Improved Algorithms for Decremental Single-Source Reachability on Directed Graphs*

Monika Henzinger† Sebastian Krinninger† Danupon Nanongkai‡

Abstract

Recently we presented the first algorithm for maintaining the set of nodes reachable from a source node in a directed graph that is modified by edge deletions with $o(mn)$ total update time, where m is the number of edges and n is the number of nodes in the graph [Henzinger et al. STOC 2014]. The algorithm is a combination of several different algorithms, each for a different m vs. n trade-off. For the case of $m = \Theta(n^{1.5})$ the running time is $O(n^{2.47})$, just barely below $mn = \Theta(n^{2.5})$. In this paper we simplify the previous algorithm using new algorithmic ideas and achieve an improved running time of $\tilde{O}(\min(m^{7/6}n^{2/3}, m^{3/4}n^{5/4+o(1)}, m^{2/3}n^{4/3+o(1)}+m^{3/7}n^{12/7+o(1)}))$. This gives, e.g., $O(n^{2.36})$ for the notorious case $m = \Theta(n^{1.5})$. We obtain the same upper bounds for the problem of maintaining the strongly connected components of a directed graph undergoing edge deletions. Our algorithms are correct with high probabililty against an oblivious adversary.

*This paper was presented at the International Colloquium on Automata, Languages and Programming (ICALP) 2015. A full version combining the findings of this paper and its predecessor [HKN14] is available at http://arxiv.org/abs/1504.07959.

†University of Vienna, Faculty of Computer Science, Austria. Supported by the Austrian Science Fund (FWF): P23499-N23 and the University of Vienna (IK I049-N). The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

‡KTH Royal Institute of Technology, Sweden. Work partially done while at University of Vienna, Faculty of Computer Science, Austria.

1 Introduction

In this paper we study the decremental reachability problem. Given a directed graph G with n nodes and m edges and a source node s in G a *decremental single-source reachability algorithm* maintains the set of nodes reachable from s (i.e., all nodes v for which there is a path from s to v in the current version of G) during a sequence of edge deletions. The goal is to minimize the *total update time*, i.e., the total time needed to process *all* deletions such that reachability queries can be answered in constant time. A *decremental s-t reachability algorithm* is given a graph G undergoing edge deletions, a source node s, and a sink node t and it determines after every deletion in G whether s can still reach t.

Related Work. The incremental version of the single-source reachability problem, in which edges are *inserted* into the graph, can be solved with a total update time of $O(m)$ by performing an incremental graph search, where m is the final number of edges. Italiano [Ita88] showed that in directed acyclic graphs the decremental problem can be solved in time $O(m)$ as well. In general directed graphs however, the problem could for a long time only be solved in time $O(mn)$ using the more general decremental single-source shortest paths algorithm of Even and Shiloach [ES81, HK95, Kin99], which maintains a breadth-first search tree rooted at s, called *ES-tree*. This upper bound of $O(mn)$ is also achieved for the seemingly more complex decremental *all-pairs* reachability problem (also known as transitive closure) [RZ08, Lac13]. In the fully dynamic version of single-source reachability both insertions and deletions of edges are possible. The matrix-multiplication based transitive closure algorithms of Sankowski [San04] give fully dynamic algorithms for single-source reachability and s-t reachability with worst-case running times of $O(n^{1.575})$ and $O(n^{1.495})$ *per update*, respectively.

These upper bounds have recently been complemented by Abboud and Vassilevska Williams [AVW14] as follows. For the decremental s-t reachability problem, a combinatorial algorithm with a *worst-case* running time of $O(n^{2-\delta})$ (for some $\delta > 0$) per update or query implies a faster combinatorial algorithm for Boolean matrix multiplication and, as has been shown by Vassilevska Williams and Williams [VWW10], for other problems as well. (For non-combinatorial algorithms, Henzinger et al. [HKN+15] showed that there is no algorithm with worst-case $O(n^{1-\delta})$ update and $O(n^{2-\delta})$ query time, assuming the so-called Online Matrix-Vector Multiplication conjecture.) Furthermore, for the problem of maintaining the number of nodes reachable from a source under deletions (which our algorithms can do) a worst-case running time of $O(m^{1-\delta})$ (for some $\delta > 0$) per update or query falsifies the strong exponential time hypothesis. Thus, amortization is indeed necessary to bypass these bounds.

In [HKN14] we recently improved upon the long-standing upper bound of $O(mn)$ for decremental single-source reachability in directed graphs. In particular, we developed several algorithms whose combined expected running time is polynomially faster than $O(mn)$ for all values of m (i.e., for all possible densities of the initial graph). By a reduction from single-source reachability, our results in [HKN14] immediately give an $o(mn)$ algorithm for maintaining strongly connected components under edge deletions. Previously, the fastest decremental algorithms for this problem had a total update time of $O(mn)$ as well [RZ08, Lac13, Rod13].

Our Results. In this paper we improve upon the upper bounds provided in [HKN14]. Furthermore, the running times achieved in this paper are arguably more natural than those in [HKN14]. Although we previously broke the $O(mn)$ barrier for all values of m, we barely did so, giving a bound of $O(n^{2.47})$, when $m = \Theta(n^{1.5})$. In this paper we also get a better improvement, namely $O(n^{2.36})$ in this notorious case. In general, we can combine the algorithms of this paper to obtain a running time of $O(mn^{0.9+o(1)})$, whereas in [HKN14] we obtained $\tilde{O}(mn^{0.984})$.

In [HKN14] the starting point was to solve the decremental s-t reachability problem, which is also the case here. For this problem we obtain two algorithms with total update times of $\tilde{O}(\min(m^{5/4}n^{1/2}, m^{2/3}n^{4/3+o(1)}))$ and $O(m^{2/3}n^{4/3+o(1)} + m^{3/7}n^{12/7+o(1)})$, respectively. Just as in [HKN14], extensions of these algorithms solve the decremental single-source reachability problem with total update times of $\tilde{O}(\min(m^{7/6}n^{2/3}, m^{3/4}n^{5/4+o(1)}))$ and $O(m^{2/3}n^{4/3+o(1)} + m^{3/7}n^{12/7+o(1)})$, respectively. Furthermore, it follows from a reduction [RZ08, HKN14] that there are algorithms for the decremental strongly connected components problem whose running times are the same up to a logarithmic factor. We compare these new results to the ones of [HKN14] in Figure 1. All our algorithms are correct with high probability who fixes its sequence of updates and queries before the algorithm is initialized and their running time bounds hold in expectation. Due to space constraints this paper only contains an overview of the algorithm that has a total update time of $O(m^{2/3}n^{4/3+o(1)} + m^{3/7}n^{12/7+o(1)})$ and is thus the current fastest for dense graphs. The other algorithm and all omitted proofs can be found in the full version of this paper.

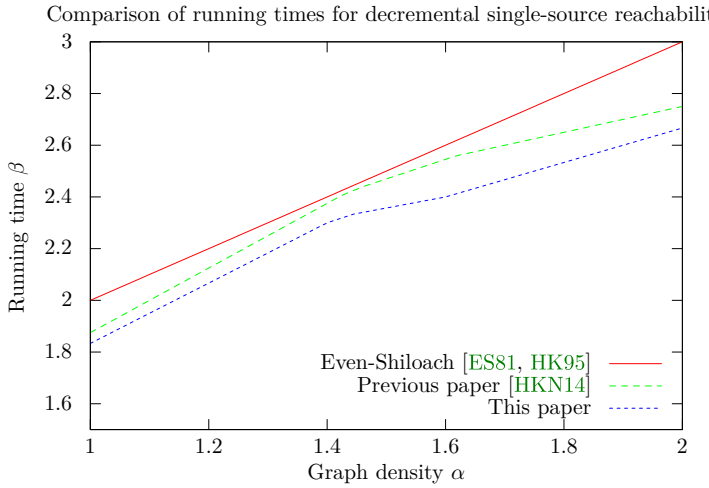

Figure 1: Running times of decremental single-source reachability algorithms dependent on the density of the initial graph. A point (α, β) in this diagram means that for a graph with $m = \Theta(n^\alpha)$ the algorithm has a running time of $O(n^{\beta+o(1)})$.

Techniques. There are two novel technical contributions: (1) The algorithm of [HKN14] uses two kinds of randomly selected nodes, called *hubs* and *centers*, each fulfilling a different purpose. Maintaining an ES-tree for each hub up to depth h, it quickly tests for every pair of centers (x, y) whether there is a path of length at most $2h$ from x to y going through a hub. If there is no such path, we build a special graph, called *path union graph*, for the pair (x, y) that contains all paths of length $O(h)$ from x to y. Since there no longer is a path from x to y through a hub of length at most h, we know that their path union graph is "smaller" than the original graph. In this paper we show how to extend this approach multiple layers of path unions graphs. Hubs and centers of the previous algorithms become level k, resp. $k-1$ centers in the new approach. Level $k-1$ centers serve as hubs for the level $k-2$ centers, and more generally level i centers serve as hubs for level $i-1$ centers. To do this efficiently we build the ES-tree for a level i center x *inside* the path-union graph of x and another, potentially higher-level center. The fact that we use the smaller path-union graph instead of the original graph for these ES-trees (together with an improved data structure for computing path-union graphs, see (2) below) gives the improvement in the running time.

(2) In [HKN14] we maintain for each center x an approximate path union data structure that computes a superset of the path union of x and any other center y. This superset is an approximation of the path union graph for (x, y) as it might contain paths between the two centers of length $O(h \log n)$ (and *not* as desired $O(h)$), but no longer. The total time spent in this data structure for x is (a) the size of the constructed path union graph and (b) a one-time "global charge" for using this data structure of $O(n^2)$. It is based on a hierarchical graph decomposition technique. Here we present a much simpler data structure that also constructs an approximate path union graph, but that does not require any hierarchical graph decomposition. This reduces the global charge per center from $O(n^2)$ to $O(m)$. We believe that this data structure is of independent interest.

Outline. In Section 2 we give the preliminaries. In Section 3 we present our new path union data structure. Finally, in Section 4 we show how to combine this idea with the multi-layer path union approach to obtain a faster decremental single-source reachability algorithm for dense graphs.

2 Preliminaries

In this section we review some notions and basic facts that we will use in the rest of this paper. We use the following notation: We consider a directed graph $G = (V, E)$ undergoing edge deletions, where V is the set of nodes of G and E is the set of edges of G. We denote by n the number of nodes of G and by m the number of edges of G *before the first edge deletion*. For every pair of nodes u and v we denote the distance from u to v in G by $d_G(u, v)$. For every subset of nodes $U \subseteq V$, we define $E(U) = E \cap U^2$ and denote by $G[U] = (U, E[U])$ the *subgraph of G induced by U*. For sets of nodes $U \subseteq V$ and $U' \subseteq V$ we define $E(U, U') = E \cap (U \times U')$, i.e., $E(U, U')$ is the set of edges $(u, v) \in E$ such that $u \in U$ and $v \in U'$. We write $\hat{O}(T(m, n))$ as an abbreviation for $O(T(m, n) \cdot n^{o(1)})$.

Like many decremental shortest paths and reachability algorithms, our algorithms internally use a data structure for maintaining a shortest paths tree up to a relatively small depth.

Theorem 2.1 (Even-Shiloach tree [ES81, HK95, Kin99])**.** *There is a decremental algorithm, called* Even-Shiloach tree *(short: ES-tree), that, given a directed graph G undergoing edge deletions, a source node s, and a parameter $h \geq 1$, maintains a shortest paths tree from s and the corresponding distances up to depth h with total update time $O(mh)$, i.e., the algorithm maintains $d_G(s, v)$ and the parent of v in the shortest paths tree for every node v such that $d_G(s, v) \leq h$. By reversing the edges of G it can also maintain the distance from v to s for every node v in the same time.*

The central concept in the algorithmic framework introduced in [HKN14] is the notion of the path union of a pair of nodes.

Definition 2.2. *For every directed graph G, every $h \geq 1$, and all pairs of nodes x and y of G, the path union $\mathcal{P}(x, y, h, G) \subseteq V$ is the set containing all nodes that lie on some path π from x to y in G of weight at most h.*

The path union has a simple characterization and can be computed efficiently.

Lemma 2.3 ([HKN14])**.** *For every directed graph G, every $h \geq 1$ and all pairs of nodes x and y of G, we have $\mathcal{P}(x, y, h, G) = \{v \in V \mid d_G(x, v) + d_G(v, y) \leq h\}$. We can compute this set in time $O(m)$.*

Our algorithms use randomization in the following way: by sampling a set of nodes with a sufficiently large probability we can guarantee that certain sets of nodes contain at least one of the sampled nodes with high probability. To the best of our knowledge, the first use of this technique in graph algorithms goes back to Ullman and Yannakakis [UY91].

Lemma 2.4. *Let T be a set of size t and let $S_1, S_2, \ldots, S_k$ be subsets of T of size at least q. Let U be a subset of T that was obtained by choosing each element of T independently with probability $p = (a \ln(kt))/q$, for some parameter a. Then, for every $1 \leq i \leq k$, the set S_i contains a node of U with high probability (whp), i.e., probability at least $1 - 1/t^a$, and the size of U is $O((t \log(kt))/q)$ in expectation.*

3 Approximate Path Union Data Structure

In this section we present a data structure for a graph G undergoing edge deletions, a fixed node x, and a parameter h. Given a node y, it computes an "approximation" of the path union $\mathcal{P}(x, y, h, G)$. Using a simple static algorithm the path union can be computed in time $O(m)$ for each pair (x, y). We give an (almost) output-sensitive data structure for this problem, i.e., using our data structure the time will be proportional to the size of the approximate path union which might be $o(m)$. Additionally, we have to pay a global cost of $O(m)$ that is amortized over *all* approximate path union computations for the node x and *all* nodes y. This will be useful because in our reachability algorithm we can use probabilistic arguments to bound the size of the approximate path unions.

Proposition 3.1. *There is a data structure that, given a graph G undergoing edge deletions, a fixed node x, and a parameter h, provides a procedure* ApproximatePathUnion *such that, given sequence of nodes $y_1, \ldots, y_k$, this procedure computes sets $F_1, \ldots F_k$ guaranteeing $\mathcal{P}(x, y, h, G) \subseteq F_i \subseteq \mathcal{P}(x, y, (\log m + 3)h, G)$ for all $1 \leq i \leq k$. The total running time is $O(\sum_{1 \leq i \leq k} |F_i| + m)$.*

3.1 Algorithm Description

Internally, the data structure maintains a set $R(x)$ of nodes, initialized with $R(x) = V$, such that the following invariant is fulfilled at any time: all nodes that can be reached from x by a path of length at most h are contained in $R(x)$ (but $R(x)$ might contain other nodes as well). Observe that thus $R(x)$ contains the path union $\mathcal{P}(x, y, h, G)$ for every node y.

To gain some intuition for our approach consider the following way of computing an approximation of the path union $\mathcal{P}(x, y, h, G)$ for some node y. First, compute $B_1 = \{v \in R(x) \mid d_{G[R(x)]}(v, y) \leq h\}$ using a backward breadth-first search (BFS) to y in $G[R(x)]$, the subgraph of G induced by $R(x)$. Second, compute $F = \{v \in R(x) \mid d_{G[B_1]}(x, v) \leq h\}$ using a forward BFS from x in $G[B_1]$. It can be shown that $\mathcal{P}(x, y, h, G) \subseteq F \subseteq \mathcal{P}(x, y, 2h, G)$.[1] Given B_1, we could charge the time for computing F to the set F itself, but we do not know how to pay for computing B_1 as $B_1 \setminus F$ might be much larger than F.

Our idea is to additionally identify a set of nodes $X \subseteq \{v \in V \mid d_G(x, v) > h\}$ and remove it from $R(x)$. Consider a second approach where we first compute B_1 as above and then compute $B_2 = \{v \in R(x) \mid d_{G[R(x)]}(v, y) \leq 2h\}$ and $F = \{v \in R(x) \mid d_{G[B_2]}(x, v) \leq h\}$. It can be shown that $\mathcal{P}(x, y, h, G) \subseteq F \subseteq \mathcal{P}(x, y, 3h, G)$. Additionally, all nodes in $X = B_1 \setminus F$ are at distance more than h from x and therefore we can remove X from $R(x)$. Thus, we can charge the work for computing B_1 and F to X and F, respectively.[2] However, we now have a similar problem as before as we do not know whom to charge for computing B_2.

We resolve this issue by simply computing $B_i = \{v \in R(x) \mid d_{G[R(x)]}(v, y) \leq ih\}$ for increasing values of i until we arrive at some i^* such that the size of B_{i^*} is at most double the size of B_{i^*-1}. We then return $F = \{v \in R(x) \mid d_{G[B_i]}(x, v) \leq h\}$ and charge the time for computing B_i to $X = B_{i-1} \setminus F$ and F, respectively. As the size of B_i can double at most $O(\log n)$ times we have $\mathcal{P}(x, y, h, G) \subseteq F \subseteq \mathcal{P}(x, y, O(h \log n), G)$, as we show below. Procedure 1 shows the pseudocode of this algorithm. Note that in the special case that x cannot reach y the algorithm returns the empty set. In the analysis below, let i^* denotes the final value of i before Procedure 1 terminates.

3.2 Correctness

We first prove Invariant (I): the set $R(x)$ always contains all nodes that are at distance at most h from x in G. This is true initially as we initialize $R(x)$ to be V and we now show that it

[1] Indeed, F might contain some node v with $d_G(x, v) = h$ and $d_G(v, y) = h$, but it will not contain any node w with either $d_G(x, w) > h$ or $d_G(w, y) > h$.

[2] Note that in our first approach removing $B_1 \setminus F$ would not have been correct as F was computed w.r.t to $G[B_1]$ and not w.r.t. $G[B_2]$.

Procedure 1: ApproximatePathUnion(y)

```
// All calls of ApproximatePathUnion(y) use fixed x and h.
1 Compute B_1 = {v ∈ R(x) | d_{G[R(x)]}(v, y) ≤ h} // backward BFS to y in subgraph
    induced by R(x)
2 for i = 2 to ⌈log m⌉ + 1 do
3     Compute B_i = {v ∈ R(x) | d_{G[R(x)]}(v, y) ≤ ih} // backward BFS to y in
        subgraph induced by R(x)
4     if |E(B_i)| ≤ 2|E(B_{i−1})| then
5         Compute F = {v ∈ B_i | d_{G[B_i]}(x, v) ≤ h} // forward BFS from x in
            subgraph induced by B_i
6         X ← B_{i−1} \ F, R(x) ← R(x) \ X
7         return F
```

continues to hold because we only remove nodes at distance more than h from x.

Lemma 3.2. *If $R(x) \subseteq \{v \in V \mid d_G(x, v) \leq h\}$, then for every node $v \in X$ removed from $R(x)$, we have $d_G(x, v) > h$.*

Proof. Let $v \in X = B_{i^*-1} \setminus F$ and assume by contradiction that $d_G(x, v) \leq h$. Since $v \in B_{i^*-1}$ we have $d_{G[R(x)]}(v, y) \leq (i^* - 1)h$. Now consider the shortest path π from x to v in G, which has length at most h. By the assumption, every node on π is contained in $G[R(x)]$. Therefore, for every node v' on π, we have $d_{G[R(x)]}(v', v) \leq h$ and thus

$$d_{G[R(x)]}(v', y) \leq d_{G[R(x)]}(v', v) + d_{G[R(x)]}(v, y) \leq h + (i^* - 1)h \leq i^* h$$

which implies that $v' \in B_{i^*}$. Thus, every node on π is contained in B_{i^*}. As π is a path from x to v of length at most h it follows that $d_{G[B_{i^*}]}(x, v) \leq h$. Therefore $v \in F$, which contradicts the assumption $v \in X$. □

We now complete the correctness proof by showing that the set of nodes returned by the algorithm approximates the path union.

Lemma 3.3. *Procedure 1 returns a set of nodes F such that $\mathcal{P}(x, y, h, G) \subseteq F \subseteq \mathcal{P}(x, y, (\log m + 3)h, G)$.*

Proof. We first argue that the algorithm actually returns some set of nodes F. Note that in Line 4 of the algorithm we always have $|E(B_i)| \geq |E(B_{i-1})|$ as $B_{i-1} \subseteq B_i$. As $E(B_i)$ is a set of edges and the total number of edges is at most m, the condition $|E(B_i)| \leq |E(B_{i-1})|$ therefore must eventually be fulfilled for some $2 \leq i \leq \lceil \log m \rceil + 1$.

We now show that $\mathcal{P}(x, y, h, G) \subseteq F$. Let $v \in \mathcal{P}(x, y, h, G)$, which implies that v lies on a path π from x to y of length at most h. For every node v' on π we have $d_G(x, v') \leq h$, which by Invariant (I) implies $v' \in R(x)$. Thus, the whole path π is contained in $G[R(x)]$. Therefore

$d_{G[R(x)]}(v', y) \leq h$ for every node v' on π which implies that π is contained in $G[B_{i^*}]$. Then clearly we also have $d_{G[B_{i^*}]}(x, v) \leq h$ which implies $v \in F$.

Finally we show that $F \subseteq \mathcal{P}(x, y, (\log m + 3)h, G)$ by proving that $d_G(x, v) + d_G(v, y) \leq (\log m + 3)h$ for every node $v \in F$. As $G[B_{i^*}]$ is a subgraph of G, we have $d_G(x, v) \leq d_{G[B_{i^*}]}(x, v)$ and $d_G(v, y) \leq d_{G[B_{i^*}]}(v, y)$. By the definition of F we have $d_{G[B_{i^*}]}(x, v) \leq h$. As $F \subseteq B_{i^*}$ we also have $d_{G[B_{i^*}]}(v, y) \leq i^* h \leq (\lceil \log m \rceil + 1)h \leq (\log m + 2)h$. It follows that $d_G(x, v) + d_G(v, y) \leq h + (\log m + 2)h = (\log m + 3)h$. □

3.3 Running Time Analysis

To bound the total running time we prove that each call of Procedure 1 takes time proportional to the number of edges in the returned approximation of the path union plus the number of edges incident to the nodes removed from $R(x)$. As each node is removed from $R(x)$ at most once, the time spent on *all* calls of Procedure 1 is then $O(m)$ plus the sizes of the subgraphs induced by the approximate path unions returned in each call.

Lemma 3.4. *The running time of Procedure 1 is $O(|E(F)| + |E(X, R(x))| + |E(R(x), X)|)$ where F is the set of nodes returned by the algorithm, and X is the set of nodes the algorithm removes from $R(x)$.*

Proof. The running time in iteration $2 \leq j \leq i^* - 1$ is $O(|E(B_j)|)$ as this is the cost of the breadth-first-search performed to compute B_j. In the last iteration i^*, the algorithm additionally has to compute F and X and remove X from $R(x)$. As F is computed by a BFS in $G[B_{i^*}]$ and $X \subseteq B_{i^*-1} \subseteq B_{i^*}$, these steps take time $O(|E(B_{i^*})|)$. Thus the total running time is $O(\sum_{1 \leq j \leq i^*} |E(B_j)|)$.

By checking the size bound in Line 4 of Procedure 1 we have $|E(B_j)| > 2|E(B_{j-1})|$ for all $1 \leq j \leq i^* - 1$ and $|E(B_{i^*})| \leq 2|E(B_{i^*-1})|$. By repeatedly applying the first inequality it follows that $\sum_{1 \leq j \leq i^*-1} |E(B_j)| \leq 2|E(B_{i^*-1})|$. Therefore we get

$$\begin{aligned}\sum_{1 \leq j \leq i^*} |E(B_j)| = \sum_{1 \leq j \leq i^*-1} |E(B_j)| + |E(B_{i^*})| \\ \leq 2|E(B_{i^*-1})| + 2|E(B_{i^*-1})| = 4|E(B_{i^*-1})|\end{aligned}$$

and thus the running time is $O(|E(B_{i^*-1})|)$. Now observe that by $X = B_{i^*-1} \setminus F$ we have $B_{i^*-1} \subseteq X \cup F$ and thus

$$\begin{aligned}E(B_{i^*-1}) &\subseteq E(F) \cup E(X) \cup E(X, F) \cup E(F, X) \\ &\subseteq E(F) \cup E(X, R(x)) \cup E(R(x), X)\,.\end{aligned}$$

Therefore the running time is $O(|E(F)| + |E(X, R(x))| + |E(R(x), X)|)$. □

4 Reachability via Center Graph

We now show how to combine the approximate path union data structure with a hierarchical approach to get an improved decremental reachability algorithm for dense graphs. The algorithm

has a parameter $1 \leq k \leq \log n$ and for each $1 \leq i \leq k$ a parameter $c_i \leq n$. We determine suitable choices of these parameters in Section 4.2. For each $1 \leq i \leq k-1$, our choice will satisfy $c_i \geq c_{i+1}$ and $c_i = \hat{O}(c_{i+1})$. Furthermore, we set $h_i = (3 + \log m)^{i-1} n/c_1$ for $1 \leq i \leq k$. At the initialization, the algorithm determines sets of nodes $C_1 \supseteq C_2 \supseteq \cdots \supseteq C_k$ such that $s, t \in C_1$ as follows. For each $1 \leq i \leq k$, we sample each node of the graph with probability $ac_i \ln n/n$ (for a large enough constant a), where the value of c_i will be determined later. The set C_i then consists of the sampled nodes, and if $i \leq k-1$, it additionally contains the nodes in C_{i+1}. For every $1 \leq i \leq k$ we call the nodes in C_i i-centers. In the following we describe an algorithm for maintaining pairwise reachability between all 1-centers.

4.1 Algorithm Description

Data Structures. The algorithm uses the following data structures:

- For every i-center x and every $i \leq j \leq k$ an approximate path union data structure (see Proposition 3.1) with parameter h_j.
- For every k-center x an incoming and an outgoing ES-tree of depth h_k in G.
- For every pair of an i-center x and a j-center y such that $l := \max(i, j) \leq k-1$, a set of nodes $Q(x, y, l) \subseteq V$. Initially, $Q(x, y, l)$ is empty and at some point the algorithm might compute $Q(x, y, l)$ using the approximate path union data structure of x.
- For every pair of an i-center x and a j-center y such that $l := \max(i, j) \leq k-1$ an ES-tree of depth h_l from x in $Q(x, y, l)$.
- For every pair of an i-center x and a j-center y such that $l := \max(i, j) \leq k-1$ a set of $(l+1)$-centers certifying that x can reach y.

Certified Reachability Between Centers (Links). The algorithm maintains the following limited path information between centers, called *links*, in a top-down fashion. Let x be a k-center and let y be an i-center for some $1 \leq i \leq k-1$. The algorithm links x to y if and only if y is contained in the outgoing ES-tree of depth h_k of x. Similarly the algorithm links y to x if and only if y is contained in the incoming ES-tree of depth h_k of x. Let x be an i-center and let y be a j-center such that $l := \max(i, j) \leq k-1$. If there is an $(l+1)$-center z such that x is linked to z and z is linked to y, the algorithm links x to y (we also say that z links x to y). Otherwise, the algorithm computes $Q(x, y, l)$ using the approximate path union data structure of x and starts to maintain an ES-tree from x up to depth h_l in $G[Q(x, y, l)]$. It links x to y if and only if y is contained in the ES-tree of x. Using a list of centers z certifying that x can reach y, maintaining the links between centers is straightforward.

Center Graph. The algorithm maintains a graph called *center graph*. Its nodes are the 1-centers and it contains the edge (x, y) if and only if x is linked to y. The algorithm maintains the transitive closure of the center graph. A query asking whether a center y is reachable from

a center x in G is answered by checking the reachability in the center graph. As s and t are 1-centers this answers s-t reachability queries.

Correctness. For the algorithm to be correct we have to show that there is a path from s to t in the center graph if and only if there is a path from s to t in G. We can in fact show more generally that this is the case for any pair of 1-centers.

Lemma 4.1. *For every pair of* 1*-centers x and y, there is a path from x to y in the center graph if and only if there is a path from x to y in G.*

4.2 Running Time Analysis

The key to the efficiency of the algorithm is to bound the size of the graphs $Q(x, y, l)$.

Lemma 4.2. *Let x be an i-center and let y be a j-center such that $l := \max(i, j) \leq k-1$. If x is not linked to y by an $(l+1)$-center, then $Q(x, y, l)$ contains at most n/c_{l+1} nodes with high probability.*

With the help of this lemma we first analyze the running time of each part of the algorithm and argue that our choice of parameters gives the desired total update time.

Parameter Choice. We carry out the running time analysis with regard to two parameters $1 \leq b \leq c \leq n$ which we will set at the end of the analysis. We set $k = \lceil (\log(c/b))/(\sqrt{\log n \cdot \log\log n}) \rceil + 1$, $c_k = b$ and $c_i = 2^{\sqrt{\log n \cdot \log\log n}} c_{i+1} = \hat{O}(c_{i+1})$ for $1 \leq i \leq k-1$. Note that the number of i-centers is $\tilde{O}(c_i)$ in expectation. Observe that

$$\begin{aligned}(3+\log m)^{k-1} = O((\log n)^k) \leq O((\log n)^{\sqrt{\log n/\log\log n}}) \\ = O(2^{\sqrt{\log n \cdot \log\log n}}) = O(n^{\sqrt{\log\log n/\log n}}) = O(n^{o(1)})\,.\end{aligned}$$

Furthermore we have

$$c_1 = \left(2^{\sqrt{\log n \cdot \log\log n}}\right)^{k-1} c_k \geq 2^{\log(c/b)} b = \frac{c}{b} \cdot b = c$$

and by setting $k' = (\log(c/b))/(\sqrt{\log n \cdot \log\log n})$ we have $k \leq k' + 2$ and thus

$$c_1 = \left(2^{\sqrt{\log n \cdot \log\log n}}\right)^{k-1} c_k \leq \left(2^{\sqrt{\log n \cdot \log\log n}}\right)^{k'+1} c_k = 2^{\sqrt{\log n \cdot \log\log n}} c = \hat{O}(c)\,.$$

Remember that $h_i = (3+\log m)^{i-1} n/c_1$ for $1 \leq i \leq k$. Therefore we have $h_i = \hat{O}(n/c_1) = \hat{O}(n/c)$.

Maintaining ES-Trees. For every k-center we maintain an incoming and an outgoing ES-tree of depth h_k, which takes time $O(mh_k)$. As there are $\tilde{O}(c_k)$ k-centers, maintaining all these trees takes time $\tilde{O}(c_k m h_k) = \hat{O}(bmn/c)$.

For every i-center x and every j-center y such that $l := \max(i,j) \leq k-1$, we maintain an ES-tree up to depth h_l in $G[Q(x,y,l)]$. By Lemma 4.2 $Q(x,y,l)$ has at most n/c_{l+1} nodes and thus $G[Q(x,y,l)]$ has at most n^2/c_{l+1}^2 edges. Maintaining this ES-tree therefore takes time $O((n^2/c_{l+1}^2)\cdot h_l) = \hat{O}(n^2/c_{l+1}^2(n/c_1)) = \hat{O}(n^3/(c_1 c_{l+1}^2))$. In total, maintaining all these trees takes time

$$\hat{O}\left(\sum_{1\leq i\leq k-1}\sum_{1\leq j\leq i} c_i c_j \frac{n^3}{c_1 c_{i+1}^2}\right) = \hat{O}\left(\sum_{1\leq i\leq k-1}\sum_{1\leq j\leq i} \frac{c_i c_1 n^3}{c_{i+1} c_1 c_k}\right)$$
$$= \hat{O}\left(\sum_{1\leq i\leq k-1}\sum_{1\leq j\leq i} \frac{n^3}{c_k}\right) = \hat{O}\left(k^2 \frac{n^3}{c_k}\right) = \hat{O}\left(\frac{n^3}{b}\right).$$

Computing Approximate Path Unions. For every i-center x and every $i \leq j \leq k$ we maintain an approximate path union data structure with parameter h_j. By Proposition 3.1 this data structures has a total running time of $O(m)$ and an additional cost of $O(|E(Q(x,y,j))|)$ each time the approximate path union $Q(x,y,j)$ is computed for some j-center y. By Lemma 4.2 the number of nodes of $Q(x,y,j)$ is n/c_{j+1} with high probability and thus its number of edges is n^2/c_{j+1}^2. Therefore, computing all approximate path unions takes time

$$\tilde{O}\left(\sum_{1\leq i\leq k-1}\sum_{i\leq j\leq k}\left(c_i m + c_i c_j \frac{n^2}{c_{j+1}^2}\right)\right) = \tilde{O}\left(\sum_{1\leq i\leq k-1}\sum_{i\leq j\leq k}\left(c_1 m + \frac{c_1 c_j n^2}{c_{j+1} c_k}\right)\right)$$
$$= \hat{O}\left(\sum_{1\leq i\leq k-1}\sum_{i\leq j\leq k}\left(c_1 m + \frac{c_1 n^2}{c_k}\right)\right) = \hat{O}(k^2 c_1 m + k^2 c_1 n^2/c_k) = \hat{O}(cm + cn^2/b).$$

Maintaining Links Between Centers. For each pair of an i-center x and a j-center y there are at most $\tilde{O}(c_{l+1})$ $(l+1)$-centers that can possibly link x to y. Each such $(l+1)$-center is added to and removed from the list of $(l+1)$-centers linking x to y at most once. Thus, the total time needed for maintaining all these links is $\tilde{O}(\sum_{1\leq i\leq k-1}\sum_{1\leq j\leq i} c_i c_j c_{i+1}) = \tilde{O}(k^2 c_1^3) = \tilde{O}(c^3)$.

Maintaining Transitive Closure in Center Graph. The center graph has $\tilde{O}(c_1)$ nodes and thus $\tilde{O}(c_1^2)$ edges. During the algorithm edges are only deleted from the center graph and never inserted. Thus we can use known $O(mn)$-time decremental algorithms for maintaining the transitive closure [RZ08, Lac13] in the center graph in time $\tilde{O}(c_1^3) = \tilde{O}(c^3)$.

Total Running Time. Since the term cn^2/b is dominated by the term n^3/b, we obtain a total running time of $\hat{O}\left(bmn/c + n^3/b + cm + c^3\right)$. By setting $b = n^{5/3}/m^{2/3}$ and $c = n^{4/3}/m^{1/3}$ the running time is $\hat{O}(m^{2/3}n^{4/3} + n^4/m)$ and by setting $b = n^{9/7}/m^{3/7}$ and $c = m^{1/7}n^{4/7}$ the running time is $\hat{O}(m^{3/7}n^{12/7} + m^{8/7}n^{4/7})$.

4.3 Decremental Single-Source Reachability

The algorithm above works for a set of randomly chosen centers. Note that the algorithm stays correct if we add any number of nodes to C_1, thus increasing the number of 1-centers for which the algorithm maintains pairwise reachability. If the number of additional centers does not exceed the expected number of randomly chosen centers, then the same running time bounds still apply. Using the reductions of [HKN14] this immediately implies decremental algorithms for maintaining single-source reachability and strongly connected components.

Theorem 4.3. *There are decremental algorithms for maintaining single-source reachability and strongly connected components with constant query time and expected total update time* $\hat{O}(m^{2/3}n^{4/3} + m^{3/7}n^{12/7})$ *that are correct with high probability against an oblivious adversary.*

References

[AVW14] Amir Abboud and Virginia Vassilevska Williams. "Popular conjectures imply strong lower bounds for dynamic problems". In: *Symposium on Foundations of Computer Science (FOCS)*. 2014, pp. 434–443 (cit. on p. 2).

[ES81] Shimon Even and Yossi Shiloach. "An On-Line Edge-Deletion Problem". In: *Journal of the ACM* 28.1 (1981), pp. 1–4 (cit. on pp. 2, 3, 5).

[HK95] Monika Henzinger and Valerie King. "Fully Dynamic Biconnectivity and Transitive Closure". In: *Symposium on Foundations of Computer Science (FOCS)*. 1995, pp. 664–672 (cit. on pp. 2, 3, 5).

[HKN14] Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. "Sublinear-Time Decremental Algorithms for Single-Source Reachability and Shortest Paths on Directed Graphs". In: *Symposium on Theory of Computing (STOC)*. 2014, pp. 674–683 (cit. on pp. 1–5, 12).

[HKN+15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. "Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture". In: *Symposium on Theory of Computing (STOC)*. 2015, pp. 21–30 (cit. on p. 2).

[Ita88] Giuseppe F. Italiano. "Finding Paths and Deleting Edges in Directed Acyclic Graphs". In: *Information Processing Letters* 28.1 (1988), pp. 5–11 (cit. on p. 2).

[Kin99] Valerie King. "Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs". In: *Symposium on Foundations of Computer Science (FOCS)*. 1999, pp. 81–91 (cit. on pp. 2, 5).

[Lac13] Jakub Łącki. "Improved Deterministic Algorithms for Decremental Reachability and Strongly Connected Components". In: *ACM Transactions on Algorithms* 9.3 (2013). Announced at SODA'11, p. 27 (cit. on pp. 2, 11).

[Rod13] Liam Roditty. "Decremental maintenance of strongly connected components". In: *Symposium on Discrete Algorithms (SODA)*. 2013, pp. 1143–1150 (cit. on p. 2).

[RZ08] Liam Roditty and Uri Zwick. "Improved Dynamic Reachability Algorithms for Directed Graphs". In: *SIAM Journal on Computing* 37.5 (2008). Announced at FOCS'02, pp. 1455–1471 (cit. on pp. 2, 3, 11).

[San04] Piotr Sankowski. "Dynamic Transitive Closure via Dynamic Matrix Inverse". In: *Symposium on Foundations of Computer Science (FOCS)*. 2004, pp. 509–517 (cit. on p. 2).

[UY91] Jeffrey D. Ullman and Mihalis Yannakakis. "High-Probability Parallel Transitive-Closure Algorithms". In: *SIAM Journal on Computing* 20.1 (1991). Announced at SPAA'90, pp. 100–125 (cit. on p. 5).

[VWW10] Virginia Vassilevska Williams and Ryan Williams. "Subcubic Equivalences between Path, Matrix and Triangle Problems". In: *Symposium on Foundations of Computer Science (FOCS)*. 2010, pp. 645–654 (cit. on p. 2).