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

Surender Baswana *
Sandeep Sen †
Department of Computer Science and Engineering
Indian Institute of Technology
Hauz Khas, New Delhi
{sbaswana,ssen}@cse.iitd.ernet.in

Ramesh Hariharan
Department of Computer Science and Automation
Indian Institute of Science
Bangalore, India
ramesh@csa.iisc.ernet.in

ABSTRACT

We present improved algorithms for maintaining transitive closure and all-pairs shortest paths/distances in a digraph under deletion of edges.

For the problem of transitive closure, the previous best known algorithms, for achieving $O(1)$ query time, require $O(\min(m, \frac{n^3}{m}))$ amortized update time, implying an upper bound of $O(n^{\frac{3}{2}})$ on update time per edge-deletion. We present an algorithm that achieves $O(1)$ query time and $O(n \log^2 n + \frac{n^2}{\sqrt{m}}\sqrt{\log n})$ update time per edge-deletion, thus improving the upper bound to $O(n^{\frac{4}{3}}\sqrt[3]{\log n})$.

For the problem of maintaining all-pairs shortest distances in unweighted digraph under deletion of edges, we present an algorithm that requires $O(\frac{n^3}{m}\log^2 n)$ amortized update time and answers a distance query in $O(1)$ time. This improves the previous best known update bound by a factor of $\log n$. For maintaining all-pairs shortest paths, we present an algorithm that achieves $O(\min(n^{\frac{3}{2}}\sqrt{\log n}, \frac{n^3}{m}\log^2 n))$ amortized update time and reports a shortest path in optimal time (proportional to the length of the path). For the latter problem we improve the worst amortized update time bound by a factor of $O(\sqrt{\frac{n}{\log n}})$.

We also present the first decremental algorithm for maintaining all-pairs $(1+\epsilon)$ approximate shortest paths/distances, for any $\epsilon > 0$, that achieves a sub-quadratic update time of $O(n \log^2 n + \frac{n^2}{\sqrt{\epsilon m}}\sqrt{\log n})$ and optimal query time.

Our algorithms are randomized and have one-sided error for query (with probability $O(1/n^c)$ for any constant c).

*Work was supported in part by a fellowship from Infosys Technologies Ltd., Bangalore.

†Work was supported in part by an IBM UPP award.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
STOC'02, May 19-21, 2002, Montreal, Quebec, Canada.
Copyright 2002 ACM 1-58113-495-9/02/0005 ...$5.00.

Categories and Subject Descriptors

E.1 [**Data Structures**]: Graphs and networks; F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Algorithms

Keywords

BFS tree, dynamic, graph, transitive closure, shortest paths

1. INTRODUCTION

The following two problems are among the most fundamental algorithmic problems in computer science :

- **Transitive Closure :** Process a given directed graph $G(V, E)$ 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 directed 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. However, there exist algorithms based on fast matrix multiplication that achieve sub-cubic time for the two problems. In particular, using the fastest known matrix multiplication algorithms (Coppersmith and Winograd [2]), the best bound on computing transitive closure is $O(n^{2.376})$, whereas for the all-pairs shortest paths problem, it is $O(n^{2.575})$ [11].

There are a lot of applications which require efficient solutions of the above problems for a dynamic graph. In these applications, an initial graph is given, followed by an online sequence of queries interspersed with updates which can be insertion or deletion of edges. We have to carry out the updates and answer the queries online in an efficient manner. Each query has to be answered with respect to the present state of the graph, i.e., incorporating all updates preceding the query. One trivial solution for solving 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 problem defined for a static graph, there exists its counterpart in dynamic graph. 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 the two problems mentioned above. The algorithms are Monte-Carlo with one sided error (the probability of error being inverse polynomial).

1.1 Previous work and our contribution

Transitive Closure : Poutre and Leeuwen [10] 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 [4] gave a decremental algorithm for the problem that requires $O(n^3/m)$ amortized update time which is better for non-sparse graphs. For an initial complete graph, the algorithm achieves $O(n)$ amortized update time per edge deletion [3], but for sparse graphs, the update time can be even $O(n^2)$. 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 [7] gave a decremental randomized algorithm that achieves amortized update time $(n \log^2 n)$ but at the expense of increased query time of $O(\frac{n}{\log n})$. The query has one sided error [1] with probability $\frac{1}{n}$. This increased (non-constant) time for answering a query is not acceptable for many applications.

In this paper, we present an efficient algorithm that achieves $O(1)$ query time w.h.p. and requires $O(n \log^2 n + \frac{n^2}{\sqrt{m}} \log n)$ 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 our algorithm with [10], we obtain an upper bound of $O(n^{\frac{4}{3}} \sqrt[3]{\log n})$ for the problem of maintaining transitive closure with optimal query time.

All-pairs shortest paths :

Demetrescu and Italiano [5] gave $O(\frac{n^3}{m} \log^3 n)$ amortized update time algorithm for maintaining all-pairs shortest paths under deletion of edges while achieving optimal query time w.h.p. Their algorithm improves the previous $O(n^2)$ update time of [8] for non-sparse graphs. We present two decremental algorithms for the all-pairs shortest path problem. For distance reporting problem, we give a simpler combinatorial algorithm that requires $O(\frac{n^3}{m} \log^2 n)$ amortized update time while achieving $O(1)$ query time. For shortest path reporting problem, we use the idea of filtering search in a novel way to design an algorithm that achieves $O(\min(n^{\frac{3}{2}} \sqrt{\log n}, \frac{n^3}{m} \log^2 n))$ update time while achieving optimal query time. Hence we reduce the upper bound on the worst case amortized update time for the latter problem by a factor of $O(\sqrt{\frac{n}{\log n}})$.

[1] The answer to any query may be incorrect in the sense that it may not report a path when there exists one. subsequently we will use w.h.p. to denote probability exceeding $1 - \frac{1}{n}$

Furthermore, we present an efficient decremental algorithm that offers a trade-off between update time and approximation factor of the shortest path. For maintaining $(1+\epsilon)$ approximate all-pairs shortest paths, our algorithm achieves $O(n \log^2 n + \frac{n^2}{\sqrt{\epsilon m}} \sqrt{\log n})$ 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. By using techniques from [9], we show that the space requirement of our data structures is $O(n^2)$.

2. OVERVIEW OF OUR ALGORITHMS

Our algorithms use an efficient data structure for maintaining the set of vertices lying within distance d from a given vertex under deletion of edges. This objective can be achieved if we can maintain BFS tree up to depth d. It follows from the work of Henzinger and King et al. [7] that BFS tree of depth d rooted at a vertex v can be maintained under deletion of edges using a data structure $T_d^{v\rightarrow}$ which can support the following queries :

- *Is there a path from v to w of length $\leq d$?*
- *Report the shortest path (or distance) from v to w of length $\leq d$ (if exists).*

in optimal time and requires $O(d)$ amortized update time per edge deletion.

We will refer to the data structure $T_d^{v\rightarrow}$ as the **out_tree** of depth d rooted at v. A similar data structure ($T_d^{v\leftarrow}$) can also be maintained (with matching performance) to dynamically keep the set of vertices for whom v lies within distance d. We shall name the data structure $T_d^{v\leftarrow}$ as the **in_tree** of depth d rooted at v.

2.1 Main Idea

In order to develop efficient decremental algorithm for transitive closure (and all-pairs shortest paths), we explore different ways of maintaining all-pairs reachability (shortest paths). One way is to maintain reachability (shortest path) *explicitly* from each vertex. Building and maintaining the **out_tree** up to depth n from every vertex $v \in V$, we can maintain all-pairs reachability (shortest paths) in $O(n^2)$ amortized update time per edge deletion.

There is an alternate implicit way of maintaining all-pairs reachability (shortest paths), where reachability (shortest paths) information from u to v is maintained by keeping a *witness*. Let $\mathcal{W}_{uv}$ be the set of vertices w such that there is a path (shortest path) from u to v passing through w. In a way, $\mathcal{W}_{uv}$ is the set of vertices that are witnesses for reachability (shortest paths) from u to v, and the data structures **in_tree** and **out_tree** of depth n built on any of $w \in \mathcal{W}_{uv}$ keep this information (implicitly). The problem is to maintain these witnesses dynamically for each pair. We present efficient algorithm for maintaining witnesses of all-pairs reachability (shortest paths) under deletion of edges.

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 $[r, 2r]$. These algorithms form the basis for developing efficient decremental algorithms for the two problems. The strategy of maintaining reachability implicitly (by keeping witnesses) proves to be efficient for maintaining all-pairs

Table 1: All the update bounds (old and new) are amortized. Throughout this paper, our query times are optimal but are randomized Monte-Carlo having one-sided error.

Comparison of the new and the previous update cost			
Problem	Query	Previous	New
All-pairs Reachability	*Is v reachable from u ?*	$O\left(n^{\frac{3}{2}}\right)$	$O\left(n^{\frac{4}{3}}\sqrt[3]{\log n}\right)$
	report a path from u to v		
Approximate APSP	*report $(1+\epsilon)$ approx. distance from u to v*	*none*	$O\left(n\log^2 n + \frac{n^2}{\sqrt{\epsilon m}}\sqrt{\log n}\right)$
	report $(1+\epsilon)$ approx. shortest path		
Exact APSP	*report the distance from u to v*	$O\left(\frac{n^3}{m}\log^3 n\right)$	$O\left(\frac{n^3}{m}\log^2 n\right)$
	report the shortest path from u to v	$O(\frac{n^3}{m}\log^3 n)$	$O\left(\min\left(n^{\frac{3}{2}}\sqrt{\log n}, \frac{n^3}{m}\log^2 n\right)\right)$

reachability corresponding to long paths. On the other hand, the strategy 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 strategy together to achieve improved update time of $O(n\log^2 n + \frac{n^2}{\sqrt{m}}\sqrt{\log n})$ per edge deletion. It turns out that our data structure for maintaining transitive closure can be directly used to maintain all-pairs stretch-2 paths[2] in unweighted digraphs.

For the problem of maintaining all-pairs shortest distances, the strategy of maintaining witness of shortest path for every vertex pair leads to achieving $O(\frac{n^3}{m}\log^2 n)$ update time. For the case when query is to report the shortest path, we use idea of filtering search [1] to reduce the update time further to $O(\min(n^{\frac{3}{2}}\log n, \frac{n^3}{m}\log^2 n))$.

3. MAINTAINING TRANSITIVE CLOSURE AND APSP FOR LENGTHS IN AN INTERVAL $[\frac{\mathbf{R}}{\mathbf{2}}, \mathbf{R}]$

In this section we design efficient algorithms for maintaining all-pairs reachability and shortest paths corresponding to paths of length $\in [\frac{r}{2}, r]$. For the reachability problem, the algorithm will maintain a witness for every pair $u, v \in V$ if there is path of length $\in [\frac{r}{2}, r]$ from u to v. For the all-pairs shortest path problem, the algorithm will maintain a witness of the shortest path for every pair $u, v \in V$ among all paths from u to v of length $\in [\frac{r}{2}, r]$.

We start with the design of efficient algorithm for maintaining all-pairs reachability (shortest paths) with respect to a given set of vertices $W \subseteq V$ as witnesses, i.e., the set of paths under consideration is restricted to the paths going through vertices of set W only. The algorithm can be used for maintaining all-pairs reachability (shortest paths) if we choose $W = V$. Subsequently to improve the update time, we reduce the number of witnesses required using random sampling based on the following well known observation (see, e.g., Greene and Knuth [6]) :

Given a path of length k in a graph, a random sample of $c\frac{n}{k}\ln n$ vertices will have at least one vertex belonging to the path with probability $1 - \frac{1}{n^c}$ for any fixed constant $c > 0$.

The observation mentioned above was exploited by Henzinger and King [7] for designing a decremental algorithm to maintain all-pairs reachability with $O(n\log^2 n)$ update time at the expense of $O(\frac{n}{\log n})$ query time. We extend this approach to its full potential for maintaining all-pairs shortest paths and transitive closure with optimal query time.

[2]Path with at most twice the length of the shortest path

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

Let W be the set of witness vertices kept in a list. There is a path from u to v of length $\leq d$ passing through some $w \in W$ if u and v lie respectively in the **in_tree** and the **out_tree** of depth d rooted at w. Thus it takes $O(1)$ time to test whether w is a witness of reachability from u to v. Finding out if there is any witness in W for reachability from u to v will require querying **in_tree** and **out_tree** data structures for each $w \in W$. Thus the query time will be $O(|W|)$. This scheme was employed in [7]. In order to achieve $O(1)$ query time, we maintain a witness matrix M. For each pair (u, v), at every stage $M[u, v]$ points to a vertex $w \in W$ such that u and v lie respectively in **in_tree** and **out_tree** of depth d rooted at w. We initialize the matrix M as follows : The matrix M has all entries pointing to **null** initially. Starting from the first vertex in the list W, we build **in_tree** and **out_tree** data structures of depth d for every vertex $w \in W$. For every $u \in$ **in_tree** of w and for every $v \in$ **out_tree** of w, we update $M[u, v]$ to point to w if it was pointing to **null** previously (w is a witness of reachability from u to v). It follows easily that that it requires $O(n^2)$ time per vertex $w \in W$ to perform this task.

Notice that as the edges are being deleted, the current $M[u, v]$ may cease to be the witness of reachability from u to v. This happens when either u ceases to belong to **in_tree** of $M[u, v]$ or v ceases to belong to **out_tree** of $M[u, v]$.

To maintain validity of entries of M after an edge deletion, we process each $w \in W$ (starting from the first vertex) as follows. We update **in_tree** and **out_tree** rooted at w for the edge deletion and compute the sets X, Y of vertices that cease to belong to **in_tree** and **out_tree** of w respectively. This step takes $O(|X|+|Y|)$ time in addition to $O(d)$ amortized time for updating the BFS trees at w. For every $u \in X$, we find the set s_u of vertices such that $M[u, v]$ is w for each $v \in s_u$. (It takes $O(n)$ time per $u \in X$ to find s_u). It can be seen that w has ceased to be a witness of reachability from u to each $v \in s_u$ and so a new witness is to be searched for. We search the list W starting from the vertex following w and find a witness of reachability from u to v (spending $O(1)$ time per vertex in W). If we reach the end of list, we update $M[u, v]$ to point to **null**, otherwise we update $M[u, v]$ to point to the new witness found. In a similar way, we process the set Y of vertices that cease to belong to **out_tree** of w.

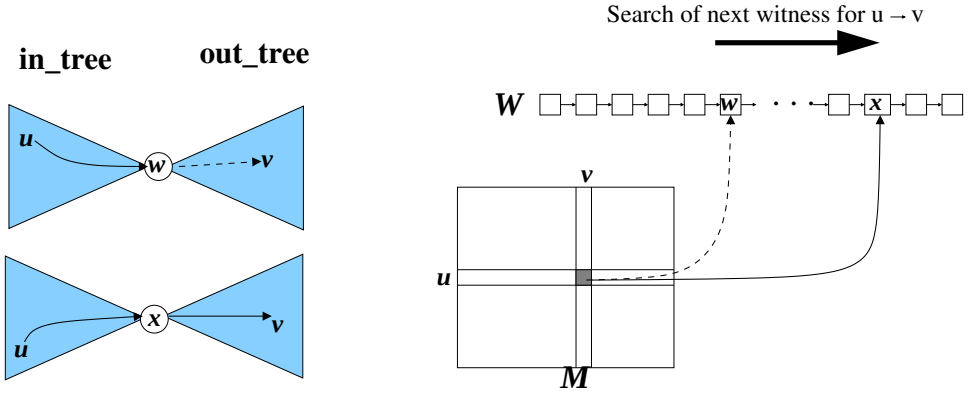

Figure 1: The search of a new witness for $u \rightarrow v$ reachability (when w ceases to be the witness) stops at vertex x.

The following invariant is maintained throughout the series of edge deletions.
$\mathcal{I}^{\tau}$: *$M[u,v]$ points to the first vertex in the list W that is a witness of reachability from u to v.*

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), therefore, we can conclude that the invariant $\mathcal{I}^{\tau}$ holds always.

Cost Analysis : As mentioned above, the total cost of initializing the witness matrix M is $O(n^2|W|)$. We now assess the total cost over all edge deletions. It takes $O(md)$ cost for maintaining the **in_tree** and the **out_tree** of depth d rooted at a vertex over the entire sequence of edge deletions, thus amounting to $O(md|W|)$ total cost for all the vertices of W. Also for every $w \in W$, we incur $O(n)$ cost per vertex that ceases to belong to the **in_tree** (or the **out_tree** of w), amounting to a total of $O(n^2)$ cost per vertex of W. To assess the total cost for searching for a new witness in W, note that we consider a vertex of W exactly once for being a witness for a pair (u,v). (This is because a vertex that ceases to be witness can never become a witness again for a given pair $u,v)$. Hence a total of $O(n^2|W|)$ cost will be spent in searching for a new witness over all vertex pairs. Summing up the cost of all operations, we conclude that total cost incurred in maintaining all-pairs reachability over any sequence of edge deletions is $O(md|W| + n^2|W|)$.

LEMMA 3.1. *Given a digraph $G(V,E)$ and a set of vertices $W \subseteq V$, the all-pairs reachability for paths of length $\leq d$ with respect to the witnesses W can be maintained in $O((d+\frac{n^2}{m})|W|)$ amortized update time per edge-deletion and $O(1)$ query 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 be the set of witness vertices kept in a list. We maintain a witness matrix M : For each $u,v \in V$, $M[u,v]$ points to the first vertex in the list W which is witness of the shortest path of length $\leq d$ among all paths from u to v passing through any vertex $w \in W$. It takes $O(n^2|W|)$ total time to initialize the matrix M after building **in_tree** and **out_tree** from each $w \in W$.

As the edges are being deleted, the current $M[u,v]$ may cease to be the witness for the shortest path from u to v. This happens when u (or v) increases its distance to (from) $M[u,v]$ in **in_tree** (**out_tree**) rooted at $M[u,v]$. To maintain validity of entries of M after an edge deletion, we process each $w \in W$ (starting from the first vertex) as follows. We update **in_tree** and **out_tree** rooted at w for the edge deletion and compute the set X (Y) of vertices whose distance to (from) w increases. This step takes $O(|X| + |Y|)$ time in addition to $O(d)$ amortized time for updating the BFS trees at w. It can be seen that for each $u \in X, v \in V$ (likewise $u \in V, v \in Y$), the length of the shortest path going through w has increased and so we need to search for a new witness if $M[u,v]$ points to w. We process each such pair (u,v) as follows :

Let r be the shortest path distance from u to v prior to the recent edge deletion. We scan the list W starting from the successor of $M[u,v]$ in search of a witness for path length r. If we find one, we stop; otherwise we conclude that there is no path from u to v passing through any $x \in W$ with length $\leq r$. In this case, we perform another scan starting from the head of the list W in search of a witness of path-length $r+1$. We increment r until we find one witness or r increases beyond d. In the latter case, we conclude that there is no path from u to v of length $\leq d$ that passes through any $x \in W$.

The following invariant is maintained throughout the series of edge deletions.
$\mathcal{I}^{APSP}$: *At every stage $M[u,v]$ points to the first vertex in the list W that is a witness of the shortest path from u to v among all paths from u to v passing through any $w \in W$.*

From the initialization of M, it follows that the invariant holds in the beginning. A simple inductive proof (induction on number of edge deletion) can be used to show that the invariant $\mathcal{I}^{APSP}$ holds always.

Cost Analysis : The total cost of initializing the matrix M and maintaining **in_tree** and **out_tree** of depth d for every vertex in W, is $O(n^2|W| + md|W|)$. For every $w \in W$, after an edge deletion we compute the set of (u,v) pair whose shortest path length passing through w has increased. Since we do not process a pair (u,v) whenever either of them falls beyond distance d from w, therefore, a pair (u,v) will be reported in this set at most $2d$ times for a witness w. It follows that a total of $O(n^2d)$ time will be spent in

computing such set over the sequence of edge deletions. Also note that for a particular distance $r \leq d$ and a pair $u, v \in V$, a vertex of the set W is considered at most once for being a witness of path length r (from u to v). (This is because a vertex that ceases to be witness of shortest distance r can never become a witness of distance r for a given pair u, v). Thus for a pair of vertices, the total cost incurred in searching the list will be $O(d|W|)$. Summing up the cost of all operations, we can conclude that the total cost incurred in maintaining all-pairs reachability is $O(n^2|W| + md|W| + n^2 d|W|)$, i.e., $O(n^2 d|W|)$.

LEMMA 3.2. *Given a digraph $G(V, E)$ and a set of vertices $W \subseteq V$, all-pairs shortest paths of length $\leq d$ with respect to the witness vertices W, can be maintained in $O(\frac{n^2 d|W|}{m})$ amortized update time per edge-deletion and $O(1)$ query time.*

3.3 Maintaining all-pairs reachability and shortest paths for paths of length $\in [\frac{r}{2}, r]$

It follows from Lemma 3.1 (and 3.2) that choosing witness set $W = V$, we can maintain all pairs reachability (shortest-paths) corresponding to paths of length $\leq d$. Notice that the update cost achieved is proportional to the size of witness set W. Therefore, to improve the update cost, a relevant question is : *Can we maintain all-pairs reachability (shortest paths) corresponding to paths of length $\leq d$ using $o(n)$ size witness set?*. It can be observed that for maintaining reachability (shortest paths) from u to v, it would suffice if at every stage at least one vertex on a path (the shortest path) from u to v is present in the witness set W. For two vertices separated by a path of length $\in [\frac{r}{2}, r]$, there are $\Omega(r)$ witnesses and therefore with probability $1 - \frac{1}{e}$, at least one of them will be present in a random sample of $\frac{2n}{r}$ vertices. The success probability bound can be made arbitrarily close to 1, as mentioned in the following Lemma.

LEMMA 3.3. *[6] Given a path p_{uv} of length l from u to v, 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_{uv}.*

Therefore, for maintaining all-pairs reachability (shortest paths) corresponding to path lengths $\in [\frac{r}{2}, r]$, the algorithms of the previous subsections would require $|W| = \frac{2n}{r} \ln n$. The query answered will have one sided error with probability $\frac{1}{n}$.

In future, W_r will denote a witness set formed by random sampling $\frac{n}{r} \ln n$ vertices and F_r will denote the forest of **in_tree** and **out_tree** of depth r from each $w \in W_r$.

THEOREM 3.4. *Given a graph $G(V, E)$ and $r \leq n$, there exists a data structure (W_r, F_r) using which the witness matrix M, for all-pairs reachability (or shortest paths) corresponding to paths of length $\in [\frac{r}{2}, r]$, can be maintained with optimal query time w.h.p. For maintaining all-pairs reachability, it requires $O(n \ln n + \frac{n^3 \ln n}{rm})$ amortized update time, whereas for maintaining all-pairs shortest paths the update time required is $O(\frac{n^3 \ln n}{m})$ per edge deletion.*

4. MAINTAINING TRANSITIVE CLOSURE

We use the idea of maintaining *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 $u, v \in V$, there is a path from u to v iff there is a short path or a long path from u to v. Therefore, for maintaining all-pairs reachability under deletion of edges it suffices to solve the following two sub-problems.

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, and the amortized update time for maintaining an **out_tree** is $O(d)$ per edge deletion. 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 amortized update time required is

$$T_{(0,d)} = O(nd) \tag{1}$$

In future we shall refer to this data structure by $\mathcal{S}_0^d$.

Maintaining reachability corresponding to long paths : The Theorem 3.4 shows that all-pairs reachability corresponding to paths in range $[\frac{r}{2}, r]$ can be maintained by keeping a witness matrix M (updated using the data structure (W_r, F_r)). In order to maintain all-pairs reachability for paths in range $[d, n]$, we partition the interval $[d, n]$ into $\log \frac{n}{d}$ 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 the $(W_{2^i d}, F_{2^i d})$ data structure and update the matrix M accordingly. Processing of an edge deletion involves updating each $F_{2^i d}$ and searching for new witness for each $u, v \in V$ if $M[u, v]$ ceases to be the witness due to the recent edge deletion. Whenever search for witness of reachability from u to v fails in list $W_{2^i d}$, we start search in next list $W_{2^{i+1} d}$. Thus the following invariant (along the lines of $\mathcal{I}^\tau$) will be maintained for each $u, v \in V$.

$\mathcal{I}^R$: At any time if lists $W_{i_1}, W_{i_2}, \cdots, W_{i_k} : i_1 < i_2 \cdots < i_k$, have witnesses of reachability from u to v, then $M[u, v]$ will point to the first witness of reachability (from u to v) in the list W_{i_1}.

Thus the matrix M maintains witnesses of all-pairs reachability corresponding to long paths. Now, using the Theorem 3.4, the update time for maintaining the witness matrix M, using the data structures $(W_{2^i d}, F_{2^i d}), 0 \leq i \leq \log \frac{n}{d}$, will be

$$\begin{aligned} T_{(d,n)} &= \sum_{i=0}^{\log \frac{n}{d}} \left(n \ln n + \frac{n^3 \ln n}{2^i dm} \right) \\ &= O\left(n \log^2 n + \frac{n^3 \ln n}{dm} \right) \end{aligned} \tag{2}$$

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

$$T_\tau = T_{(0,d)} + T_{(d,n)}$$

Using equations 1 and 2, it follows that

$$T_\tau = O\left(nd + n\log^2 n + \frac{n^3 \ln n}{dm}\right)$$
$$= O\left(n\log^2 n + \frac{n^2\sqrt{\ln n}}{\sqrt{m}}\right) \quad \text{for} \quad d = \frac{n\sqrt{\ln n}}{\sqrt{m}}$$

THEOREM 4.1. *Given a digraph $G(V,E)$, its transitive closure can be maintained using an algorithm achieving $O(1)$ query time w.h.p. and $O(n\log^2 n + \frac{n^2\sqrt{\ln n}}{\sqrt{m}})$ amortized update time per edge deletion.*

Previously there were two algorithms for maintaining transitive closure under deletion of edges with $O(1)$ query time. The first algorithm due to Poutre and Leeuwen [10] achieved $O(m)$ amortized update time while the second algorithm due to Demetrescu and Italiano [4] achieved $O(\frac{n^3}{m})$ 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 [10], we can state the following Corollary.

COROLLARY 4.2. *There exists a decremental algorithm for maintaining transitive closure that achieves $O(n^{\frac{4}{3}}\sqrt[3]{\log n})$ amortized update time and optimal query time w.h.p.*

5. MAINTAINING ALL-PAIRS APPROXIMATE SHORTEST PATHS

We show that the algorithm for maintaining all-pairs reachability described in the previous section can be used to maintain all-pairs shortest paths with stretch at most 2.

If the shortest path length from a vertex u to a vertex v is $\leq d$, we can even report exact shortest distance using the **out_tree** rooted at u. Otherwise let $2^i d \leq r \leq 2^{i+1}d$ be the shortest path distance from u to v at a given time. The witness list $W_{2^i d}$ has a witness w_{uv} for a path from u to v with very high probability. The length of the path from u to v passing through w_{uv} is at most twice the length of the shortest path. It follows from invariant $\mathcal{I}^R$ that $M[u,v]$ will point to w_{uv}. Hence for each pair (u,v) with shortest path $> d$, $M[u,v]$ points to a witness of path from u to v of stretch at most 2. Thus we can maintain all-pairs shortest paths of stretch at most 2 under deletion of edges. To achieve a stretch of $(1+\epsilon)$ for arbitrarily small ϵ, we build $\mathcal{L}_d^n$ using a set of (W_r, F_r) for each $r \in \{d, d(1+\epsilon), \cdots, d(1+\epsilon)^i, \cdots, n\}$. The update time required will be $O(n\log^2 n + \frac{n^2}{\sqrt{m}}\sqrt{\log_{1+\epsilon} n}) = O(n\log^2 n + \frac{n^2}{\sqrt{\epsilon m}}\sqrt{\log n})$.

THEOREM 5.1. *Given an unweighted digraph $G(V,E)$, there exists a data structure for maintaining all-pairs $(1+\epsilon)$ approximate shortest paths/distance with optimal query time w.h.p. and $O(n\log^2 n + \frac{n^2}{\sqrt{\epsilon m}}\sqrt{\log n})$ amortized update time.*

King et al. [8] gave fully dynamic algorithm for maintaining all-pairs $(1+\epsilon)$ approximate shortest paths that require $O(\frac{n^2\log^3 n}{\epsilon^2})$ update time per edge deletion. For updates consisting of edge deletions only, our algorithm thus improves the update time by a factor of $O(\sqrt{\frac{m\log^5 n}{\epsilon^3}})$.

6. MAINTAINING ALL-PAIRS SHORTEST PATHS/DISTANCES

We maintain all-pairs shortest paths/distances by keeping a witness of shortest path for each vertex pair. The Theorem 3.4 shows that the matrix M for witnesses of all-pairs shortest paths in the range $[\frac{r}{2}, r]$ can be maintained using the data structure (W_r, F_r). In order to maintain all pairs shortest paths, we build and maintain (W_r, F_r) for each $r \in \{1, 2, 4, \cdots, 2^i, \cdots, n\}$. It follows (see subsection 3.2) that using the set $\{(W_r, F_r)\}$, the entry $M[u,v]$ gets initialized to a witness of the shortest path from u to v for all $u, v \in V$ in total time $O(n^3)$.

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

THEOREM 6.1. *Given an unweighted digraph $G(V,E)$, there exists a data structure for maintaining all-pairs shortest distances in $O(\frac{n^3\log^2 n}{m})$ amortized update time per edge-deletion and taking $O(1)$ time to answer a distance query w.h.p.*

Remark : Demetrescu and Italiano [5] designed an $O(\frac{n^3}{m}\log^3 n)$ 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 [8] for non-sparse graphs. Our algorithm improves the update time further by a factor of $O(\log n)$.

For initial complete graph, the new algorithm achieves $O(n\log^2 n)$ amortized update time. But for sparse graphs the update time may be $O(n^2)$. So there is still no decremental algorithm for maintaining all-pairs shortest distances that achieves sub-quadratic update time for all digraphs 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 $O(n^{\frac{3}{2}}\sqrt{\log n})$ update time for any digraph and answers any query in optimal time. The improvement is achieved by combining the concept of filtering search [1] with the data structures $\mathcal{S}_0^d$, $\mathcal{L}_d^n$ in a novel way as follows.

Recall the data structures $\mathcal{S}_0^d$ and $\mathcal{L}_d^n$ described earlier. If the shortest path is of length $l_{uv} \leq d$, we are able to answer the distance query exactly in $O(1)$ time using **out_tree** rooted at u. Otherwise $2^{i-1}d \leq l_{uv} < 2^i d$ holds for some $2 \leq i < \log\frac{n}{d}$. Note that w.h.p. the list $W_{2^i d}$ of sampled vertices has a witness of the shortest path from u to v. But finding the witness may take $O(|W_{2^i d}|) = O(\frac{n}{2^i d}\log n)$ time (since it takes $O(1)$ time to find length of the shortest path from u to v going through a witness w). This forced us to change our approach for achieving $O(1)$ query time for distance reporting problem. However, for a shortest-path reporting query, we must in any case spend $O(l_{uv})$ time to report all the edges on the path. Therefore, if we spend additional $O(l_{uv})$ time to find shortest path witness, we will still be achieving $O(l_{uv})$ query time for path reporting (which is

optimal). This is indeed the idea of filtering search [1]. Now choosing $d = \sqrt{n \log n}$ ensures that $|W_{2^i d}| \leq d < l_{uv}$ for all i. Our algorithm employs the data structures $\mathcal{S}_0^d$ and $\mathcal{L}_d^n$ for $d = \sqrt{n \log n}$ and processes a shortest-path (say, from u to v) reporting query in optimal time as follows :

First inquire if v 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 v is reachable from u or not, using the witness matrix M. If $M[u, v]$ is pointing to* **null**, *we report in $O(1)$ time that there is no path from u to v. Otherwise, it can be concluded that the shortest path length $l_{uv} \in [\sqrt{n \log n} + 1, n - 1]$. In this case, at least one of the vertex of the shortest path from u to v 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 $\sqrt{n \log n}$, we can thus find the witness of the shortest path from u to v in $O(\sqrt{n \log n})$ time and then report the shortest path from u to v passing through it in additional $O(l_{uv})$ time. Since $l_{uv} > \sqrt{n \log n}$, the total time taken for processing a shortest path reporting query is $O(l_{uv})$ (and hence optimal).*

The amortized update time per edge deletion for the two data structures $\mathcal{S}_0^d$ and $\mathcal{L}_d^n$ will be $O(nd + \frac{n^3}{md} \log n)$, which is $O(n^{\frac{3}{2}} \sqrt{\log n})$ for $d = \sqrt{n \log n}$.

THEOREM 6.2. *An unweighted digraph $G(V, E)$ can be preprocessed to build a data structure that answers any online shortest path reporting query in optimal time w.h.p. while ensuring $O(\min(n^{\frac{3}{2}} \sqrt{\log n}, \frac{n^3 \log^2 n}{m}))$ 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 [8] required $\theta(m)$ space for maintaining an **in_tree** (or an **out_tree**), it was later improved to $O(n)$ by King and Thorup [9]. 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 gave improved decremental algorithms for maintaining transitive closure and All-pairs shortest paths with optimal query time. We pose the following problem :

Can a given graph $G(V, E)$ be preprocessed so that the set of vertices lying within distance d from a vertex can be maintained under deletion of edges while taking $o(md)$ total update time?

An answer to the above problem will lead to improved upper bound on the update time for the decremental problem of transitive closure (and all-pairs approximate shortest paths).

9. ACKNOWLEDGMENTS

We sincerely thank Camil Demetrescu for providing a very helpful clarification about the paper [4]. We also thank L. Sunil Chandran and L. Shankar Ram for their valuable suggestions and comments on an earlier draft of this paper.

10. REFERENCES

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

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

[3] C. Demetrescu. Private communication.

[4] 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.

[5] 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 381–389, 2001.

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

[7] 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.

[8] 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.

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

[10] H. Poutre and J. van 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.

[11] 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.