Maintaining All-Pairs Approximate Shortest Paths under Deletion of Edges

Surender Baswana* Ramesh Hariharan† Sandeep Sen‡

Abstract

We present a hierarchical scheme for efficiently maintaining all-pairs approximate shortest-paths in undirected unweighted graphs under deletions of edges.

An α-approximate shortest-path between two vertices is a path of length at-most α times the length of the shortest path. For maintaining α-approximate shortest paths for all pairs of vertices separated by distance $\leq d$ in a graph of n vertices, we present the first $o(nd)$ update time algorithm based on our hierarchical scheme. In particular, the update time per edge deletion achieved by our algorithm is $\tilde{O}(\min\{\sqrt{n}d, (nd)^{2/3}\})$ for 3-approximate shortest-paths, and $\tilde{O}(\min\{\sqrt[3]{n}d, (nd)^{4/7}\})$ for 7-approximate shortest-paths. For graphs with $\theta(n^2)$ edges, we achieve even further improvement in update time : $\tilde{O}(\sqrt{nd})$ for 3-approximate shortest-paths, and $\tilde{O}(\sqrt[3]{nd^2})$ for 5-approximate shortest-paths.

For maintaining all-pairs approximate shortest-paths, we improve the previous $\tilde{O}(n^{3/2})$ bound on the update time per edge deletion for approximation factor ≥ 3. In particular, update time achieved by our algorithm is $\tilde{O}(n^{10/9})$ for 3-approximate shortest-paths, $\tilde{O}(n^{14/13})$ for 5-approximate shortest-paths, and $\tilde{O}(n^{28/27})$ for 7-approximate shortest-paths.

All our algorithms achieve optimal query time and are simple to implement.

1 Introduction

The problem of computing all-pairs shortest paths (*APSP*) in a given graph is one of the most fundamental graph problem in computer science. To this date, there is no sub-cubic time combinatorial algorithm for computing APSP. However, in recent years, a number of combinatorial algorithms have been proposed for computing all-pairs *approximate* shortest paths (*APASP*) in undirected graphs that achieve sub-cubic running times. Dor et al. [3] presented an $\tilde{O}(n^2)$ time[1] algorithm for computing 3-approximate shortest paths in undirected unweighted graphs. Cohen and Zwick [2] extended their result for weighted undirected graphs with matching time bounds. The space requirement of these algorithms is $O(n^2)$. Thorup and Zwick [7] gave a $O(n^{1+\frac{1}{k}})$ size data-structure that can report $(2k-1)$-approximate distance between any two vertices in $O(k)$.

There are a number of applications that require efficient solutions of the APASP problem for a dynamic graph. In these applications, an initial graph is given, followed by an online sequence of queries interspersed with updates that can be insertion or deletion of edges. We have to carry out the updates and answer the queries online in an efficient manner. The goal of a dynamic graph algorithm is to update the solution efficiently after the dynamic changes, rather than having to recompute it from scratch each time.

In this paper we present efficient dynamic (decremental[2]) algorithms for maintaining APASP in undirected unweighted graphs under edge deletions.

1.1 Previous work and our techniques Let $G(V, E)$ be an undirected, unweighted graph. A breadth-first-search (BFS) tree rooted at a vertex $u \in V$ stores distance information with respect to the vertex u. In addition to the explicit distance information : *shortest-paths from u to all the vertices*, the BFS tree also stores the following distance information implicitly : *the shortest-path between any two vertices that passes through u.* Due to these properties, a BFS tree is employed as a fundamental data-structure by the existing dynamic algorithms [1, 6] for maintaining all-pairs approximate/exact shortest-paths. In order to maintain APASP dynamically, these algorithms require an efficient solution for the following sub-problem :

$\mathcal{P}$: *With respect to a given set $X \subset V$, maintain distance information (explicit/implicit) for all-pairs $u, v \in V$ separated by distance $\in [1, d]$ for some $d \leq n$.*

For solving the problem $\mathcal{P}$, a collection of BFS trees of depth d is maintained under deletion of edges at each $u \in X$. (Let B_u^d denote a BFS tree of depth d rooted

*Dept. of Comp. Sc. and Engg., I.I.T. Delhi, Hauz Khas, New Delhi. Work was supported in part by a fellowship from Infosys Technologies Ltd., Bangalore. Email : sbaswana@cse.iitd.ernet.in

†Dept. of Comp. Sc. and Automation, I.I.Sc., Bangalore. Email : ramesh@csa.iisc.ernet.in

‡Dept. of Comp. Sc. and Engg., I.I.T. Delhi, Hauz Khas, New Delhi. Work was supported in part by an IBM UPP award. Email : ssen@cse.iitd.ernet.in

[1] m and n denote respectively the number of edges and vertices in a given graph. $\tilde{O}$ suppresses polylog factors

[2] the dynamic algorithm if the updates are only edge-deletions.

at u). It turns out that the update time required for problem $\mathcal{P}$ is dependent on the following two parameters : the number of vertices $\nu(B_u^d)$, and the number of edges $\mu(B_u^d)$ in the sub-graph induced by each BFS tree $B_u^d, u \in X$. In particular, for maintaining (*explicit* distance information with respect to set X) shortest-paths from each $u \in X$ to all the vertices of the set V that lie within distance d from u, the total update time required is $\sum_{u\in X} \mu(B_u^d)$. Whereas, for maintaining (*implicit* distance information) the shortest-paths passing through any vertex of the set X for all pairs separated by distance $\leq d$, it requires $\sum_{u\in X}(\nu(B_u^d))^2$ total update time.

Potentially in the worst case $\mu(B_u^d)$ can be as large as $\theta(m)$ and $\nu(B_u^d)$ can be as large as $\theta(n)$. Therefore, in order to improve the update time for the problem $\mathcal{P}$, and hence for the problem APASP, the following ideas come to mind.

- Is it possible to solve the problem $\mathcal{P}$ by keeping depth-d BFS trees on vertices of some smaller-size set $S \subset X$ (of size $o(|X|)$)?

- Is there some other alternative t for depth bounded BFS tree B_u^d that has $o(m)$ bound on $\mu(t)$ and $o(n)$ bound on $\nu(t)$?

While it appears difficult for any of the above ideas to succeed individually, they can be combined in the following way : *Build and maintain BFS trees of depth $2d$ on vertices of a set S of size $o(|X|)$, called* the set of special vertices*, and for each remaining vertex $u \in X \backslash S$, maintain BFS tree (denoted by B_u^S) rooted at u, and containing all the vertices that lie closer to u than the nearest special vertex, say $\mathcal{N}(u,S)$.*

Along the above lines, we present a 2-level scheme (and its generalization to k-levels) for maintaining all-pairs approximate shortest-paths under deletion of edges. It can be seen that unlike the tree B_u^d, the new BFS tree B_u^S might not contain all the vertices lying within distance d from u. In order to ensure that our scheme leads to a solution of problem $\mathcal{P}$, we use the following observation carefully. If v is a vertex lying within distance d from u but not present in B_u^S, an *approximate* distance from u to v can be extracted from the tree rooted at the nearest special vertex $\mathcal{N}(u,S)$. This is because (by triangle inequality) the distance from $\mathcal{N}(u,S)$ to v is at most twice the distance from u to v.

For our hierarchical scheme to lead to improved update time, it is crucial that we establish sub-linear upper bounds on $\mu(B_u^S)$ and $\nu(B_u^S)$. We show that if the set S is formed by picking each vertex independently with *suitable* probability, then $\mu(B_u^S) = \tilde{O}(m/|S|)$ and $\nu(B_u^S) = \tilde{O}(n/|S|)$ with probability arbitrarily close to 1. A straightforward random sampling of vertices would have sufficed to bound the distance of u to $\mathcal{N}(u,S)$, which is similar to the previous use of random sampling in the context of dynamic graph algorithms [5, 1]. However, we are not aware of any work where there was a need for bounding the number of edges based on vertex sampling.

Previously, a hierarchical distance reporting scheme was given by Thorup and Zwick [7] for building a static data-structure to answer approximate distance-reporting queries. However, there does not seem to be any efficient way to dynamize their scheme for answering the queries under deletion of edges.

We use our hierarchical scheme to achieve better update time for the following problems.

Maintaining shortest paths up to length $\leq$ d : Given an undirected unweighted graph G under deletion of edges, and a distance parameter d; the objective is to efficiently maintain shortest paths for all-pairs separated by distance $\leq d$. Using the data-structure from [4], the existing upper bound on the amortized update time for this problem is $O(nd)$ per edge deletion. We present an algorithm that achieves $o(nd)$ update time and maintains α-approximate shortest-paths (the path with length at most α times the length of the shortest path). In particular, our algorithm requires $\tilde{O}(\min(\sqrt{n}d, (nd)^{2/3})$ update time for maintaining 3-approximate shortest paths, and $\tilde{O}(\min(\sqrt[3]{n}d, (nd)^{4/7}))$ update time for maintaining 7-approximate shortest paths for all-pairs separated by distance $\leq d$.

We present another algorithm that achieves even better bound for dense graphs. For graphs with $m = \theta(n^2)$, it requires $\tilde{O}(\sqrt{nd})$ update time for maintaining 3-approximate shortest paths, and $\tilde{O}(\sqrt[3]{nd^2})$ update time for maintaining 5-approximate shortest paths for all-pairs separated by distance $\leq d$.

Maintaining all-pairs shortest paths : Given an undirected, unweighted graph under deletion of edges, the aim is to efficiently maintain all-pairs approximate shortest-paths. The previous best algorithm [1] requires $\tilde{O}(n^2/\sqrt{m})$ amortized update time ($\theta(n^{3/2})$ in worst case) for maintaining all-pairs 2-approximate shortest-paths. We present an algorithm that gives improved update time (approaching $\tilde{O}(n)$) for higher approximation factor). The update time achieved is : $\tilde{O}(n^{10/9})$ for 3-approximate shortest-paths, $\tilde{O}(n^{14/13})$ for 5-approximate shortest-paths, $\tilde{O}(n^{28/27})$ for 7-approximate shortest-paths. Like the previous best decremental algorithm [1] for APASP, the query reporting has one sided error [3] with probability $< \frac{1}{n}$.

Our algorithms achieve optimal query time : re-

[3] The answer to any query may be incorrect in the sense that it may not report path/distance when there exists one.

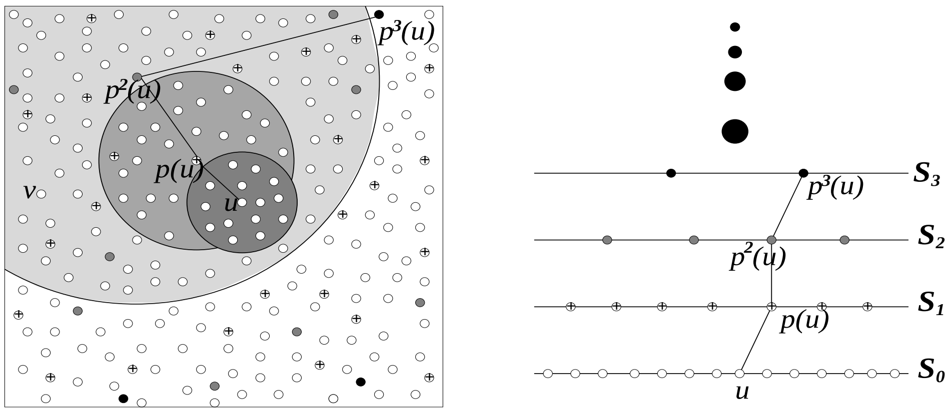

Figure 1: Hierarchical scheme for maintaining approximate distance

porting approximate distance between a pair of vertices in $O(1)$ time, and reporting approximate shortest-path in time proportional to the length of the path. Though we present algorithms for maintaining approximate distances only, these algorithms can be adapted to report approximate shortest paths also (details omitted from this extended abstract).

Organization of the paper : After giving description of various notations used in this paper, we present our hierarchical distance maintaining scheme in section 3.

In section 4, we describe a random sampling scheme to form a set $S \subset V$ that establishes bounds on $\mu(B_u^S)$ and $\nu(B_u^S)$ in terms of the size $|S|$. This is followed by an algorithm for maintaining a BFS tree B_u^S under deletion of edges.

In section 5, we present an improved decremental algorithm for maintaining approximate distances for all-pairs of vertices that are separated by distance $\leq d$, for some $d \leq n$. This algorithm is suitable for maintaining approximate distances for pairs of vertices separated by *small* distances.

In section 6, we present an algorithm that maintains all-pairs approximate distances implicitly (by maintaining witnesses of approximate distances for each pair). This algorithm improves the update time for maintaining approximate distances for all-pairs separated by distance $\in [d, n]$. The algorithm is suitable for maintaining approximate distances for pairs of vertices separated by *long* distances.

In section 7, we suitably combine the algorithms of section 5 and 6 in order to get improved bounds on the update time for maintaining all-pairs approximate distances.

2 Notations

For an undirected graph $G(V, E)$, $S \subset V$, and a distance parameter $d \leq n$,

- $deg(u)$: degree of vertex $u \in V$..
- $\delta(u, v)$: distance between u and v.
- $\mathcal{N}(v, S)$: the vertex of the set $S \subset V$ nearest to v.
- B_v^d : The BFS tree of depth d rooted at $v \in V$.
- B_v^S : The BFS tree of depth $(\delta(u, \mathcal{N}(u, S)) - 1)$ rooted at v
- $B_v^{d,S}$: The BFS tree of depth $\min\{d, \delta(v, \mathcal{N}(v, S)) - 1\}$ rooted at v.
- $\mu(t)$: the number of edges in the sub-graph (of G) induced by the tree t.
- $\nu(t)$: the number of vertices in tree t.
- For a sequence $\{S_0, S_1, \cdots S_{k-1}\}, S_i \subset V$, and a vertex $u \in S_0$, we define
  $p^0(u) = u$.
  $p^{i+1}(u) =$ the vertex from set S_{i+1} nearest to $p^i(u)$.
- $\mathcal{F}^d$: The set of BFS trees from the set $\mathcal{F}$ having depth d.
- $\overline{\alpha}$: the smallest integer of the form 2^i which is greater than α.

3 Hierarchical Distance Maintaining Scheme

Based on the idea of "keeping *many small trees, and a few large trees*", we define a k-level hierarchy for maintaining approximate distances as follows.

Let $\mathcal{S} = \{S_0, S_1, \cdots, S_{k-1} : S_i \subset V, |S_{i+1}| < |S_i|\}$ be a sequence. For a given distance parameter $d \leq n$, let $\mathcal{F}_i$ be the set $\{B_u^{2^i d, S_{i+1}} : u \in S_i\}$ of BFS trees for $i < k-1$, and $\mathcal{F}_{k-1}$ be the set of BFS trees of depth $2^{k-1}d$ rooted at each $u \in S_{k-1}$. The k-level hierarchy $\mathcal{H}_d^k$ induced by the sequence $\mathcal{S}$ is the set $\{(S_0, \mathcal{F}_0), (S_1, \mathcal{F}_1), \cdots, (S_{k-1}, \mathcal{F}_{k-1})\}$.

Let v be a vertex within distance d from u. If v is present in B_u^{d,S_1}, we can report exact distance between them. Otherwise, (as will soon become clear) we can extract the approximate distance between u and v from the collection of the BFS trees rooted at the vertices $u, p(u), \cdots, p^{k-1}(u)$ (see the Figure 1). The following Lemma is the basis for estimating the distance between two vertices using the hierarchy $\mathcal{H}_d^k$.

LEMMA 3.1. *Given a hierarchy $\mathcal{H}_d^k$, if $j < k-1$ is such that v is not present in any of the BFS trees $\{B_{p^i(u)}^{S_{i+1}} | 0 \le i \le j\}$, then*
$\delta(p^{i+1}(u), p^i(u)) \le 2^i \delta(u,v)$ and
$\delta(p^{i+1}(u), v) \le 2^{i+1}\delta(u,v)$, for all $i \le j$.

Proof. We give a proof by induction on i.
Base Case $(i = 0)$ **:** Since v is not present in $B_u^{S_1}$, so the vertex $p(u)$ must be lying equidistant or closer to u than v. Hence $\delta(p(u), u) \le \delta(u,v)$. Using triangle inequality, it follows that $\delta(p(u), v) \le \delta(p(u), u) + \delta(u,v) = 2\delta(u,v)$.
Induction Hypothesis :
$\delta(p^{i+1}(u), p^i(u)) \le 2^i\delta(u,v)$, and
$\delta(p^{i+1}(u), v) \le 2^{i+1}\delta(u,v)$, for all $i < l$.
Induction Step $(i = l)$ **:** if $v \notin B_{p^l(u)}^{S_{l+1}}$, then the distance between p^{l+1} and $p^l(u)$ must not be longer than $\delta(p^l(u), v)$, which is less than $2^l\delta(u,v)$ (using induction hypothesis).

Now using triangle inequality (see the Figure 2) we can bound $\delta(p^{l+1}(u), v)$ as follows.

$$\begin{aligned}
\delta(p^{l+1}(u), v) &\le \delta(p^{l+1}(u), p^l(u)) + \delta(p^l(u), v) \\
&\le 2^l\delta(u,v) + \delta(p^l(u), v) \\
&\le 2^l\delta(u,v) + 2^l\delta(u,v) \quad \{\text{ using I.H.}\} \\
&= 2^{l+1}\delta(u,v)
\end{aligned}$$

Since the depth of a BFS tree at $(k-1)$th level of hierarchy $\mathcal{H}_d^k$ is $2^{k-1}d$, therefore the following corollary holds true.

COROLLARY 3.1. *If $\delta(u,v) \le d$, then there is some $p^i(u), i < k$ such that v is present in the BFS tree rooted at $p^i(u)$ in the hierarchy $\mathcal{H}_d^k$.*

COROLLARY 3.2. *Given a hierarchy $\mathcal{H}_d^k$, if $j < k-1$ is such that v is not present in any of the BFS trees $\{B_{p^i(u)}^{S_i} | 0 \le i \le j\}$, then $\delta(p^{i+1}(u), u) \le (2^{i+1} - 1)\delta(u,v)$, for all $i \le j$.*

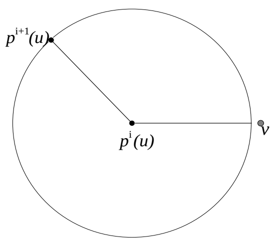

Figure 2: Bounding the approximate distance between $p^{i+1}(u)$ and v

Proof. Using simple triangle inequality, it follows that

$$\begin{aligned}
\delta(p^{i+1}(u), u) &\le \sum_{l \le i} \delta(p^{l+1}(u), p^l(u)) \\
&\le \sum_{l \le i} 2^l\delta(u,v) = (2^{i+1} - 1)\delta(u,v)
\end{aligned}$$

It follows from the Lemma 3.1 and the Corollary 3.2 that if l is the smallest integer such that v is present in the BFS tree rooted at $p^l(u)$ in the hierarchy $\mathcal{H}_d^k$, then we can report $\delta(p^l(u), u) + \delta(p^l(u), v)$ as an approximate distance between u and v. Along these lines in section 5, we present an improved decremental algorithms for maintaining approximate distance for all-pairs separated by distance $\le d$.

4 Bounding the Size of $\mathbf{B_u^{d,S}}$ under Edge-deletions

In this section we present a scheme based on random sampling to find a set $S \subset V$ of vertices that will establish a bound on the number of vertices $(\nu(B_u^S))$ as well as the number of edges $(\mu(B_u^S))$ induced by B_u^S under deletion of edges. Since $B_u^{d,S} \subset B_u^S$, so these upper bounds also hold for $B_u^{d,S}$.

Build the set S of vertices by picking each vertex from V independently with probability $\frac{n^c}{n}$. The expected size of S is $O(n^c)$. Consider an ordering of vertices V according to their levels in the BFS tree B_u^S (see Figure 3). The set of vertices lying at higher levels than the nearest sampled vertex in this ordering is what constitutes the BFS tree B_u^S. It follows from elementary probability that the expected size of this set (and hence $\nu(B_u^S)$) is $\frac{n}{n^c}$. Moreover, $\nu(B_u^S)$ is less than $\frac{4n\ln n}{n^c}$ with probability $> 1 - \frac{1}{n^4}$. Now as the edges are being deleted, the levels of the vertices in the tree B_u^S may change, and so will the ordering of the vertices. There will be a total of m such orderings during the entire course of edge deletions. Since the vertices are picked randomly and independently, therefore, the upper bound of $\frac{4n\ln n}{n^c}$ holds for $\nu(B_u^S)$ with probability $(1 - \frac{1}{n^4})$ for any of these orderings. So we can conclude

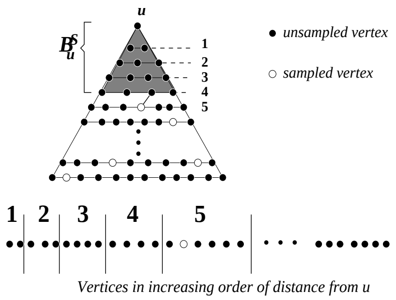

Figure 3: Bounding the size of BFS tree B_u^S

that $\nu(B_u^S)$, the number of vertices of tree B_u^S never exceeds $(\frac{4n \ln n}{n^c})$ during the entire course of edge deletions with probability $> 1 - \frac{1}{n^2}$.

To bound the number of edges induced by B_u^S, consider the following scheme. Pick every edge independently with probability $\frac{n^c}{m}$. The set S consists of the end points of the edges sampled. Clearly expected size of S is $O(n^c)$. Consider an ordering of the edges according to their level in B_u^S (level of an edge is defined as the minimum of the levels of its end points). Along the lines of arguments given above (for bounding the the number of vertices of B_u^S), it can be shown that $\mu(B_u^S)$, the number of edges induced by B_u^S remains $\leq \frac{4m \ln n}{n^c}$ with probability $> 1 - \frac{1}{n^2}$ during the entire course of edge deletions.

Note that in the sampling scheme to bound the number of vertices of tree B_u^S, a vertex v is picked with probability $\frac{n^c}{n}$. Whereas in the sampling scheme for bounding the number of edges in the sub-graph induced by B_u^S, a vertex v is picked with probability $\frac{deg(v) \cdot n^c}{m}$. It can thus be seen that both the bounds can be achieved simultaneously by the following random sampling scheme :

$\mathcal{R}(c)$: Pick each vertex $v \in V$ independently with probability $\frac{n^c}{n} + \frac{deg(v) \cdot n^c}{m}$.

It is easy to see that the expected size of the set formed by the sampling scheme $\mathcal{R}(c)$ will be $O(n^c)$.

THEOREM 4.1. *Given an undirected unweighted graph $G(V, E)$, a $c \in (0, 1)$, a distance parameter d; a set S of size $O(n^c)$ vertices can be found that will ensure the following bound on the number of vertices and number of edges in the sub-graph of G induced by $B_u^{d,S}$.*

$$\nu(B_u^{d,S}) = O\left(\frac{n \ln n}{n^c}\right), \qquad \mu(B_u^{d,S}) = O\left(\frac{m \ln n}{n^c}\right)$$

with probability $\Omega(1 - \frac{1}{n^2})$ during the entire sequence of edge deletions.

4.1 Maintaining the BFS tree $\mathbf{B_u^{d,S}}$ under edge deletion

Even and Shiloach [4] gave an algorithm for maintaining a depth-d BFS tree in an undirected graph.

LEMMA 4.1. (EVEN, SHILOACH [4]) *Given a graph under deletion of edges, a BFS tree $B_u^d, u \in V$ can be maintained in $O(d)$ amortized time per edge deletion.*

For maintaining a $B_u^{d,S}$ tree under edge deletions, we shall use the same algorithm of [4] with the modification that whenever the depth of $B_u^{d,S}$ has to be increased (due to recent edge deletion), we grow the tree to its new level $\min\{d, \delta(u, \mathcal{N}(u, S)) - 1\}$. We analyze the total update time required for maintaining $B_u^{d,S}$ as follows.

There are two computational tasks : one extending the level of the tree, and another that of maintaining the levels of the vertices in the tree $B_u^{d,S}$ under edge deletions. For the first task, the time required is bounded by the edges of the new level introduced which is $O(\mu(B_u^{d,S}))$. For the second task, we give a variant of the proof of Even and Shiloach [4] (for details, please refer [4]). The running time is dominated by the processing of the edges in this process. Whenever an edge is processed, level of one of its end-point is going to fall down by at least one unit. The processing cost of an edge can thus be charged to the level from which it has fallen. Clearly the maximum number of edges passing a level i is bounded by $\mu(B_u^{d,S})$. The number of levels in the tree $B_u^{d,S}$ is $\min\{d, \nu(B_u^{d,S})\}$. Thus the total cost for maintaining the BFS tree $B_u^{d,S}$ over the entire sequence of edge deletions is $O(\mu(B_u^{d,S}) \cdot \min\{d, \nu(B_u^{d,S})\})$.

LEMMA 4.2. *Given an undirected unweighted graph $G(V, E)$ under edge deletions, a distance parameter d, and a set $S \subset V$; a BFS tree $B_u^{d,S}$ can be maintained in*

$$O\left(\frac{\mu(B_u^{d,S})}{m} \cdot \min\{d, \nu(B_u^{d,S})\}\right)$$

amortized update time per edge deletion.

4.2 Some Technical Detail

As the edges are being deleted, we need an efficient mechanism to detect any increase in the depth of tree $B_u^{d,S}$. We outline one such mechanism as follows.

For every vertex $v \notin S$, we keep a count $C[v]$ of the vertices of the S that are neighbors of v. It is easy to maintain $C[u], \forall u \in V$ under edge-deletions. We use the count $C[v]$ in order to detect any increase in the depth of a tree $B_u^{d,S}$ as follows. Note that when depth of a tree $B_u^{d,S}$ is less than d, there has to be at-least one vertex w at leaf-level in $B_u^{d,S}$ with $C[w] \geq 1$ (as an indicator that the vertex $p(u)$ is at next level). Thus whenever due to an edge deletion there is no vertex w at leaf level with $C[w] \geq 1$, we grow the BFS tree $B_u^{d,S}$ beyond its previous level until either depth becomes d or we reach some vertex w' with $C[w'] \geq 1$.

Another technical issue is that when an edge e_{uv} is deleted, we must update only those trees which contain u and v. For this purpose, we maintain for each vertex, a set of roots of all the BFS trees containing it. We maintain this set using any dynamic search tree.

5 Improved Decremental Algorithm for APASP up to Distance d

Let $\{(S_0, \mathcal{F}_0), (S_1, \mathcal{F}_1), \cdots, (S_{k-1}, \mathcal{F}_{k-1})\}$ be a k-level hierarchy $\mathcal{H}_d^k$ with $S_0 = V$ and $n^{c_i} = |S_i|$, where each $c_i, i < k$ is a fraction to be specified soon. Each set $S_i, i > 0$ is formed by picking the vertices from set V using the random sampling scheme $\mathcal{R}$ mentioned in section 4.

To report distance from u to v, we start form the level 0. We first inquire if v lies in $B_u^{S_1}$. If v does not lie in the tree, we move to the first level and inquire if v lies in $B_{p(u)}^{S_2}$. It follows from the Corollary 3.1 that if $\delta(u,v) \leq d$, then proceeding in this way, we eventually find a vertex $p^l(u), l < k-1$ in the hierarchy $\mathcal{H}_d^k$ such that v is present in the BFS tree rooted at $p^l(u)$. (See the Figure 1). We then report the sum of distance from $p^i(u)$ to both u and v.

Algorithm for reporting approximate distance using $\mathcal{H}_d^k$

Distance(u, v)
{ $D \longleftarrow 0; l \longleftarrow 0$
While $(v \notin B_{p^l(u)}^{S_{l+1}} \wedge l < k-1)$ **do**
{
If $u \in B_{p^l(u)}^{S_{l+1}}$, **then** $D \leftarrow \delta(p^l(u), u)$,
$D \leftarrow D + \delta(p^l(u), p^{l+1}(u))$
$l \leftarrow l + 1;$
}
If $v \notin B_{p^l(u)}^{S_{l+1}}$, **then** *"$\delta(u,v)$ is greater than d"*,
else return $\delta(p^l(u), v) + D$
}

The approximation factor ensured by the above algorithm can be bounded as follows.

It follows from the Corollary 3.2 that the final value of D in the algorithm given above is bounded by $(2^l - 1)\delta(u,v)$, and it follows from Lemma 3.1 that $\delta(p^l(u), v)$ is bounded by $2^l\delta(u,v)$. Since $l \leq k-1$, therefore the distance reported by the algorithm is bounded by $(2^k - 1)\delta(u,v)$ if v is at distance $\leq d$.

Lemma 5.1. *Given an undirected unweighted graph $G(V,E)$, and a distance parameter d. If α is the desired approximation factor, then there exists a hierarchical scheme $\mathcal{H}_d^k$ with $k = \log_2 \overline{\alpha}$, that can report α-approximate shortest distance between any two vertices separated by distance $\leq d$, in time $O(k)$.*

Update time for maintaining the hierarchy $\mathcal{H}_{\mathbf{k}}^{\mathbf{d}}$: The update time per edge deletion for maintaining the hierarchy $\mathcal{H}_k^d$ is the sum total of the update time for maintaining the set of BFS trees $\mathcal{F}_i, i \leq k-1$.

Each BFS tree from the set $\mathcal{F}_{k-1}$ has depth $2^{k-1}d$, and edges $O(m)$. Therefore, using Lemma 4.1, each tree from set $\mathcal{F}_{k-1}$ requires $O(2^{k-1}d)$ amortized update time per edge deletion. So, the amortized update time T_{k-1} per edge deletion for maintaining the set $\mathcal{F}_{k-1}$ is

$$T_{k-1} = O(n^{c_{k-1}} 2^{k-1} d)$$

It follows from the Theorem 4.1 that a tree t from a set $\mathcal{F}_i, i < (k-1)$, has $\mu(t) = m \ln n / n^{c_{i+1}}$, and depth $= \min\{2^i d, n \ln n / n^{c_{i+1}}\}$. Therefore, using the Lemma 4.2, each tree $t \in \mathcal{F}_i, i < k-1$ requires $O(\min\{2^i d / n^{c_{i+1}}, n \ln n / n^{2c_{i+1}}\})$ amortized update time per edge deletion. So the amortized update time T_i per edge deletion for maintaining the set $\mathcal{F}_i$ is

$$T_i = O\left(\min\left\{2^i d \frac{n^{c_i}}{n^{c_{i+1}}} \ln n, \frac{n^{1+c_i}}{n^{2c_{i+1}}} \ln^2 n\right\}\right), \quad i < k-1$$

Hence, the amortized update time T per edge deletion for maintaining the hierarchy $\mathcal{H}_k^d$ is

$$\begin{aligned} T &= T_{k-1} + \sum_{i<k-1} T_i \\ &= O(n^{c_{k-1}} 2^{k-1} d) + \\ &\quad \sum_{i=0}^{i=k-2} O\left(\min\left\{2^i d \frac{n^{c_i}}{n^{c_{i+1}}} \ln n, \frac{n^{1+c_i}}{n^{2c_{i+1}}} \ln^2 n\right\}\right) \end{aligned}$$

To minimize the sum on right hand side in the above equation, we balance all the terms constituting the sum, and get

$$T = \tilde{O}\left(2^{k-1} \cdot \min\left\{\sqrt[k]{n}d, (nd)^{\frac{2^{(k-1)}}{2^k - 1}}\right\}\right)$$

If α is the desired approximation factor, then it follows from Lemma 5.1 that the number of levels k, in the hierarchy are $\log_2 \overline{\alpha}$. So the amortized update time required is $\tilde{O}(\alpha \cdot \min\{\sqrt[\log_2 \overline{\alpha}]{n}d, (nd)^{\frac{\overline{\alpha}}{2(\overline{\alpha}-1)}}\})$.

Theorem 5.1. *Let $G(V,E)$ be an undirected graph undergoing edge deletions, d be a distance parameter, and $\alpha > 2$ be the desired approximation factor. There exists a data-structure $\dot{\mathcal{D}}_\alpha(1,d)$ for maintaining α-approximate distances for all-pairs separated by distance $\leq d$ in $\tilde{O}(\alpha \cdot \min\{\sqrt[\log_2 \overline{\alpha}]{n}d, (nd)^{\frac{\overline{\alpha}}{2(\overline{\alpha}-1)}}\})$ amortized update time per edge deletion, and $O(\log \overline{\alpha})$ query time.*

Based on the data-structure of [4], the previous best algorithm for maintaining all-pairs exact shortest paths of length $\leq d$ requires $O(nd)$ amortized update time. We have been able to achieve $o(nd)$ update time at the expense of introducing approximation as shown in the following table.

Table 1

Maintaining α-approximate distances for all-pairs of vertices separated by distance $\leq d$		
Data-structure	α	Update time
$\dot{\mathcal{D}}_3(1,d)$	3	$\tilde{O}(\min(\sqrt{n}d,(nd)^{2/3}))$
$\dot{\mathcal{D}}_7(1,d)$	7	$\tilde{O}(\min(\sqrt[3]{n}d,(nd)^{4/7}))$
$\dot{\mathcal{D}}_{15}(1,d)$	15	$\tilde{O}(\min(\sqrt[4]{n}d,(nd)^{8/15}))$

6 Maintaining Witnesses of Approximate Distance Efficiently

In addition to the explicit way of maintaining all-pairs approximate distances (by keeping a BFS tree on each vertex), there is an alternate implicit way of maintaining approximate distances by keeping a *witness* of approximate distance for each vertex-pair. It can be observed that if two vertices u and v are present in a BFS tree of depth d rooted at $w \in V$, then $\delta(u,v)$ is bounded by $2d$, and so the tree B_w^d stands as a witness of $\frac{2d}{\delta(u,v)}$-approximate distance between u and v.

The idea of maintaining all-pairs shortest distances implicitly (by keeping witnesses) was explored for the first time for directed graphs in [1]. For sake of completeness, we first give analogous and simpler description for undirected graphs, and then use our hierarchical scheme to improve the update time.

We define a terminology here : Let $\mathcal{W}$ be a set of BFS trees called a *witness-set*. The *witness-count* for a pair of vertices $u,v \in V$ with respect to a witness-set $\mathcal{W}$ is the number of the BFS trees from the set $\mathcal{W}$ that contain both u and v.

Given a witness set $\mathcal{W}$, *witness-count* for all-pairs of vertices can be maintained under deletion of edges as follows.

We maintain the witness-count for all-pairs of vertices in a matrix M. We initialize the matrix as follows. For each tree $t \in \mathcal{W}$, we increment $M[u,v]$ by one for every $u,v \in t$. The total time required for initializing the matrix will be $O(\sum_{t\in\mathcal{W}}(\nu(t))^2)$.

We handle deletion of an edge e_{uv} as follows. We update each BFS tree $t \in \mathcal{W}$ that contains the edge e_{uv}. Let s_e be the set of the vertices that cease to belong to the tree t due to deletion of e_{uv}. If S_t is the set of vertices belonging to the tree t prior to the deletion of e_{uv}, then it can be seen that the tree t is no longer a witness for the set of vertex-pairs $R = \{(x,y) : x \in s_e \wedge y \in S_t\}$. So we decrement entry $M[x,y]$ by one for each $(x,y) \in R$. In this way we update the witness-counts in the matrix M with respect to the changes in t. We perform this task for each $t \in \mathcal{W}$ containing e_{uv}. The task of updating the matrix M during the sequence of edge deletions can be visualized as undoing the process of initialization of matrix M, and thus requires $O(\sum_{t\in\mathcal{W}}(\nu(t))^2)$ total update time in addition to the time required for maintaining the trees $t \in \mathcal{W}$.

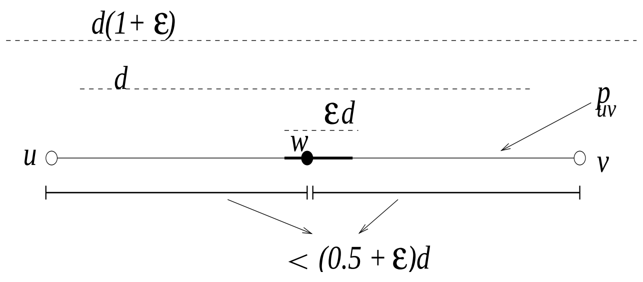

Figure 4: Bounding the approximation factor of distance reported by $B_w^{(1/2+\epsilon)d}$

LEMMA 6.1. *Given a set of $\mathcal{W}$ of BFS-trees under deletion of edges, maintaining the all-pairs witness-count matrix M with respect to the set $\mathcal{W}$ requires $O(\sum_{t\in\mathcal{W}}(\nu(t))^2)$ total update time (in addition to the time required for maintaining the set $\mathcal{W}$ of BFS trees).*

The following Lemma is the basis for maintaining witnesses of $(1+2\epsilon)$-approximate distances for all-pairs separated by distance $\in [d, d(1+\epsilon)]$.

LEMMA 6.2. *Given a random sample $\mathcal{S} \subset V$ of size $\frac{4n}{\epsilon d}\ln n$, the set $\mathcal{W} = \{B_w^{(1/2+\epsilon)d} : w \in \mathcal{S}\}$ contains a witness of $(1+2\epsilon)$-approximate distance with probability $1-1/n^2$, for each pair of vertices separated by distance $\in [d, d(1+\epsilon)]$.*

Proof. Consider any two vertices u and v separated by a shortest-path p_{uv} of length $\in [d, d(1+\epsilon)]$ (see Figure 4). Let S_{uv} be the set of the vertices lying on the path p_{uv} that are at distance $\leq \epsilon d/2$ from the mid-point of p_{uv}. Clearly $|S_{uv}| = \epsilon d$. It can be observed that a tree $B_w^{(1/2+\epsilon)d}$ rooted at any $w \in S_{uv}$ contains both u and v. In other words, each tree $B_w^{(1/2+\epsilon)d}, w \in S_{uv}$, is a witness of $(1+2\epsilon)$-approximate distance between u and v. A uniform random sample $\mathcal{S} \subset V$ of size $\frac{4n}{\epsilon d}\ln n$ is going to pick at-least one vertex from the set S_{uv} with probability $\geq 1-1/n^4$. The pair of vertices in the graph separated by distance $\in [d, d(1+\epsilon)]$ is $O(n^2)$, therefore, it follows that with probability $> 1 - 1/n^2$, the set $\mathcal{W} = \{B_w^{(1/2+\epsilon)d} : w \in S\}$ contains a witness of $(1+2\epsilon)$-approximate distance for each pair of vertices separated by distance $\in [d, d(1+\epsilon)]$.

It follows from the Lemma given above that if $\mathcal{W}$ is the set of BFS trees of depth $(1/2+\epsilon)d$ built on a random set $S \subset V$ of size $\frac{4n \ln n}{\epsilon d}$, then witness-count for each pair of vertices separated by distance $\in [d, (1+\epsilon)d]$ is going to remain non-zero with probability $\geq 1 - 1/n^2$ at each stage during the process of edge-deletions. Therefore, in order to maintain $(1+2\epsilon)$-approximate distances for all-pairs of vertices separated by distance $\in [d, (1+\epsilon)d]$, it suffices if we maintain witness-count for all-pairs with respect to the witness set $\mathcal{W}$. It follows from Lemma 4.1 that the total update cost of maintaining a tree $t \in \mathcal{W}$ is $O(\mu(t)d)$. So the total update cost for maintaining witness-count matrix with respect to the set $\mathcal{W}$ (using Lemma 6.1) is given by

$$T_d^{d(1+\epsilon)}(\mathcal{W}) = \sum_{t \in \mathcal{W}} O(|\mu(t)|d + (\nu(t))^2)$$

Note that $|\mathcal{W}| = O(\frac{n \ln n}{\epsilon d})$. The worst-case bounds $\mu(t) = \theta(m)$ and $\nu(t) = \theta(n)$ for a tree $t \in \mathcal{W}$ imply that $T_d^{d(1+\epsilon)}(\mathcal{W}) = O((mn + n^3/d) \ln /\epsilon)$.

Using the idea of *"keeping many small trees and a few large trees"* that underlies our hierarchical scheme, we shall improve this update time at the expense of increasing the approximation factor in the following section.

At this point we would like to mention that the technique for maintaining $(1+2\epsilon)$-approximate distances can be extended to any interval $[a, b]$ as follows. We maintain a witness count matrix M_i for each interval $[a(1+\epsilon)^i, a(1+\epsilon)^{i+1}], i \leq \ln_{1+\epsilon} b/a$. We keep an auxiliary matrix $\mathcal{A}$ such that $\mathcal{A}[u, v]$ point to the $M_i[u, v]$ for the smallest i such that $M_i[u, v] \neq 0$. It can be seen that $\mathcal{A}$ stores implicitly the $(1+2\epsilon)$-approximate distance for each pair of vertices separated by distance $\in [a, b]$.

Note that if query requires reporting an approximate shortest-path instead of approximate distance, an entry $M[x, y]$ stores all the witnesses for the pair (x, y) (instead of their count). It can be seen that the update time for maintaining this witness-matrix is same as that of maintaining a witness-count matrix (Lemma 6.1).

6.1 Maintaining witnesses of $(2+\epsilon)$-approximate distance We form a two level hierarchy $\mathcal{H}^2_{(1/2+\epsilon)d} = \{(S_0, \mathcal{F}_0), (S_1, \mathcal{F}_1)\}$ where the set $S_0 \subset V$ is a random sample of size $4n \ln n/\epsilon d$, whereas the set $S_1 \subset V$ (of size $n^c = o(|S_0|)$) is formed using the sampling scheme $\mathcal{R}(c)$ (described in section 4).

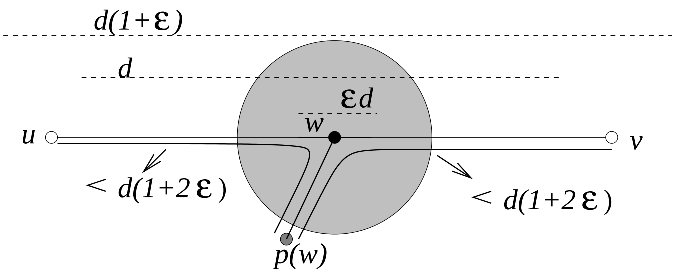

Figure 5: Bounding the approximation factor of distance reported by $B_{p(w)}^{(1+2\epsilon)d}$

LEMMA 6.3. *The set $\mathcal{F}_0^{(1/2+\epsilon)d} \cup \mathcal{F}_1$ contains a witness of $2(1+2\epsilon)$-approximate distance with probability $\geq 1 - 1/n^2$, for each pair of vertices separated by distance $\in [d, d(1+\epsilon)]$*

Proof. It follows from Lemma 6.2 that there is some $w \in S_0$ such that the distance of w from either u or v is at most $(1/2+\epsilon)d$. Therefore, if $B_w^{(1/2+\epsilon)d, S_1}$ is of depth $(1/2+\epsilon)d$ (i.e., it belongs to $\mathcal{F}_0^{(1/2+\epsilon)d}$), then surely $B_w^{(1/2+\epsilon)d, S_1}$ contains both u and v, and so is a witness of $(1+2\epsilon)$-approximate distance between the two. Otherwise it implies that the vertex $p(w)$ is at distance $\leq (1/2+\epsilon)d$ from w (see the Figure 5). Therefore by triangle inequality, it follows that the distance from $p(w)$ to either of u or v is at most $(1+2\epsilon)d$. Hence $B_{p(w)}^{(1+2\epsilon)d} \in \mathcal{F}_1$ contains both u and v, and so is a witness of $2(1+\epsilon)$-approximate distance between u and v.

From Lemma 6.3 given above it follows that for maintaining $2(1+2\epsilon)$-approximate distances for all-pairs of vertices separated by distance $\in [d, (1+\epsilon)d]$, it suffices if we maintain a witness-count matrix M with respect to the set $\mathcal{F}_0^{(1/2+\epsilon)d} \cup \mathcal{F}_1$. Note that as the edges are being deleted, the depth of a tree from the set $\mathcal{F}_0 \backslash \mathcal{F}_0^{(1/2+\epsilon)d}$ may increase (and eventually become $(1/2+\epsilon)d$). So the set $\mathcal{F}_0^{(1/2+\epsilon)d}$ grows during the sequence of edge-deletions. In order to ensure that after each edge-deletion, the matrix M keeps all-pairs witness-count with respect to the dynamic set $\mathcal{F}_0^{(1/2+\epsilon)d} \cup \mathcal{F}_1$, we perform the following additional updates in matrix M. Whenever a tree t from the set $\mathcal{F}_0 \backslash \mathcal{F}_0^{(1/2+\epsilon)d}$ grows to depth $(1/2+\epsilon)d$ (and thus moves to the set $\mathcal{F}_0^{(1/2+\epsilon)d}$), we increment $M[u, v]$ by one for each $u, v \in t$.

Thus the total update time for maintaining $2(1+\epsilon)$-approximate distances for all-pairs of vertices separated by distance $\in [d, d(1+\epsilon)]$ is the time required for maintaining the set $\mathcal{F}_0 \cup \mathcal{F}_1$ of BFS trees and the witness-count matrix M with respect to the set $\mathcal{F}_0^{(1/2+\epsilon)d} \cup \mathcal{F}_1$, and is given by

$$T_d^{(1+\epsilon)d} = \sum_{t\in\mathcal{F}_0\cup\mathcal{F}_1}\mu(t)\cdot d + \sum_{t\in\mathcal{F}_0^{(1/2+\epsilon)d}\cup\mathcal{F}_1}\nu(t)^2$$
$$= O\left(\frac{n\ln n}{\epsilon d}\frac{m\ln n}{n^c}d + n^c md\right) + O\left(\frac{n\ln n}{\epsilon d}\left(\frac{n\ln n}{n^c}\right)^2 + n^{2+c}\right)$$

Choosing c to minimize the total sum, we get

$$T_d^{(1+\epsilon)d} = O\left(\left(n^2\sqrt[3]{\frac{n}{\epsilon d}} + n^2\sqrt{\frac{m}{\epsilon n}} + m\sqrt{\frac{nd}{\epsilon}}\right)\ln n\right)$$

The algorithm for maintaining $2(1+2\epsilon)$-approximate distances can be extended to any interval $[a,b]\subset[1,n]$ just the same way the algorithm for maintaining $(1+2\epsilon)$-approximate distance is extended (as described earlier in this section). The total update time over any sequence of edge-deletions will be

$$T_a^b = \sum_{i\le\log_{1+\epsilon}b/a} T_{(1+\epsilon)^i d}^{(1+\epsilon)^{i+1}d}$$
$$= O\left(\left(n^2\sqrt[3]{\frac{n}{\epsilon a}} + n^2\sqrt{\frac{m}{\epsilon n}} + m\sqrt{\frac{nb}{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$$

THEOREM 6.1. *Given an undirected unweighted graph under deletion of edges, and an interval $[a,b]\subset[1,n]$, there exists a data-structure $\ddot{\mathcal{D}}_{2+\epsilon}(a,b)$ for maintaining $(2+\epsilon)$-approximate distance for all-pairs separated by distance $\in[a,b]$ in $O\left(\left(\frac{n^2}{m}\sqrt[3]{\frac{n}{\epsilon a}} + n\sqrt{\frac{n}{\epsilon m}} + \sqrt{\frac{nb}{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$ amortized update time per edge-deletion.*

6.2 Maintaining witnesses of $(4+\epsilon)$-approximate distance We form a three level hierarchy $\mathcal{H}^3_{(1/2+\epsilon)d} = \{(S_0,\mathcal{F}_0),(S_1,\mathcal{F}_1),(S_2,\mathcal{F}_2)\}$, where the set $S_0\subset V$ is a random sample of size $\frac{4n\ln n}{\epsilon d}$, whereas the set $S_1,S_2\subset V$ are formed using the sampling scheme $\mathcal{R}$ described in section 4. Let $|S_1| = n^{c_1}$, $|S_2| = n^{c_2}$, where c_1,c_2 are two constants <1 to be chosen so as to get improved update time (as described below).

Along the lines of the Lemma 6.3, we can state the following Lemma.

LEMMA 6.4. *The set $\mathcal{F}_0^{(1/2+\epsilon)d}\cup\mathcal{F}_1^{(1+2\epsilon)d}\cup\mathcal{F}_2$ contains a witness of $4(1+2\epsilon)$-approximate distance with probability $\ge 1-1/n^2$, for each pair of vertices separated by distance $\in[d,(1+\epsilon)d]$.*

From the Lemma given above it follows that for maintaining $4(1+2\epsilon)$-approximate distances for all-pairs of vertices separated by distance $\in[d,(1+\epsilon)d]$, it suffices if we maintain a witness-count matrix M with respect to the set $\mathcal{F}_0^{(1/2+\epsilon)d}\cup\mathcal{F}_1^{(1+2\epsilon)d}\cup\mathcal{F}_2$. Thus the total update-time for maintaining $4(1+2\epsilon)$-approximate distances for all-pairs separated by distance $\in[d,d(1+\epsilon)]$ is the time required for maintaining the set $\mathcal{F}_0\cup\mathcal{F}_1\cup\mathcal{F}_2$ of BFS trees and the witness-count matrix M with respect to the set $\mathcal{F}_0^{(1/2+\epsilon)d}\cup\mathcal{F}_1^{(1+2\epsilon)d}\cup\mathcal{F}_2$, and is given by

$$T_d^{(1+\epsilon)d} = \sum_{t\in\mathcal{F}_0\cup\mathcal{F}_1\cup\mathcal{F}_2}\mu(t)\cdot d + \sum_{t\in\mathcal{F}_0^{(1/2+\epsilon)d}\cup\mathcal{F}_1^{(1+2\epsilon)d}\cup\mathcal{F}_2}\nu(t)^2$$
$$= O\left(\frac{n\ln n}{\epsilon d}\frac{m\ln n}{n^{c_1}}d + n^{c_1}\frac{m\ln n}{n^{c_2}}d + n^{c_2}md\right) + O\left(\frac{n\ln n}{\epsilon d}\left(\frac{n\ln n}{n^{c_1}}\right)^2 + n^{c_1}\left(\frac{n\ln n}{n^{c_2}}\right)^2 + n^{2+c_2}\right)$$

Choosing c_1,c_2 so as to minimize the total sum, we get

$$T_d^{(1+\epsilon)d} = O\left(\left(n^2\sqrt[5]{\frac{n}{\epsilon d}} + n^2\sqrt[3]{\frac{m}{\epsilon n}} + m\sqrt[3]{\frac{nd^2}{\epsilon}}\right)\ln n\right)$$

The algorithm for maintaining $4(1+2\epsilon)$-approximate distances can be extended to any interval $[a,b]\subset[1,n]$ just the same way the algorithm for maintaining $(1+2\epsilon)$-approximate distance was extended (as described earlier in this section). The total update time required over any sequence of edge-deletions will be

$$T_a^b = \sum_{i\le\log_{1+\epsilon}b/a} T_{(1+\epsilon)^i d}^{(1+\epsilon)^{i+1}d}$$
$$= O\left(\left(n^2\sqrt[5]{\frac{n}{\epsilon a}} + n^2\sqrt[3]{\frac{m}{\epsilon n}} + m\sqrt[3]{\frac{nb^2}{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$$

THEOREM 6.2. *Given an undirected unweighted graph under deletion of edges, and an interval $[a,b]\subset[1,n]$, there exists a data-structure $\ddot{\mathcal{D}}_{4+\epsilon}(a,b)$ for maintaining $(4+\epsilon)$-approximate distance for all-pairs separated by distance $\in[a,b]$ in $O\left(\left(\frac{n^2}{m}\sqrt[5]{\frac{n}{\epsilon a}} + \frac{n^{5/3}}{\epsilon^{1/3}m^{2/3}} + \sqrt[3]{\frac{nb^2}{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$ amortized update time per edge-deletion.*

Using the Theorems 6.1 and 6.2, the following improved bounds on update time are achieved for maintaining all-pairs approximate shortest paths of length $\le d$.

Table 2

Maintaining α-approximate distances for all-pairs of vertices separated by distance $\leq d$ in graphs with $m = \theta(n^2)$		
Data-structure	α	Update time
$\ddot{\mathcal{D}}_3(1,d)$	3	$\tilde{O}\left(\sqrt{nd}\right)$
$\ddot{\mathcal{D}}_5(1,d)$	5	$\tilde{O}\left(\sqrt[3]{nd^2}\right)$

For maintaining approximate distances for all-pairs separated by distance in interval $[d,n]$, the Theorems 6.1 and 6.2 imply the results given in **Table 3**.

Table 3

Maintaining α-approximate distances for all-pairs of vertices separated by distance $\in [d,n]$		
Data-structure	α	Update time
$\ddot{\mathcal{D}}_{2+\epsilon}(d,n)$	$2+\epsilon$	$O\left(\left(\frac{n^2}{m}\sqrt[3]{\frac{n}{\epsilon d}} + \frac{n}{\sqrt{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$
$\ddot{\mathcal{D}}_3(d,n)$	3	$O\left(\left(\frac{n^2}{m}\sqrt[3]{\frac{n}{d}} + n\right)\ln^2 n\right)$
$\ddot{\mathcal{D}}_{4+\epsilon}(d,n)$	$4+\epsilon$	$O\left(\left(\frac{n^2}{m}\sqrt[5]{\frac{n}{\epsilon d}} + \frac{n}{\sqrt[3]{\epsilon}}\right)\frac{\ln^2 n}{\epsilon}\right)$
$\ddot{\mathcal{D}}_5(d,n)$	5	$O\left(\left(\frac{n^2}{m}\sqrt[5]{\frac{n}{d}} + n\right)\ln^2 n\right)$

7 Improved Decremental Algorithms for APASP

In section 5 and 6, we described two data-structures for maintaining approximate distances under deletion of edges for all-pairs separated by distance $\in [a,b], 1 \leq a < b \leq n$. Both these data-structures can be used for maintaining all-pairs approximate distances. However, it should be noted that the data-structure $\dot{\mathcal{D}}_\alpha$ is more suitable for maintaining α-approximate distances for pairs of vertices separated by *small* distance (refer to **Table 1** for dependence of update time on d). Whereas the data-structure $\ddot{D}_\alpha$ is more suitable for maintaining α-approximate distances for pairs of vertices separated by *large* distances (refer to **Table 3** for dependence of update time on d). Thus we use both the data-structures simultaneously to achieve better update time as follows.

If α is the desired approximation factor, then we pick a suitable distance parameter d. We maintain $\dot{\mathcal{D}}_\alpha(1,d)$ and maintain $\ddot{\mathcal{D}}_\alpha(d,n)$. We choose the value for d that balances the update time for both the data-structures.

For reporting α-approximate distance between two vertices u and v, we first query the data-structure $\dot{\mathcal{D}}_\alpha(1,d)$ which will report approximate distance between the two if they are separated by distance $\leq d$. We query the data-structure $\ddot{\mathcal{D}}_\alpha(d,n)$ in case $\dot{\mathcal{D}}_\alpha(1,d)$ reports that they are separated by distance $\geq d$.

Combining the two data-structures $\dot{\mathcal{D}}, \ddot{\mathcal{D}}$ together as described above, we achieve *practically* linear update time per edge deletion for maintaining all-pairs approximate distances. See the **Table 4** (please refer to **Table 1**, and **Table 3** for notations used for the data-structures). This is an improvement over the previous bound of $\tilde{O}(\frac{n^2}{\sqrt{m}})$ on the update time which is $\tilde{O}(n^{1.5})$ for sparse graphs.

Table 4

Maintaining α-approximate distances for all-pairs of vertices				
Data-structures		α	Update time	Maximum Update time
$\dot{\mathcal{D}}_3(1,d)$	$\ddot{\mathcal{D}}_3(d,n)$	3	$\tilde{O}\left(\frac{n^{\frac{16}{9}}}{m^{\frac{2}{3}}} + n\right)$	$\tilde{O}(n^{\frac{10}{9}})$
$\dot{\mathcal{D}}_3(1,d)$	$\ddot{\mathcal{D}}_5(d,n)$	5	$\tilde{O}\left(\frac{n^{\frac{24}{13}}}{m^{\frac{10}{13}}} + n\right)$	$\tilde{O}(n^{\frac{14}{13}})$
$\dot{\mathcal{D}}_7(1,d)$	$\ddot{\mathcal{D}}_5(d,n)$	7	$\tilde{O}\left(\frac{n^{\frac{16}{9}}}{m^{\frac{20}{27}}} + n\right)$	$\tilde{O}(n^{\frac{28}{27}})$

References

[1] S. Baswana, R. Hariharan, and S. Sen. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, pages 117-123, 2002.

[2] Edith Cohen and Uri Zwick. All-pairs small-stretch paths. *Journal of Algorithms*, 38:335–353, 2001.

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

[4] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of association for computing machinery*, 28:1-4, 1981.

[5] Monika Rauch Henzinger and Valerie King. Fully dynamic bi-connectivity and transitive closure. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 664-672, 1995.

[6] Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 325-334, 1999.

[7] M. Thorup and U. Zwick. Approximate distance oracles. In *Proceedings of the 33rd Annual Symposium on Theory of Computing (STOC)*, pages 183-192, 2001.