Improved Decremental Algorithms for Maintaining Transitive Closure and All-Pairs Shortest Paths ⋆

Surender Baswana [a,∗,1] Ramesh Hariharan [b] Sandeep Sen [a,2]

[a] *Dept. of Comp. Sc. and Engg., Indian Institute of Technology Delhi, New Delhi, India*

[b] *Dept. of Comp. Sc. and Automation, Indian Institute of Science, Bangalore, India*

Abstract

This paper presents improved algorithms for the following problem : Given an unweighted directed graph $G(V, E)$ and a sequence of on-line shortest-path/reachability queries interspersed with edge-deletions, develop a data-structure that can answer each query in optimal time, and can be updated efficiently after each edge-deletion.

The central idea underlying our algorithms is a scheme for implicitly storing all-pairs reachability/shortest-path information, and an efficient way to maintain this information.

Our algorithms are randomized and have one-sided inverse polynomial error for query.

Key words: BFS tree, dynamic, graph, transitive closure, shortest paths

⋆ Preliminary version of this paper appeared in 34th *ACM Symposium on Theory of Computing (STOC)*, May 19-21, 2002, Montreal,Quebec, Canada.

∗ Corresponding author.

Email addresses: `sbaswana@cse.iitd.ernet.in` (Surender Baswana), `ramesh@csa.iisc.ernet.in` (Ramesh Hariharan), `ssen@cse.iitd.ernet.in` (Sandeep Sen).

[1] Work was supported in part by a PhD fellowship from Infosys Technologies Ltd., Bangalore.

[2] Work was supported in part by an IBM UPP award

1 Introduction

The following two problems are among the most fundamental algorithmic graph problems :

- **Transitive Closure :** Process a given directed graph[3] $G(V, E)$ so that any query of the form, "*Is there a path from u to v in the graph?*", can be answered efficiently.
- **All-Pairs Shortest Paths Problem :** Process a given unweighted graph $G(V, E)$ so that any query of the form, "*Report the shortest path (or distance) from vertex u to a vertex v?*", can be answered efficiently.

There exist classical algorithms that take $O(mn)$ time for the two problems so that any query can be answered in $O(1)$ time. There also exist algorithms based on fast matrix multiplication that require sub-cubic time for the two problems. In particular, using the fastest known matrix multiplication algorithms (Coppersmith and Winograd [4]), the best bound on computing transitive closure is $O(n^{2.376})$, whereas for the all-pairs shortest paths problem, it is $O(n^{2.575})$ [20].

There are many applications in communication networks, incremental parser generation [14] and relational databases augmented with transitive closure [15,11] that require efficient solutions of the above problems for a dynamic graph. In a dynamic graph problem, an initial graph is given, followed by an sequence of on-line queries interspersed with updates which can be insertion or deletion of edges. We have to carry out the updates and answer the queries on-line in an efficient manner. Each query has to be answered with respect to the present state of the graph, i.e., incorporating all the updates preceding the query. One trivial way to solve such a dynamic graph problem is that we run the static graph algorithm after every update. The goal of a dynamic graph algorithm is to update the solution efficiently after the dynamic changes, rather than having to re-compute it from scratch each time.

For every static graph problem, there exists its dynamic counterpart. We can classify dynamic graph problems according to the types of updates allowed. A problem is said to be *fully dynamic* if the update operations include both insertions and deletions of edges. A problem is called *partially dynamic* if only one type of update, either insertion or deletions, is allowed. If only insertions are allowed, the problem is called *incremental*; if only deletions are allowed, it is called *decremental*.

We present efficient decremental algorithms for maintaining all-pairs shortest-paths and transitive closure in an unweighted graph. The query algorithms

[3] In this paper graphs will imply directed graphs

are Monte-Carlo with one sided error (the probability of error being inverse polynomial).

1.1 Previous work and our contribution

Transitive Closure :

La Poutre and Van Leeuwen [18] gave a decremental algorithm for maintaining transitive closure with $O(m)$ amortized update time per edge deletion and answering each query in $O(1)$ time. Demetrescu and Italiano [6] gave a decremental algorithm for the problem that requires $O(n^3/m)$ amortized update time which is better for dense graphs. For an initial complete graph, the algorithm achieves $O(n)$ amortized update time per edge deletion [5], but for sparse graphs, the update time can be even $\theta(n^2)$. It can be seen that a combination of these two algorithms yields an upper bound of $O(n^{\frac{3}{2}})$ on the update time while keeping $O(1)$ query time. Henzinger and King [13] gave a decremental randomized algorithm that achieves $O(n \ln^2 n)$ amortized update time but at the expense of increased query time of $O(\frac{n}{\ln n})$. The query has one sided inverse polynomial error, i.e., the answer to any query may be incorrect in the sense that it may not report a path when there exists one. However, the probability of an error can be made smaller than $\frac{1}{n^c}$ for arbitrary $c > 0$. Subsequently we will use the acronym w.h.p. to denote probability exceeding $1 - \frac{1}{n^c}$ for any $c > 0$.

In this paper, we present an efficient algorithm that achieves $O(1)$ query time w.h.p. and requires $O(n \ln^2 n + \frac{n^2}{\sqrt{m}} \ln n)$ amortized update time per edge-deletion. Our algorithm achieves an improvement in the update time compared to the deterministic algorithms while ensuring $O(1)$ query time. By suitably combining with [18], our algorithm achieves an improved upper bound of $O(n^{\frac{4}{3}} \sqrt[3]{\ln n})$ on amortized update time per edge deletion for the problem of maintaining transitive closure with optimal query time.

All-pairs shortest paths :

Demetrescu and Italiano [7] gave a decremental algorithm for maintaining all-pairs shortest paths under deletion of edges that achieves $O(\frac{n^3}{m} \ln^3 n)$ amortized update time while achieving optimal query time w.h.p. Their algorithm improves the previous $O(n^2)$ update time of [16] for dense graphs.

We present two decremental algorithms for the all-pairs shortest paths problem. For distance reporting problem, we give a simpler combinatorial algorithm that requires $O(\frac{n^3}{m} \ln^2 n)$ amortized update time while achieving $O(1)$ query time. For the shortest path reporting problem (i.e. the actual sequence of edges), we use the idea of filtering search in a novel way to design an al-

Maintaining All-Pairs Reachability		
Query	Previous Update Time	Improved Update Time
Is v reachable from u ?	$O\left(n^{\frac{3}{2}}\right)$	$O\left(n^{\frac{4}{3}}\sqrt[3]{\ln n}\right)$
report a path from u to v		
Maintaining All-Pairs $(1+\epsilon)$-Approximate Shortest Paths		
Query	Previous Update Time	Improved Update Time
report $(1+\epsilon)$-approx. distance	*none*	$O\left(\frac{n\ln n}{\epsilon^2}+\frac{n^2\sqrt{\ln n}}{\epsilon\sqrt{m}}\right)$
report $(1+\epsilon)$-approx. shortest path		
Maintaining All-Pairs Shortest Paths		
Query	Previous Update Time	Improved Update Time
report the distance from u to v	$O\left(\frac{n^3}{m}\ln^3 n\right)$	$O\left(\frac{n^3}{m}\ln^2 n\right)$
report the shortest path from u to v	$O(\frac{n^3}{m}\ln^3 n)$	$\min\begin{cases} O(n^{\frac{3}{2}}\sqrt{\ln n}), \\ O(\frac{n^3}{m}\ln^2 n) \end{cases}$

Table 1
All the update bounds (old and new) are amortized. Throughout this paper, our query times are optimal but there is one sided inverse polynomial error in the query-answer.

gorithm that achieves $O(\min(n^{\frac{3}{2}}\sqrt{\ln n}, \frac{n^3}{m}\ln^2 n))$ update time while achieving optimal query time. Hence we reduce the upper bound on the worst case amortized update time for the problem of maintaining all-pairs shortest paths under deletion of edges by a factor of $O(\sqrt{\frac{n}{\ln n}})$.

Next, we present an efficient decremental algorithm that offers a trade-off between update time and approximation factor of the shortest path. For maintaining all-pairs $(1+\epsilon)$-approximate shortest paths, our algorithm achieves $O(\frac{n\ln n}{\epsilon^2}+\frac{n^2\sqrt{\ln n}}{\epsilon\sqrt{m}})$ amortized update time per edge-deletion for arbitrarily small $\epsilon > 0$. We summarize our results in Table 1.

It may be noted that our algorithms are simple to implement and do not make use of any sophisticated matrix multiplication algorithms. We show that the space requirement of our data structures is $O(n^2)$.

2 Overview of our algorithms

2.1 Notations

Given an unweighted graph $G(V,E)$, $v \in V$, and a positive integer d, we define the following notations that will be used in the remaining paper.

- *out_tree*(v) : a complete breadth-first-search (BFS) tree rooted at the vertex v in the graph $G(V,E)$.
- *in_tree*(v) : a complete BFS tree rooted at the vertex v in the graph formed by reversing all the edge-directions in $G(V,E)$.
- *out_tree*(v,d) : a BFS tree of depth d rooted at the vertex v in the graph $G(V,E)$.
- *in_tree*(v,d) : a BFS tree of depth d rooted at the vertex v in the graph formed by reversing all the edge-directions in $G(V,E)$.
- (*in_tree*,*out_tree*)(v,d) : a pair of *in_tree*(v,d) and *out_tree*(v,d).
- $W(S,d)$: $\{$(*in_tree*,*out_tree*)(v,d) $|v \in S$ $\}$, for a given $S \subset V$.
- S_r : A random set of vertices formed by picking each vertex randomly independently with probability $\frac{\ln n}{r}$. The expected size of the set S_r is $\frac{n \ln n}{r}$.

With a BFS tree, we also keep an auxiliary array to determine in constant time whether a vertex belongs to the tree or not. In addition, for each vertex that belongs to the tree we keep information about its level (its distance from the root) and its parent in the tree. With all this information that we maintain in a BFS tree, it can be seen that in addition to maintaining reachability information from v, the tree *out_tree*(v,d) can also serve the purpose of maintaining the distance/shortest-path information to all the vertices lying within distance d from v. In the same way, *in_tree*(v,d) can be used for maintaining reachability information as well as the shortest path to v from all the vertices for whom v lies within distance d.

While edges are being deleted in a graph, the levels of vertices in a BFS tree may increase. Thus maintaining a BFS tree of depth d, after an edge deletion, involves finding the set of vertices whose level has increased, eliminating those from this set whose level has fallen beyond d, and assigning the rest of the vertices (with level $\leq d$ still) to their new levels.

The best known algorithm for maintaining an *in_tree* or an *out_tree* is given by Henzinger and King [13] along the lines of [10].

Lemma 1 *[13] Given an unweighted graph $G(V,E)$, a vertex $v \in V$ and a positive integer d, an* in_tree*(v,d) (or an* out_tree*(v,d)) can be maintained in $O(d)$ amortized update time per edge deletion.*

2.2 Main Idea

The simplest way of maintaining all-pairs reachability (shortest paths) in an unweighted graph is to maintain an *out_tree* up to depth n from each vertex. It follows from Lemma 1 that the amortized update time required by this approach (maintaining n *out_trees* of depth n) will be $O(n^2)$ per edge deletion.

In order to develop efficient decremental algorithm for transitive closure and all-pairs shortest paths, we explore alternate schemes of maintaining all-pairs reachability and shortest paths. In this endeavor, we present schemes that maintains reachability and exact (or approximate) shortest path information *implicitly.*

Consider a pair of vertices $u, w \in V$ such that there is a path from vertex u to w, and let v be any intermediate vertex on the path. It can be seen that the vertex u belongs to *in_tree* rooted at v, and the vertex w belongs to *out_tree* rooted at v. Thus combined together *in_tree*, *out_tree* rooted at v stores the path from u to w implicitly (the complete path and its length can be retrieved by querying *in_tree*(v) and *out_tree*(v)). In other words, the pair (*in_tree*(v),*out_tree*(v)) acts as a *witness of reachability* from u to w. Analogously if the vertex v lies on the shortest path from u to w, the pair (*in_tree*(v),*out_tree*(v)) pair acts as *a witness of shortest path* from u to w.

The above mentioned scheme of keeping reachability (shortest path) information implicitly suggests that in order to maintain all-pairs reachability (shortest paths) under deletion of edges, it suffices to maintain a witness (if one exists) of reachability (shortest path) for each vertex-pair (u, w). In the following section, we design efficient algorithms for maintaining witnesses of all-pairs reachability and shortest paths corresponding to paths of length in an interval $[d', d]$ for any $1 \leq d' < d \leq n$. These algorithms form the basis for developing efficient decremental algorithms for the main problems as follows.

Maintaining transitive closure : The scheme of maintaining reachability implicitly (by keeping witnesses) proves to be efficient for maintaining all-pairs reachability corresponding to long paths. On the other hand, the scheme of maintaining reachability explicitly (by keeping *out_tree* from every vertex) proves to be efficient for maintaining reachability corresponding to short paths. We combine the two strategies together to achieve improved update time of $O(n \ln^2 n + \frac{n^2}{\sqrt{m}}\sqrt{\ln n})$ per edge deletion.

Maintaining all-pairs approximate shortest paths : Analogous to witness of reachability and shortest paths, we propose the terminology of witness of *approximate shortest paths.* It turns out that our data structure for maintaining transitive closure can be suitably adapted to maintain all-pairs

$(1+\epsilon)$-approximate shortest paths [4] in unweighted graphs.

Maintaining all-pairs exact shortest paths : For the problem of maintaining all-pairs shortest distances, the strategy of maintaining witness of shortest path for every vertex-pair leads to achieving $O(\frac{n^3}{m}\ln^2 n)$ update time. For the case when query is to report the shortest path, we use the idea of filtering search [2] to reduce the update time further to $O(\min(n^{\frac{3}{2}}\sqrt{\ln n}, \frac{n^3}{m}\ln^2 n))$ which is bounded by $O(n^{\frac{3}{2}}\sqrt{\ln n})$ for all graphs.

3 Maintaining witnesses of reachability/shortest-paths for all-pairs of vertices separated by distance $\in [\mathbf{d}', \mathbf{d}]$

In this section we design efficient algorithms for maintaining reachability and shortest paths for all the vertex-pairs (u, w) such that there is a path from u to w of length $\in [d', d]$. For the reachability problem, the algorithm will maintain a witness of reachability for every vertex-pair (u, w) if there is path of length $\in [d', d]$ from u to w. For the all-pairs shortest path problem, the algorithm will maintain a witness of the shortest path for every vertex-pair (u, w) among all paths from u to w of length $\in [d', d]$.

Let $S \subset V$, and let $W(S, d)$ be the set of pairs of *in_tree* and *out_tree* of depth d rooted at each vertex of the set S (see the notations defined in section 2.1). We begin with the design of efficient algorithms for maintaining all-pairs reachability and shortest paths with tree-pairs from the set $W(S, d)$ as witnesses, i.e., the paths to be considered for reachability (shortest-path) are only those paths that are captured by the tree-pairs of the set $W(S, d)$. The algorithm can be used for maintaining reachability (shortest paths) for all-pairs of vertices separated by distance $\in [d', d]$, if we choose $S = V$. It turns out that the update time of the algorithm is directly proportional to the size of the witness set. Subsequently, to improve the update time, we reduce the number of witnesses, i.e. $|S|$ by using random sampling. This observation was exploited by Henzinger and King [13] for designing a decremental algorithm to maintain all-pairs reachability with $O(n\ln^2 n)$ update time at the expense of $O(\frac{n}{\ln n})$ query time. We extend this approach to its full potential for maintaining all-pairs shortest paths and transitive closure with optimal query time.

3.1 Maintaining all-pairs reachability with respect to a witness set

Let $W(S, d)=\{(\textit{in\_tree}, \textit{out\_tree})(v, d) | v \in S\}$ be the set of witnesses kept in a list, also denoted by $W(S, d)$. Let u, w be any two vertices in the graph. If u

[4] Path with length at most $(1+\epsilon)$ times the length of the shortest path

and w lie respectively in *in_tree*(v,d) and *out_tree*(v,d) for some vertex $v \in S$, then (*in_tree*,*out_tree*)(v,d) is a witness of reachability from u to w. Finding out if there is any witness in the set $W(S,d)$ for reachability from u to w will require querying each (*in_tree*,*out_tree*)$(v,d) \in W(S,d)$. Thus the query time will be $O(|S|)$. In order to achieve $O(1)$ query time, we maintain a witness matrix M. For each pair (u,w), at every stage $M[u,w]$ points to some vertex $v \in S$ if (*in_tree*,*out_tree*)(v,d) is a witness of reachability from u to w.

We initialize the matrix M as follows : The matrix M has all entries pointing to **null** initially. For every (*in_tree*,*out_tree*)$(v,d) \in W(S,d)$, we do the following. For each $u \in$ *in_tree*(v,d) and for every $w \in$ *out_tree*(v,d), we update $M[u,w]$ to point to v if it was pointing to **null** previously. Thus it requires $O(n^2)$ time per tree-pair from the set $W(S,d)$ to initialize the witness matrix M.

Lemma 2 *Given a graph $G(V,E)$, a set of tree pairs $W(S,d)$ for a set $S \subset V$ and a positive integer d, it takes $O(n^2|S|)$ time to initialize matrix M for storing witnesses of reachability for all-pairs of vertices with respect to the set $W(S,d)$.*

Notice that as the edges are being deleted, a tree-pair (*in_tree*,*out_tree*)$(v,d) \in W(S,d)$ may cease to be a witness of reachability for a vertex-pair (u,w). This happens when either u ceases to belong to *in_tree*(v,d) or the vertex w ceases to belong to *out_tree*(v,d). We now describe an algorithm for updating the entries of matrix M after an edge deletion.

Updating matrix M for witnesses of reachability :
We perform the following three operations after deletion of an edge.

(1) *Updating the BFS trees of the set $W(S,d)$* :
For each $v \in S$, we update *in_tree*(v,d) and *out_tree*(v,d) for the edge deletion.

(2) *Finding the pairs of vertices whose witness of reachability has expired* :
Let (*in_tree*,*out_tree*)(v,d) be a tree-pair from the set $W(S,d)$, and let X and Y be the set of vertices that cease to belong to *in_tree*(v,d) and *out_tree*(v,d) respectively. (These sets are already computed in step 1.) We process the set X as follows (the set Y is processed in a similar fashion). For every $u \in X$, we find the set s_u of vertices such that $M[u,w]$ is v for each $w \in s_u$. By inspecting the row $M[u,\_]$ it requires $O(n)$ time per $u \in X$ to find s_u. It can be seen that (*in_tree*,*out_tree*)$(v,d) \in W(S,d)$ has ceased to be a witness of reachability from u to each $w \in s_u$.

By processing each tree-pair in the manner describer above, we can compute all-pairs of vertices whose current witness of reachability in M has expired.

(3) *Searching for a new witness of reachability* :
Let (u,w) be a pair of vertices whose current witness of reachability, say

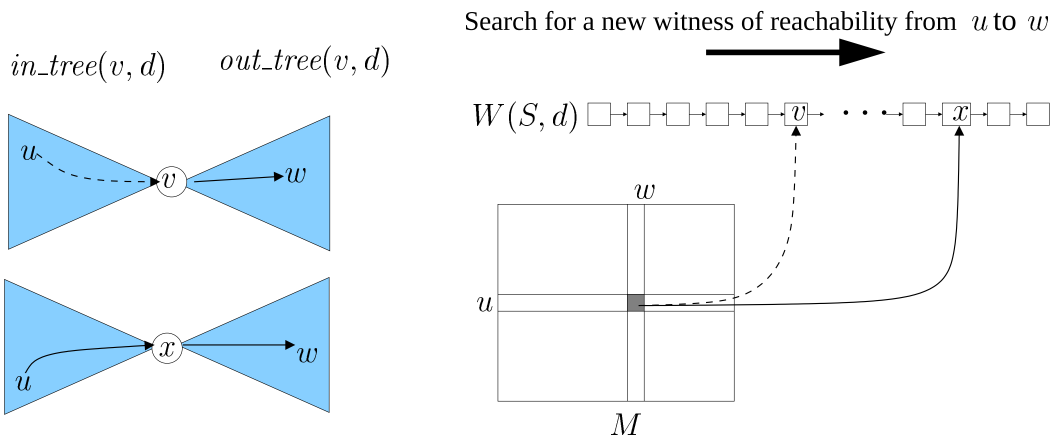

Fig. 1. The search for a new witness of reachability from u to w when $(in\_tree, out\_tree)(v,d)$ ceases to be the witness.

> $(in\_tree, out\_tree)(v,d)$ has expired due the recent edge-deletion. We search the list $W(S,d)$ starting from the tree-pair following $(in\_tree, out\_tree)(v,d)$, and find a witness of reachability from u to w (spending $O(1)$ time per tree-pair in $W(S,d)$). If we reach the end of the list $W(S,d)$, we update $M[u,w]$ to point to **null**, otherwise we update $M[u,w]$ to point to the root of the new witness found.

The following invariant is maintained throughout the series of edge deletions. I^τ : $M[u,w]$ *points to the first tree-pair in the list* $W(S,d)$ *that is a witness of reachability from* u *to* w*.*

The invariant holds just after the initialization of M. It can be easily verified that the procedure described above preserves the invariant after each edge deletion. By induction on the number of edge deletions, we can conclude that the invariant I^τ always holds.

Cost Analysis : As mentioned in Lemma 2, the total cost of initializing the witness matrix M is $O(n^2|S|)$. We now assess the total cost incurred in the three kinds of operations that we perform to maintain matrix M.

The first operation deals with updating each tree-pair under edge deletion. It requires $O(d)$ amortized cost per edge deletion for maintaining the *in_tree* and *out_tree* of depth d rooted at a vertex. Hence the total cost of maintaining tree-pairs of set $W(S,d)$ will be $O(md|S|)$ over any sequence of edge deletions.

Now consider the second operation that computes the pairs of vertices whose witness of reachability has expired after an edge deletion. It follows from the description given above that for a tree-pair $(in\_tree, out\_tree)(v,d) \in W(S,d)$, we incur $O(n)$ cost per vertex when the vertex ceases to belong to $in\_tree(v,d)$ (or $out\_tree(v,d)$). Hence the total cost incurred in second operation is $O(n^2|S|)$ over any sequence of edge deletions.

To assess the total cost incurred in the third operation, that is, searching for a new witness of reachability, note that we consider a tree-pair from the list $W(S,d)$ exactly once for being a witness of reachability for a pair (u,w). This is because once a tree-pair $(in\_tree, out\_tree)(v,d)$ ceases to be a witness of reachability from u to w, it can never become a witness of reachability from u to w in future. Hence the total cost incurred in searching for new witnesses will be $O(n^2|S|)$ over the entire sequence of edge-deletions.

Summing up the cost of all operations, we conclude that the total cost incurred in maintaining all-pairs reachability over any sequence of edge deletions is $O(md|S| + n^2|S|)$.

Theorem 3 *Given a graph $G(V,E)$, a set of tree-pairs $W(S,d)$ for a set $S \subset V$ and a positive integer d, all-pairs reachability with respect to the witnesses $W(S,d)$ can be maintained under edge-deletions with $O(1)$ query time and $O((md+n^2)|S|)$ total update time.*

3.2 Maintaining all-pairs shortest paths with respect to a witness set

The algorithm we describe is analogous to the algorithm described in the previous subsection. Let $W(S,d)=\{(in\_tree,\ out\_tree)(v,d) | v \in S\}$ be the set of witnesses kept in a list, also denoted by $W(S,d)$. We maintain a witness matrix M such that for each $u,w \in V$, $M[u,w]$ points to the first tree-pair in the list $W(S,d)$ which is a witness of the shortest path among all paths from u to w captured by the tree-pairs from the set $W(S,d)$. The matrix M is initialized as follows. The matrix M has all entries pointing to **null** initially. For every $(in\_tree, out\_tree)(v,d) \in W(S,d)$, we do the following. For each $u \in in\_tree(v,d)$ and for every $w \in out\_tree(v,d)$, we update $M[u,w]$ to point to v if it is pointing to **null** or if the tree-pair associated with the existing $M[u,w]$ is a witness of a path from u to w with longer length than the tree-pair $(in\_tree, out\_tree)(v,d)$. It follows easily that it requires $O(n^2)$ time per element of the set $W(S,d)$ to initialize the matrix.

Lemma 4 *Given a graph $G(V,E)$, a set of tree pairs $W(S,d)$ for a set $S \subset V$ and a positive integer d, it takes $O(n^2|S|)$ time to initialize matrix M for storing witnesses of shortest paths for all-pairs of vertices with respect to the set $W(S,d)$.*

For any pair of vertices $u,w \in V$, let $M[u,w] = v$ after the initialization of the matrix M. Notice that as the edges are being deleted, the tree-pair $(in\_tree, out\_tree)(v,d) \in W(S,d)$ may cease to be a witness of the shortest path for the vertex-pair (u,w). This happens when either u increases its level in $in\_tree(v,d)$ or the vertex w increases its level in $out\_tree(v,d)$. We now describe an algorithm for updating the entries of M after an edge deletion.

Updating matrix M for witnesses of shortest paths : We perform the following three operations after deletion of an edge.

(1) *Updating the BFS trees of the set $W(S, d)$:*
For each $v \in S$, we update *in_tree*(v, d) and *out_tree*(v, d) for the edge deletion.

(2) *Finding the pairs of vertices whose witness of shortest path has to be updated :*
Let (*in_tree,out_tree*)(v, d) be a tree-pair from the set $W(S, d)$, and let X and Y be the sets of vertices whose level has increased in *in_tree*(v, d) and *out_tree*(v, d) respectively. We process the set X as follows (the set Y is processed in a similar fashion). We scan row $M[u, \_]$ for each $u \in X$ to compute all those vertices $w \in V$ such that $M[u, w] = v$. It can be seen that the length of the shortest path from u to w passing through v has increased. As a result, we can not be sure that (*in_tree,out_tree*)(v, d) would still be a witness of shortest path from u to w. This is because there might be some other tree-pair in the set $W(S, d)$ that could be a witness of a path from u to w with length less than that of the new path from u to w passing through v. So we need to find a new witness of shortest path from u to w.
By processing each tree-pair in the manner describer above, we can compute all-pairs of vertices whose current witness of shortest path has to be updated in M.

(3) *Searching for a new witness of shortest path* : Let (u, w) be a pair of vertices such that a new witness of shortest path from u to w has to be searched for due to the recent edge-deletion (as mentioned in the second operation mentioned above). Let (*in_tree,out_tree*)(v, d) be the witness of shortest path (of length say r) from u to w prior to the edge deletion. We scan the list $W(S, d)$ starting from the successor of the tree-pair (in_tree,out_tree)(v, d) in search for a witness of a path of length r from u to w. If we find one, we stop; otherwise there is no path from u to w of length $\leq r$ in any tree-pair from the set $W(S, d)$. In this case, we perform another scan starting from the head of the list $W(S, d)$ in search of a witness of path-length $r + 1$. We increment r until we find one witness or r becomes $2d$. In the latter case, we conclude that there is no path from u to w captured in any tree-pair of the set $W(S, d)$.

The following invariant is maintained throughout the series of edge deletions.
I^{APSP} : *At every stage $M[u, w]$ points to the first tree-pair in the list $W(S, d)$ that is a witness of the shortest path from u to w among all paths from u to w captured by tree-pairs of the set $W(S, d)$.*
From the initialization of M, it follows that the invariant holds in the beginning. A simple inductive proof on the number of edge deletions can be used

to show that the invariant I^{APSP} always holds.

Cost Analysis : As mentioned in Lemma 4, the total cost of initializing the matrix M is $O(n^2|S|)$. Now we shall assess the total cost incurred in the three kinds of operations that we perform to maintain the matrix M.

The total cost of the first operation over any sequence of edge deletions is $O(md|S|)$.

During the second operation, for every $v \in S$, we compute the set of vertex-pairs whose distance in (*in_tree,out_tree*)(v,d) has increased. Since we do not process a vertex-pair (u,w) whenever either of them falls beyond distance d from v, therefore, a vertex-pair will be reported in this set at most $2d$ times by a tree-pair. It follows that a total of $O(n^2 d)$ time will be spent per tree-pair in computing such sets over the entire sequence of edge deletions. Since there are $|S|$ tree-pairs, the total cost incurred in the second operation will be $O(n^2 d|S|)$.

To assess the total cost incurred in the third operation, the key observation is that for a particular distance $r \le d$ and a vertex-pair (u,w), a tree-pair from the set $W(S,d)$ is considered at most once for being a witness of path-length r (from u to w). This is because a tree-pair that ceases to be a witness of path-length r from u to w can never become a witness of path-length r from u to w in future. Thus for a pair of vertices, the total cost incurred in searching the list $W(S,d)$ will be $O(d|S|)$ over any sequence of edge deletions. Since there are $O(n^2)$ pairs of vertices, the total cost incurred in the third operation will be $O(n^2 d|S|)$.

Summing up the cost of all operations, we can conclude that the total cost incurred in maintaining all-pairs shortest paths with respect to the witness set $W(S,d)$ is $O(n^2|S| + md|S| + n^2 d|S|)$, i.e., $O(n^2 d|S|)$.

Theorem 5 *Given a graph $G(V,E)$, a set of tree-pairs $W(S,d)$ for a set $S \subset V$ and a positive integer d, all-pairs shortest paths with respect to the witnesses $W(S,d)$ can be maintained under edge-deletions with optimal query time and $O(n^2 d|S|)$ total update time.*

3.3 Maintaining reachability and shortest paths corresponding to all-paths of length $\in [d', d]$

It follows from Theorem 3 (and 5) that by choosing witness set $W(V,d)$, we can maintain all-pairs reachability (shortest paths) corresponding to paths of length $\le d$. Notice that the update cost achieved is proportional to the size of the witness set. Therefore, to improve the update cost, a relevant question is : *Can we maintain all-pairs reachability (shortest paths) corresponding to paths of length $\le d$ using $o(n)$ size witness set?* We shall now use random sampling to show that indeed a sub-linear size set S suffices.

Consider any two vertices u and w in the graph. At an instance T, suppose we receive a query asking for reachability (shortest-path) from u to w. Let p_{uw} be a path (shortest path) from u to w of length $\in [d', d], d' < d$ in the graph at that instance. There are $\Omega(d')$ vertices lying on the path p_{uw}, and at least one of them will be present in a random sample of $\frac{n}{d'}$ vertices with probability $\geq 1 - \frac{1}{e}$. So in the witness set $W(S, d)$, that we are maintaining, if S is a uniform random sample of $\frac{n}{d'}$ vertices, then the reachability query from u to w at instance T will be answered correctly with probability $\geq 1 - \frac{1}{e}$. The success probability can be made arbitrarily close to 1 at the expense of increasing the sample size by a factor of $c \ln n$ as mentioned in the following lemma.

Lemma 6 *[12] Given a path p_{uw} of length l from u to w, if we sample $c\frac{n}{l} \ln n$ vertices (for any $c > 0$), then with probability $1 - \frac{1}{n^c}$, at least one of the vertices will be picked from the path p_{uw} in the sample.*

Therefore, for maintaining all-pairs reachability (shortest paths) corresponding to paths of length $\in [d', d]$, it would suffice to maintain a witness-matrix with respect to a set $W(S_{d'}, d)$, where $S_{d'}$ is a set formed by picking each vertex of the graph independently with probability $\frac{\ln n}{d'}$. Each reachability (shortest-path) query from any vertex u to another vertex w in the graph will be answered correctly with probability $\geq 1 - 1/n$ provided there is a path (shortest-path) of length $\in [d', d]$ from u to w. Expected size of the set $S_{d'}$ is $\frac{n \ln n}{d'}$. This observation can be combined with Theorems 3 and 5 to yield the following result.

Theorem 7 *Given a graph $G(V, E)$ and integers d, d' with $1 \leq d' < d \leq n$, all-pairs reachability (or shortest paths) corresponding to paths of length $\in [d', d]$, can be maintained with optimal query time w.h.p. by maintaining a witness matrix M with respect to the set $W(S_{d'}, d)$. It requires $O(\frac{d}{d'} mn \ln n + \frac{n^3 \ln n}{d'})$ total update time for maintaining all-pairs reachability, whereas for maintaining all-pairs shortest paths the total update time required is $O(\frac{d}{d'} n^3 \ln n)$.*

4 Decremental algorithm for maintaining transitive closure

We maintain all-pairs reachability corresponding to *short* and *long* paths separately. We call the paths of lengths $\leq d$ the *short paths*, and the paths of length $> d$ the *long paths*, where d will be fixed shortly. It follows that for a pair of vertices $u, w \in V$, there is a path from u to w if and only if there is a short path or a long path from u to w. Therefore, for maintaining all-pairs reachability under deletion of edges it suffices to solve the following two subproblems.

Maintaining reachability corresponding to short paths : It follows from the description given in section 2 that *out_tree* of depth d rooted at a vertex u maintains the set of vertices reachable within distance d from u. The set of *out_trees* of depth d from every vertex can thus be used for maintaining all-pairs reachability corresponding to short paths, and the total update time required is $O(mnd)$. In future we shall refer to this data structure by $\mathbf{S}_0^d$.

Maintaining reachability corresponding to long paths : Theorem 7 shows that all-pairs reachability corresponding to paths of length $\in [r, 2r]$ can be maintained by keeping a witness matrix M with respect to the set $W(S_r, 2r)$. To maintain all-pairs reachability for paths of length $\in [d, n]$, we partition the interval $[d, n]$ into $\log_2 \frac{n}{d}$ sub-intervals : $[d, 2d], \cdots, [2^i d, 2^{i+1} d], \cdots$, $[n/2, n]$. For each sub-interval $[2^i d, 2^{i+1} d]$ starting from $i = 0$, we build and maintain the set $W(S_{2^i d}, 2^{i+1} d)$. It can be seen that at any stage in the sequence of edge-deletions, if there is a path of length $\geq d$ from a vertex u to a vertex w, the set $\cup_{i=0}^{i<\log_2 n/d} W(S_{2^i d}, 2^{i+1} d)$ has a witness of reachability from u to w. Therefore, in order to maintain all-pairs reachability corresponding to long paths, it suffices to maintain a witness matrix M with respect to the set $\cup_{i=0}^{i<\log_2 n/d} W(S_{2^i d}, 2^{i+1} d)$. Now, using Theorem 7, the total update time for maintaining the witness matrix M with respect to this set will be

$$\sum_{i=0}^{i<\log_2 \frac{n}{d}} \left(mn \ln n + \frac{n^3 \ln n}{2^i d} \right) = O\left(mn \ln^2 n + \frac{n^3 \ln n}{d} \right)$$

We shall denote the above data structure by $\mathbf{L}_d^n$. We maintain all-pairs reachability by keeping the data structures $\mathbf{S}_0^d$ and $\mathbf{L}_d^n$ simultaneously. Deleting an edge will require updating both the data structures, and thus the total update time for maintaining all-pairs reachability (transitive closure) over any sequence of edge-deletions will be

$$O\left(mnd + mn \ln^2 n + \frac{n^3 \ln n}{d} \right) = O\left(mn \ln^2 n + n^2 \sqrt{m \ln n} \right) \quad \text{for} \quad d = \frac{n\sqrt{\ln n}}{\sqrt{m}}$$

Theorem 8 *Given a graph $G(V, E)$, all-pairs reachability information can be maintained under deletion of edges by an algorithm that achieves $O(1)$ query time w.h.p. and $O(n \ln^2 n + \frac{n^2\sqrt{\ln n}}{\sqrt{m}})$ amortized update time per edge deletion.*

There are two previous algorithms for maintaining transitive closure under deletion of edges with $O(1)$ query time. The first algorithm due to La Poutre and Van Leeuwen [18] achieves $O(m)$ amortized update time while the second algorithm due to Demetrescu and Italiano [6] achieves $O(\frac{n^3}{m})$ amortized update time. These two algorithms together establish an upper bound of $O(n^{\frac{3}{2}})$ on the update time. By suitably combining our algorithm with [18], we can state the following Corollary.

Corollary 9 *There exists a decremental algorithm for maintaining transitive* *closure with $O(n^{\frac{4}{3}}\sqrt[3]{\ln n})$ amortized update time and optimal query time w.h.p.*

5 Decremental algorithm for maintaining all-pairs approximate shortest paths

We introduced the terminologies of witness of reachability and witness of shortest path. In this section, we introduce one more type of witness, namely a witness of *approximate shortest path.*

Consider a pair of vertices $u, w \in V$ such that the vertex u lies in *in_tree* and the vertex w lies in *out_tree* of depth d rooted at a vertex v. The pair (*in_tree*,*in_tree*)(v, d) not only acts as a witness of reachability from the vertex u to the vertex w, it also gives an upper bound (thus estimates) of $2d$ on the distance from u to w. In other words, if $\delta(u, w)$ is the distance from vertex u to vertex w, the tree-pair (*in_tree*,*in_tree*)(v, d) acts as a witness of $\frac{2d}{\delta(u,w)}$-approximate shortest-path from u to w.

We present an efficient decremental algorithm for maintaining all-pairs $(1+\epsilon)$-approximate shortest paths for any $\epsilon > 0$. The algorithm can be visualized as a careful refinement of the algorithm for maintaining all-pairs reachability described in the previous section. The following lemma is the basis for maintaining witnesses of $(1 + 2\epsilon)$-approximate distances for all-pairs of vertices separated by distance $\in [d, d(1 + \epsilon)]$.

Lemma 10 *Given a sample $S_{\epsilon d} \subset V$ formed by picking each vertex independently with probability $\frac{\ln n}{\epsilon d}$, the set $W(S_{\epsilon d}, (1/2 + \epsilon)d)$ has a witness of $(1 + 2\epsilon)$-approximate distance with probability $1 - 1/n$, for a pair of vertices separated by distance $\in [d, d(1 + \epsilon)]$.*

PROOF. Let p_{uw} be a shortest-path from vertex u to vertex w of length $\in [d, d(1 + \epsilon)]$ (see Figure 2). Let s_{uw} be the set of the vertices lying on

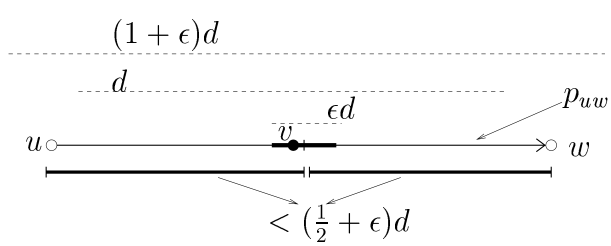

Fig. 2. a witness of $(1/2 + \epsilon)$-approximate shortest path from u to w is rooted at v

the path p_{uw} that are at distance $\leq \epsilon d/2$ from the mid-point of p_{uw}. Clearly $|s_{uw}| = \epsilon d$. It can be observed that an *in_tree* and an *out_tree* of depth $(1/2+\epsilon)d$ rooted at any $v \in s_{uw}$ captures the path p_{uw} completely, and thus establishes an upper bound of $(1 + 2\epsilon)d$ on the distance from u to w. Since the actual distance from u to w lies in the interval $\in [d, (1+\epsilon)d]$, therefore, the tree-pair $(in\_tree, out\_tree)(v, d) : v \in s_{uw}$ is a witness of $(1+2\epsilon)$-approximate shortest path from u to w.

A sample $S_{\epsilon d} \subset V$ formed by picking each vertex independently with probability $\frac{\ln n}{\epsilon d}$ has at-least one vertex from the set s_{uw} with probability $\geq 1 - 1/n$. So the set $W(S_{\epsilon d}, (\frac{1}{2}+\epsilon)d)$ has a witness of $(1+2\epsilon)$-approximate distance for a pair of vertices separated by distance $\in [d, d(1+\epsilon)]$ with probability $> 1 - \frac{1}{n}$.

It follows from Lemma 10 that in order to maintain $(1+2\epsilon)$-approximate distances/shortest-paths for all-pairs of vertices separated by distance $\in [d, (1+\epsilon)d]$, it suffices if we maintain witness matrix for all-pairs reachability with respect to the witness set $W(S_{\epsilon d}, (\frac{1}{2}+\epsilon)d)$. So we can state the following theorem along the lines of Theorem 7.

Theorem 11 *Given a graph $G(V, E)$ and $d \leq n$, all-pairs $(1+2\epsilon)$-approximate shortest paths for all-pairs of vertices separated by distance $\in [d, (1+\epsilon)d]$, can be maintained under edge-deletions with optimal query time w.h.p. by maintaining a witness matrix M for all-pairs reachability with respect to the set $W(S_{\epsilon d}, (\frac{1}{2}+\epsilon)d)$. The total update time required over any sequence of edge-deletions is $O(\frac{mn \ln n}{\epsilon} + \frac{n^3 \ln n}{\epsilon d})$.*

We use the idea of maintaining $(1+\epsilon)$-approximate shortest paths corresponding to *short* and *long* paths separately (also used for maintaining transitive closure in the previous section). We call the paths of lengths $\leq d$ the *short paths*, and the paths of length $> d$ the *long paths*, where d will be determined in the analysis.

Maintaining $(1+\epsilon)$-approximate shortest paths for vertex-pairs separated by long paths:

In order to maintain $(1+\epsilon)$-approximate shortest paths for pair of vertices separated by distance $\in [d, n]$, we partition the interval $[d, n]$ into $\ln_{1+\epsilon} \frac{n}{d}$ sub-intervals : $\{((1+\epsilon)^i d, (1+\epsilon)^{i+1} d) | i < \ln_{1+\epsilon} \frac{n}{d}\}$. For each sub-interval $[(1+\epsilon)^i d, (1+\epsilon)^{i+1} d]$ starting from $i = 0$, we build and maintain the sets $\mathbf{W}_i = W(S_{\epsilon(1+\epsilon)^i d}, (1+\epsilon)^i (\frac{1}{2}+\epsilon)d)$ and update the matrix M accordingly.

Processing of an edge deletion involves updating each tree-pair, and searching for new witness for each $u, w \in V$ if $M[u, w]$ ceases to be the witness due to the recent edge deletion. Whenever search for witness of reachability from u to w fails in list $\mathbf{W}_i$, we start search in next list $\mathbf{W}_{i+1}$. Thus the following invariant (along the lines of I^τ) will be maintained for each $u, w \in V$.

I^R : At any time if sets $\mathbf{W}_{i_1}, \mathbf{W}_{i_2}, \cdots, \mathbf{W}_{i_k} : i_1 < i_2 \cdots < i_k$ are the only sets that have witnesses of reachability from u to w, then $M[u, w]$ will point to the first witness of reachability (from u to w) in the list $\mathbf{W}_{i_1}$

Let the length of the shortest-path from vertex u to vertex w lies in interval $((1+\epsilon)^j d, (1+\epsilon)^{j+1} d)$. Lemma 10 ensures that the list $\mathbf{W}_j$ must have a tree-pair which is a witness of $(1+2\epsilon)$-approximate shortest path from u to w, and the invariant I^R implies that $M[u,w]$ will point to this tree-pair. In other words, the reachability-witness matrix M with respect to the set $\cup_{i=0}^{i<\ln_{1+\epsilon} n/d} \mathbf{W}_i$ maintains witnesses of $(1+2\epsilon)$-approximate shortest path for each pair (u,w) separated by distance $\in [d,n]$. Using Theorem 11, the total update time required for maintaining the witness matrix M over any sequence of edge-deletions will be

$$\sum_{i=0}^{i<\ln_{1+\epsilon} \frac{n}{d}} \left(\frac{mn \ln n}{\epsilon} + \frac{n^3 \ln n}{(1+\epsilon)^i \epsilon d} \right) = O\left(\frac{mn \ln n}{\epsilon^2} + \frac{n^3 \ln n}{\epsilon^2 d} \right)$$

So the amortized update time per edge deletion required for maintaining $(1+\epsilon)$-approximate shortest paths for all-pairs of vertices separated by long paths is $O(\frac{n \ln n}{\epsilon^2} + \frac{n^3 \ln n}{\epsilon^2 dm})$.

Maintaining shortest paths for vertex-pairs separated by short paths : We maintain *out_tree* of depth d from each vertex to report exact distance between any pair of vertices separated by short paths. The amortized update time per edge deletion is $O(nd)$

Balancing the update bounds for maintaining all-pairs $(1+\epsilon)$-approximate shortest path for long and short paths, we get $d = \frac{n\sqrt{\ln n}}{\epsilon\sqrt{m}}$. Hence the amortized update time per edge deletion for maintaining $(1+\epsilon)$-approximate shortest paths will be $O(\frac{n \ln n}{\epsilon^2} + \frac{n^2\sqrt{\ln n}}{\epsilon\sqrt{m}})$.

Theorem 12 *Given an unweighted graph $G(V,E)$, there exists a data structure for maintaining all-pairs $(1+\epsilon)$-approximate shortest paths/distance with optimal query time w.h.p. and $O(\frac{n \ln n}{\epsilon^2} + \frac{n^2\sqrt{\ln n}}{\epsilon\sqrt{m}})$ amortized update time per edge-deletion.*

King [16] gave a fully dynamic algorithm for maintaining all-pairs $(1+\epsilon)$ approximate shortest paths that require $O(\frac{n^2 \ln^3 n}{\epsilon^2})$ update time per edge deletion. For updates consisting of edge deletions only, our algorithm improves the update time by a factor of $O(\frac{\sqrt{m}\ln^5 n}{\epsilon})$.

6 Decremental algorithm for maintaining all-pairs distances and all-pairs shortest paths

We maintain all-pairs shortest paths/distances by keeping a witness of shortest path (if exists) for each vertex-pair. It follows from Theorem 7 that for maintaining shortest paths for all-pairs of vertices separated by distance $\in [r, 2r]$, it suffices to maintain a matrix of shortest-path witnesses with respect to the set $W(S_r, 2r)$. In order to maintain shortest paths for all-pairs of vertices (irrespective of the distance), we build and maintain $W(S_r, 2r)$ for each $r \in \{1, 2, 4, \cdots, 2^i, \cdots, n/2\}$. It follows from subsection 3.2 that with respect to the set $\cup_{i=0}^{i<\log_2 n} W(S_{2^i}, 2^{i+1})$, an entry $M[u, w]$ gets initialized to a witness of the shortest path from u to w for all $u, w \in V$ in total time $O(n^3)$.

Processing of an edge deletion involves updating each tree-pair and searching for new shortest-path witness for each pair $u, w \in V$ if path length from u to w passing through $M[u, w]$ has increased due to the recent edge deletion. While searching for a witness of path length r from u to w, we will confine our search within the list $W(S_{2^i}, 2^{i+1})$ for $2^i < r \leq 2^{i+1}$. Whenever search for witness of path of length 2^{i+1} fails in the list $W(S_{2^i}, 2^{i+1})$, we move onto the next list $W(S_{2^{i+1}}, 2^{i+2})$. We proceed in this way for vertex-pair (u, w) performing $O(n \ln n)$ work per witness list (scanning list of length $\frac{n}{2^i} \ln n$, for 2^i times). Since there are now $\log_2 n$ lists and a total of n^2 vertex-pairs, the total update cost will be $O(n^3 \ln^2 n)$ over any sequence of edge deletions. We can thus state the following theorem.

Theorem 13 *Given an unweighted graph $G(V, E)$, there exists a data structure for maintaining all-pairs shortest distances in $O(\frac{n^3 \ln^2 n}{m})$ amortized update time per edge-deletion and taking $O(1)$ time to answer a distance query w.h.p.*

Remark : Demetrescu and Italiano [7] designed an $O(\frac{n^3}{m} \ln^3 n)$ update time algorithm for maintaining all-pairs shortest paths under deletion of edges. Their algorithms achieves $O(1)$ query time w.h.p. and improves the previous $O(n^2)$ update time of King [16] for dense graphs. Our algorithm improves the update time further by a factor of $O(\ln n)$.

For initial complete graph, the new algorithm achieves $O(n \ln^2 n)$ amortized update time. But for sparse graphs the update time may be $\theta(n^2)$. So there is still no decremental algorithm for maintaining all-pairs shortest distances that achieves sub-quadratic update time for all graphs and takes $O(1)$ time to answer distance query. However, for the case of shortest path reporting problem, we are able to design a decremental algorithm that requires $O(n^{\frac{3}{2}} \sqrt{\ln n})$ update time for any graph and answers any query in optimal time. The improvement is achieved by combining the concept of filtering search [2] with the data structures $\mathbf{S}_0^d$, $\mathbf{L}_d^n$ in a novel way as follows.

Consider any two vertices u and w in a graph $G(V, E)$ under deletion of edges. Suppose we receive a query "*report distance from u to w*" at an instance. Let w be reachable from u at that instance. If the shortest path from u to w is of length $l_{uw} \leq d$, we are able to answer the query in $O(1)$ time using *out_tree*(u,d) of the data-structure $\mathbf{S}_0^d$. Otherwise $2^i d < l_{uw} \leq 2^{i+1}d$ holds for some $0 \leq i < \log_2 \frac{n}{d}$. Note that w.h.p. the set $W(S_{2^i d}, 2^{i+1}d)$ has a witness of the shortest path from u to w. But finding the witness may require $O(|S_{2^i d}|)$ time (not constant !) since it takes $O(1)$ time to find the length of the path from u to w lying in a tree-pair $\in W(S_{2^i d}, 2^{i+1}d)$. This forced us to maintain a matrix of all-pairs shortest-path witnesses so that we can answer any distance reporting query in constant time. However, for a shortest-path reporting query, we must in any case spend $O(l_{uw})$ time to report all the edges on the shortest path. Therefore, if we spend additional $O(l_{uw})$ time to find the shortest path witness, we will still be achieving $O(l_{uw})$ query time for shortest path reporting, which is optimal. (This is indeed the idea of filtering search [2]). Now to materialize this idea, we need to ensure that whenever the shortest-path to be reported lies in the interval $[d,n]$, the number of witnesses to be searched for finding the shortest path should be $O(d)$. In other words, the following inequality must hold :

$$|S_{2^i d}| < d \qquad \text{for all} \qquad 0 \leq i < \log_2 n/d$$

Since $|S_{2^i d}| = \frac{n}{2^i d} \ln n$, therefore, $\sqrt{n \ln n}$ is the right choice for d. Note that the total number of witnesses in $\mathbf{L}_d^n$ will be $O(\sqrt{n \ln n})$ for this value of d. Using the data structures $\mathbf{S}_0^d$ and $\mathbf{L}_d^n$ for $d = \sqrt{n \ln n}$, we can process any shortest-path reporting query (say, from u to w) in optimal time as follows :

First inquire if w is present in the out_tree *rooted at u. This operation takes constant time. If the answer is yes, we can report the shortest path using the* out_tree *rooted at u; otherwise inquire if w is reachable from u or not, using the witness matrix M of the data-structure $\mathbf{L}_d^n$. If $M[u,w]$ is pointing to* **null**, *we report in $O(1)$ time that there is no path from u to w. Otherwise, it can be concluded that the shortest path length $l_{uw} \in [\sqrt{n \ln n}+1, n-1]$. In this case, at least one of the vertex of the shortest path from u to w must be present w.h.p. in the sampled set of witnesses. We search the entire set of witnesses to find the witness of shortest path. The size of the witness set being $O(\sqrt{n \ln n})$, we can thus find the witness of the shortest path from u to w in $O(\sqrt{n \ln n})$ time and then report the shortest path from u to w passing through it in additional $O(l_{uw})$ time. Since $l_{uw} > \sqrt{n \ln n}$, the total time taken for processing a shortest path reporting query is $O(l_{uw})$ (and hence optimal).*

The amortized update time per edge deletion for the two data structures $\mathbf{S}_0^d$ and $\mathbf{L}_d^n$ will be

$$O\left(nd + \frac{n^3}{md} \ln n\right) \quad = \quad O\left(n^{\frac{3}{2}} \sqrt{\ln n}\right) \quad \text{for} \quad d = \sqrt{n \ln n}$$

Theorem 14 *An unweighted graph $G(V,E)$ can be preprocessed to build a data structure that answers any on-line shortest path reporting query in optimal time w.h.p. while ensuring $O(\min(n^{\frac{3}{2}}\sqrt{\ln n}, \frac{n^3 \ln^2 n}{m}))$ amortized update time per edge-deletion.*

7 Space Requirement of our Algorithms

It can be seen that all the algorithms given in this paper employ a witness matrix M and $O(n)$ number of *in_trees* and *out_trees*. The matrix M clearly occupies $\theta(n^2)$ space. Although the earlier scheme given in [16] requires $\theta(m)$ space for maintaining an *in_tree* (or an *out_tree*), it has been improved to $O(n)$ by King and Thorup [17]. Thus the total space requirement for maintaining $O(n)$ number of *in_trees* or *out_trees* is $O(n^2)$. Hence the space requirement of all the algorithms given in this paper is $O(n^2)$.

8 Conclusion and Open Problems

In this paper, we presented improved decremental algorithms for maintaining transitive closure and all-pairs exact/approximate shortest paths with optimal query time. The key concept underlying our algorithms has been the implicit scheme for maintaining reachability/shortest-path information. Later we also use this scheme along with a new hierarchical distance reporting scheme in paper [1] for the problem of maintaining all-pairs approximate shortest paths in undirected unweighted graphs under deletion of edges. We show that it is possible to achieve $O(n^{1+\delta})$ bound (for arbitrarily small $\epsilon > 0$) on amortized update cost at the expense of increased approximation factor of the distance reported.

There have been new developments in the field of dynamic graph algorithms ever since we submitted this paper. For the problem of all-pairs reachability under edge-deletions, Roditty and Zwick [19] design an algorithm that requires $O(n)$ amortized update cost, and thus improves our bounds for all-pairs reachability problem. For fully dynamic maintenance of all-pairs shortest path, Demetrescu and Italiano [8] present an algorithm that achieves $O(n^2)$ amortized cost per update which is superior to the previous $O(n^{2.5})$ bound achieved by King [16]. Note that for all-pairs approximate shortest paths in undirected graphs, there are static algorithms [3,9] that require $O(n^2 \ln n)$ time. In these cases, the fully-dynamic algorithm of [8] provides only logarithmic improvement. Therefore, it is desirable to design a fully dynamic algorithm for maintaining all-pairs approximate shortest paths with $o(n^2)$ update time, possibly at the expense of increased query time.

9 Acknowledgments

We sincerely thank Camil Demetrescu for providing a very helpful clarification about the paper [6]. We also thank L. Sunil Chandran and L. Shankar Ram for their valuable suggestions and comments on an earlier draft of this paper.

References

[1] S. Baswana, R. Hariharan, and S. Sen. Maintaining all-pairs approximate shortest paths under deletion of edges. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete A lgorithms (SODA)*, pages 394–403, 2003.

[2] B. Chazelle. Filtering search : A new approach to query-answering. *SIAM J. Comput.*, 15:703–724, 1986.

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

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[5] C. Demetrescu. Personal communication.

[6] C. Demetrescu and G. Italiano. Fully dynamic transitive closure : Breaking through the $o(n^2)$ barrier. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 381–389, 2000.

[7] C. Demetrescu and G. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 260–267, 2001.

[8] C. Demetrescu and G. Italiano. A new approach to dynamic all-pairs shortest paths. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 159–166, 2003.

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

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

[11] R. Goldman, N. Shivakumar, S. Venkatasubramanian, and H. Garcia-Molina. Proximity searches in databases. In *Proceedings of the 24th VLDB Conference*, pages 26–37, 1998.

[12] D. Greene and D. Knuth. Mathematics for the analysis of algorithms. *Birkhauser, Boston*, 1982.

[13] M. R. Henzinger and V. 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.

[14] N. Horspool. Incremental generation of LR parsers. *Computer Languages*, 15:205–223, 1990.

[15] H. V. Jagadish. A compression technique to materialize transitive closure. *ACM Transaction on Database Systems (TODS)*, 15:558–598, 1990.

[16] V. 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.

[17] V. King and M. Thorup. A space saving trick for directed dynamic transitive closure and shortest path algorithms. In *Proceedings of 7th International Computing and Combinatorics Conference, COCOON*, volume 2108 of *LNCS*, pages 268–277. Springer Verlag, 2001.

[18] H. L. Poutre and J. V. Leeuwen. Maintenance of transitive closure and transitive reduction of a graph. In *Proceedings of Workshop on Graph-Theoretic Concepts in Computer Science*, volume 314 of *LNCS*, pages 106–120. Springer Verlag, 1988.

[19] L. Roditty and U. Zwick. Improved dynamic reachability algorithms for directed graphs. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 679–688, 2002.

[20] U. Zwick. All-pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 310–319, 1998.