Subquadratic Dynamic Path Reporting in Directed Graphs against an Adaptive Adversary

Adam Karczmarz
a.karczmarz@mimuw.edu.pl
Institute of Informatics, University of Warsaw and IDEAS NCBR
Warsaw, Poland

Anish Mukherjee
anish@mimuw.edu.pl
Institute of Informatics, University of Warsaw and IDEAS NCBR
Warsaw, Poland

Piotr Sankowski
sank@mimuw.edu.pl
Institute of Informatics, University of Warsaw and IDEAS NCBR
Warsaw, Poland

ABSTRACT
We study reachability and shortest paths problems in dynamic directed graphs. Whereas algebraic dynamic data structures supporting edge updates and reachability/distance queries have been known for quite a long time, they do not, in general, allow reporting the underlying paths within the same time bounds, especially against an adaptive adversary.

In this paper we develop the first known fully dynamic reachability data structures working against an adaptive adversary and supporting edge updates and path queries for two natural variants: (1) point-to-point path reporting, and (2) single-source reachability tree reporting. For point-to-point queries in DAGs, we achieve $O(n^{1.529})$ worst-case update and query bounds, whereas for tree reporting in DAGs, the respective worst-case bounds are $O(n^{1.765})$.

More importantly, we show how to lift these algorithms to work on general graphs at the cost of increasing the bounds to $n^{1+5/6+o(1)}$ and making the update times amortized. On the way to accomplishing these goals, we obtain two interesting subresults. We give subquadratic fully dynamic algorithms for topological order (in a DAG), and strongly connected components. To the best of our knowledge, such algorithms have not been described before.

Additionally, we provide deterministic incremental data structures for (point-to-point or single-source) reachability and shortest paths that can handle edge insertions and report the respective paths within subquadratic worst-case time bounds. For reachability and $(1+\varepsilon)$-approximate shortest paths in weighted directed graphs, these bounds match the best known dynamic matrix inversion-based randomized algorithms for fully dynamic reachability [v.d.Brand, Nanongkai and Saranurak, FOCS’19].

1 INTRODUCTION
Dynamic reachability (or transitive closure) and dynamic all-pairs shortest paths are among the most fundamental and well-studied dynamic problems on directed graphs. In these problems, we are given a dynamic directed graph $G = (V,E)$ with $n = |V|$, and the goal is to devise a data structure maintaining $G$ and supporting edge set updates interleaved with reachability (or shortest path) queries between arbitrary source-target pairs of vertices of $G$. A dynamic data structure is called incremental if it can handle edge insertions only, decremental if it can handle edge deletions only, and fully dynamic if it can handle both. One is typically interested in optimizing both the (amortized or worst-case) update time of the data structure, and the query time (which is usually worst-case anyway). In the partially dynamic settings one usually optimizes the total update time, i.e., the time needed to process the entire sequence of updates.

Efficient and non-trivial combinatorial algorithms have been developed for dynamic transitive closure in the incremental [24], decremental [21], and fully dynamic settings [17, 31, 32]. For all-pairs shortest paths, Demetrescu and Italiano [15] developed a fully dynamic algorithm (later slightly improved by [37]) with $O(n^2)$ amortized update time and optimal query time. That algorithm works in the most general comparison-addition model which allows real edge weights. Partially dynamic algorithms for all-pairs shortest paths with non-trivial total update time bounds and optimal query time are known in the incremental [3, 7, 26, 27] and decremental [7, 20, 27] settings; however, these algorithms work only for unweighted graphs (or for small-integer weights) and/or produce $(1+\varepsilon)$-approximate answers.

One can observe the following phenomenon: the known combinatorial dynamic algorithms for reachability and shortest paths typically have good amortized, but much worse worst-case update bounds. Moreover, in the fully dynamic settings, none of the state-of-the-art combinatorial algorithms [15, 17, 31, 32, 37] achieves $O(n^{2-\varepsilon})$ update time and query time at the same time.
for dense graphs (for any $\delta > 0$). King and Sagert [28] were the first to observe that path counting modulo a sufficiently large prime can be used to obtain a fully dynamic transitive closure data structure with $O(n^2)$ worst-case update time in the case of acyclic graphs. Demetrescu and Italiano [16] combined this technique with fast rectangular matrix multiplication and obtained a fully dynamic reachability algorithm for DAGs supporting single-edge updates in $O(n^{1+\rho})$ worst-case time and queries in $O(n^\rho)$ time, where $\rho \approx 0.529$ equals the smallest real number such that $\omega(1, p, 1) = 1 + 2p$ [22]. Here, $\omega(1, a, 1)$ denotes an exponent such that a product of $n \times n^a$ and $n^a \times n$ matrices can be computed in $O(n^{\omega(1, a, 1)})$ time. Finally, Sankowski [33] obtained the same bounds for fully dynamic reachability in general graphs by reducing the problem to dynamically maintaining a matrix inverse. This technique also led to subquadratic fully dynamic algorithms for exact distances in unweighted graphs [34], approximate distances in weighted graphs [39], and maximum matchings [35]. The state-of-the-art bounds for dynamic matrix inverse (and thus also for some of these graph problems) were given by [40]; we refer to that work for more graph applications of dynamic matrix inverse.

The common drawback of algebraic dynamic transitive closure and distances algorithms based on either dynamic path counting or dynamic matrix inverse is that they are Monte Carlo randomized (as they involve the Zippel-Schwartz lemma). More importantly, they do not, in general, allow constructing a path that certifies reachability or achieves the reported distance. To the best of our knowledge, the only exception to the latter drawback is the recent trade-off algorithm of [6] which, if the maximum of query/update time bounds is optimized, supports single-edge updates and shortest path reporting queries in $O(n^{1.897})$ worst-case time. However, the high probability correctness of the dynamic shortest paths data structure of [6] is only guaranteed against an oblivious adversary that does not base its future updates on the answers that the data structure produces. In the recent years, there has been a significant effort of the dynamic graph algorithms community to obtain solutions that work well against an adaptive adversary, possibly by completely avoiding randomization (see, e.g., [9, 13, 20, 30]). Such algorithms are not only more general, but can also be used in a black-box way as building blocks for static algorithms.

To the best of our knowledge, none of the currently known dynamic algorithms for reachability or shortest paths that relies on algebraic techniques and has subquadratic update/query bounds is able to report the certifying paths and still perform well under the adaptive adversary assumption.

1.1 Our Results

1.1.1 Fully Dynamic Path Reporting. As our first contribution, we show the first known fully dynamic reachability algorithms that support edge updates and certificate reporting queries in subquadratic time and at the same time work against an adaptive adversary. We consider two natural certificate reporting variants:

1. point-to-point path reporting for any requested source-target pair $s, t \in V$, and
2. single-source reachability tree reporting for any requested source $s \in V$.

Note that the former variant constitutes a reporting analogue to reachability (transitive closure) queries, whereas the latter is an analogue to single-source reachability queries. Recall that $\rho \approx 0.529$ is such that $\omega(1, p, 1) = 1 + 2p$. Moreover, if $\omega = \omega(1, 1, 1) = 2$, then $\rho = 0.5$. The following theorem summarizes the bounds that we obtain for directed acyclic graphs.

**Theorem 1.1.** Let $G$ be a directed graph subject to fully dynamic single-edge updates that keep $G$ acyclic at all times. There exist data structures with the following worst-case update and query bounds:

1. $\widetilde{O}(n^{1+\rho}) = O(n^{1.529})$ for point-to-point path reporting,
2. $\widetilde{O}(n^{3+\rho})/2 = O(n^{1.765})$ for single-source reachability tree reporting.

The data structures are Monte Carlo randomized and produce answers correct with high probability against an adaptive adversary.

In order to obtain the former of the above data structures, we observe that a topological order of a fully dynamic acyclic graph can be maintained in $O(n^{1+\rho}) = O(n^{1.529})$ worst-case time per update as well. Whereas this observation is a simple consequence of the known dynamic reachability algorithms with subquadratic update time and sublinear query time [16, 33], to the best of our knowledge, it has not been described before. Dynamic topological ordering has been mostly studied in the incremental setting, and multiple algorithms with non-trivial total update time bounds are known [4, 8, 12].

Even more importantly, we show the following reductions of the respective variants on general graphs to suitable reporting variants of the decremental strongly connected components problem. In this paper, we use the term strongly connected components to refer to the set $S$ of equivalence classes of the strong connectivity relation, and not the subgraphs $G[S], S \in S$.

**Theorem 1.2.** Let $C$ be a decremental data structure with total update time $\delta T(n, m) = \Omega(m + n)$ (1) maintaining strongly connected components explicitly and (2) supporting queries reporting a single path $P$ between arbitrary strongly connected vertices $u, v$ of the maintained graph in $O(|P| \cdot n^\rho)$ time. Then, there exists a fully dynamic data structure supporting single-edge updates and point-to-point path reporting queries with amortized update time and worst-case query time of $\widetilde{O}\left(\sqrt{T(n, m) \cdot n + n^{1+\rho}}\right)$.

If the data structure $C$ works against an adaptive adversary, so does the fully dynamic path reporting data structure.

**Theorem 1.3.** Let $C$ be a decremental data structure with total update time $T(n, m) = \Omega(m + n)$ (1) maintaining strongly connected components $S$ of $G$ explicitly and (2) supporting queries reporting, for a chosen strongly connected component $S \in S$, a (possibly...
sparser) strongly connected subgraph \( Z \subseteq G[S] \) with \( V(Z) = S \) in \( O(|S| \cdot n^{14p/5}) \) time. Then, there exists a fully dynamic data structure supporting single-edge updates and single-source reachability tree reporting queries with amortized update time and worst-case query time of \( O(\sqrt{T(n, m) + n + n^{3p/5}}) \).

If the data structure \( C \) works against an adaptive adversary, so does the fully dynamic reachability tree reporting data structure.

In particular, a deterministic decremental strongly connected components data structure with total update time \( T(n, m) = mn^{2/3+o(1)} \) that satisfies the requirements of both reductions has been recently shown by \([9]\), i.e., the query time of their data structure is \( |P| \cdot n^{o(1)} = O(|P| \cdot n^p) \) for path reporting and \( |S| \cdot n^{o(1)} \) for strongly connected subgraph reporting, respectively. Hence, we obtain \( n^{14/5+o(1)} \) amortized update bound and worst-case query time for both variants against an adaptive adversary. Thanks to our reductions, further progress on deterministic (or adaptive\(^7\)) decremental strongly connected components problem will lead to improved bounds for subquadratic fully dynamic path- and tree-reporting data structures.

We also note that if the oblivious adversary assumption is acceptable, by plugging in the near-optimal decremental strongly connected components data structure of \([11]\) with \( T(n, m) = O(n + m) \) as \( C \), we obtain the same asymptotic (but still amortized) bounds for the respective reporting variants as in Theorem 1.1 for DAGs.

As a warm-up to proving Theorems 1.2 and 1.3, we also show that the strongly connected components of a fully dynamic graph can be explicitly maintained under single-edge updates in \( O(n^{14p} \cdot \text{time per update}) \). To the best of our knowledge, no non-trivial bounds have ever been described for the strongly connected components problem in the fully dynamic setting for general digraphs. Such a bound (in fact, a worst-case bound) has only been given for an easier problem of testing whether the graph is strongly connected \([40]\). It is unlikely that one can achieve a subquadratic update bound for this problem using combinatorial methods \([1]\).

1.1.2 Deterministic Incremental Algorithms with Subquadratic Worst-Case Bounds. As our second contribution, we show that randomization is not always required for obtaining subquadratic worst-case update bounds for dynamic reachability or shortest paths problems in directed graphs.\(^5\) Namely, we show that in the incremental setting, there exist deterministic path-reporting algebraic data structures for reachability, and (approximate) shortest paths. These data structures completely avoid using the Zippel-Schwartz lemma.

Whereas in the incremental setting one usually studies the total update time, obtaining incremental data structures with good worst-case bounds is important for the following additional reasons. First of all, such data structures can efficiently handle rollbacks, i.e., are capable of reverting the most recent insertion within the same worst-case time bound. As a result, they are useful in certain limited fully dynamic settings as well. This property can be also used to obtain offline fully dynamic algorithms\(^6\) with the same update bound in a black-box way (up to polylogarithmic factors). Formally, we have the following transformation\(^9\) (see, e.g., \([29, \text{Theorem 1}]\)).

**Lemma 1.4.** Suppose there is an incremental data structure maintaining some information about the graph \( G \) with initialization time \( I(n, m) \), worst-case update time \( U(n, m) \), and query time \( Q(n, m) \). Then, one can preprocess a sequence of fully dynamic updates to \( G \) (given offline) in \( O(I(n, m) + T \cdot U(n, m)) \) time, so that queries about any of the \( t+1 \) versions of the graph are supported in \( O(Q(n, m)) \) time. The transformation is deterministic.

Offline algorithms are, in turn, important from the hardness perspective - many of the known conditional lower bound techniques for dynamic problems (see, e.g., \([1, 23]\)) apply to the offline setting as well, and thus obtaining a faster offline algorithm can exclude the possibility that a certain conditional lower bound exists.

For incremental reachability, we show the following.

**Theorem 1.5.** There exist deterministic incremental reachability data structures supporting:

- single-edge insertions in \( O(n^{1+4p}) = O(n^{1.529}) \) worst-case time and path-reporting queries in \( O(n^p + |P|) = O(n^{0.529} + |P|) \) time, where \( P \) is the reported simple path.\(^10\)
- insertions of at most \( n \) incoming edges of a single vertex, and single-source reachability tree queries, both in \( O(n^{1.529}) \) worst-case time.

The bounds in Theorem 1.5 match the best-known dynamic matrix inverse-based bounds for fully dynamic transitive closure in the respective update/query variants (that is, single-edge updates/single-pair queries, or incoming edges updates/single-source queries, respectively) \([40]\). However, as Theorem 1.5 shows, in the incremental (or offline fully dynamic, by Lemma 1.4) setting, one can reproduce these bounds deterministically and allow for very efficient certificate reporting.

For incremental \((1+\epsilon)\)-approximate shortest paths, we show the following.

**Theorem 1.6.** Let \( \epsilon \in (0, 1] \) and suppose \( G \) is a weighted digraph with real edge weights in \([1, C]\). There exist deterministic \((1+\epsilon)\)-approximate data structures supporting:

- single-edge insertions in \( O(n^{1+4p} \cdot (1/\epsilon) \cdot \log(C/\epsilon)) \) worst-case time and approximate shortest path-reporting queries in \( O(n^{0.529} + |P|) \) time.
- single-edge insertions in \( O(n^{1+4p} \cdot (1/\epsilon) \cdot \log(C/\epsilon)) \) worst-case time and approximate shortest path-reporting queries in \( O(n^{1.407} + |P|) \) time, where \( P \) is the reported not necessarily simple path.

Interestingly, both trade-offs in Theorem 1.6 match the best-known dynamic matrix inverse-based bounds for fully dynamic transitive closure in the regime of single-edge updates and single-pair queries \([40]\). For comparison, the state-of-the-art fully dynamic \((1+\epsilon)\)-approximate all-pairs distances data structure \([39]\) has \( O(n^{1.865}/\epsilon^2) \) update time and \( O(n^{0.666}/\epsilon^2) \) query time.

\(^7\)That is, allowing path-reporting queries within a strongly connected component against an adaptive adversary.

\(^5\)See \([36]\) for subquadratic deterministic fully dynamic approximate distances algorithms in unweighted undirected graphs.

\(^6\)That is, in the case when the entire sequence of updates issued is known beforehand.

\(^9\)Technically, the transformation requires making the incremental data structure fully persistent first. However, this can be easily achieved deterministically and without introducing additional amortization using standard methods \([18, 19]\), at the cost of only a polylogarithmic slowdown of updates and queries.

\(^10\)We stress that if one only cares about path existence, and not a certificate, the \(|P|\) term can be omitted. This also applies to other results stated in this section.
Whereas in our approximate data structures the reported paths need not be simple, in the important scenario with \( \varepsilon = O(1) \) and \( C = O(1) \) (e.g., for unweighted graphs), the reported non-necessarily simple path (that is nevertheless approximately shortest in terms of length, but not necessarily in terms of hop-length) may contain only a constant factor more edges than the shortest simple path (since the minimum allowed weight is 1). As a result, the latter data structure in Theorem 1.6 may also be used to obtain an \( O(n^{1.407}) \) worst-case update/query time trade-off for simple-path-reporting incremental reachability (by setting, e.g., \( \varepsilon = 1 \)).

Finally, for incremental exact shortest paths in unweighted directed graphs, we obtain the following trade-offs.

**Theorem 1.7.** Let \( G \) be an unweighted digraph. There exist deterministic incremental data structures supporting:

- single-edge insertions and shortest path-reporting queries in \( O(n^{1.62}) \) worst-case time,
- insertions of at most \( n \) incoming edges of a single vertex, and single-source shortest paths tree-reporting queries in \( O(n^{1.724}) \) worst-case time.

The former bound is polynomially smaller than the best-known \( O(n^{1.724}) \) bound on the update/query time of a fully dynamic all-pairs distances data structure [34, 40]. The latter bound is even more interesting – it is still not known whether there exists an exact fully dynamic data structure maintaining single-source distances in an unweighted graph that would achieve subquadratic update time, even for a fixed source \( s \). By our incremental bound and Lemma 1.4, we obtain that an offline fully dynamic data structure with subquadratic worst-case time is possible.

Due to space constraints, the details of our deterministic incremental algorithms with subquadratic worst-case update time can only be found in the full version. We give an overview of how these results are obtained in Section 4.3.

## 2 PRELIMINARIES

In this paper we deal with (possibly weighted) directed graphs. We write \( V(G) \) and \( E(G) \) to denote the sets of vertices and edges of \( G \), respectively. We omit \( G \) when the graph in consideration is clear from the context. A graph \( H \) is a subgraph of \( G \), which we denote by \( H \subseteq G \), if and only if \( V(H) \subseteq V(G) \) and \( E(H) \subseteq E(G) \). We write \( e = uv \in E(G) \) when referring to edges of \( G \) and use \( w_G(uv) \) to denote the weight of \( uv \) (in case of weighted digraphs). We call \( u \) the tail of \( e \), and \( v \) the head of \( e \). If \( uv \notin E \), we assume \( w_G(uv) = \infty \).

For some set \( E' \subseteq V \times V \), we denote by \( G \cup E' \) the graph \( (V(G), E(G) \cup E') \). Similarly, we denote by \( G \setminus E' \) the graph \( (V(G), E(G) \setminus E') \). We also denote the notation \( G - e := G \setminus \{e\} \).

A sequence of edges \( P = e_1 \ldots e_k \), where \( k \geq 1 \) and \( e_i = u_i u_{i+1} \in E(G) \), is called an \( s \rightarrow t \) path in \( G \) if \( s = u_1, v_k = t \) and \( u_i = u_{i+1} \) for each \( i = 1, \ldots, k \). For brevity, we sometimes also express \( P \) as a sequence of \( k + 1 \) vertices \( u_1, u_2 \ldots u_k \) or as a subgraph of \( G \) with vertices \( \{u_1, \ldots, u_k\} \) and edges \( \{e_1, \ldots, e_k\} \). The length of a path \( P \) equals \( \sum_{i=1}^{k} w_G(e_i) \). The hop-length \( |P| \) is equal to the number \( k \) of edges in \( P \). We also say that \( P \) is a \( k \)-hop path. For convenience, we sometimes consider a single edge \( uv \) as a path of hop-length 1, as well as zero length sequence is used to denote an empty path. If \( P_1 \) is a \( u \rightarrow v \) path and \( P_2 \) is a \( v \rightarrow w \) path, we denote by \( P_1 \cdot P_2 \) (or simply \( P_1 P_2 \)) a path obtained by concatenating \( P_1 \) with \( P_2 \).

An out-tree \( T_{\text{out}} \subseteq G \) is a subgraph of \( G \) for which there exists a vertex \( s \in V(T_{\text{out}}) \) (the root), such that each vertex \( v \in V(T_{\text{out}}) \setminus \{s\} \) has precisely one incoming edge in \( T_{\text{out}} \), and \( s \) has no incoming edges in \( T_{\text{out}} \). Symmetrically, an in-tree \( T_{\text{in}} \subseteq G \) is a subgraph of \( G \) for which there exists a vertex \( t \in V(T_{\text{in}}) \) (the root), such that each vertex \( v \in V(T_{\text{in}}) \setminus \{t\} \) has precisely one outgoing edge in \( T_{\text{in}} \), and \( t \) has no outgoing edges in \( T_{\text{in}} \).

A vertex \( t \) is reachable from \( s \), if there is an \( s \rightarrow t \) path in \( G \). A single-source reachability tree \( T_G(s) \) from \( s \) is an out-tree in \( G \) whose root is \( s \) and \( V(T_G(s)) \) equals the set of vertices reachable from \( s \) in \( G \).

Two vertices \( u, v \in V \) are strongly connected if both \( u \) is reachable from \( v \) and \( v \) is reachable from \( u \). Strong connectivity is an equivalence relation. We use the term strongly connected components (SCCs) to refer to the set \( S \) of equivalence classes of the strongly connectivity relation.

For some partition \( V = V_1 \cup \ldots \cup V_k \) of \( V \), we define \( G/G' \) to be a graph with vertices \( V \) obtained from \( G \) by contracting each subset \( V_i \) into a single vertex labeled \( V_i(G[V_i]) \) does not necessarily need to be a connected subgraph). For any \( xy \in E(G) \) such that \( x \in V_i \) and \( y \in V_j \), we have a corresponding edge \( V_i V_j \) in \( G/G' \) if and only if \( V_i \neq V_j \). As a result, \( G/G' \) can be a multigraph.

## 3 FULLY DYNAMIC PATH REPORTING IN DAGS

In this section we assume that the graph \( G \) remains acyclic at all times and give dynamic path reporting algorithms under this assumption. We will first show a fully dynamic algorithm for maintaining a topological order. This is a crucial element in our point-to-point path reporting algorithm. Finally, we will use another idea to construct an algorithm for reporting single-source reachability trees. These results will be lifted to general digraphs in Section 6.

We will repeatedly make use of the following dynamic transitive closure data structure of [33] that allows for subquadratic updates and sublinear queries.

**Theorem 3.1.** [33] Let \( G \) be a digraph and let \( \delta \in (0, 1) \). There exists a data structure that supports single-edge insertions/deletions to \( G \) in \( O(n^{(1,0,\delta)} - \delta + n^{1+\delta}) \) worst-case time, and point-to-point reachability queries in \( G \) in \( O(n^\delta) \) worst-case time.

The data structure is Monte Carlo randomized and produces answers correct with high probability.

It is important to note that as long as the data structure does not err (which happens with low probability over its random choices for the fixed sequence of \( \text{poly}(n) \) updates), the produced answers depend only on the current graph (and not the previous answers) and thus are unique. Therefore, the data structure can be obviously used against an adaptive adversary.

### 3.1 Fully Dynamic Topological Order

Let us identify \( V \) with \( \{1, \ldots, n\} \). Let \( \pi : V \rightarrow \{1, \ldots, n\} \) be a permutation such that \( uv \in E \) implies \( \pi(u) < \pi(v) \). Upon initialization, some \( \pi \) can be computed in linear time using one of the classical algorithms.
We will maintain a data structure $D$ of Theorem 3.1 on $G$ and pass all the issued edge updates to it after updating the topological order. Moreover, we will store the vertices $V$ in an array $A$ sorted according to $\pi$, i.e., we have $A[i] = v$ iff $\pi(v) = i$. Since $\pi$ is simply an inverse of $A$, we will only focus on maintaining $A$: all the changes to $A$ can be reflected in $\pi$ in a straightforward way.

Now let us describe how to handle updates. If the update is a deletion of an edge then we do not have to do anything, since $\pi$ remains a topological order of $G - e$. So suppose we are inserting an edge $e = uv$. Again, if we currently have $\pi(u) < \pi(v)$, the topological order does not need to be updated. Now consider the case that $\pi(u) > \pi(v)$. We need to modify the topological numbering only for vertices $v$ currently satisfying $\pi(v) \in [\pi(u), \pi(u)]$, as for the remaining vertices the topological order can remain the same. Let us call the set of these vertices $W$. In other words, $W$ contains the vertices between $v$ and $u$ in $A$ (including $u$ and $v$).

Let $T \subseteq W$ be the vertices of $W$ reachable from $v$ before the insertion, including $v$. Let $S \subseteq W$ be the vertices of $W$ that can reach $u$ before the insertion, including $u$. Note that each of $S, T, W$ can be found by issuing $|W|$ = $O(n)$ queries to the data structure $D$, i.e., in $O(n \cdot \delta)$ worst-case time.

We will update $A$ in the following way. Let $Z = W \setminus (S \cup T)$. The subarray $A[\pi(x), \pi(u)]$ will be replaced by a sequence of vertices $S \cdot Z \cdot T$, where each of the sets $S, T, Z$ is ordered according to $\pi$ (before the insertion). The correctness of such a change follows also from the following lemma.

**Lemma 3.2.** Let $S, T, Z$ be defined as above. Then, after the edge insertions:

- no vertex $t \in T$ can reach a vertex $s \in S$,
- no vertex $t \in T$ can reach a vertex $z \in Z$,
- no vertex $z \in Z$ can reach a vertex $v \in S$.

**Proof.** Note that no vertex $t \in T$ can reach a vertex $s \in S$ as otherwise there would be a path $s \rightarrow t \rightarrow s \rightarrow u$ in $G$, which, along with the inserted edge $uv$, would form a cycle. This would contradict that $G$ is acyclic. In particular, we have $S \cap T = \emptyset, v \in T$, and $u \in S$.

Observe that no vertex $z \in Z$ can reach a vertex $t \in T$, since, by definition, $T$ contains all vertices in $W$ reachable from $v$. Similarly, no vertex $z$ can reach a vertex from $S$. □

Finally, observe that if for two vertices $(x, y)$, one of the following holds: $(x, y) \in E^2, (x, y) \in E^2, (x, y) \in E^2, \pi(x) < \pi(y)$, or $\pi(y) > \pi(u)$ before the insertion, then the same holds also after the insertion. This is because we don’t change the relative order of vertices in each of the sets $S, T, Z$, $\{x : \pi(x) < \pi(u)\}, \{y : \pi(y) > \pi(u)\}$.

Let $\rho$ be the smallest number such that $\omega(1, \rho, 1) = 1 + 2\rho$. With the current best known bounds on the values $\omega(1, \cdot, 1)$, we have $\rho \approx 0.529$ [22]. By setting $\delta = \rho$, we obtain the following lemma.

**Lemma 3.3.** Let $G$ be a digraph. There exists a data structure supporting fully dynamic single-edge updates to $G$ that keep $G$ acyclic, and maintaining a topological order of $G$ in $O((n^{1 + \rho}) \cdot \rho) = O(n^{1.529})$ worst-case time. The algorithm is Monte Carlo randomized. With high probability, the maintained topological order is correct and is uniquely determined by the sequence of updates.

**Proof.** Note that here randomization is only used inside the data structure $D$. Consequently, high probability correctness follows by Theorem 3.1. As the answers produced by that component are uniquely determined by the graph updates (as the graph itself is), so is the maintained topological order. □

### 3.2 Point-to-Point Path Queries

In order to support point-to-point path queries under fully dynamic edge updates, we will use two data structures. The first one is a data structure $D$ of Theorem 3.1 with $\delta$ set to $\rho$. The other data structure is that of Lemma 3.3, maintaining a topological order of $G$.

No additional information is maintained, and so in the updates we are simply passing each edge update to those data structures.

Now consider queries. Suppose we are requested to find an $s \rightarrow t$ path, where $s, t \in V$ are the query vertices. Using a single query to $D$ we check whether such a path exists in $O(n^{1 + \rho})$ time. If this is not the case, we are done. Otherwise, we infer that $\pi(s) < \pi(t)$.

The algorithm for constructing an $s \rightarrow t$ path (known to exist), is recursive. If $s = t$, then clearly an empty path can be returned.

Otherwise, we scan through the outgoing edges $\epsilon_x = sx$ of $s$ in the order of $\pi(x)$. Note that for each such edge we have $\pi(x) > \pi(s)$ and the values $\pi(x)$ are distinct. When scanning the edge $\epsilon_x$, we stop if $x$ can reach $t$ in $G$. Each such test takes $O(n^{\rho})$ time using a single query to $D$. Then we recursively construct a path $P = x \rightarrow t$ and return the path $e_x P$.

Observe that since an $s \rightarrow t$ path exists, some edge $e_x$ will surely lead to a recursive call: this will happen for $e_x$ with minimum $\pi(x)$ such that a path $x \rightarrow t$ exists in $G$.

**Lemma 3.4.** If $s$ can reach $t$ in $G$ then the above algorithm constructs an $s \rightarrow t$ path in $O((\pi(t) - \pi(s)) \cdot n^{\rho} + 1)$ time.

**Proof.** We proceed by induction on $\pi(t) - \pi(s)$. If $\pi(t) - \pi(s) = 0$, i.e., $s = t$, then the algorithm obviously finishes with a correct answer in $O(1)$ time.

Suppose $\pi(s) < \pi(t)$. Consider a path $P = s \rightarrow t$ in $G$ whose first edge $e = sx$ has the minimum value $\pi(x)$. Then, the algorithm issues exactly one query to $D$ for the existence of a path $y \rightarrow t$ for each $y \in V$ with $\pi(y) \in (\pi(s), \pi(x))$. This amounts to at most $\pi(x) - \pi(s) - 1$ queries. Each such query returns a negative answer. Then, the answer to a subsequent $x \rightarrow t$ query is affirmative. As a result, the algorithm recursively searches for an $x \rightarrow t$ path. Since $\pi(x) - \pi(s) < \pi(x) - \pi(s)$, by the inductive hypothesis, the $x \rightarrow t$ path will be constructed in $O((\pi(t) - \pi(x)) \cdot n^{\rho} + 1)$ time. So the total time needed to construct an $s \rightarrow t$ path is $O(1 + ((\pi(x) - \pi(s)) + \pi(t) - \pi(x)) \cdot n^\rho + 1) = O(1 + (\pi(x) - \pi(s)) \cdot n^\rho)$, as desired. □

Using the above, we obtain the following lemma.

**Lemma 3.5.** Let $G$ be an acyclic digraph. There exists a data structure supporting fully dynamic single-edge updates to $G$ that keep $G$ acyclic, and point-to-point path queries, both in $O(n^{1 + \rho}) = O(n^{1.529})$ worst-case time.

The algorithm is Monte Carlo randomized and produces correct answers with high probability and against an adaptive adversary.

**Proof.** By Lemma 3.3, with high probability (if the data structure $D$ does not err) the maintained topological order is uniquely
3.3 Single-Source Reachability Tree Queries

Suppose we want to compute a single-source reachability tree from a query vertex \( s \in V \). We could, in principle, reuse the path reporting procedure developed for point-to-point queries to construct such a tree. By taking for each \( t \in V \setminus \{ s \} \) that is reachable from \( s \), the ultimate edge \( e_t = y_t \) on the \( s \to t \) path, we obtain an out-tree rooted at \( s \). Such an edge \( e_t = y_t \) could be computed by running a single step of the recursive algorithm on the reverse graph of \( G \). Unfortunately, in general, finding that edge could require \( \pi(t) - \pi(y) \) reachability queries to \( D \). This could result in \( \Omega(n\Delta) \) time per just a single edge of the tree when \( \pi(t) - \pi(y) = \Theta(n) \).

As a consequence, we need to use a different approach. Let \( S \) be the set of all vertices reachable from \( s \) in an acyclic digraph \( G \). Recall that \( S \) can be computed using \( O(n) \) reachability queries issued to \( D \) in \( O(n\Delta) \) time. We rely on the following observation that holds for DAGs.

**Observation 3.6.** For each \( t \in S \setminus \{ s \} \), let \( e_t = vt \) be an arbitrary edge of \( G \) with \( v \in V \). Then, the edges \( \{ e_t : t \in S \setminus \{ s \} \} \) form an out-tree on \( S \) rooted at \( s \).

By the above observation, it is enough to pick, for each \( t \in S \setminus \{ s \} \), any incoming edge with its tail in \( S \). In order to work against an adaptive adversary we will pick these edges in a consistent deterministic manner. For example, we could pick for each \( t \) an incoming edge \( e_t = vt \) where \( v \in S \) has the minimum possible label (recall that we identify \( V \) with \( \{ 1, \ldots, n \} \)).

To achieve that we use an approach reminiscent of the algorithms for computing minimum witnesses for boolean matrix multiplication [14]. Let \( \Delta + 1 \leq n \) be an integer parameter to be chosen later. We will maintain \( G = [n/\Delta] \) data structures \( D_1, \ldots, D_q \) of Theorem 3.1, where the underlying graph \( G_t \) maintained in \( D_t \) is defined as follows. Let \( V', V'' \) be two copies of \( V \). Denote by \( v' \in V' \) and \( v'' \in V'' \) the corresponding copies of a vertex \( v \in V \). We have \( V(G_t) = V \cup V' \cup V'' \) and \( G \subseteq G_t \). For each edge \( uv \in E(G) \), we have \( u \in E(G_t) \) and \( u'v'' \in E(G_t) \). Moreover, for each \( u \in V \) such that \( u \in [i - 1] \cdot \Delta + 1, i \cdot \Delta \), we add an edge \( uu' \) to \( G_t \).

**Lemma 3.7.** Let \( u, v \in V(G) \), \( u \neq v \). Then, a path \( u \rightarrow v' \) exists in \( G_t \) if and only if there exists a path \( P = u \rightarrow v \) in \( G \) such that the ultimate edge \( yo \) of \( P \) satisfies \( y \in [i - 1] \cdot \Delta + 1, i \cdot \Delta \).

**Proof.** For the forward direction, let \( P = P', y v \). Then, by \( G \subseteq G_t \) it follows that \( P' \) is a \( u \rightarrow y \) path in \( G_t \). But since \( y \in [i - 1] \cdot \Delta + 1, i \cdot \Delta \), there exist edges \( yy' \) in \( G_t \) as well. Therefore, \( G_t \) indeed contains a \( u \rightarrow y' \) path \( P' \rightarrow y'v' \). Now suppose there is an \( u \rightarrow y' \) path \( Q \) in \( G_t \). Since the vertex \( u'p \) only has incoming edges from \( V' \), and a vertex from \( V' \) has an incoming edge (necessarily one, from \( V' \)) only if that vertex lies in \( [i - 1] \cdot \Delta + 1, i \cdot \Delta \), \( Q \) is of the form \( Q' \rightarrow yy' \rightarrow y'v' \), where \( y \in V \) is such that \( yv \in E(G) \) and \( y \in [i - 1] \cdot \Delta + 1, i \cdot \Delta \). Moreover, \( Q' \subseteq G_t[V] = G' \) as \( V' \cup V'' \) has no outgoing edges to \( V \). So, since \( yo \in E(G) \), \( Q' \rightarrow yo \) is an \( u \rightarrow \) path in \( G \) and its penultimate vertex indeed lies in the desired interval.

Every issued edge update to \( G \) is passed to each of the \( q \) data structures \( D_t \). Note that such an update translates to two edge updates to \( G_t \): one in \( V \times V \) and one in \( V' \times V'' \). As a result, an edge update is processed in \( O((n/\Delta) \cdot n^{1+\rho}) \) worst-case time.

Now, at query time, in order to find for \( t \in S \setminus \{ s \} \) an incoming edge \( \sigma_t \) such that \( v \in S \) and \( v \) is minimum possible, we first find the smallest \( j \in [1, \ldots, q] \) such that there is an \( s \rightarrow t \) path with the penultimate vertex in the interval \( [(j - 1) \cdot \Delta + 1, j \cdot \Delta] \). By Lemma 3.7, this can be achieved in \( O((n/\Delta) \cdot n^\rho) \) time by issuing \( q = O(n/\Delta) \) queries, one per each of the data structures \( D_t \). Afterwards, in \( O(\Delta) \) time we iterate over at most \( \Delta \) edges of \( t \) coming from vertices in the interval \( [(j - 1) \cdot \Delta + 1, j \cdot \Delta] \) in order to locate the desired minimum labeled vertex \( v \). Since there are \( O(n) \) different vertices \( t \), finding the desired incoming edges for all of them costs \( O(n/\Delta \cdot n^{1+\rho} + n\Delta) \) worst-case time. By setting \( \Delta = n^{1+\rho}/2 \), we obtain the following.

**Lemma 3.8.** Let \( G \) be an acyclic digraph. There exists a data structure supporting fully dynamic single-edge updates to \( G \) that keep \( G \) acyclic, and reporting a single-source reachability tree from any query vertex, both in \( O(n/\Delta) \) worst-case time. The algorithm is Monte Carlo randomized and produces answers correct with high probability against an adaptive adversary.

**Proof.** By Theorem 3.1, each of the data structures \( D_1, \ldots, D_q \) produces only correct answers with high probability. The answers produced by our algorithm are uniquely determined by the graph – we always choose a minimum labeled feasible edge for Observation 3.6. Hence, the answers of the algorithm do not reveal any random bits used by the underlying data structures.

The Lemmas 3.5 and 3.8 combined yield Theorem 1.1.

4 OVERVIEW OF THE REMAINING RESULTS

4.1 Fully Dynamic Path Reporting in General Digraphs

The path finding algorithm behind Lemma 3.5 fails for digraphs with non-trivial (i.e., consisting of at least two vertices) strongly connected components. In order to deal with this issue, we apply the usual idea of solving the problem separately on the “acyclic” part of the graph, and separately on the individual strongly connected components. Of course, the acyclic part corresponds to the condensation \( G/S \), where \( S \) denotes the family of strongly connected components of \( G \).

Fully dynamic path reporting inside strongly connected components against an adaptive adversary is a challenge by itself, and no prior tools for this task have been developed. However, the known combinatorial methods [9] allow us to solve the “strongly connected” problem in subquadratic amortized time in the decremental setting. This often captures some of the critical difficulties of the fully dynamic setting. To apply this tool, however, we need to abandon handling edge insertions and deletions in a uniform way, as is typical in algebraic dynamic graph algorithms (such as Theorem 3.1) based on path counting or dynamic matrix inverse.
In comparison to the algorithm for acyclic graphs from Section 3.2, in order to handle insertions, the algorithm for general graphs operates in phases of $F$ edge insertions. At the beginning of each phase, a path reporting decrementally strongly connected components data structure $C$ is initialized for the current graph $G$. This data structure maintains a graph $G^-$ defined as the graph at the beginning of the current phase minus the edges deleted in that phase. The data structure $C$ can be extended (in a standard way, see Lemma 6.1) to also maintain the condensation $G^-/S$ efficiently, where $S$ denotes the set of strongly connected components of $G^-$. If there were no edge insertions issued, given a query $s,t$, applying the algorithm of Lemma 3.5 to the condensation $G^-/S$ could produce, in $O(n^{1+\rho})$ time, a path $P$ in $G^-/S$ between components $X,Y\in S$ such that $s\in X$ and $t\in Y$. Then, the data structure $C$ and the condensation itself could be used to lift the path $P$ to an actual $s\rightarrow t$ path in $G$, again in $O(n^{1+\rho})$ time.

However, in presence of insertions, to compute a desired $s\rightarrow t$ path upon query, one first needs to identify which of the current phase’s inserted edges (and in which order) necessarily appear on a sought $s\rightarrow t$ path $P$. This can be decided in $O(n^{1+\rho}+nf)$ time using a simple but powerful generalization of Theorem 3.1 given in Lemma 5.1 - that efficiently maintains an incremental subset of rows/columns of the transitive closure. Moreover, using Lemma 5.1, one can compute a partition of $G^-$ into clusters, such that (at most $F+1$) individual maximal subpaths of $P$ entirely contained in $G^-$ can be sought in separate clusters (see Lemma 6.2). This enables constructing them in $O(n^{1+\rho})$ total time, instead of $O((F+1)n^{1+\rho})$ time, which one would need to pay if each of the subpaths was computed in the entire graph $G^-$. For details, see Section 6.

### 4.2 Fully Dynamic Reachability Tree Reporting in Directed Graphs

Unfortunately, Observation 3.6 does not hold for general graphs: given some source $s\in V$, choosing an arbitrary incoming edge (e.g., that with the minimum label) from each vertex reachable from $s$ might lead to a disconnected (in the undirected sense) graph containing cycles.

In order to deal with this problem, we could, again, apply Observation 3.6 to the condensation $G/S$ (with the source set to the SCC containing $s$). The obtained out-tree $T^\prime$ in $G/S$ could then be extended to a single-source reachable tree $T$ from $s$ in $G$ in two steps: first, expand each vertex of $T^\prime$ (i.e., a strongly connected component $S$ reachable from $s$) into a sparse strongly connected subgraph of $S$ using the data structure $C$. Then, compute a single-source reachability tree from $s$ in the obtained subgraph of $G$ using any graph search procedure in $O(n)$ time.

For similar reasons as applied to path reporting, we need to operate in phases of edge insertions, so that the strongly connected components of $G^-$ (defined as in Section 4.1) only split. However, it is not clear how to efficiently handle the condensation $G^-/S$ (which, critically, does not include some of the original edges of the decremental graph $G^-$) using the algebraic data structure of Theorem 3.1, so that an interval of length $\Delta$ containing the minimum labeled tail of an incoming edge can be located efficiently. Recall that when the strongly connected components split, the condensation undergoes vertex splits, and each vertex split (revealed online) might require $\Theta(n)$ edges changing endpoints. Moreover, each of the $O(n^2)$ original edges of $G^-$ can be inserted, at some point, to $G^-/S$. Consequently, we might need to perform $\Theta(n^2)$ edge updates on the data structure of Theorem 3.1 if we want it to reflect $G^-/S$. However, the updates in Theorem 3.1 are relatively costly and we could not afford performing $\Theta(n^2)$ such updates within a single phase.

We avoid the above problem by picking, for each vertex of $G^-$ (and not $G^-/S$), the minimum-labeled incoming edge wrt. the topological order $\pi$ of $G^-/S$, instead of a minimal edge wrt. an arbitrary order that is fixed initially. This, of course, leads to other complications: the topological order of $G^-/S$ evolves in time. However, we can maintain a topological labeling $\pi$ of the dynamic graph $G^-/S$ that admits a certain nesting property (see Lemma 6.1). This property guarantees that if the interval $[\pi(S), \pi(S)+|S|-1]$ for $S\in S$ is contained in some interval $I_j$ of the form $[i-1, i+1, j]$, then we will have $\pi(S') \subseteq I$ for any $S' \subseteq S$ that becomes an SCC of $G^-$ in the future. We call such an SCC $S$ non-special, and all other SCCs special. In other words, the topological order of all vertices within a non-special component $S$ is fixed, up to the interval $I_j$ it is currently contained in. This enables us to locate the minimum labeled edges coming from non-special SCCs using the approach of Section 3.3.

Luckily, the number of special SCCs that do not fall into the above category is always $O(n/\Delta)$, so we can handle them using a different, more straightforward approach.

The fully dynamic tree-reporting data structure constitutes the most technically involved part of this paper and heavily relies on the developments of Section 6. Details can be found in Section 7.

### 4.3 Deterministic Incremental Algorithms with Subquadratic Worst-Case Bounds

In order to obtain Theorems 1.5, 1.6, and 1.7, we follow the general approach behind the subquadratic trade-offs for reachability – based on either path counting [16] or dynamic matrix inverse [33, 40]. Namely, the algorithms operate in phases of $F=O(n^2)$ updates. Each phase starts with a computationally heavy recomputation step based on rectangular matrix multiplication, that, roughly speaking, recomputes the reachability/shortest paths matrix $A$ for all pairs of vertices $u,v\in V$ in the graph $G_0$ (equal to $G$ when the phase starts) based on the analogously defined matrix $A'$ and the $O(n^2)$ updates from the previous phase. The cost of such a recomputation step should be thought of being amortized over the $F$ updates of the phase, and this is the only source of amortization. However, there is a well-known standard technique (also used in the previous dynamic algebraic graph algorithms, e.g., [16, 33, 40]) for converting such amortized bounds into worst-case bounds by maintaining two copies of the data structure that switch roles every $F/2$ updates.

The queries are answered using the matrix $A$ and possibly some auxiliary data structure build on the current phase’s updates that depends on the trade-off we want to achieve.

In the fully dynamic algorithms for reachability, the matrix $A$ stores path counts modulo a prime [16], or values of certain multivariate polynomials at suitable points modulo a prime [33, 40]. In the fully dynamic shortest paths algorithms [34, 39, 40], the matrix
A stores coefficients of low-degree terms of multivariate polynomials with suitably chosen coefficients. All these fully dynamic algorithms critically rely on the Zippel-Schwartz lemma and thus are inherently Monte-Carlo randomized.

On the other hand, we show that in the incremental setting a much simpler idea is sufficient. Roughly speaking, in our incremental algorithms, the matrix \( A \) stores \emph{paths} in \( G_0 \). These paths are implemented as optimal purely functional concatenatable dequeues of \([25]\) to allow for convenient and efficient path manipulations, such as concatenation and iteration. More specifically, for reachability these are simple paths, for approximate shortest paths \( A \) stores approximate shortest paths, whereas for exact shortest paths, \( A \) stores short paths of length no more than a certain threshold \( h \).

For reachability, the recomputation is performed using boolean matrix multiplication, for approximate shortest paths, we use the \((1 + \epsilon)\)-approximate min-plus product \([42]\), whereas for exact shortest paths – the bounded exact min-plus product \([42]\). Turning these respective (rectangular) matrix products of number matrices into products of path matrices is possible since each of these products allows for computing witnesses deterministically \([2, 42]\).

The trade-offs that we obtain for incremental reachability match the best known fully dynamic transitive closure trade-offs for single-edge and single-vertex-incoming-edges updates obtained via dynamic matrix inverse \([40]\). However, these fully dynamic trade-offs seem hopelessly randomized and do not allow for path reporting.

Interestingly, for \((1 + \epsilon)\)-approximate shortest paths in the single-edge updates setting, we obtain the same trade-offs (up to polylogarithmic factors) as in the case of reachability. To this end, we need to carefully control the error and perform recomputations using a hierarchical binary-tree-like circuit, as opposed to a linear path-like circuit that is sufficient for reachability.

Finally, for exact shortest paths in unweighted graphs, we exploit a very simple observation (also used in e.g. \([26]\)), that a deterministically computed \( \widetilde{O}(F) \)-size hitting set of length-\( n \) \( F \) paths remains valid and is \( \widetilde{O}(F) \)-sized throughout a phase of insertions if we simply augment it with the endpoints of inserted edges.

Due to space limitations, the details of the discussed incremental algorithms can only be found in the full version of this paper.

5 FULLY DYNAMIC STRONGLY CONNECTED COMPONENTS

As a warm-up, let us first consider fully dynamic maintenance of strongly connected components. We start with the following variant of the data structure from Theorem 3.1 which will be crucial to all our developments in general graphs.

**Lemma 5.1.** Let \( G = (V, E) \) be a digraph. Let \( R \subseteq V \). There exists a data structure explicitly maintaining the information whether there exists an \( r \to v \) path in \( G \) for each of the pairs \((r, v) \in R \times V\) and supporting each of the following update operations in \( O(n^{1+\epsilon} + n \cdot |R|) \) worst-case time:

- single-edge insertions/deletions to \( G \),
- adding a new vertex to the set \( R \),
- resetting \( R \) to \( \emptyset \).

The data structure is Monte Carlo randomized and produces correct answers with high probability.

**Proof.** We follow the idea of \([34]\), which is an extension of the data structure of Theorem 3.1 supporting single edge insertions and deletions in \( O(n^{1+\epsilon}) \) worst-case time, and queries in \( O(n^\delta) \) time (for \( \delta := \rho \)). From \([33, \text{Theorem 7}]\), maintaining the reachability information reduces, within the same update time and with high probability correctness, to maintaining the inverse of a matrix \( A \) which is a symbolic adjacency matrix with variables replaced with random elements of the field \( \mathbb{Z}/p\mathbb{Z} \) for a sufficiently large prime number \( p = \Theta(\text{poly} \ n) \). Indeed, to maintain the reachability information from \( R \) to \( V \), it is enough to explicitly maintain the submatrix \((A^{-1})_{RV}\) under the sequence of updates.

The key idea is to write, after a single entry change, the updated matrix as the product of two matrices \( A' = A \cdot B \). Here \( B \) has a special form, i.e., it has non-zero elements only on the diagonal and one column (say \( j \)) where the update occurs \([34, \text{Theorem 4}]\). This implies that \( B^{-1} \) has similar non-zero structure \([5, \text{Fact 5.4}]\). This way, in the multiplications \( A^{-1} = B^{-1}A^{-1} = A^{-1} + (B^{-1} - I)A^{-1} \) only the \( j \)-th row of \( A \) is used. In particular, we can obtain the submatrix \((A^{-1})_{RV}\) by the following equation:

\[
(A^{-1})_{RV} = (A^{-1})_{RV} + (B^{-1} - I)_{(j,j)} \cdot (A^{-1})_{(j,V)}.
\]

The entries of \((A^{-1})_{(j,V)}\) can be found in \( O(n^{1+\epsilon}) \) time using Theorem 3.1. Subsequently, recomputation of the part of the inverse can be done via a vector-vector product in \( O(n \cdot |R|) \) time. To add a vertex \( v \) to the set \( R \) we additionally need to query the \( v \)-th row of \( A^{-1} \) which again takes \( O(n^{1+\epsilon}) \) time by Theorem 3.1. \( \square \)

We will also need a near-optimal decremental strongly connected components data structure.

**Theorem 5.2.** \([11]\) Let \( G \) be a directed graph. There exists a Las Vegas randomized data structure maintaining the strongly connected components \( S \) of \( G \) explicitly\(^{12}\) subject to edge deletions in \( \tilde{O}(n + m) \) expected total time.

**Remark 5.3.** The data structure of Theorem 5.2 can be converted into a Monte Carlo data structure that runs in worst-case \( \tilde{O}(n + m) \) total time and is correct with high probability. Indeed, it is enough to maintain \( \Theta(\log n) \) independent copies of the data structure, such that each of them is terminated prematurely when its actual time used exceeds the expected running time by more than a fixed constant factor. A failure is declared if all of the copies are terminated prematurely. By Markov’s inequality, this happens with low probability.

Our algorithm operates in phases. Each phase spans \( F \) edge insertions. At the beginning of a phase, we reinitialize a decremental data structure \( C \) of Theorem 5.2 that will be responsible for maintaining the components that are not affected by the insertions. The remaining components will be handled using a data structure \( D \) of Lemma 5.1 with the (growing) set \( R \) storing the (at most \( F \)) heads of edges inserted in the current phase.

More specifically, using the data structures \( C \) and \( D \), we will build a not very dense graph \( H \) which will preserve the strongly connected components of \( G \). The following lemma defines such a graph and proves that it possesses the desired property.

\(^{12}\)That is, the algorithm maintains a mapping \( s : V \to S \) such that \( s(v) \) is the SCC of \( v \), and an identifier of each \( s(v) \) is explicitly stored in a memory cell, so that one is notified every time that identifier changes.
Lemma 5.4. Let \( G \) be a digraph. Let \( E^+ \) be some set of at most \( f \) edges. Let a graph \( H \) on \( V \) with \( O(nf) \) edges contain the following:

1. for each strongly connected component \( S \) of \( G \), a directed cycle on the vertices \( S \),
2. for each \( w \in E^+ \), and each \( w \in V \), an edge \( euv \) if we can reach \( w \) in \( G \cup E^+ \), and an edge \( euv \) if we can reach \( w \) in \( G \cup E^+ \).

Then, for all \( u, v \in V \), \( u \) and \( v \) are strongly connected in \( G \cup E^+ \) if and only if they are strongly connected in \( H \).

Proof. First, each edge \( xy \) in \( H \) certifies the existence of an \( x \to y \) path in \( G \cup E^+ \). As a result, if \( u \) and \( v \) are strongly connected in \( H \), then they are strongly connected in \( G \cup E^+ \) as well. This proves the "\( \Rightarrow \)" direction.

Now, suppose \( u \) and \( v \) are strongly connected in \( G \cup E^+ \). If these vertices are strongly connected in \( G \) as well, they lie on a single cycle in \( H \), so they are strongly connected in \( H \) as well.

So consider the case when \( u \) and \( v \) are not strongly connected in \( G \). Then, there exists either a \( u \to v \), or a \( v \to u \) path through some edge \( xy = e \in E^+ \) if there existed paths in both directions not using any edge from \( E^+ \). \( u \) and \( v \) would be already strongly connected in \( G \). It follows that \( y \) is strongly connected with \( u \) and \( v \) in \( G \cup E^+ \). Consequently, we have edges \( uy, yv, yx \), and \( vx \) in \( H \). It follows that there exist paths \( u \to v \) and \( v \to u \) in \( H \), so \( u \) and \( v \) are indeed strongly connected in \( H \).

Let \( E^- \) denote the set of edges inserted in the current phase (and not deleted by any of the updates in that phase). Let \( G^- \) denote the graph \( G \) at the beginning of the phase minus the edges deleted in the current phase. So we have \( G = G^- \cup E^+ \) at all times. We maintain the invariant that \( D \) stores the graph \( G \) (i.e., \( D \) is passed all the fully dynamic updates), and the set \( R \) in the data structure \( D \) equals the set of heads \( \{ y : xy \in E^- \} \). By Lemma 5.1, this can be guaranteed in \( O(n^{1+\rho} + nF) \) worst-case time per update. The set \( R \) is reset to \( \emptyset \) when a new phase starts.

The data structure \( C \), on the other hand, is only passed the edge deletions and thus maintains the graph \( G^- \).

Now, we leverage Lemma 5.4 as follows. We use \( C \) to construct the first type of edges of the graph \( H \) of Lemma 5.4 (applied to \( G = G^- \)) in \( O(n) \) time – recall that \( C \) stores the SCCs of \( G^- \) explicitly. Moreover, the edges of \( H \) of the second type are constructed by reading the \( O(nF) \)-size information stored by \( D \). Finally, to compute the strongly connected components of the current \( G \) (which equals \( G^- \cup E^+ \)), we run any classical linear-time strongly connected components algorithm on the graph \( H \). Recall that the graph \( H \) has size \( O(nF) \).

Lemma 5.5. The strongly connected components of a digraph can be maintained explicitly subject to edge insertions and deletions in \( O(n^{1+\rho}) = O(n^{1.329}) \) amortized time per update. The algorithm is Monte Carlo randomized and correct with high probability.

Proof. Since the data structure \( C \) of Theorem 5.2 is rebuilt in each \( F \) insertions, the amortized cost of maintaining it is no more than \( O(n^{1/2}F) \). By Lemma 5.1, the data structure \( D \) is updated in \( O(n^{1+\rho} + nF) \) worst-case time per edge update. Both data structures are Monte Carlo randomized and correct with high probability (see Remark 5.3). To optimize the running time, set \( F = n^{1/2} \). Since \( \rho \geq 1/2 \), the \( n^{1+\rho} \) term dominates the amortized update time.

Finally, we note that as strongly connected components are defined in a unique way, the above algorithm can be used against an adaptive adversary, as the uniqueness guarantees that the structure of \( S \) does not leak randomness.

6 FULLY DYNAMIC PATH REPORTING IN GENERAL DIGRAPHS

In this section, we will prove the following theorem.

Theorem 1.2. Let \( C \) be a decremental data structure with total update time \( T(n,m) = \Omega(m + n) \) (1) maintaining strongly connected components explicitly and (2) supporting queries reporting a simple path \( P \) between arbitrary strongly connected vertices \( u, v \) of the maintained graph in \( O(|P| \cdot n^\rho) \) time. Then, there exists a fully dynamic data structure supporting single-edge updates and point-to-point path reporting queries with amortized update time and worst-case query time of \( O \left( \sqrt{\left( \frac{m}{n} \right)} \cdot n + n^{1+\rho} \right) \).

If the data structure \( C \) works against an adaptive adversary, so does the fully dynamic path reporting data structure.

We start with the following black-box extension of decremental strongly connected components maintenance, whose proof can be found in the full version. Similar extensions have been previously used in, e.g., [9, 10], albeit with the goal of maintaining approximate single-source shortest paths.

Lemma 6.1. Let \( G \) be a digraph and suppose the strongly connected components \( S \) of \( G \) are maintained explicitly as a mapping \( s : V \to S \) subject to edge deletions.

Then, in additional \( O(n + m \log n) \) total time, we can also achieve the following.

1. Explicitly maintain topological labels \( \pi : S \to \{1, n\} \) of the SCCs so that:
   - All the intervals \( [\pi(S), \pi(S) + |S| - 1] \), \( S \in S \) are disjoint and form a partition of \( \{1, n\} \).
   - If for some \( S' \in S \), \( S \neq S' \) there is a path from \( S \to S' \) in \( G \), then \( \pi(S) < \pi(S') \).
   - If an SCC \( S \) splits due to an edge deletion into some number of smaller SCCs \( S_1, \ldots, S_t \), then the intervals \( [\pi(S_1), \pi(S_1) + |S_1| - 1] \) form a partition of \( [\pi(S), \pi(S) + |S| - 1] \).

2. Explicitly maintain a condensation \( G/S \) of \( G \), which is a single acrylic digraph \( (S, E') \) such that for each \( X, Y \subseteq S \) there is an edge \( XY \in E' \) if and only if there is an edge \( e_{XY} \in E \) such that \( x \in X \) and \( y \in Y \). Moreover, one such edge \( e_{XY} \in E \) is maintained as well.

The algorithm is deterministic and the maintained information depends only on the initial graph \( G \) and the sequence of updates.

The algorithm will again operate in phases spanning \( F \) insertions, forming a set \( E^+ \), where \( F \) is to be set later. Each phase will involve initializing a decremental data structure \( C \) as stated in Theorem 1.2, along with the auxiliary data structures from Lemma 6.1. So, \( C \) will actually maintain a graph \( G^- \) defined as the graph \( G_0 \) from the beginning of the phase minus the edges of \( G_0 \) deleted in the current phase. Denote by \( S \) the strongly connected components of \( G^- \).

Again, we will use a data structure \( D \) of Lemma 5.1 with the set \( R \) storing the heads of the edges of \( E^+ \), but in a slightly different manner. Namely, when the phase proceeds, \( D \) will only accept edge
deletions. As a result, throughout the phase, $\mathcal{D}$ will also store the graph $G^-$. However, when the phase ends, all the edges $E^+$ will be added to $\mathcal{D}$ at once.

Additionally, we use a data structure $Q$ of Theorem 3.1 (with $\delta = \rho$) in a similar way as $\mathcal{D}$ in order to enable $O(nP)$-time reachability queries on $G^-$ throughout the phase.

Given source $s$ and target $t$, the query algorithm will first identify a minimal subset $E^+_{st}$ of edges of $E^+$ such that a path $P = s \to t$ exists in $G^- \cup E^+_{st}$, along with the order $e_1, \ldots, e_j$ in which the edges $E^+_{st}$ appear on $P$. Then, $P = \rho_0 e_1 \rho_2 e_2 \rho_3 \ldots \rho_j e_j$ and each of the paths $\rho_0, \ldots, \rho_j$ is contained in $G^-$. We will show that all these paths can be found using $O(n)$ queries to $Q$ and $O(n)$ path reporting queries to $G$ that will report paths of total length $O(n)$.

More specifically, we will first partition the graph $G^-$ so that each of the paths $\rho_0, \ldots, \rho_j$ can be searched for in a separate disjoint region of the graph $G$. The following lemma describes such a partition in an even more general setting that will prove useful when reporting a reachability tree instead of a path.

**Lemma 6.2.** Let $E^+ = \{u_i v_j : 1 \leq i \leq k\}$, where $k \leq L$. Let $s \in V$ and $V_0 := s$. Let $V_0$ be the set of vertices reachable from $s$ in $G^- \cup E^+$. In $O(n(P + n)\text{-time})$ one can partition $V_0$ into possibly empty and pairwise disjoint subsets $V_{s,0}, \ldots, V_{s,k}$ such that for any $i$ and $z \in V_{s,i}$, every $v_i \to z$ path in $G^-$ is fully contained in $V_{s,z}$. Moreover, let $J = \{j : V_{s,j} \neq \emptyset\}$. For every $i \in J \setminus \{0\}$, there is a parent $p(i) \in J \setminus \{i\}$ such that $u_i \in V_{s,p(i)}$ and the edges in $\{p(j) : j \in J \setminus \{0\}\}$ form an out-tree on $J$ with root at $0$.

**Proof.** We build the sets using the following procedure which gradually extends the set $V_0$ of vertices reachable from $s$ in $G^- \cup E^+$ and inserts new elements to $J$.

We start with $J = \{0\}$. We set $V_{s,0}$ to be the vertices that $s$ can reach in $G^-$. Note that those can be computed using $O(n)$ queries to $Q$. We initialize $V_0$ to $V_{s,0}$.

Next, while for some $u_i v_j \in E^+$ we have $u_i \in V_0$ and $v_j \not\in V_0$, we set $V_{s,j}$ to be the vertices reachable from $v_j$ in $G^-$ minus those already in $V_0$. Moreover, we add $i$ to the set $J$. Note that since $s \in R$ in the data structure $D$, $V_{s,i}$ can be computed in $O(n)$ time by simply reading the vertices reachable from $v_i$ from $D$. Finally, if $u_i \in V_{s,j}$, then we set the parent $p(i)$ of $i$ to $j$ and add $v_i$ to $V_0$. Observe that as we have $v_i \in V_0$ afterwards, this step is performed at most $|E^+| \leq F$ times. As a result, all the steps take $O(nP)$ time in total.

Since the parent $p(i)$ of each added $i$ to $J$ is set to an element of $J$ that was added to $J$ before $i$, the edges $(p(i))_{i \in J}$ indeed form an out-tree $T$ rooted at the first element added to $J$, i.e., $0$, as desired.

By the construction, it is clear that the sets $V_{s,i}, i \in J$, are pairwise disjoint. Let us now prove that their union $V_0$ indeed contains precisely the vertices reachable from $s$ in $G^- \cup E^+$. To prove that each vertex ever put in some $V_{s,i}$ is indeed reachable from $s$ one can proceed by induction on the depth of $i$ in the tree $T$. For $i = 0$ this is clear since $V_{s,0}$ contains precisely the vertices reachable from $s$ in $G^- \subseteq G^- \cup E^+$. If $i \geq 0$, then by the inductive hypothesis, every vertex in $V_{s,p(i)}$, in particular $u_i$, is reachable from $s$ in $G^- \cup E^+$. So $v_j$ is reachable as well since $u_i v_j \in E^+$. But every vertex in $V_{s,j}$ is reachable from $v_j$ in $G^-$, so it is also reachable from $s$ in $G^- \cup E^+$.

Now suppose that $s$ can reach some $t \in V$ in $G^- \cup E^+$. Let $h$ be the minimum possible number of edges from $E^+$ on a $s \to t$ path in $G^- \cup E^+$. We prove that $t \in V_0$ by induction on $h$. If $h = 0$, this means that $t$ is reachable from $s$ in $G^-$, and as a result we have $t \in V_{s,0}$. If $h \geq 1$, then let $P$ be some $s \to t$ path in $G^- \cup E^+$ with exactly $h$ edges from $E^+$ and suppose $e = u_j v_j$ is the edge of $E^+$ that appears on $P$ last. Let us first observe that $v_j$ will be put, at some point, to some set $V_{s,i}$, possibly with $i = j$. Indeed, there exists an $s \to u_j$ path in $G^- \cup E^+$ with less than $h$ edges in $E^+$. So, by the inductive hypothesis, we have $u_j \in V_0$. As a result, the main loop of the algorithm constructing the sets $V_{s,j}$ will ensure that $v_j \in V_{s,j}$ for some $j$ as well. But there is a path $v_j \to t$ in $G^-$, so whenever $v_j$ is included in some $V_{s,j}$, the construction ensures that all vertices reachable from $v_j$ (in particular $t$) in $G^-$ are included in $V_0$ as well.

It remains to prove that if $z \in V_{s,j}$, then every $v_i \to z$ path in $G^-$ contains vertices of $V_{s,j}$ exclusively. Suppose this is not the case and there exists a path $v_i \to w \to z$, where $w \in V_{s,j}$ for $j \neq i$. Note that $j$ could not be added to $J$ later than $i$ because when $V_{s,i}$ is built, all vertices reachable from $v_j$ (in particular $w$) that were not in $V_0$ before are put into $V_{s,i}$. But if $j$ was added to $J$ earlier than $i$, then $w$ is reachable from $v_j$ in $G^-$ and so is $z$. So $z$ should be put into $V_0$ at the time of insertion of $j$ into $J$ (or earlier), a contradiction. $\square$

We will also need the following generalization of the path-finding algorithm of Section 3.2 analyzed in Lemma 3.4.

**Lemma 6.3.** Let $s, t \in V$ and let $W \subseteq V$. Suppose every $s \to t$ path in $G^-$ lies within the subgraph $G^-\{W\}$. Let $S_W = \{s : s \in S : s \in W\}$, where $S$ are the strongly connected components of $G^-$. Let $S_{s,j}$ be the SCCs of $G^-$ containing $s$ and $t$ respectively. Given the relative topological order of $S_W$ within the condensation $G^-/S$, one can find an $s_T \to t_T$ path in $G^-/S$ in $O(|W| \cdot n^P)$ time.

**Proof.** Recall the recursive algorithm of Section 3.2 that worked for acyclic graphs. Given a source $s$ and a target $t$, the algorithm searched for the topologically earliest vertex $x$ such that (1) there is an edge $sx$, and (2) there exists an $x \to t$ path.

We apply the same algorithm to the acyclic graph $G^\{\setminus W\}/S_W$. We first find the components $G^-\{\setminus W\}/S_W$ in $O(|W|)$ time using the explicitly stored mapping from vertices to SCCs, and their relative topological order using the labels $\pi$ from Lemma 6.1. Since the condensation $G^-/S$ is explicitly maintained (Lemma 6.1) along with $S$, for any $x \in S$, we can query for the existence of an edge $S_X$ in $G^-\{\setminus W\}/S_W$ (as required by (1)) in constant time. Note that a path between two vertices $X, Y$ of $G^-/S$ exists if and only if a path between arbitrary $x \in X, y \in Y$ exists in $G^-$. As a result, we can handle (2) using a simple query to the data structure $Q$.

Recall that the algorithm of Section 3.2 run in time proportional to the number of the graph’s vertices times $O(n^P)$. So in our case, the algorithm takes $O(|V| \cdot n^P) = O(|W| \cdot n^P)$ time to complete. $\square$

Let us now describe the algorithm reporting an $s \to t$ path more formally. In the first step, we compute a partition $V_{s,0}, \ldots, V_{s,k}$ of $V_{s,0}$, along with the set $J$ and the tree structure on it in $O(n^P + nP)$ time. If $t \not\in \bigcup_{s \in J} V_{s,j}$, it is not reachable from $s$ in $G$, so we are done. Otherwise, suppose $t \in V_{s,j}$, and let $0 = y_0, \ldots, y_l = j$ be the ancestors of $j$ (including $j$) in the tree $J$, furthest to nearest. 

\footnote{Technically speaking, such an efficient access requires the edges of the condensation maintained by Lemma 6.1 also be stored in a hash table. Doing this introduces no additional asymptotic overhead.}
Let the edges $u_i v_i \in E'$, $i = 1, \ldots, k$, be defined as in Lemma 6.2. By Lemma 6.2, there exists an $s \rightarrow t$ path $P = P_0 e_0 v_0 \cdot P_1 e_1 v_1 \cdot \cdots \cdot P_{t-1} e_{t-1} v_{t-1} \cdot P_t$ in $G'$ where $u_0 v_0 \in E'$, $v_i := s$ and $u_{t+1} = t$, such that every $v_i \rightarrow u_{t+1}$ path in $G^-$ (for $i = 0, \ldots, t$), in particular $P_t$, is fully contained in $G^-[V_{\Psi,G}].$

The above reduces our problem to computing, for all $i = 0, \ldots, t$, some $p \rightarrow q$ path in $G^-\{V_{\Psi,G}\}$, where $p := v_t$ and $q := u_{t+1}$. We accomplish this for each $i$ separately. Let $X, Y \in S$ be such that $p \in X$ and $q \in Y$. Since every $p \rightarrow q$ path in $G'$ lies in $G^-\{V_{\Psi,G}\}$, by Lemma 6.3 we can find an $X \rightarrow Y$ path $P'$ in $G^-/S$ in $O(\sum_{i=1}^k |V_i| \cdot n^p)$ time. We now discuss how such a path can be lifted to an actual $p \rightarrow q$ path in $G$. From the auxiliary data structures of Section 6.1 we can obtain, for each edge $e'$ of $P'$ connecting some $A, B \in S$, an edge $e = e \in E(G^-)$ such that $e$ is in $A$ and $B \in B$. As a result, from $P'$ we obtain a sequence of edges $a_i b_i$, $i = 1, \ldots, g$, such that for each $i = 0, \ldots, g + 1$, $(1)$ $b_i = a_{i+1}$ and $(2)$ each edge $b_i \in$ the same strongly connected component of $G^-$. We can convert this sequence into a $p \rightarrow q$ path in $G'$, we query the data structure $C$ for the respective $b_i \rightarrow a_i$ paths. Note that all the requested paths lie within single but distinct strongly connected components of $G^-$. As the paths returned by $C$ are simple and pairwise disjoint, and all are fully contained within $G^-\{V_{\Psi,G}\}$, the total time needed to compute them all will be $O(\sum_{i=1}^k |V_i| \cdot n^p)$.

We conclude that the total time needed to report an $s \rightarrow t$ path is $O(n\rho \cdot \sum_{i=1}^k |V_i|)$, which, since the sets $V_{\Psi,G}$ are pairwise disjoint, is $O(n^{1+\rho})$. We stress that the above algorithm is deterministic, modulo the operation of data structures $D$ and $Q$, whose output is also unique with high probability. As a result, we can report paths against an adaptive adversary if and only if the paths within strongly connected components of $G^-$ can be reported against an adaptive adversary.

Finally, note that a new data structure $C$ is initialized once per every $F$ insertions. Thus, the amortized update time can be bounded as $O(n^{1+\rho} + nF + T(n, m)/F)$. To obtain Theorem 1.2, it is enough to set $F = \sqrt{T(n, m)/n}$.

**Theorem 1.3.** Let $C$ be a decremental data structure with total update time $T(n, m) = \Omega(m + n)$ (1) maintaining strongly connected components $S$ of $G$ explicitly and (2) supporting queries reporting, for a chosen strongly connected component $S \subseteq S$, (a possibly sparser) strongly connected subgraph $Z \subseteq G[S]$ with $V(Z) = S$ in $O(\sum_{i=1}^k n^{1+\rho}/n^p)$ time. Then, there exists a fully dynamic data structure supporting single-edge updates and single-source-reachability tree reporting queries with amortized update time and worst-case query time of $O(\sqrt{T(n, m) - n + n^{3+\rho}/2})$.

If the data structure $C$ works against an adaptive adversary, so does the fully dynamic reachability tree reporting data structure.

Our overall strategy will be to generalize the approach of Section 3.3 to general directed graphs. Unfortunately, Observation 3.6 does not hold for general graphs: given some source $s \in V$, choosing an arbitrary incoming edge (e.g., that with the minimum label) from each vertex reachable from $s$ might lead to a disconnected (in the undirected sense) graph containing cycles.

In order to deal with this problem, we could apply Observation 3.6 to the condensation $G/S$ (with the source vertex set to the SCC containing $s$). The obtained out-tree $T'$ in $G/S$ could be then extended to an out-tree $T$ in $G$ spanning the vertices reachable from $s$ in two steps. First, expand each vertex of $T'$ (i.e., a strongly connected component $S$ reachable from $s$) into a sparse strongly connected subgraph of $S$, e.g., consisting of a pair of reachability trees from/to a single vertex in $S$. Then, compute a single-source-reachability tree from $s$ in the obtained subgraph of $G$ using any graph search procedure in $O(n)$ time.

However, it is not clear how to efficiently operate on the condensation $G/S$ (which is a DAG) of a decremental graph $G$ using the algebraic data structures of Theorem 3.1 and Lemma 5.1. Recall that when the strongly connected components split, the condensation undergoes vertex splits, and each vertex split (revealed online) might require $\Theta(n)$ edges changing endpoints.

We will nevertheless avoid this problem. Let us start with the following technical lemma.

**Lemma 7.1.** Let $\pi : V \rightarrow [1, n]$ be such that: (a) if $x, y \in V$ lie in distinct strongly connected components of $G$ and there exists an $x \rightarrow y$ path in $G$, then $\pi(x) < \pi(y)$, and (b) if $x$ and $y$ are strongly connected, then $\pi(x) = \pi(y)$.

Let $s \in V$ and let $V_{\Psi,G} \subseteq V$ be the vertices reachable from $s$ in $G$. Moreover, let $S_s$ be the set of strongly connected components reachable from $s$ in $G$. Let $S_s'$ be an arbitrary subset of $S_s$.

Suppose $H \subseteq G[V_{\Psi,G}]$ with $V(H) = V_s$ satisfies the following:

1. For each $t \in V_s \setminus \{s\}$, $H$ contains an edge $vt$ such that $\pi(v) < \pi(t)$ is minimal possible among $v \in \bigcup_{S \in S_s'} S_s'$,

2. For all $(x, y) \in (S_s \setminus S_s') \times S_s'$, $H$ contains an edge $xy$ in $E(G) \cap (X \times Y)$, if such an edge exists.

3. For each $S \in S_s$, $H[S]$ is strongly connected.

Then, all vertices of $V_s$ are reachable from $s$ in $H$.

**Proof.** By item (3), it is enough to prove that in the condensation $H/S_s$, every $Y \in S$ is reachable from the component $S'$ containing $s$. Since $H/S_s$ is a DAG, by Observation 3.6, we only need to prove that every $Y \in S \subseteq S'$ has an incoming edge in that graph. Note that $Y$ is reachable from $S'$ in $G[V_s]/S_s$. Consequently, $Y$ has an incoming edge from some $X \in S_s$, $X \neq Y$, in $G[V_s]/S_s$. If
there exists such an $X$ that $X \not\subseteq S_i'$, then by item (2), we also have an edge $xy \in E(G) \cap (X \times Y)$ in $H$. So, then in $H/S_i$ indeed $Y$ has an incoming edge.

So assume that for all $X$ such that $Y$ has an incoming edge $XY$ in $G[V_i]/S_i, X \subseteq S_i'$. There exists such an edge $xy \in E(G) \cap (X \times Y)$ and we have $x \in (\mathcal{S}_i' \setminus \{y\})$ and $\pi(x) < \pi(y)$. Consequently, by item (1), $H$ contains an edge $vy$ with $v \in \bigcup \mathcal{S}_i'$ such that $\pi(v) \leq \pi(x) < \pi(y)$. By $\pi(v) < \pi(y)$, $v$ is in a different strongly connected component than $y$ and $v$ is reachable from $s$ as well. This proves that in $H/S_i$ the vertex $Y$ indeed has an incoming edge. □

Generally speaking, the query procedure will construct a subgraph $\mathcal{H} \subseteq G$ of moderate size satisfying the requirements of Lemma 7.1 and then produce a single-source reachability tree from $s$ in $H$ (and thus also in $G$) in $O(|E(\mathcal{H})|)$ time. To allow constructing such a subgraph we will require a number of components.

We reuse much of the notation and developments from Section 6. Again, the algorithm will operate in phases spanning $E$'s edge insertions. The data structure $C$, accompanied with the auxiliary data structures of Lemma 6.1, is reinitialized at the beginning of each phase. Recall that $E^*$ denotes the set of edges inserted in the current phase, and $G$ is the graph from the beginning of the phase minus the edges deleted in the current phase, so that at all times the current graph $G$ satisfies $G = G^* \cup E^*$. Moreover, similarly as in Section 6, we use a pair of data structures $Q, D$ of Theorem 3.1 and Lemma 5.1 respectively. Both these data structures maintain the graph $G^*$ when a phase proceeds, and are passed the edge insertions of $E^*$ as late as when a new phase is started.

We will maintain a labeling $\pi : V \rightarrow [1, n]$ based on the topological labels $\pi$ of the strongly connected components $\mathcal{S}$ of $G^*$, as given by Lemma 6.1. We set $\pi(v) = \pi(S)$ if $v \in S \in \mathcal{S}$.

Finally, for $\Lambda$ to be chosen later, we maintain $q = \lceil n/\Lambda \rceil$ data structures $D_1, \ldots, D_q$ of Theorem 3.1. Recall that in Section 3.3, $D_i$ was responsible for detecting paths whose penultimate vertex had its label in $[(i - 1) \cdot \Lambda + 1, i \cdot \Lambda]$: this was possible since the $V \times V^{\prime}$ layer of the underlying graph $G_i$ contained only edges $uv'$ where $v \in [(i - 1) \cdot \Lambda + 1, i \cdot \Lambda]$. Here, we proceed a bit differently: the data structure $D_i$ will be responsible for detecting paths whose penultimate vertex’s topological label $\pi(y)$ will surely remain in the interval $[(i - 1) \cdot \Lambda + 1, i \cdot \Lambda]$ till the end of the current phase. In particular, not all possible penultimate vertices will be assigned to one of $D_i$. For example, it may happen that for some vertex, the range of its possible topological labels after the future updates of the phase is too large.

More formally, for each $i = 1, \ldots, q$, we maintain a growing subset $Y_i \subseteq V$ such that the underlying graph $G_i = (V \cup V' \cup V''', E')$ of $D_i$ satisfies $u \in E'$ and $v'' \in E'$ for each $uv \in E(G^*)$, and moreover $uv' \in E'$ iff $u \in Y_i$. $Y_i$ will contain the vertices whose topological label will stay in the interval $[(i - 1) \cdot \Lambda + 1, i \cdot \Lambda]$ till the end of the current phase. By proceeding identically as in the proof of Lemma 3.7, one shows that a $u \rightarrow v''$ path exists in $G_i$ iff there exists a $u \rightarrow v$ path in $G''$ whose penultimate vertex is in $Y_i$. Let $S_i$ contain the subset of the SCCs of $S$ of $G^*$ contained in $G^{-}\{V_{i-1}\}$. For each $Z \in S_i$ that is special, and all other $X \in S_i \setminus \{Z\}$, if there exists an edge $XZ$ in $G^{-}\{V_{i-1}\}/S_i$ we include in $H_i$ the edge $e_{XZ} \in E(G^*)$, as defined and maintained by Lemma 6.1.

For each vertex $t \in V_{i-1} \setminus \{v_i\}$, we find the smallest $j$ (if one exists) such that a path $v_i \rightarrow t^{j'}$ exists in $G_j$. Note that such a $j$ can be found using at most $q = \lceil n/\Lambda \rceil$ reachability queries to the data structures $D_1, \ldots, D_q$. The cost of these queries is clearly $O((n/\Lambda) \cdot n^p)$. We then add to $H_i$ all the $t$’s incoming edges (in $G^{-}\{V_{i-1}\}$) from the vertices of the set $Y_j$, whose size is at most $\Lambda$.

Finally, for each $Z \in S$ (regardless of whether it is special or not), using the assumed decremental strongly connected components data structure $C$ maintaining $G^*$, we compute a strongly connected subgraph $Z''$ of $G^{-}[Z]$ with $V(Z'') = Z \cup O(|Z| \cdot n^{(p+2)/2})$ time and add it to $H_i$.}

1654
The following lemma proves the above construction correct.

**Lemma 7.3.** Let \( V_{s,l} \neq \emptyset \). Then, in the subgraph \( H_l \), \( v_i \) can reach every vertex of \( V_{s,l} \).

**Proof.** We prove that \( H_l \) satisfies the properties (1)-(3) from Lemma 7.1 applied to \( G = G - [V_{s,l}] \), \( s = v_i \), \( H = H_l \), \( S_i \subseteq S_l \), and \( S'_i \) equal to the non-special strongly connected components in \( S_l \). Then, the lemma will follow by Lemma 7.1.

Let \( t \in V_{s,l} \backslash \{ v_i \} \). For contradiction, suppose that property (1) is not satisfied, i.e., \( H_l \) does not contain an edge \( uv \) such that \( u \) is in a non-special strongly connected component \( Z \subseteq S_l \) and \( \pi(u) \) is minimal. Since \( Z \) is non-special, by the maintained invariant we have \( Z \subseteq Y_l \) for some \( l \). Moreover, since \( v \) is reachable from \( v_i \) in \( G - [V_{s,l}] \), there exists a path \( v_i \rightarrow t \rightarrow u \) in \( G - [V_{s,l}] \), whose penultimate vertex is \( u \). As a result, there exists a path \( v_i \rightarrow t'' \) in \( G_l \), i.e., the query about a \( v_i \rightarrow t'' \) path in \( D_l \) returns true. So, the minimal \( j \) such that a path \( v_i \rightarrow t'' \) exists in \( G_l \) satisfies \( j \leq l \). If \( j = l \), then all edges \( wt \in E(G_l) \) with \( w \in Y_l \), in particular \( x \), are added to \( H_l \), which contradicts our assumption. On the other hand, if \( j < l \), then there exists a \( u_j \rightarrow t \) path in \( G - [V_{s,l}] \), whose penultimate vertex \( w \) lies in \( Y_j \). But since \( j < l \), \( w \in E(G - [V_{s,l}]) \), \( \pi(w) < \pi(u) \) and \( w \) lies in a non-special strongly connected component as well, so \( \pi(u) \) was not minimal — a contradiction.

Property (2) from Lemma 7.1 follows easily by construction: for each special strongly connected component \( X \), we add to \( H_l \) an outgoing edge to every other component of \( G - [V_{s,l}] \) if it exists.

Property (3) also follows trivially by construction. □

**Lemma 7.4.** The total time needed to compute the subgraph \( H \) is \( O(n^{2p}/\Delta + \Delta n + n^{(3p)/2} + nf) \).

**Proof.** First, recall that computing the partition \( V_{s,0}, \ldots, V_{s,k} \) takes \( O(n^{1+p} + nF) \) time.

Let \( r_i \) denote the number of special strongly connected components in \( S_i \). Then, the total time needed to compute \( H \) is:

\[
O\left(\frac{n}{\Delta} \cdot n^p + \Delta \right) + \left| V_{s,i} \right| \cdot n^{(1+p)/2}
\]

Summing through all \( i = 0, \ldots, k \), and taking into account that there are \( O(n/\Delta) \) special strongly connected components in total (see Observation 7.2), we bound the time as follows:

\[
O\left(\frac{n^{2p}}{\Delta} + n^{(3p)/2} + \Delta \cdot \sum_{i=1}^{k} \left| V_{s,i} \right| + \frac{k}{\Delta} \cdot \left| V_{s,i} \right| \cdot r_i\right)
\]

\[
= O\left(\frac{n^{2p}}{\Delta} + \Delta n + n^{(3p)/2} + \frac{n^2}{\Delta}\right). \quad \Box
\]

Note that the time bound from Lemma 7.4 also bounds the size of \( H \), and thus the query time.

**Lemma 7.5.** The amortized update time is:

\[
O\left(T(n, m) + n^{2p}/\Delta + n^{2p}/F\right).
\]

**Proof.** Initializing the data structure \( C \) along with the auxiliary data of Lemma 6.1 happens once per \( F \) insertions, this gives \( O(T(n, m)/F) \) amortized time per update. Since each edge update is passed to all the data structures \( D, D_1, \ldots, D_H \), which gives \( O(n^{2p}/\Delta) \) amortized cost per update. As discussed earlier, handling changes to the sets \( Y_i \) (and also their initialization) in data structures \( D_1, \ldots, D_H \) costs \( O(n^{2p}/F) \) time per phase, which gives \( O(n^{2p}/F) \) amortized time per update. □

To balance the update and query time, we set \( \Delta = F = \max\left(\sqrt{\frac{T(n, m)}{n}}, \sqrt{n^{(1+p)/2}}\right) \). Again, all the components except the subgraphs certifying strong connectivity inside the components of \( S \) are deterministic, so the algorithm works against an adaptive adversary if and only if these subgraphs are reported against an adaptive adversary.

**Corollary 7.6.** There exist data structures supporting single-source reachability tree queries on a fully dynamic graph with the following amortized update time and worst-case query time bounds:

1. \( O(m^{1/2}, n^{5/6+o(1)} + n^{(3p)/2}) = O(n^{1.834}) \) against an adaptive adversary.
2. \( O(n^{(3p)/2}) = O(n^{1.765}) \) against an oblivious adversary.

**Proof.** For the former item, we note that the deterministic decremental strongly connected components data structure with total update time \( m^{1/3+o(1)} \) [9] is actually capable of finding the first edge on some simple \( s \rightarrow t \) path, where \( s \) and \( t \) are strongly connected, in \( n^{(1+p)^4} \) worst-case time. As a result, for any strongly connected component \( S \), we can find a sparse strongly connected subgraph of \( G[S] \) spanning vertices \( S \) in \( |S| \cdot n^{(1+p)^4} \) time as follows. Pick some root \( r \) in \( S \). The subgraph will consist of an out-tree from \( r \), and an in-tree to \( r \), both within \( G[S] \). Let us focus on finding an in-tree \( T_S \), since an out-tree can be found by proceeding identically on the reverse graph. Initially, \( T_S = (\emptyset, \emptyset) \). While there exists some \( x \in S \setminus V(T_S) \), pick an arbitrary such \( x \). Issue a query about an \( x \rightarrow r \) path in \( G[S] \) to \( C \) — the query will produce, one by one, the subsequent edges \( e_1 = x_1 y_1, e_2 = x_2 y_2, \ldots \) of some \( x \rightarrow r \) path, in each \( n^{(1+p)/2} \) worst-case time. We stop when the algorithm outputs, for the first time, an edge such \( e_j \) such that \( y_j \in V(T_S) \). Observe that then the algorithm terminates in \( j \cdot n^{(1+p)} \) time. Moreover, by connecting the path \( e_1, \ldots, e_j \) to the tree \( T_S \), we make \( T_S \) span \( j \) more vertices. As a result, \( V(T_S) \) will finally grow to size \( |S| \) and thus constructing \( T_S \) will require \( |S| \cdot n^{(1+p)/2} \), which is \( O(|S| \cdot n^{(1+p)/2}) \), as desired.

Consider item 2. The near-optimal data structure of [11] (Theorem 5.2) actually maintains, for each strongly connected component \( S \) in \( S \), a spanning in-tree and a spanning out-tree. Combined, those can serve as a sparse subgraph of \( G[S] \) spanning \( S \) that is strongly connected. Unfortunately, these maintained trees might reveal the random choices made by the data structure. So the tree reporting in this case works against an oblivious adversary only. □

**REFERENCES**

[1] Amir Abboud and Virginia Vassilevska Williams. 2014. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In 55th IEEE Annual Symposium on Foundations of Computer Science. FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014. IEEE Computer Society, 434–443. https://doi.org/10.1109/FOCS.2014.53

[2] Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. 1992. Witnesses for Boolean Matrix Multiplication and for Shortest Paths. In 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, Pennsylvania, USA, 24-27 October 1992 IEEE Computer Society. 417–426. https://doi.org/10.1109/SFCS.1992.267748

[3] Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. 1991. Incremental Algorithms for Minimal Length Paths. J. Algorithms 12, 4 (1991), 615–638. https://doi.org/10.1016/0196-6774(91)90036-X
