using the emulator. More specifically, we run the decremental algorithm of Roditty and Zwick on the emulator rather than on the original graph. Since the emulator has only $\widetilde{O}(n\alpha)$ edges, and the maximal depth is n, the resulting total update time is $\widetilde{O}((\alpha n)(n^{1/k})(n)) = \widetilde{O}(\alpha n^{2+1/k})$.

This approach does not work as stated because the data structure of Roditty and Zwick [21] only supports edge deletions, but we know that edges can be *inserted* into our emulator. Thus, just as with King's SSSP algorithm, we rely on the properties of our emulator (see Theorem 3.3) to extend Roditty and Zwick's decremental algorithm to also work in our partially incremental setting. Before doing so, we need to review their original construction [21].

5.1 Decremental distance oracle of Roditty and Zwick

(See Section 2.2 for a review of definitions). In order to decrementally maintain the static distance oracle of Thorup and Zwick [25], one must, for every vertex w, maintain $C(w)$ (cluster maintenance) and all the $p_i(w)$ (witness maintenance). Recall that the set hierarchy $A_0 \supseteq A_1 \supseteq A_2 \supseteq \cdots \supseteq A_k$ is picked in an oblivious manner, without ever looking at the graph $G = (V, E)$, so we can use the same set hierarchy for all versions of the graph.

Roughly speaking, Roditty and Zwick [21] maintain each $C(w)$ by maintaining a decremental shortest path tree from w up to level $\bar{d} = (2k-1)d$; we can represent a cluster with a tree because if v is in $C(w)$, where $w \in A_i - A_{i+1}$, then so is any vertex on the shortest path from v to w. Now, recall that $v \in C(w)$ implies that $\delta(w, v) < \delta(v, A_{i+1})$. We are in a decremental setting, so both $\delta(w, v)$ and $\delta(v, A_{i+1})$ can only increase. However, the order relation between them can change many times, so vertices can both join and leave $C(w)$.

Since Roditty and Zwick maintain each $C(w)$ using the decremental shortest path tree of King [18] (see Section 2.1), we know that the edges of v are only scanned when the distance to v within some cluster changes (*i.e* $v \in C(w)$ and $\delta(v, w)$ changes). By Lemma 2.1, v is in only $\widetilde{O}(n^{1/k})$ clusters at any given time. Moreover, the distance to v within a cluster can change at most $\bar{d}$ times, so intuitively, we want to argue that the edges of v are scanned at most $\widetilde{O}(n^{1/k}\bar{d})$ times.

Unfortunately, it is not this simple because even though v is in $\widetilde{O}(n^{1/k})$ clusters at any given time, it can belong to different clusters at different times, so it is difficult to bound the overall number of clusters in which it participates. Hence, Roditty and Zwick used a more sophisticated analysis to bound the total number of times that the edges of a given vertex are scanned over all deletions. The proof was implicit in their time analysis, but we provide an explicit proof which is orientated toward our later improvements. In the next section we will show that it is possible to prove a similar lemma in our partially incremental setting.

LEMMA 5.1. *For every* $v \in V$ *and* $0 \leq i \leq k-1$*, the expected total number of times the edges of* v *are scanned, in all trees rooted at vertices of* A_i*, is* $\widetilde{O}(\bar{d}n^{1/k})$.

Proof. Let $w \in A_i \setminus A_{i+1}$. The edges of v are scanned in $C(w)$ once when v joins $C(w)$ and then each time $\delta(v, w)$ changes until v leaves $C(w)$. We first separately analyze the cost of joining new clusters. In the decremental setting, v can only join $C(w)$ if $\delta(v, A_{i+1})$ increases, which can happen at most $\bar{d}$ times. Each time, v joins $\widetilde{O}(n^{1/k})$ clusters (see Lemma 2.1). Thus, the total number of times the edges of v are scanned because of v joining a cluster

is $\widetilde{O}(\bar{d}n^{1/k})$.

We now turn to analyze the case where the distance between v and the cluster center decreases. Let $w_{t,1}, w_{t,2}, \ldots$ be the vertices of A_i arranged in non-decreasing order of distance from v after the t-th deletion. To resolve ties we arrange the vertices in a non-decreasing lexicographic order of $(\delta_t(v,w), \delta_{t+1}(v,w))$. Thus, if w and w' have the same distance from v at time t, and the next deletion increases the distance to w' but not to w, then w appears before w' in the ordering.

It follows from this ordering that for every $v \in V$ and $j \geq 1$, the sequence $\delta_t(v, w_{t,j})$ is non-decreasing. Furthermore, if $\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t,j})$ then also $\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t+1,j})$. Note that $v \in C_t(w_{t,j})$ only if for every $j' < j$ we have that $w_{t,j'}$ is in A_i but not A_{i+1}. Because of how we chose the sets A_i (see Definition 2.2), this implies that $v \in C_t(w_{t,j})$ with probability at most $(1-p)^{j-1}$, where $p = n^{-1/k}$.

Let $I = \{\, (t,j) \mid \delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t,j}) \leq \bar{d} \,\}$. Clearly, the expected number of times the edges of v are scanned, in all trees rooted at vertices of A_i, is at most $\sum_{(t,j)\in I} \Pr[v \in C_t(w_{t,j})]$. For each j, the set I contains at most $\bar{d}$ pairs of the form (t,j). (In other words, the distance to the j-th closest vertex to v increases at most $\bar{d}$ times.) Thus $\sum_{(t,j)\in I} \Pr[v \in C_t(w_{t,j})] \leq \bar{d}\sum_{j\geq 1}(1-p)^{j-1} \leq \bar{d}p^{-1} = \bar{d}n^{1/k}$, as required. ∎

This implies that cluster maintenance requires a total of $\widetilde{O}(mn^{1/k}d)$ time in expectation. It is not hard to show that *witness maintenance can be done as a part of cluster maintenance*. The basic idea is that while maintaining clusters, we can also maintain bunches in the same time bounds because bunches are just inverse clusters (see Definition 2.4). But note that $p_i(v)$ is in $B(v)$, and that in particular $p_i(v) = \operatorname{argmin}_{w\in A_i \bigcap B(v)} \delta(v,w)$. This is easy to return if we store each bunch as a union of k min-heaps – one for each set A_i (technical note: using min-heaps might increase our running time by $O(\log(n))$, but it is easy to overcome this. Details omitted).

5.2 Proof of Theorem 5.1 Similarly to our decremental SSSP algorithm, our construction of decremental distance oracles is based on the sparse emulator H introduced in Section 3 (its main properties are summarized in Theorem 3.3). The emulator H only serves as a $(1+\epsilon)$-emulator for distances larger than β (property 2 of Theorem 3.3), so we maintain distances smaller than β directly with the decremental distance oracle of Roditty and Zwick [21] (total update time $\widetilde{O}(mn^{1/k}\beta)$). For distances larger than β, our approach is to maintain Roditty and Zwick's oracle for the sparse emulator H rather than for the original graph G. We do so by extending their decremental algorithm to also support insertions into H. We focus on cluster maintenance because as described in the previous section, witness maintenance can be done in the same time bounds.

The main technical tool used in Roditty and Zwick's original algorithm is King's decremental SSSP algorithm [18] (see Section 2.1). In Lemma 4.1, we extended King's algorithm to handle insertions in such a way that the edges incident on a vertex v are scanned $O(1)$ times per distance change to v. This allows us to maintain our cluster trees under insertions, but we must now analyze how these trees change over time. In particular, we need to bound the number of times that the edges of a vertex v are scanned by the cluster maintenance.

Let $w \in A_i$. Recall that $v \in C(w)$ if and only if $\delta(w,v) < \delta(v, A_{i+1})$. Thus, whether v is in the cluster of w depends on the order relation between $\delta(v,w)$ and $\delta(v, A_{i+1})$. This can obviously change as a result of either $\delta(v,w)$ or $\delta(v, A_{i+1})$ changing. We deal with these two cases separately. Notice that as opposed to Lemma 5.1, in the following two lemmas we prove our results with high probability and not in expectation. This is necessary because Lemma 4.2 requires a strict (not an expected) upper bound on the number of times the edges of a vertex are scanned.

LEMMA 5.2. *For every $v \in V$ and $0 \leq i \leq k-1$, w.h.p, the total number of times the edges of v are scanned in all trees rooted at vertices of A_i, as a result of a change in $\delta(v, A_{i+1})$, is $\widetilde{O}(n^{1+1/k}\alpha\beta^2)$.*

Proof. Each time that $\delta(v, A_{i+1})$ changes, v can leave and join (w.h.p) a total of at most $\widetilde{O}(n^{1/k})$ clusters (Lemma 2.1), so its edges are scanned at most $\widetilde{O}(n^{1/k})$ times. Thus, we only need to bound the total number of changes to $\delta(v, A_{i+1})$. Consider a vertex s_{i+1} that is added to the graph and is connected to the vertices of A_{i+1} with edges of zero weight. It is easy to see that $\delta(s_{i+1}, v) = \delta(v, A_{i+1})$. Using the same arguments that are used in the proof of Lemma 3.4, it is possible to show that $\delta(s_{i+1}, v)$ changes at most $\widetilde{O}(n\alpha\beta^2)$ times. Thus, the edges of v are scanned at most $\widetilde{O}(n^{1+1/k}\alpha\beta^2)$ times. ∎

LEMMA 5.3. *For every $v \in V$ and $0 \leq i \leq k-1$, w.h.p, the total number of times the edges of v are scanned in all trees rooted at vertices of A_i, as a result of a change in the distance between v and the cluster center (*i.e *$\delta(v,w)$ changes and $v \in C(w)$), is $\widetilde{O}(n^{1+1/k}\alpha\beta^2)$.*

Proof. As in Lemma 5.1, let $w_{t,1}, w_{t,2}, \ldots$ be the vertices of A_i arranged in non-decreasing order of

distance from v after the t-th deletion. Note that for every $j \geq 1$, the sequence $\delta_t(v, w_{t,j})$ is non-decreasing. As in Lemma 5.1, we resolve ties by arranging the vertices in a non-decreasing lexicographic order of $(\delta_t(v,w), \delta_{t+1}(v,w))$. This ensures that if a *decremental* update at time $t+1$ causes $\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t,j})$ then it also causes $\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t+1,j})$. However, this is *not true of incremental updates*, so we must do more work than in Lemma 5.1.

Let $I^1 = \{(t,j) \mid \delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t,j})\}$ and let $I^2 = \{(t,j) \mid \delta_{t+1}(v, w_{t,j}) < \delta_t(v, w_{t,j})\}$. Notice that I^1 corresponds to $\delta(v, w_{t,j})$ increasing (as a result of a deletion), and I^2 corresponds to $\delta(v, w_{t,j})$ decreasing (as a result of an insertion). The number of times the edges of v are scanned (as a result of a change in $\delta(v,w)$, where w is a cluster center), in all trees rooted at vertices of A_i, is at most:

$$\sum_{(t,j)\in I^1} 1_{C_t(w_{t,j})}(v) + \sum_{(t,j)\in I^2} 1_{C_{t+1}(w_{t,j})}(v),$$

where 1_S is the indicator function for the set S, that is, $1_S(x) = 1$ if $x \in S$ and $1_S(x) = 0$ if $x \notin S$. We start with the second sum, which corresponds to edge insertions. From Theorem 3.3 (property 4) we know that the number of edge insertions is at most $\widetilde{O}(\alpha\beta n)$. Also, at any given time, v is (w.h.p) in at most $\widetilde{O}(n^{1/k})$ clusters (Lemma 2.1), so even if after every edge insertion v gets closer to every $w_{t,j}$, we have $\sum_{(t,j)\in I^2} 1_{C_{t+1}(w_{t,j})}(v) \leq \widetilde{O}((\alpha\beta n)(n^{1/k})) = \widetilde{O}(\alpha\beta n^{1+1/k})$, as required.

We now bound the first sum. *Fix an index j* and let $t^1, \ldots, t^\ell$ be the times in which an *incremental* update took place and the distance between v and the jth closest vertex in A_i decreased right after it. Let t^0 be the time before the first update and let $t^{\ell+1}$ be the time after the last update. Notice that for every $1 \leq r \leq \ell$ it holds that $\delta_{t^r-1}(v, w_{t^r-1,j}) \geq \delta_{t^r}(v, w_{t^r,j})$, and that for every other time t ($t \neq t^r$ for any r), we have $\delta_{t-1}(v, w_{t-1,j}) \leq \delta_t(v, w_{t,j})$.

Let $\Delta(v,j)$ be the total amount of distance changes between v and the jth closest vertex from A_i. Since I^1 corresponds to *decremental* updates, the tie breaking property of Lemma 5.1 must hold ($\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t,j})$ implies $\delta_t(v, w_{t,j}) < \delta_{t+1}(v, w_{t+1,j})$); thus, just as in Lemma 5.1, $\Delta(v,j)$ is an upper bound on the number of pairs in I^1 of the form $(\cdot, j)$. Let $\Delta^{\text{INC}}(v,j)$ (resp. $\Delta^{\text{DEC}}(v,j)$) be the total distance changes caused by incremental (resp. decremental) updates. From the definition of $t^1, \ldots, t^\ell$ it follows that:

$$\Delta^{\text{INC}}(v,j) = \sum_{r=1}^{\ell} (\delta_{t^r-1}(v, w_{t^r-1,j}) - \delta_{t^r}(v, w_{t^r,j}))$$

$$\Delta^{\text{DEC}}(v,j) \leq \sum_{r=0}^{\ell} (\delta_{t^{r+1}-1}(v, w_{t^{r+1}-1,j}) - \delta_{t^r}(v, w_{t^r,j}))$$

Using a simple manipulation of the second summation we get that:

$$\Delta^{\text{DEC}}(v,j) \leq$$

$$\delta_{t^{\ell+1}-1}(v, w_{t^{\ell+1}-1,j}) - \delta_{t^0}(v, w_{t^0,j}) + \Delta^{\text{INC}}(v,j)$$

We can now bound $\Delta(v,j)$ as follows:

$$\Delta(v,j) = \Delta^{\text{INC}}(v,j) + \Delta^{\text{DEC}}(v,j) \leq$$

$$\delta_{t^{\ell+1}-1}(v, w_{t^{\ell+1}-1,j}) - \delta_{t^0}(v, w_{t^0,j}) + 2\Delta^{\text{INC}}(v,j)$$

By Theorem 3.3, $\delta_{t^r-1}(v, w_{t^r-1,j}) - \delta_{t^r}(v, w_{t^r,j}) \leq O(\beta)$ for every $1 \leq r \leq \ell$ (property 5), and the number of incremental updates is $\widetilde{O}(\alpha\beta n)$ (property 6). Thus, $\Delta^{\text{INC}}(v,j) \leq \widetilde{O}(\alpha\beta^2 n)$, so

$$\Delta(v,j) \leq \widetilde{O}(\alpha\beta^2 n) + \delta_{t^{\ell+1}-1}(v, w_{t^{\ell+1}-1,j}) \leq$$

$$\widetilde{O}(\alpha\beta^2 n) + n = \widetilde{O}(\alpha\beta^2 n)$$

Notice that by Lemma 2.1, (w.h.p) for every time t we have that $1_{C_t(w_{t,j})}(v) = 0$ for every $j > \Theta(n^{1/k}\log n)$. Thus, $\sum_{(t,j)\in I^1} 1_{C_t(w_{t,j})} \leq \sum_{j=1}^{j=\Theta(n^{1/k}\log n)} \Delta(v,j) \leq \widetilde{O}(\alpha\beta^2 n^{1+1/k})$. ∎

COROLLARY 5.1. *The total time needed in order to maintain all the clusters (and hence also all the witnesses) while the graph G undergoes a sequence of edge deletions is $\widetilde{O}(n^{2+1/k}\alpha^2\beta^3)$. This stems trivially from Lemma 4.2 and the two lemmas above.*

6 Future Work

We believe that our general framework can lead to further improvements. Firstly, it can potentially be applied to decremental SSSP in *directed* graphs, because although no sparse emulator of a directed graph can preserve *all* shortest distances, there do exist emulators that preserve *long* distances, which were our main obstacle in the first place. Secondly, the key to both our algorithms was an emulator which preserves the following property of a purely decremental setting: the shortest distance between two vertices can only change around n times. However, it fails to preserve the more general property that for any d, the shortest distance between two vertices can only change around d times before this distance exceeds d. Developing a spanner with this property would not only be helpful for small-diameter graphs, but it would also lead to improved decremental algorithms for *general* graphs.

References

[1] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28:1167–1181, 1999.

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

[3] B. Awerbuch, B. Berger, L. Cowen, and D. Peleg. Near-linear time construction of sparse neighborhood covers. *SIAM Journal on Computing*, 28:263–277, 1999.

[4] S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In *Proc. of 34th STOC*, pages 117–123, 2002.

[5] S. Baswana, R. Hariharan, and S. Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *SODA*, pages 394–403, 2003.

[6] S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007.

[7] A. Bernstein. Fully dynamic approximate all-pairs shortest paths with query and close to linear update time. In *Proc. of the 50th FOCS*, pages 50–60, Atlanta, GA, 2009.

[8] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28:210–236, 1999.

[9] E. Cohen and U. Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

[10] C. Demetrescu and G. F. Italiano. A new approach to dynamic all pairs shortest paths. *JACM: Journal of the ACM*, 51, 2004.

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

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

[13] D. Dor, S. Halperin, and U. Zwick. All pairs almost shortest paths. *SIAM Journal on Computing*, 29:1740–1759, 2000.

[14] M. Elkin. Computing almost shortest paths. In *Proc. of 20th PODC*, pages 53–62, 2001.

[15] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM J. Computing*, 33(3):608–631, 2004.

[16] S. Even and Y. Shiloach. An on-line edge deletion problem. *J. ACM*, 28(1):1–4, 1981.

[17] M. R. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *FOCS*, pages 664–672, 1995.

[18] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. of the 40th FOCS*, pages 81–91, 1999.

[19] 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*, pages 268–277. Springer, 2001.

[20] G. Ramalingam and T. W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305, 1996.

[21] L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. In *Proc. of the 45th FOCS*, pages 499–508, Rome, Italy, 2004.

[22] L. Roditty and U. Zwick. On dynamic shortest paths problems. In *Proc. of the 12th ESA*, pages 580–591, 2004.

[23] M. Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *SWAT: Scandinavian Workshop on Algorithm Theory*, 2004.

[24] M. Thorup. Worst-case update times for fully-dynamic all-pairs shortest paths. In *Proc. of the 37th STOC*, pages 112–119, 2005.

[25] M. Thorup and U. Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[26] Mikkel Thorup and Uri Zwick. Spanners and emulators with sublinear distance errors. In *Proc. of the 17th SODA*, pages 802–809, Miami, Florida, 2006.

[27] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.