Fully Dynamic Shortest Paths and Reachability in Sparse Digraphs

Adam Karczmarz
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Piotr Sankowski
University of Warsaw, Poland
IDEAS NCBR, Warsaw, Poland

Abstract

We study the exact fully dynamic shortest paths problem. For real-weighted directed graphs, we show a deterministic fully dynamic data structure with $\widetilde{O}(mn^{4/5})$ worst-case update time processing arbitrary s,t-distance queries in $\widetilde{O}(n^{4/5})$ time. This constitutes the first non-trivial update/query tradeoff for this problem in the regime of *sparse weighted* directed graphs.

Moreover, we give a Monte Carlo randomized fully dynamic *reachability* data structure processing single-edge updates in $\widetilde{O}(n\sqrt{m})$ worst-case time and queries in $O(\sqrt{m})$ time. For sparse digraphs, such a tradeoff has only been previously described with amortized update time [Roditty and Zwick, SIAM J. Comp. 2008].

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms; Theory of computation → Shortest paths

Keywords and phrases dynamic shortest paths, dynamic reachability, dynamic transitive closure

Digital Object Identifier 10.4230/LIPIcs.ICALP.2023.84

Category Track A: Algorithms, Complexity and Games

Funding *Adam Karczmarz*: Partially supported by the ERC CoG grant TUgbOAT no 772346.
Piotr Sankowski: Partially supported by the ERC CoG grant TUgbOAT no 772346 and National Science Center (NCN) grant no. 2020/37/B/ST6/04179.

1 Introduction

Computing all-pairs shortest paths (APSP) is among the most fundamental algorithmic problems on directed graphs. This classical problem is often generalized into a data structure "oracle" variant: given a graph G, preprocess G so that efficient point-to-point distance or shortest paths queries are supported. Computing APSP can be viewed as an extreme solution to the oracle variant; if one precomputes the answers to all the n^2 possible queries in $\widetilde{O}(nm)$ time, the queries can be answered in constant time. The other extreme solution is to not preprocess G at all and run near-linear-time Dijkstra's algorithm upon each query. Interestingly, for general directed weighted graphs, no other tradeoffs for the exact oracle variant of static APSP beyond these trivial ones are known.

In this paper, we consider the exact APSP problem, and its easier relative *all-pairs reachability* (or, in other words, *transitive closure*), in the *fully dynamic* setting, where the input graph G evolves by both edge insertions and deletions.

1.1 Prior work

There has been extensive previous work on APSP and transitive closure in the fully dynamic setting. Notably, Demetrescu and Italiano [16] showed that APSP in a real-weighted digraph can be maintained deterministically in $\widetilde{O}(n^2)$ amortized time per vertex update (changing all edges incident to a single vertex). Thorup [36] later slightly improved and simplified their result. These data structures maintain an explicit distance matrix and the corresponding collection of shortest paths, and thus allow querying distances and shortest paths in optimal time. Similar amortized bounds have been earlier obtained for transitive closure [17, 30, 32] albeit using different combinatorial techniques. Polynomially worse (but nevertheless subcubic) *worst-case* update bounds for real-weighted fully dynamic APSP are also known: randomized $\widetilde{O}(n^{2+2/3})$ [2, 24] and slightly worse deterministic $\widetilde{O}(n^{2+41/61})$ [13].

For dense *unweighted* digraphs, non-trivial fully dynamic data structures for all-pairs reachability and APSP can be obtained using algebraic techniques. Via a reduction to dynamic matrix inverse, Sankowski [35] obtained $O(n^2)$ *worst-case* update bound for explicitly maintaining the transitive closure, and also gave update/query tradeoffs. In particular, he showed a reachability data structure with subquadratic $O(n^{1.529})$ update time and sublinear $O(n^{0.529})$ query time. Using the same general algebraic framework, van den Brand, Nanongkai, and Saranurak [40] showed $O(n^{1.407})$ worst-case update bound for st-reachability (that is, fixed single-pair reachability), whereas van den Brand, Forster, and Nazari [38] gave an $O(n^{1.704})$ worst-case update bound for maintaining exact st-distance in unweighted digraphs.[1] That framework, however, inherently leads to Monte Carlo randomized solutions and does not generally allow reporting (shortest) paths within the stated query bounds.[2]

Interestingly, neither the known fully dynamic APSP data structures for *real-weighted* digraphs (or even for integer weights between 1 and n) nor the algebraic data structures tailored to dense graphs yield any improvement over the extreme recompute-from-scratch approaches for *sparse* graphs with $m = \widetilde{O}(n)$. This is especially unfortunate as such graphs are ubiquitous in real-world applications. Indeed, for $m = \widetilde{O}(n)$, recomputing APSP from scratch takes $\widetilde{O}(n^2)$ worst-case update time and $O(1)$ query time (which matches the amortized bound in [17, 36]), whereas naively running Dijkstra's algorithm upon query costs $\widetilde{O}(n)$ time (which already improves upon the update bound of the algebraic st-distance data structure of [38]). The only non-trivial fully dynamic APSP data structure in the sparse regime has been described by Roditty and Zwick [34]. Their randomized data structure has $\widetilde{O}(m\sqrt{n})$ amortized update time and $O(n^{3/4})$ query time. Unfortunately, it works only for *unweighted digraphs*. To the best of our knowledge, no non-trivial update/query tradeoffs for fully dynamic APSP in sparse weighted digraphs have been described to date. A step towards this direction has been made by Karczmarz [27] who showed that some *fixed* – in a crucial way – m distance pairs can be maintained in $\widetilde{O}(mn^{2/3})$ worst-case time per update.

For the simpler fully dynamic reachability problem, the $O(n^{1.529})$ update time and $O(n^{0.529})$ query time algebraic tradeoff of [35] is already non-trivial for all graph densities. However, specifically for sparse graphs, a deterministic and combinatorial tradeoff of Roditty and Zwick [33] is more efficient; they showed a data structure with $O(m\sqrt{n})$ amortized update time and $O(\sqrt{n})$ query time. Moreover, the data structure of [35] requires fast matrix multiplication algorithms [3, 21] and these are considered impractical. That being said, the downside of [33] is that the update bound holds only in the amortized sense.

[1] The single-pair data structures [40, 38] can be easily extended to support arbitrary-pair queries. Then, the query time matches the update time.

[2] As shown quite recently, reporting (shortest) paths in subquadratic time can be possible via a combination of algebraic and combinatorial techniques [8, 28]. However, this comes with a polynomial time overhead.

1.2 Our results

Dynamic shortest paths. Most importantly, we show the first fully dynamic APSP data structure with non-trivial update and query bounds for *sparse weighted* digraphs.

▶ **Theorem 1.** *Let G be a real-weighted directed graph. There exists a deterministic data structure maintaining G under fully dynamic vertex updates and answering arbitrary s,t-distance queries with $\widetilde{O}(mn^{4/5})$ worst-case update time and $\widetilde{O}(n^{4/5})$ query time and using $\widetilde{O}(n^2)$ space. The queries are supported only when G has no negative cycles. After answering a distance query, some corresponding shortest path $P = s \to t$ can be reported in $O(|P|)$ time.*

Compared to the data structure of Roditty and Zwick [34] for the unweighted case, our obtained update/query bounds are polynomially higher. However, our data structure has some very significant advantages. It is deterministic, handles real-edge-weighted graphs (possibly with negative edge weights and negative cycles), and the update time bounds holds in the worst case, as opposed to only in the amortized sense in [34]. Moreover, if path reporting is required, then the bounds in [34] hold only against an oblivious adversary. We also remark that a slightly more efficient variant of Theorem 1, with $\widetilde{O}(mn^{3/4})$ worst-case update time and $\widetilde{O}(n^{3/4})$ query time, can be obtained for the unweighted case.

The near-quadratic space requirement in Theorem 1 is clearly undesirable in the sparse setting, but also applies to all the other known fully dynamic reachability and shortest paths data structures. Moreover, this phenomenon is not specific to the dynamic setting. To the best of our knowledge, even for the *static* transitive closure problem, it is not known whether one can preprocess a general sparse directed graph into a data structure of size $O(n^{2-\epsilon})$ supporting arbitrary reachability queries in $O(n^{1-\epsilon})$ time.[3]

Dynamic reachability. For fully dynamic all-pairs reachability in sparse digraphs, we show that the amortized update bound of Roditty and Zwick [33] can also hold in the worst case.

▶ **Theorem 2.** *Let G be a directed graph. Let $t \in [1, \sqrt{m}]$. There exist a Monte Carlo randomized data structure maintaining G subject to fully dynamic single-edge updates with $\widetilde{O}(mn/t)$ worst-case update time and supporting arbitrary-pair reachability queries in $O(t)$ time. The answers produced are correct with high probability*[4]*.*

Note that for $t = \sqrt{m}$, Theorem 2 yields $O(n^{2-\epsilon})$ update time and $O(n^{1-\epsilon})$ for some $\epsilon > 0$ for all but dense graphs. The data structure of Roditty and Zwick [33], on the other hand, has amortized update time at least $\Theta(m\sqrt{n})$, which is $o(n^2)$ only if $m = o(n^{3/2})$. However, the downsides of Theorem 2 compared to [33] are: supporting more restricted single edge (as opposed to vertex-) updates, using randomization, and not being able to report the underlying path efficiently.

Our data structure should also be compared with the $O(n^{1.529})/O(n^{0.529})$ worst-case update/query bounds obtained in [35]. Theorem 2 gives polynomially better bounds for very sparse graphs, with $m = O(n^{1.057})$. Moreover, although it is also algebraic in nature, it does not rely on fast matrix multiplication [3, 21], thus avoiding this potential practical efficiency bottleneck.

[3] Such a tradeoff is possible, for example, if the graph has a sublinear *minimum path cover*, see, e.g., [31].
[4] That is, with probability at least $1 - 1/n^c$, where the constant $c \geq 1$ can be set arbitrarily. We will also use the standard abbreviation w.h.p.

1.3 Technical overview

Shortest paths. In order to obtain a basic randomized variant of Theorem 1, we combine ideas from the known data structures for fully dynamic APSP with subcubic worst-case update bound [2, 24, 27]. These data structures all build upon hitting set arguments (dating back to the work of Ullman and Yannakakis [37]) yielding a sublinear $\widetilde{O}(n/h)$-sized set of vertices of the graph that lie on the shortest paths whose number of edges (*hops*) is at least $h = \operatorname{poly} n$. With this in hand, the main challenge is to recompute pairwise *small-hop* shortest paths, i.e., those with at most h hops, under edge deletions. As usual, edge insertions are rather easy to handle since the potential new paths created by insertions necessarily pass through the inserted edges' endpoints.

For efficient recomputation of small-hop paths, our data structure once in a while chooses a collection Π of n^2 pairwise $\leq h$-hop paths in G, and a set $C \subseteq V$ of *congested vertices* of truly sublinear (in n) size, so that the chosen paths are at least as short as shortest $\leq h$-hop paths in $G - C$ (i.e., the graph G with edges incident to the vertices C removed). The congested vertices are picked in such a way that no individual vertex $v \in V$ appears on the chosen paths too often. As a result, the number of precomputed paths destroyed by a vertex deletion that have to be restored is bounded. This idea is due to Probst Gutenberg and Wulff-Nilsen [24]. However, as opposed to [24], we cannot afford to recompute shortest $\leq h$-hop paths upon update in a hierarchical way which is inherently quadratic in n (albeit advantageous in the case of dense graphs). Instead, recomputation upon deletions is performed using a Dijkstra-like procedure (as in [2]), crucially with the sparsity-aware enhancements of [27] (such as the degree-weighted congestion scheme). These techniques, combined with the standard random hitting set argument [37] are enough to get the stated bounds, albeit Monte Carlo randomized.

Derandomization. Randomization above is only required for the sake of the hitting set argument. Curiously, we do not (and do not know how to) exploit the often-used property that a random hitting set, once sampled, is valid through multiple versions of the evolving graph as long as the adversary is oblivious to the hitting set. Therefore, we may as well sample the hitting set from scratch after each update. This is as opposed to [2, 27], where avoiding that leads to polynomially better bounds. If a fresh hitting set can be used upon each update, the standard derandomization method is to use a folklore greedy algorithm (see Lemma 8) for constructing a minimum hitting set that is $O(\log n)$-approximate, first used in the context of static and dynamic APSP algorithms in [30, 42]. The greedy algorithm runs in linear time in the input size. For constructing a hitting set of explicitly given pairwise $\leq h$-hop paths, this gives an $O(n^2 h)$ time bound per update. This is enough for deterministic variants of [2] and [42]. However, the incurred cost is prohibitive in the sparse case.

Derandomization of our data structure without a polynomial slowdown turns out to be non-trivial and requires some new tools. First, when precomputing $\leq h$-hop paths Π, we construct a hitting set H_0 of those paths in Π that have $\Theta(h)$ hops. When G is subject to deletions, H_0 hits the precomputed paths in Π that are not destroyed as a result of deletions. Hence, in order to lift H_0 into a hitting set after an update, it is enough to extend it so that it hits all the restored paths. If we wanted to run the greedy algorithm on the restored paths, the data structure would suffer from a factor-h polynomial slowdown. This is because the representation of the restored paths (constructed using Dijkstra's algorithm) can be computed more efficiently that their total hop-length and encoded using a collection of shortest paths trees $\mathcal{Z}$. The goal can be thus achieved by finding a hitting set of all $\Theta(h)$-hop root-leaf paths in $\mathcal{Z}$. King [30] gave a variant of the aforementioned deterministic greedy algorithm precisely for this task. The algorithm of [30] runs in $O(\min(Nh, |\mathcal{Z}|n))$ time, where N denotes the total size of trees in $\mathcal{Z}$. While this is optimal when $\mathcal{Z}$ contains $\Theta(n)$ trees of size $\Theta(n)$ (as required in [30]), for small enough N and large enough $|\mathcal{Z}|$, this is not better than the standard greedy algorithm which could also solve the task in $O(Nh)$ time.

We deal with this problem by designing a novel near-optimal deterministic algorithm computing an $\widetilde{O}(n/h)$-sized hitting set of h-hop root-leaf path in a collection of trees that runs in $O(N \log^2 N)$ time independent of h (see Theorem 9). We believe that this algorithm might be of independent interest. The main idea here is to simulate the greedy algorithm only approximately, which enables taking advantage of dynamic tree data structures [4].

Reachability and sparse matrix inverse. Our improved worst-case bounds for fully dynamic reachability in sparse digraphs are obtained via a small change in the subquadratic update-sublinear query tradeoff of [35] based on dynamic matrix inverse. That algorithm once in a while explicitly recomputes the inverse of a certain matrix associated with the graph using fast rectangular matrix multiplication. That inverse encodes the transitive closure of the graph G. We observe that for sparse graphs, it is beneficial to recompute the inverse in a more naive way, entirely from scratch. This is because for large enough finite fields (with more than n^2 elements), it is in fact possible to compute the inverse of a sparse matrix with $m = \widetilde{\Theta}(n)$ non-zero elements in near-optimal $\widetilde{O}(mn)$ time without fast Strassen-style matrix multiplication algorithms (see Theorem 13). This is a relatively easy consequence of the classical work of Kaltofen and Pan [26], and has been, to the best of our knowledge, overlooked and not explicitly stated before. Sparse matrix inversion has been recently viewed (see, e.g., [12, 18]) mainly through the lens of *black-box matrix computations*, i.e., parameterized by the cost $\phi(n)$ of multiplying the input matrix (or its transpose) by a vector. For sparse matrices, we clearly have $\phi(n) = \widetilde{O}(n)$, but the best described bound for sparse matrix inversion in finite fields in that literature seems to be $O(n^{2.214})$ [12]. However, $\phi(n) = \widetilde{O}(n)$ holds for sparse matrices even in a less general so-called *straight-line program* computation model (also called the *algebraic circuit* model) which allows employing powerful tools such as the Baur-Strassen theorem [7].

1.4 Further related work

Exact all-pairs shortest paths in unweighted graphs have been studied also in partially dynamic settings: incremental [5] and decremental [6, 19]. Fully dynamic data structures are also known for $(1+\epsilon)$-approximate distances in weighted directed graphs [9, 39]. A significant research effort has been devoted to finding fully- and partially dynamic (approximate) all-pairs shortest paths data structures for *undirected* graphs, e.g., [10, 14, 15, 20, 38].

Dynamic reachability and shortest paths problems have also been studied from the perspective of conditional lower bounds [1, 23, 25, 34, 40].

2 Preliminaries

We work with directed graphs $G = (V, E)$. We denote by $w_G(e) \in \mathbb{R}$ the weight of an edge $uv = e \in E$. The graph G is called *unweighted* if $w_G(e) = 1$ for all $e \in E$. If the graph whose edge we refer to is clear from the context, we may sometimes skip the subscript and write $w(e)$. For simplicity, we do not allow parallel directed edges between the same endpoints of G, as those with non-minimum weights can be effectively ignored in reachability and shortest paths problems we study. As a result, we sometimes write $w_G(uv)$ or $w(uv)$.

For $u, v \in V$, an $u \to v$ path P in G is formally a sequence of vertices $v_1 \ldots v_k \in V$, where $k \geq 1$, $u = v_1$, $v = v_k$, such that $v_i v_{i+1} \in E$ for all $i = 1, \ldots, k-1$. The hop-length $|P|$ of P equals $k - 1$. The length $\ell(P)$ of P is defined as $\sum_{i=1}^{k-1} w_G(v_i v_{i+1})$. P is a *simple path* if $|V(P)| = |E(P)| + 1$. We sometimes view P as a subgraph of G with vertices $\{v_1, \ldots, v_k\}$ and edges (hops) $\{v_1 v_2, \ldots, v_{k-1} v_k\}$.

For any $k \geq 0$, $\delta_G^k(s,t)$ is the minimum length of an $s \to t$ path in G with at most k hops. A *shortest k-hop-bounded $s \to t$* path in G is an $s \to t$ path with length $\delta_G^k(s,t)$ and at most k hops. We define the s,t-distance $\delta_G(s,t)$ as $\inf_{k \geq 0} \delta_G^k(s,t)$. For $s, t \in V$, we say that t is *reachable* from s in G if there exists an $s \to t$ path in G, that is, $\delta_G(s,t) < \infty$. If $\delta_G(s,t)$ is finite, there exists a simple $s \to t$ path of length $\delta_G(s,t)$. Then, we call any $s \to t$ path of length $\delta_G(s,t)$ a *shortest s,t-path*.

If G contains no negative cycles, then $\delta_G(s,t) = \delta_G^{n-1}(s,t)$ for all $s, t \in V$. Moreover, in such a case there exists a *feasible price function* $p : V \to \mathbb{R}$ such that reduced weight $w_p(e) := w(e) + p(u) - p(v) \geq 0$ for all $uv = e \in E$. For any path $s \to t = P \subseteq G$, the reduced length $\ell_p(P)$ (i.e., length wrt. weights w_p) is non-negative and differs from the original length $\ell(P)$ by the value $p(s) - p(t)$ which does not depend on the shape of P.

For any $S \subseteq V$, we denote by $G - S$ the subgraph of G on V obtained from G by removing all edges incident to vertices S.

We sometimes talk about rooted out-trees T with all edges directed from a parent to a child. In such a tree T with root s, a *root path* $T[s \to t]$ is the unique path from the root to the vertex t of T. A subtree of T rooted in some of its vertices v is denoted by $T[v]$.

3 Fully dynamic shortest paths data structure

This section is devoted to proving the main theorem of this paper.

▶ **Theorem 1.** *Let G be a real-weighted directed graph. There exists a deterministic data structure maintaining G under fully dynamic vertex updates and answering arbitrary s,t-distance queries with $\widetilde{O}(mn^{4/5})$ worst-case update time and $\widetilde{O}(n^{4/5})$ query time and using $\widetilde{O}(n^2)$ space. The queries are supported only when G has no negative cycles. After answering a distance query, some corresponding shortest path $P = s \to t$ can be reported in $O(|P|)$ time.*

First, let us assume that all the edge weights are non-negative. Let us also make a simplifying assumption that any shortest k-hop-bounded $s \to t$ path in G always has a minimum possible number of hops and is simple. If there are no negative cycles, this is easy to guarantee by replacing each edge weight $w(e)$ in G with a pair $(w(e), 1)$, adding weights coordinate-wise, and comparing them lexicographically. We discuss how to extend the data structure to also handle negative edge weights and negative cycles later in Section 3.7.

We will first present a simple Monte Carlo randomized data structure, and show how to make it deterministic with no asymptotic time penalty (wrt. $\widetilde{O}(\cdot)$ notation) in Section 3.5.

Some further variants of the data structure are sketched in the Appendix. A variant for unweighted digraphs is given in Section A.1. In the weighted case, one can also achieve polynomially faster update at the cost of polynomially slower query and randomization. For details, see Section A.2.

The data structure operates in phases of Δ vertex updates. At the beginning of each phase, we apply a rather costly preprocessing described in the next subsection.

3.1 Preprocessing at the beginning of a phase

The preprocessing follows the general approach of [24] adjusted with some ideas from [27].

Let $h \in [2, n]$, and let τ be a *congestion threshold*, to be set later. We compute a certain collection of paths Π in G containing, for every pair $s, t \in V$, at most one $s \to t$ path $\pi_{s,t}$, satisfying $|\pi_{s,t}| = O(h)$, and a subset $C \subseteq V$ of *congested vertices.*

First of all, the collection Π and the set C satisfy:

$$\delta^h_G(s,t) \leq \ell(\pi_{s,t}) \leq \delta^h_{G-C}(s,t), \text{ for all } s,t \in V. \tag{1}$$

Above, we abuse the notation a bit and set $\ell(\pi_{s,t}) := \infty$ if there is no path $\pi_{s,t}$ in Π. Moreover, for any $v \in V$, let us define:

$$\Pi(v) := \{\pi_{s,t} \in \Pi : v \in V(\pi_{s,t})\},$$
$$\alpha(v) := \sum_{\pi_{s,t} \in \Pi(v)} \deg(t).$$

Crucially, Π additionally satisfies:

$$\alpha(v) \leq \tau, \text{ for all } v \in V. \tag{2}$$

▶ **Lemma 3.** *Let $h \in [1, n]$. For any $\tau \geq 2m$, in $O(nmh)$ time one can compute the congested set $C \subseteq V$ and a set of paths Π satisfying conditions (1) and (2) so that $|C| = O(nmh/\tau)$.*

Proof. We start with empty sets C and Π. Note that (2) is satisfied initially since all values $\alpha(\cdot)$ are zero. We will gradually add new paths to Π while maintaining (2) and ensuring that (1) holds for more and more pairs s, t. While introducing new paths to Π, we will also maintain the values $\alpha(v)$ (as defined above) for all $v \in V$.

We process source vertices $s \in V$ one by one, in arbitrary order. For each such s, we first move to C all the vertices $v \in V \setminus C$ with $\alpha(v) > \tau/2$. Next, we compute, for all $t \in V$, a shortest h-hop-bounded path $\pi_{s,t} = s \to t$ in $G - C$ (if such a path exists). For a fixed s, all the paths $\pi_{s,t}$ can be computed in $O(mh)$ time using a variant of Bellman-Ford algorithm. We add the newly computed paths to Π. Afterwards, (1) clearly holds for s and all $t \in V$. Moreover, (1) also holds for all $\pi_{s',t'} \in \Pi$ that have been added for a source s' processed earlier than s. Indeed, extending the set C only weakens the upper bound in (1). The values $\alpha(v)$ can be updated easily in $O(nh)$ time. Observe that for any $v \in V \setminus C$, $\alpha(v)$ grows by at most $\sum_{t \in V} \deg(t) = m$ when processing s. As a result, after processing s, we have $\alpha(v) \leq \tau/2 + m \leq \tau/2 + \tau/2 = \tau$ and hence (2) is satisfied. At the same time, since we use paths from $G - C$, for any $y \in C$, $\alpha(y)$ does not increase and thus we still have $\alpha(y) \leq \tau$.

Finally, note that for any $\pi_{s,t}$ added to Π, since $|\pi_{s,t}| \leq h$, $\alpha(v)$ grows by $\deg(t)$ for at most h distinct vertices v. As a result, we have $\sum_{v \in V} \alpha(v) \leq \sum_{t \in V} \deg(t) \cdot \left(\sum_{s \in V} h\right) \leq m \cdot nh$. But for each $y \in C$, we have $\alpha(y) > \tau/2$, so there is at most $2nmh/\tau$ such vertices y. ◀

Applying Lemma 3 constitutes the only preprocessing that we apply at the beginning of a phase in the Monte Carlo randomized variant. The computed paths Π are stored explicitly and thus the used space might be $\Theta(n^2 h)$. Note that with the help of additional $O\left(\sum_{\pi_{s,t} \in \Pi} |\pi_{s,t}|\right) = O(n^2 h)$ vertex-path pointers, we can report the elements of any $\Pi(v)$, $v \in V$, in constant time per element. We will discuss how to improve the space to $\widetilde{O}(n^2)$ using a trick due to Probst Gutenberg and Wulff-Nilsen [24] in Section 3.6.

3.2 Update

When a phase proceeds, let D be the set of at most Δ *affected* vertices in the current phase, that is, D contains every v such that a vertex update around v has been issued in this phase.

In the query procedure, we will separately consider paths going through $C \cup D$, and those lying entirely in $G - (C \cup D)$. To handle the former, upon each update we simply compute single-source shortest-path trees from and to each $s \in C \cup D$ in the current graph G. This takes $\widetilde{O}(|C \cup D|m)$ worst-case time using Dijkstra's algorithm.

As a matter of fact, we will not quite compute shortest paths in $G - (C \cup D)$, but instead, we will find paths in $G - D$ that are not longer than the distances between their corresponding endpoints in $G - (C \cup D)$. This is acceptable since $G - D \subseteq G$.

To prepare for queries about the paths in $G - (C \cup D)$, we do the following. We will separately handle *short* $\leq h$-hop shortest paths, and *long* $> h$-hop shortest paths.

Short paths. Denote by G_0 the graph at the beginning of the phase. Recall that we use G to refer to the current graph. Clearly, we have $G - D \subseteq G_0$. Fix some $s \in V$. First of all, note that if for some $t \in V$, $V(\pi_{s,t}) \cap D = \emptyset$, then $\pi_{s,t} \subseteq G - D$, so by (1):

$$\delta_{G-D}(s,t) \leq \ell(\pi_{s,t}) \leq \delta^h_{G_0-C}(s,t) \leq \delta^h_{G-(C\cup D)}(s,t).$$

The paths $\pi_{s,t}$ going through D are not preserved in $G - (C \cup D)$ and thus we cannot use them. We replace them with other paths $\pi'_{s,t}$ constructed using the following lemma.

▶ **Lemma 4.** *For $s \in V$, let Q_s contain all t such that $V(\pi_{s,t}) \cap D \neq \emptyset$. In $\widetilde{O}\left(\sum_{t\in Q_s} \deg(t)\right)$ time we can compute a representation of paths $\pi'_{s,t} \subseteq G - D$ (where $t \in Q_s$), each with possibly $\Theta(n)$ hops, satisfying:*

$$\delta_{G-D}(s,t) \leq \ell(\pi'_{s,t}) \leq \delta^h_{G-(C\cup D)}(s,t).$$

The representation is a tree T_s rooted at s such that:
(1) *some edges $sv \in E(T_s)$ represent paths $\pi_{s,v} \subseteq G - D$ from Π and have corresponding weights $\ell(\pi_{s,v})$,*
(2) *all other edges of T_s come from $E(G - D)$,*
(3) *for all $t \in Q_s$, $\pi'_{s,t}$ equals $T_s[s \to t]$ with possibly the first edge sw of that path uncompressed into the corresponding path $\pi_{s,w} \in \Pi$.*

Proof. Let Y be an edge-induced directed graph obtained as follows. For all $t \in Q_s$, and every of at most $\deg(t)$ edges $vt \in E(G - D)$, we add to Y the following:
- the edge vt itself (with the same weight),
- if $V(\pi_{s,v}) \cap D = \emptyset$, an edge sv of weight $\ell(\pi_{s,v})$ corresponding to the path $\pi_{s,v} \in \Pi$.

The algorithm is to simply compute a shortest paths tree T_s from s in Y in $\widetilde{O}(|E(Y)|) = \widetilde{O}\left(\sum_{t\in Q_s} \deg(t)\right)$ time using Dijkstra's algorithm. Clearly, any path $T_s[s \to t]$ corresponds to an $s \to t$ path in $G - D$. It is thus sufficient to prove that for all $t \in Q_s$, we have $\ell(T_s[s \to t]) \leq \delta^h_{G-(C\cup D)}(s,t)$.

If t is unreachable in $G - (C \cup D)$ from s using a path with at most h hops, there is nothing to prove. Otherwise, let a simple path P be a shortest h-hop-bounded $s \to t$ path in $G - (C \cup D)$. Let p be the last vertex on P such that $V(\pi_{s,p}) \cap D = \emptyset$, that is, $p \notin Q_s$. Note that p exists since $\delta^h_{G-C}(s,v) \neq \infty$ for all $v \in V(P)$ (which implies $\pi_{s,v} \in \Pi$) and $p \neq t$. Let P' be the $s \to p$ subpath of P. Let $e_1, \ldots, e_k \in E(G - (C \cup D))$ be the edges following p on P. Here, p is the tail of e_1. By the definition of Y and p, we have $e_i \in E(Y)$ for all $i = 1, \ldots, k$ since the head of each e_i is in Q_s. Moreover, there is an edge sp of weight $\ell(\pi_{s,p})$ in Y. Now, since $\pi_{s,p}$ is a path in $G - D$ of length at most $\delta^h_{G_0-C}(s,p)$, whereas the path $P' \subseteq G - (C \cup D) = G_0 - (C \cup D)$ has less than h hops, we obtain $\ell(\pi_{s,p}) \leq \ell(P')$ and hence:

$$\ell(T_s[s \to t]) = \delta_Y(s,t) \leq \ell(\pi_{s,p}) + \sum_{i=1}^{k} w(e_i) \leq \ell(P') + \sum_{i=1}^{k} w(e_i) = \ell(P) = \delta^h_{G-(C\cup D)}(s,t). \quad ◀$$

We compute the paths $\pi'_{s,t}$ from Lemma 4 for all $s \in V$, $t \in Q_s$. Recall that $t \in Q_s$ implies that $V(\pi_{s,t}) \cap D \neq \emptyset$ and thus $\pi_{s,t} \in \Pi(d)$ for some $d \in D$. Therefore, the time needed for computing the paths $\pi'_{s,t}$ can be bounded as follows:

$$\widetilde{O}\left(\sum_{s\in V}\sum_{t\in Q_s}\deg(t)\right) = \widetilde{O}\left(\sum_{d\in D}\sum_{\pi_{s,t}\in\Pi(d)}\deg(t)\right) = \widetilde{O}\left(\sum_{d\in D}\alpha(d)\right) = \widetilde{O}(|D|\tau) = \widetilde{O}(\Delta\tau).$$

Note that the sets Q_s can also be constructed within this bound, since they can be read from $\bigcup_{d\in D}\Pi(d)$ which also has size $\widetilde{O}(\Delta\tau)$ and the paths from any $\Pi(v)$ can be reported in $O(1)$ time per path.

For all $s \in V$ and $t \notin Q_s$, let us simply set $\pi'_{s,t} := \pi_{s,t}$ and put $\Pi' = \{\pi'_{s,t} : s,t \in V\}$. To summarize, in $\widetilde{O}(\Delta\tau)$ time we can find, for all $s,t \in V$, a representation of paths $\pi'_{s,t}$ in $G - D$ that are at least as short as the corresponding shortest h-hop-bounded $s \to t$ paths in $G - (C \cup D)$. Storing a representation of the paths $\Pi' \setminus \Pi$ requires $\widetilde{O}(\min(\Delta\tau, n^2))$ additional space since, by the construction of Lemma 4, each of these paths can be encoded using its last edge and a pointer to another path in Π' with less hops.

Long paths. In order to handle long paths, we use the following standard hitting set trick from [37].

▶ **Lemma 5.** *Let $G = (V,E)$ be a directed graph with no negative cycles. For any $s,t \in V$, fix some simple shortest $s \to t$ path $p_{s,t}$ in G. Let $H \subseteq V$ be obtained by sampling, uniformly and independently (also from the choice of paths $p_{s,t}$), $c \cdot (n/h)\log n$ elements of V, where $c \geq 1$ is a constant. Then, with high probability (controlled by the constant c), for all $s,t \in V$, if $|p_{s,t}| \geq h$, then $V(p_{s,t}) \cap H \neq \emptyset$.*

On update, we simply apply Lemma 5 to the graph $G - (C \cup D)$ and an arbitrary choice of pairwise shortest paths therein. This way, with high probability, we obtain an $\widetilde{O}(n/h)$-sized hitting set H of shortest paths in $G - (C \cup D)$ that have at least h hops. Finally, we simply compute shortest paths trees from and to the vertices H in $G - (C \cup D)$ in $\widetilde{O}(|H|m) = \widetilde{O}(mn/h)$ worst-case time using Dijkstra's algorithm.

3.3 Query

To answer a query about s,t distance in the current graph, we simply return:

$$\min\left(\min_{v\in C\cup D}\{\delta_G(s,v)+\delta_G(v,t)\}, \min_{v\in H}\{\delta_{G-(C\cup D)}(s,v)+\delta_{G-(C\cup D)}(v,t)\}, \ell(\pi'_{s,t})\right). \quad (3)$$

The first term above is responsible for considering all s,t paths in G going through $C \cup D$. If all shortest s,t paths in G do not pass through $C \cup D$, then the second term captures (with high probability) one of such paths provided that it has at least h hops. Finally, if every shortest s,t path in G does not go through $C \cup D$ and has less than h hops, then $\delta_G(s,t) = \delta_{G-D}(s,t) = \delta^h_{G-(C\cup D)}(s,t)$. Moreover, by Lemma 4, $\pi'_{s,t}$ belongs to $G - D \subseteq G$ and $\delta_G(s,t) = \delta_{G-D}(s,t) \leq \ell(\pi'_{s,t}) \leq \delta^h_{G-(C\cup D)}(s,t) = \delta_G(s,t)$, so indeed $\delta_G(s,t) = \ell(\pi'_{s,t})$.

Finally, note that finding the minimizer in (3) allows for reconstruction of some shortest s,t path P in G in $O(|P|)$ time using the stored data structures.

3.4 Time analysis

The total time spent handling a single update is:

$$\widetilde{O}\left((|D|+|C|+|H|)m+\Delta\tau\right)=\widetilde{O}(m\Delta+nm^2h/\tau+mn/h+\Delta\tau).$$

There is also an $O(mnh)$ preprocessing cost spent every Δ updates which yields an amortized cost of $\widetilde{O}(mnh/\Delta)$ per update. Since $\tau \geq 2m$, the term $m\Delta$ is negligible above.

Balancing the terms mnh/Δ and mn/h yields $\Delta = h^2$. Next, balancing with $\Delta\tau$ yields $\tau = mn/h^3$ under the assumption $h = O(n^{1/3})$. Finally, balancing mn/h and $nm^2h/\tau = mh^4$ yields $h = n^{1/5}$, $\Delta = n^{2/5}$, and $\tau = mn^{2/5}$. For such a choice of parameters, the amortized update time is $\widetilde{O}(mn^{4/5})$. Since the only source of amortization here is a costly preprocessing step happening in a coordinated way every Δ updates, the bounds can be made worst-case using a standard technique, see, e.g., [2, 40].

The query time is $O(|C|+|D|+|H|+1)$. For the obtained parameters, the bound becomes $\widetilde{O}(\Delta+nmh/\tau+n/h)=\widetilde{O}(h^4+n/h)=\widetilde{O}(n^{4/5})$.

▶ Remark 6. In the above analysis, we have silently assumed that the "current" number of edges m does not decrease significantly (say, by more than a constant factor) during a phase due to vertex deletions, so that $m = \Omega(m_0)$ holds at all times, where $m_0 = |E(G_0)|$. Since the preprocessing of Lemma 3 is applied to G_0, for the chosen parameters h, Δ, and $\tau = m_0 n^{2/5}$, the update bound should more precisely be bounded by $\widetilde{O}(\max(m, m_0) \cdot n^{4/5})$. In general, it might happen that m becomes polynomially smaller that m_0 while the phase proceeds, e.g., if $m_0 = O(n\Delta)$. This could make the update bound higher than $\widetilde{O}(mn^{4/5})$.

There is a simple fix to this shortcoming, described in [27]: when a phase starts, it is enough to put aside a set $B \subseteq V$ of Δ vertices with largest degrees in G_0 and preprocess the graph $G_0 - B$ instead. The edges incident to vertices B can be viewed as added during the first Δ "auxiliary" updates in the phase, and effectively included in the affected set D from the beginning of the phase. One can easily prove that this guarantees that $m = \Omega(m_0)$ throughout the phase, where m_0 is now defined as $|E(G_0 - B)|$.

3.5 Derandomization

The only source of randomization so far was sampling a subset of vertices that hits shortest paths in $G - (C \cup D)$ with at least h hops. To derandomize the data structure, we will construct a hitting set H of size $\widetilde{O}(n/h)$ such that H hits all the paths in $\Pi' = \{\pi'_{s,t} : s,t \in V\}$ (constructed during update) with at least h distinct vertices. Recall that the paths Π' have been used to handle short paths so far. We first show that a hitting set H defined this way can serve the same purpose as the randomly sampled hitting set.

▶ **Lemma 7.** *Let $H \subseteq V$ be such that for all $s,t \in V$ satisfying $|V(\pi'_{s,t})| \geq h$, $V(\pi'_{s,t}) \cap H \neq \emptyset$ holds. Let $a, b \in V$ be such that every shortest $a \to b$ path in G has more than h hops and* does not *go through $C \cup D$. Then there exists a shortest $a \to b$ path in G that goes through a vertex of H.*

Proof. Let Q be the shortest $a \to b$ path in G that has the minimum number of hops. By the assumption, $|Q| > h$ and $V(Q) \cap (C \cup D) = \emptyset$. Let $Q = RS$, where $R = a \to c$ is the h-hop prefix of Q. We have $R \subseteq G - (C \cup D)$ and, since Q is a shortest path in G, R is also shortest in G and

$$\ell(R) = \delta_G(a,c) = \delta^h_G(a,c) = \delta^h_{G-(C\cup D)}(a,c).$$

Since $\delta^h_{G-(C\cup D)}(a,c)$ is finite, the path $\pi'_{a,c} \subseteq G - D \subseteq G$ satisfies

$$\delta_G(a,c) \leq \delta_{G-D}(a,c) \leq \ell(\pi'_{a,c}) \leq \delta^h_{G-(C\cup D)}(a,c) = \delta_G(a,c).$$

We conclude that the path $Q' = \pi'_{a,c} \cdot S$ satisfies $\ell(Q') = \ell(Q)$ and thus Q' is also a shortest $a \to b$ path in G. Since G has no negative cycles, one can obtain a *simple* $a \to c$ path $\pi''_{a,c}$ from $\pi'_{a,c}$ by eliminating zero-weight cycles, so that $\ell(\pi''_{a,c}) = \ell(\pi'_{a,c}) = \delta_G(a,c)$ and $V(\pi''_{a,c}) \subseteq V(\pi'_{a,c})$. By the definition of Q, $|V(\pi'_{a,c})| \geq |\pi''_{a,c}| \geq |R| \geq h$, since otherwise Q would not have a minimum number of hops. By the assumption we have $V(\pi'_{a,c}) \cap H \neq \emptyset$, so Q' is a shortest $a \to b$ path in G going through a vertex of H. ◀

Additional preprocessing. When a phase starts, we additionally do the following. Let Π_0 be a set of paths obtained as follows. For all $\pi_{s,t} \in \Pi$, if $|\pi_{s,t}| \geq h/2$, we add $\pi_{s,t}$ to Π_0.

Let us now recall a folklore greedy algorithm (used, e.g., in [42]) for computing a hitting set of a collection of sufficiently large sets over a common ground set, summarized by the following lemma.

▶ **Lemma 8.** *Let X be a ground set of size n and let $\mathcal{Y}$ be a family of subsets of X, each with at least k elements. Then, in $O\left(\sum_{Y\in\mathcal{Y}}|Y|\right)$ time one can deterministically compute a* hitting set *$H \subseteq X$ of size $O(n/k \cdot \log n)$ such that $H \cap Y \neq \emptyset$ for all $Y \in \mathcal{Y}$.*

We skip the proof of Lemma 8 since we later prove a more general result in Theorem 9. Using Lemma 8 we can compute a hitting set $H_0 \subseteq V$ of Π_0 in $O(n^2h)$ time. H_0 has size $\widetilde{O}(n/h)$.

Computing a hitting set upon update. To compute a hitting set $H \subseteq V \setminus D$ as required by Lemma 7, we perform the following additional steps upon update. Recall that the precomputed set $H_0 \subseteq V$ hits all (simple) paths in $\Pi \cap \Pi'$ with at least $h/2$ hops, and thus also those that have at least h distinct vertices. We will augment H_0 into H so that it also hits all the paths in $\Pi' \setminus \Pi$ with at least h distinct vertices.

Recall from Lemma 4 that for a fixed $s \in V$, all the paths $\pi'_{s,t}$, where $t \in Q_s$, are encoded using a tree T_s. By construction, for each edge e of T_s, we have that the tail of e is s, or the head of e is in Q_s. Consider a subtree $T_s[u]$ rooted at some child u of s in T_s. If the edge su in T_s corresponds to the path $\pi_{s,u}$ with $|\pi_{s,u}| \geq h/2$ then H_0 hits $\pi_{s,u}$. As a result, for all $t \in Q_s \cap V(T_s[u])$, $V(\pi_{s,u}) \subseteq V(\pi'_{s,t})$ and hence if $|V(\pi'_{s,t})| \geq h$ then H_0 hits $V(\pi'_{s,t})$. Otherwise either su is a single edge from $G - D$, or it corresponds to a path $\pi_{s,u}$ with $|\pi_{s,u}| < h/2$. In either of these cases, if some t is at depth less than $h/2 - 1$ in $T_s[u]$, then $|V(\pi'_{s,t})| < h/2 + 1 + (h/2 - 1) = h$, so the path $\pi'_{s,t}$ does not need to be hit by H. Consequently, observe that it is enough for H to hit all the $(h/2-1)$-hop root paths in $T_s[u]$ in order to have $V(\pi'_{s,t}) \cap H \neq \emptyset$ for each $t \in T_s[u]$ with $|V(\pi'_{s,t})| \geq h$.

Let $\mathcal{Z}$ be the collection of all the subtrees $T_s[u]$, where $s \in V$ and $su \in E(T_s)$. It is now enough to compute an $\widetilde{O}(n/h)$-sized hitting set H_1 of each of the $(h/2-1)$-hop root paths in all trees in $\mathcal{Z}$. Then, $H_0 \cup H_1$ will form a desired hitting set H of all the paths in Π' with at least h distinct vertices. To this end, we could use a well-known variant of Lemma 8 due to King [30, Lemma 5.2]. However, the running time of that algorithm cannot be easily bounded with the total size N of $\mathcal{Z}$ (i.e., $N = \sum_{T\in\mathcal{Z}}|T|$) exclusively; its running time is $O\left(N + \sum_{T\in\mathcal{Z}}\min(n\log n, |T|k)\right) = O(\min(Nk, |\mathcal{Z}|n\log n))$ if one desires to hit k-hop root paths. Though, for some important cases, e.g., when $\mathcal{Z}$ contains n trees with $\Theta(n)$ vertices each, the running time is near-linear in N for any k. Unfortunately, this might not be the case in our scenario. Instead, we present a more sophisticated near-linear (independent of k) time algorithm for this task.

▶ **Theorem 9.** *Let V be a vertex set of size n and let $\mathcal{Z}$ be a family of trees on V. Let $N = \sum_{T\in\mathcal{Z}} |T|$. For any $k \in [1,n]$, in $O(N\log^2 n)$ time one can deterministically compute an $O(n/k\cdot\log n)$-sized* hitting set *$H \subseteq V$ of all the k-hop root paths in all the trees in $\mathcal{Z}$.*

Proof. We first iteratively prune the trees in $\mathcal{Z}$ of all the leaves at depths not equal to k: this does not alter the set of subpaths required to be hit. Afterwards, the task is to hit all the root-leaf paths in the collection $\mathcal{Z}$, each of exactly k hops.

Similarly as in [30], we would like to simulate the greedy algorithm behind Lemma 8, that is, repeatedly pick a vertex $v \in V$ hitting the largest number of paths not yet hit, and add it to the constructed set H. However, we cannot afford to follow this approach directly. Instead, when $L \geq 1$ paths are remaining to be hit, and there is $n' \geq k+1$ vertices $V \setminus H$ that have not yet been chosen, we pick a vertex hitting at least $\frac{L(k+1)}{2n'}$ remaining paths. Note that there always exists a vertex hitting at least $\frac{L(k+1)}{n'}$ remaining paths, since otherwise some of the remaining paths would contain a vertex from outside $V \setminus H$, a contradiction. A single step in our approach reduces L to at most $\left(1-\frac{k+1}{2n'}\right)L$, so $\lceil 2n'/(k+1)\rceil = O(n/k)$ steps reduce L to at most L/e. Hence, since L is an integer, after $O\left(\frac{n}{k}\ln L\right) = O\left(\frac{n}{k}\ln N\right)$ steps L will drop to 0, i.e., all required paths will be hit.

Our strategy can be also rephrased as follows: maintain 2-approximate counters $\{c_v : v \in V\}$ such that the vertex v hits between c_v and $2c_v$ of the remaining paths, and repeatedly pick a vertex z with the maximum value of c_z. By the above discussion, the picked z will always satisfy $c_z \geq L(k+1)/2n'$, as desired. To implement this strategy, we proceed as follows.

For each $T \in \mathcal{Z}$, and $v \in V(T)$, let $d_{v,T}$ be the *exact* number of *previously not hit* root-leaf paths in T that v hits. Note that through the entire collection, v hits $D_v := \sum_{T\in\mathcal{Z}} d_{v,T}$ paths not yet hit. Observe that when a root-leaf path to the leaf l in T is hit for the first time, the value $d_{v,T}$ of all the ancestors v of l gets decreased by one. In fact, the algorithm of [30] can be seen to maintain such values $d_{v,T}$ and D_v explicitly. However, this is too costly for us; we will instead maintain the exact values $d_{v,T}$ only implicitly, in a data structure.

For each $T \in \mathcal{Z}$, we keep $V(T)$ (explicitly) partitioned into subsets $V_{T,0},\ldots,V_{T,\ell}$, where $\ell = O(\log|V(T)|)$, so that $v \in V_{T,i}$ iff $d_{v,T} \in [2^i, 2^{i+1})$. Throughout the process, the values $d_{v,T}$ will only decrease, so a vertex $v \in V(T)$ can only move $O(\log n)$ times to a subset $V_{T,j}$ with a lower value j. Let us first argue that maintaining such partitions yields the desired 2-approximate counters rather straightforwardly.

For $v \in V$, let us define $c_v = \sum_{T,j:v\in V_{T,j}} 2^j$. Then, we have:

$$D_v = \sum_{T\in\mathcal{Z}} d_{v,T} \geq \sum_{T,j:v\in V_{T,j}} 2^j = c_v = \sum_{T,j:v\in V_{T,j}} 2^j = \frac{1}{2}\sum_{T,j:v\in V_{T,j}} 2^{j+1} > \frac{1}{2}\sum_{T\in\mathcal{Z}} d_{v,T} = \frac{1}{2}D_v.$$

As a result, the counters c_v indeed 2-approximate the values D_v and can be maintained subject to changes in the partitions $V_{T,i}$, for all T, i, in $O\left(\sum_{T\in\mathcal{Z}}|T|\log n\right) = O(N\log n)$ time.

Fix some $T \in \mathcal{Z}$. To maintain the partition $V_{T,0},\ldots,V_{T,\ell}$, we maintain the values $d_{v,T}$ using ℓ data structures $S_{T,0},\ldots,S_{T,\ell}$. The data structure $S_{T,i}$ associates (implicitly) the following vertex weights to the individual vertices v of T. If $d_{v,T} \geq 2^i$, then v has weight $d_{v,T}$ in $S_{T,i}$. Otherwise, if $d_{v,T} < 2^i$, then v has weight ∞ in $S_{T,i}$. In particular, $S_{T,0}$ associates the exact values $d_{v,T}$ to the vertices of T.

Fix some $i = 0,\ldots,\ell$. $S_{T,i}$ is implemented using, e.g., a top-tree [4, Theorem 2.4] that allows performing the following operations, both in $O(\log n)$ time[5]:

[5] As a matter of fact, in [4] this is shown for edge weights. However, vertex weights can be simulated easily using edge weights by assigning each vertex its parent edge, and explicitly maintaining the weight of the root.

(1) adding the same $\delta \in \mathbb{R}$ to the weights of vertices on some specified path in the tree, and
(2) querying for a vertex of the tree with minimum weight.

Clearly, $S_{T,i}$ can be initialized at the beginning of the process in $O(|T| \log n)$ time. When a new vertex z is added to H, and $z \in V(T)$, we iterate through all the (previously unvisited) descendants of z to identify the (original) leaves y at depth k such that the root-to-y path in T has not been previously hit. For each such y, we decrease the weights of all the ancestors of y in T (all lying on a single path in T) by 1. This requires a single top-tree operation on $S_{T,i}$. Afterwards, for all $w \in V(T)$ whose value $d_{w,T}$ was at least 2^i before adding z to H, $S_{T,i}$ contains (in an implicit way) the correctly updated exact value $d_{w,T}$. Some of these values in $S_{T,i}$ might drop below 2^i, though. To deal with this, we repeatedly attempt to extract the minimum-valued vertex $x \in V(T)$ from $S_{T,i}$. If the value of x is less than 2^i, we reset the value of x in $S_{T,i}$ to ∞. Otherwise, we stop; at this point all the values in $S_{T,i}$ are at least 2^i; the invariant posed on $S_{T,i}$ is fixed.

The above update procedure is performed for each i. Observe that $v \in V_{T,i}$ iff i is the maximum index such that v has assigned a finite value in $S_{T,i}$. Since for all i we can explicitly track which vertices in $S_{T,i}$ are assigned ∞ while performing updates, the time needed to maintain the partition $V_{T,0}, \ldots, V_{T,\ell}$ can be charged to the cost of maintaining the data structures $S_{T,0}, \ldots, S_{T,\ell}$.

Let us now analyze the time cost of this algorithm. For each $T \in \mathcal{Z}$, we iterate through every vertex of T at most $O(1)$ times. For $i = 0, \ldots, \ell$, at most $O(|V(T)| + |H \cap V(T)|) = O(|V(T)|)$ top-tree operations are performed on $S_{T,i}$. Hence, the cost of maintaining all $S_{T,i}$ for all $i = 0, \ldots, O(\log n)$ is $O(|T| \log^2 n)$. Through all $T \in \mathcal{Z}$, this is $O(N \log^2 n)$.

To implement finding a next vertex $z \in H$ with the largest c_z, one may simply store the counters c_z in a priority queue. Since the counters are updated $O(N \log n)$ times in total, the priority queue operations cost is $O(N \log^2 n)$ as well. ◀

Observe that through all s, the total number of edges in trees added to $\mathcal{Z}$ can be bounded by the number of edges in the (compressed) trees T_s of Lemma 4, and thus also by $\widetilde{O}(\min(\tau\Delta, n^2))$. As a result, by Theorem 9, the desired set H hitting all paths $\pi'_{s,t}$ with at least h distinct vertices can be computed in $\widetilde{O}(\tau\Delta)$ time, using at most quadratic space. This does not increase the running time of the update procedure in the asymptotic sense.

3.6 Reducing the space usage

So far, the space used by the preprocessing phase could only be bounded by $O(n^2 h)$ as we have explicitly stored the $O(n^2)$ preprocessed paths $\pi_{s,t} \in \Pi$, each with $O(h)$ hops.

We do not, however, need to store the paths $\pi_{s,t} \in \Pi$ explicitly. For performing updates and answering distance queries, we only require storing the values $\ell(\pi_{s,t})$, $|\pi_{s,t}|$, and being able to efficiently access the sets $\Pi(v)$, for any $v \in V$. If we want to also support path queries, then constant-time reporting of the subsequent edges of $\pi_{s,t}$ is also needed. Probst Gutenberg and Wulff-Nilsen [24, Section 4.2] showed an elegant way of achieving that in a slightly relaxed way using only $\widetilde{O}(n^2 \log h)$ space.

▶ **Lemma 10** ([24])**.** *Let $G = (V, E)$ be a real-weighted digraph with no negative cycles. Let $s \in V$ and let $h \in [1, n]$. Using $O(mh)$ time and $O(nh)$ space, one can build an $\widetilde{O}(n)$-space data structure representing a collection $\{\pi_t : t \in V\}$ of (not necessarily simple) $O(h)$-hop paths from s to all other vertices in G such that for any t, $\ell(\pi_t) \le \delta^h_G(s, t)$.*

For any $v \in V$, the data structure allows:

- *accessing $\ell(\pi_v)$ and $|\pi_v|$ in $O(1)$ time,*
- *reporting the set $P_v = \{t \in V : v \in V(\pi_t)\}$ in $\widetilde{O}(|P_v|)$ time,*
- *reporting the edges of π_v in $O(|\pi_v|)$ time.*

Proof sketch. Suppose we compute shortest h-hop-bounded $s \to t$ paths p_t from s to all $t \in V$. This takes $O(mh)$ time, but storing the computed paths explicitly would require $\Theta(nh)$ space. Recall that if G has no negative cycles, then we may wlog. assume that the paths p_t are all simple. As a result, one can deterministically compute an $\widetilde{O}(n/h)$-sized hitting set H of the $\lceil h/3 \rceil$-hop infixes starting at the $(\lceil h/3 \rceil + 2)$-th hop of those of the computed p_t that satisfy $|p_t| \geq \lceil 2h/3 \rceil$. We explicitly store the paths p_y for all $y \in H$ which costs only $\widetilde{O}(|H| \cdot h) = \widetilde{O}(n)$ space.

Let G' be obtained from G be adding shortcut edges $e_y = sy$ of weight $w(e_y) = \ell(p_y)$ for all $y \in H$. Note that for all $v \in V$, $\delta_{G'}^{\lceil 2h/3 \rceil}(s,v) \leq \delta_G^h(s,v) = \ell(p_v)$ and every $\leq \lceil 2h/3 \rceil$-hop path in G' corresponds to a path in G with at most $h + \lceil h/3 \rceil$ hops.

We recursively solve the problem on the graph G' with hop-bound $h' = \lceil 2h/3 \rceil$. Let $\{\pi'_t : t \in V\}$ be the obtained set of paths. For every $t \in V$, we define π_t to be π'_t with possibly the first shortcut edge e_z expanded to the corresponding path p_z. One can easily prove by induction that $|\pi_t| = O(h)$ and $\ell(\pi_t) \leq \delta_G^h(s,t)$. The recursion depth is clearly $O(\log h)$.

Finally, each of the explicitly stored $\widetilde{O}(n/h)$ paths p_t at some level of the recursion can be imagined to point to at most one path of the previous level (corresponding to a shortcut edge) and some $O(h)$ distinct vertices of G. By keeping only the nodes reachable from the paths at the last level of the recursion in this pointer system, and storing reverse pointers, we can report the elements of each P_v so that every element gets reported $O(\log n)$ times. ◀

To reduce the space to $\widetilde{O}(n^2)$, we simply replace the Bellman-Ford like procedure run on $G - C$ in the preprocessing of Lemma 3 with the construction of Lemma 10. The total congestion of all the vertices can increase only by a constant factor then. In Section 3.5 we have assumed that the preprocessed paths $\pi_{s,t}$ were simple when hitting all $\pi_{s,t}$ satisfying $|\pi_{s,t}| \geq h/2$ with H_0. But we can as well assume that H_0 hits all $\pi_{s,t}$ with $|V(\pi_{s,t})| \geq h/2$ instead. Even though the paths represented by the data structure of Lemma 10 might be non-simple, we can compute the sizes $|V(\pi_v)|$ within the same bound easily. Moreover, the algorithm behind Lemma 8 can be implemented so that it requires only $O(n)$ additional space if it is possible to (1) iterate through the elements of individual sets of $\mathcal{Y}$ in $O(1)$ time per element, and (2) report the sets $Y \in \mathcal{Y}$ containing a given $x \in X$ in near-linear time in the number of reported sets. This is precisely what Lemma 10 enables.

3.7 Negative edges and cycles

In this section we briefly describe the modifications to the data structure needed to handle negative edge weights and possibly negative cycles.

First of all, we run in parallel a deterministic fully dynamic negative cycle detection algorithm with $\widetilde{O}(m)$ worst-case update time (see, e.g., [27]). That algorithm also maintains a feasible price function p of the current graph G. With this in hand, whenever G has a negative cycle, we refrain from running the update procedure and forbid issuing queries. Otherwise, p is also a feasible price function of $G - D$, and thus the Dijkstra-based update procedure can simply use p to ensure that all the edge and path lengths accessed are non-negative.

In the basic randomized variant of our data structure we don't need to alter the preprocessing at the beginning of a phase at all. Indeed, our analysis did not require that the paths $\pi'_{s,t}$ are simple or with no negative cycles, and h-hop-bounded shortest paths are well-defined even in presence of negative cycles. In the $O(n^2h)$-space deterministic variant (Section 3.5), similarly as in Section 3.6, we may compute the hitting set H_0 only for those $\pi_{s,t}$ that satisfy $|V(\pi_{s,t})| \geq h/2$. Recall that if the update procedure is run, then $G - D$ has no negative cycle and hence no path $\pi_{s,t}$ containing a negative cycle survives in $G - D$ anyway.

Finally, the preprocessing algorithm behind Lemma 10 internally uses hitting-set arguments (valid for simple paths) and requires, out-of-the-box, that there are no negative cycles. We now sketch how to deal with negative cycles while using the space-saving Lemma 10.

Whenever the preprocessing in Lemma 10 for source s encounters a path p_t containing a negative cycle, we use it as the desired path π_t, but discard it when computing a hitting set and thus also in the recursive preprocessing in Lemma 10 – effectively making reporting π_t (in any way) during update or query impossible. Similarly, such a path is included as $\pi_{s,t} \in \Pi$ in Lemma 3 only implicitly and marked as *negative*, but nevertheless used for updating the congestion counters $\alpha(\cdot)$ during the preprocessing. Note that during the update procedure, if G has no negative cycles, then for each "negative" path $\pi_{s,t}$, we have $V(\pi_{s,t}) \cap D \neq \emptyset$. The used charging scheme ensures that we can afford reconstructing the path $\pi'_{s,t}$ within the $\widetilde{O}(\tau\Delta)$ bound even though we do not know which vertices of D lie on $\pi_{s,t}$.

4 Algebraic fully dynamic reachability in sparse digraphs

In this section we show how the algebraic approach to dynamic reachability [35] can be applied in the case of sparse graphs, even without resorting to fast matrix multiplication [3, 21].

Assume for simplicity that $m = |E(G)| \geq n$ at all times. We prove the following.

▶ **Theorem 2.** *Let G be a directed graph. Let $t \in [1, \sqrt{m}]$. There exist a Monte Carlo randomized data structure maintaining G subject to fully dynamic single-edge updates with $\widetilde{O}(mn/t)$ worst-case update time and supporting arbitrary-pair reachability queries in $O(t)$ time. The answers produced are correct with high probability.*

Let us first review the approach of [35]. Identify the vertices of $G = (V, E)$ with $\{1, \ldots, n\}$. Assume G has a self-loop at every vertex, i.e., $vv \in E$ for all $v \in V$; self-loops do not change the reachability relation in G. Let $A(G)$ be an adjacency matrix of G, that is, an $n \times n$ matrix with the entry $A(G)_{ij}$ equal to 1 if $ij \in E(G)$, and 0 otherwise.

Let us choose a field $\mathbb{F} = \mathbb{Z}/p\mathbb{Z}$, for a prime number $p = \Theta(n^c)$, where $c \geq 3$ is a constant. Let the matrix $\tilde{A}(G)$ be obtained from $A(G)$ by replacing each 1 with a random element from $\mathbb{F}$. Sankowski [35, Theorem 6] showed the following.

▶ **Theorem 11.** [35] *With high probability (controlled by the constant c), the matrix $\tilde{A}(G)$ is invertible over $\mathbb{F}$ and for all $u, v \in V$, $(\tilde{A}(G)^{-1})_{u,v} \neq 0$ holds if and only if there exists a $u \to v$ path in G.*

Theorem 11 reduces fully dynamic reachability to the *dynamic matrix inverse* problem. Note that a single-edge update to G translates to a single-entry matrix update on $\tilde{A}(G)$, whereas a reachability query corresponds to an element query on the inverse $\tilde{A}(G)^{-1}$.

Sankowski [35] studied update/query tradeoffs for the dynamic matrix inverse problem. One tradeoff, summarized by the following theorem, is of our particular interest.

▶ **Theorem 12.** [35] *Suppose a matrix $A \in \mathbb{F}^{n \times n}$ is subject to single-element updates that keep A non-singular at all times.*

Let $\delta \in (0, 1)$. There exists a data structure maintaining A^{-1} with $\widetilde{O}(n^{\omega(1,\delta,1)-\delta} + n^{1+\delta})$ worst-case update time and supporting element queries on A^{-1} in $O(n^\delta)$ time.

Above, $\omega(1, \delta, 1) \geq 2$ denotes the rectangular matrix multiplication exponent (see [21]), i.e., a value such that one can multiply an $n \times n^\delta$ matrix by an $n^\delta \times n$ matrix in $\widetilde{O}\left(n^{\omega(1,\delta,1)}\right)$ time. Here, the time is measured in field operations. By applying Theorem 12 with $\delta \approx 0.529$ such that $\omega(1, \delta, 1) = 1 + 2\delta$ to the matrix $\tilde{A}(G)$ (whose inverse correctly encodes the transitive closure of G throughout poly n updates, w.h.p. against an adaptive adversary), Sankowski [35] obtains a Monte Carlo randomized fully dynamic reachability algorithm with $\widetilde{O}(n^{1.529})$ worst-case update and $O(n^{0.529})$ query time.

To continue, we need to discuss some of the internals of the data structure of Theorem 12 [35, Section 6]. That data structures operates in phases of n^{δ} updates. At the end of each phase, the inverse A^{-1} is *explicitly* recomputed from (1) the explicitly stored inverse $(A_0)^{-1}$ of the matrix A_0 from the beginning of the phase, and (2) the n^{δ} updates in the current phase, via rectangular matrix multiplication. This is the sole reason why the term $n^{\omega(1,\delta,1)-\delta}$ appears in the update bound. In particular, at the beginning of each phase, we could also recompute the inverse of the current matrix A *from scratch* in $O(n^{\omega})$ time and thus obtain a slightly worse update bound of $\widetilde{O}(n^{\omega-\delta}+n^{1+\delta})$, which in turn leads to the $\widetilde{O}(n^{(\omega+1)/2})=O(n^{1.687})$ update bound if optimized wrt. δ. The query time is proportional to the phase length n^{δ}.

Speaking more generally, if we could compute the inverse of the maintained matrix at any time in T time, then by following the approach behind Theorem 12, for any parameter $t\in[1,n]$ (denoting the phase length) we could obtain a data structure with $\widetilde{O}(T/t+nt)$ worst-case update time and $O(t)$ query time. For $T=\Omega(n)$, it only makes sense to use $t\in\left[1,\sqrt{T/n}\right]$, and the update bound then simplifies to $\widetilde{O}(T/t)$. To obtain our fully dynamic reachability algorithm for sparse digraphs, we use this observation combined with the below theorem following from a classical result on solving linear systems in parallel [26].

► **Theorem 13.** *Let $A\in\mathbb{F}^{n\times n}$ be a non-singular matrix with $m=\Omega(n)$ non-zero entries. Assume the finite field $\mathbb{F}$ has at least n^{2+c} elements, where $c\geq 1$ is a constant.*

There is a Las Vegas randomized algorithm that computes A^{-1} in $\widetilde{O}(nm)$ time. The success probability is at least $1-O(n^{-c})$.

Proof sketch. Kaltofen and Pan [26, Theorem 4] show, using techniques of [41], that finding the determinant of A can be reduced, in $\widetilde{O}(n^2)$ time, to solving the following subproblems:

(a) For a given vector $v\in\mathbb{F}^{n\times 1}$, computing vectors $\tilde{A}^i\cdot v$, for $i=0,\ldots,2n-1$, where $\tilde{A}=A\cdot H\cdot D$, $H\in\mathbb{F}^{n\times n}$ is a Hankel matrix, and $D\in\mathbb{F}^{n\times n}$ is a diagonal matrix.

(b) For a given vector $v\in\mathbb{F}^{n\times 1}$, computing vectors $T^i\cdot v$ for $i=0,\ldots,n-1$, where $T\in\mathbb{F}^{n\times n}$ is a Toeplitz matrix.

Then, in [26, Section 4] it is proven that if the determinant algorithm is realized using a randomized algebraic circuit, or, in other words, a straight-line program with no conditional branches, loops, etc., that possibly can divide by zero with low probability, then the Baur-Strassen theorem [7] implies that the matrix inverse can be computed within the same asymptotic bound as the determinant, even in parallel.

The subproblems (a) and (b) for general dense $n\times n$ matrices can be solved within this model in $\widetilde{O}(n^{\omega})$ time using a folklore combination of repeated squaring and fast matrix multiplication (see, e.g., [29]). In our case, to obtain a desired $\widetilde{O}(mn)$-time sparse matrix inverse algorithm, it is enough to argue that the subproblems (a) and (b) admit $\widetilde{O}(mn)$ time straight-line program (SLP) solutions for matrices with m non-zero entries.

Consider the subproblem (a), since (b) is very similar. We compute each subsequent vector $\tilde{A}^{i+1}\cdot v$ as $A\cdot(H\cdot(D\cdot(\tilde{A}^i v)))$. Multiplying a vector by a matrix with $m=\Omega(n)$ non-zero entries can clearly be realized in $O(m)$ time using an SLP with no conditional statements. This justifies that multiplications by the matrices A and D can be realized in the required model. It is also well-known that multiplying a vector by a Hankel/Toeplitz matrix reduces to polynomial multiplication (see, e.g., [22]), and thus also can be realized using an $\widetilde{O}(n)$-gate straight-line program (see, e.g., [11]). This proves that each $\tilde{A}^{i+1}v$ can be obtained from $\tilde{A}^{i}v$ in $\widetilde{O}(m)$ time in the SLP model. This implies the desired $\widetilde{O}(nm)$ SLP time bound for subproblem (a). The theorem follows. ◀

▶ **Corollary 14.** *Suppose a matrix $A \in \mathbb{F}^{n\times n}$ is subject to single-element updates that keep A non-singular at all times and the number of non-zero elements in A is always $O(m)$, where $m \geq n$. Let $t \in [1, \sqrt{m}]$. There exists a data structure maintaining A^{-1} with $\widetilde{O}(mn/t)$ worst-case update time and supporting element queries on A^{-1} in $O(t)$ time.*

The above corollary applied to the matrix $\tilde{A}(G)$ combined with Theorem 11 implies Theorem 2.

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 434–443. IEEE Computer Society, 2014. doi:10.1109/FOCS.2014.53.

2 Ittai Abraham, Shiri Chechik, and Sebastian Krinninger. Fully dynamic all-pairs shortest paths with worst-case update-time revisited. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 440–452. SIAM, 2017. doi:10.1137/1.9781611974782.28.

3 Josh Alman and Virginia Vassilevska Williams. A refined laser method and faster matrix multiplication. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 522–539. SIAM, 2021. doi:10.1137/1.9781611976465.32.

4 Stephen Alstrup, Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Maintaining information in fully dynamic trees with top trees. *ACM Trans. Algorithms*, 1(2):243–264, 2005. doi:10.1145/1103963.1103966.

5 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental algorithms for minimal length paths. *J. Algorithms*, 12(4):615–638, 1991. doi:10.1016/0196-6774(91)90036-X.

6 Surender Baswana, Ramesh Hariharan, and Sandeep Sen. Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths. *J. Algorithms*, 62(2):74–92, 2007. doi:10.1016/j.jalgor.2004.08.004.

7 Walter Baur and Volker Strassen. The complexity of partial derivatives. *Theor. Comput. Sci.*, 22:317–330, 1983. doi:10.1016/0304-3975(83)90110-X.

8 Thiago Bergamaschi, Monika Henzinger, Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New techniques and fine-grained hardness for dynamic near-additive spanners. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1836–1855. SIAM, 2021. doi:10.1137/1.9781611976465.110.

9 Aaron Bernstein. Maintaining shortest paths under deletions in weighted directed graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.

10 Aaron Bernstein, Maximilian Probst Gutenberg, and Thatchaphol Saranurak. Deterministic decremental SSSP and approximate min-cost flow in almost-linear time. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021*, pages 1000–1008. IEEE, 2021. doi:10.1109/FOCS52979.2021.00100.

11 David G. Cantor and Erich Kaltofen. On fast multiplication of polynomials over arbitrary algebras. *Acta Informatica*, 28(7):693–701, 1991. doi:10.1007/BF01178683.

12 Sílvia Casacuberta and Rasmus Kyng. Faster sparse matrix inversion and rank computation in finite fields. In *13th Innovations in Theoretical Computer Science Conference, ITCS 2022*, volume 215 of *LIPIcs*, pages 33:1–33:24. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.33.

13 Shiri Chechik and Tianyi Zhang. Faster deterministic worst-case fully dynamic all-pairs shortest paths via decremental hop-restricted shortest paths. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023*, pages 87–99. SIAM, 2023. doi:10.1137/1.9781611977554.ch4.

14 Li Chen, Gramoz Goranci, Monika Henzinger, Richard Peng, and Thatchaphol Saranurak. Fast dynamic cuts, distances and effective resistances via vertex sparsifiers. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020*, pages 1135–1146. IEEE, 2020. doi:10.1109/FOCS46700.2020.00109.

15 Julia Chuzhoy. Decremental all-pairs shortest paths in deterministic near-linear time. In *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 626–639. ACM, 2021. doi:10.1145/3406325.3451025.

16 Camil Demetrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *J. ACM*, 51(6):968–992, 2004. doi:10.1145/1039488.1039492.

17 Camil Demetrescu and Giuseppe F. Italiano. Trade-offs for fully dynamic transitive closure on dags: breaking through the $O(n^2)$ barrier. *J. ACM*, 52(2):147–156, 2005. doi:10.1145/1059513.1059514.

18 Wayne Eberly, Mark Giesbrecht, Pascal Giorgi, Arne Storjohann, and Gilles Villard. Faster inversion and other black box matrix computations using efficient block projections. In *Symbolic and Algebraic Computation, International Symposium, ISSAC 2007, Proceedings*, pages 143–150. ACM, 2007. doi:10.1145/1277548.1277569.

19 Jacob Evald, Viktor Fredslund-Hansen, Maximilian Probst Gutenberg, and Christian Wulff-Nilsen. Decremental APSP in unweighted digraphs versus an adaptive adversary. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 64:1–64:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.64.

20 Sebastian Forster, Yasamin Nazari, and Maximilian Probst Gutenberg. Deterministic incremental APSP with polylogarithmic update time and stretch. *CoRR*, abs/2211.04217, 2022. doi:10.48550/arXiv.2211.04217.

21 Francois Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 1029–1046. SIAM, 2018. doi:10.1137/1.9781611975031.67.

22 Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

23 Maximilian Probst Gutenberg, Virginia Vassilevska Williams, and Nicole Wein. New algorithms and hardness for incremental single-source shortest paths in directed graphs. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020*, pages 153–166. ACM, 2020. doi:10.1145/3357713.3384236.

24 Maximilian Probst Gutenberg and Christian Wulff-Nilsen. Fully-dynamic all-pairs shortest paths: Improved worst-case time and space bounds. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020*, pages 2562–2574. SIAM, 2020. doi:10.1137/1.9781611975994.156.

25 Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 21–30. ACM, 2015. doi:10.1145/2746539.2746609.

26 Erich Kaltofen and Victor Y. Pan. Processor efficient parallel solution of linear systems over an abstract field. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '91*, pages 180–191. ACM, 1991. doi:10.1145/113379.113396.

27 Adam Karczmarz. Fully dynamic algorithms for minimum weight cycle and related problems. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 83:1–83:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.ICALP.2021.83.

28 Adam Karczmarz, Anish Mukherjee, and Piotr Sankowski. Subquadratic dynamic path reporting in directed graphs against an adaptive adversary. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1643–1656. ACM, 2022. doi:10.1145/3519935.3520058.

29 Walter Keller-Gehrig. Fast algorithms for the characteristic polynomial. *Theor. Comput. Sci.*, 36:309–317, 1985. doi:10.1016/0304-3975(85)90049-0.

30 Valerie King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *40th Annual Symposium on Foundations of Computer Science, FOCS 1999*, pages 81–91. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814580.

31 Veli Mäkinen, Alexandru I. Tomescu, Anna Kuosmanen, Topi Paavilainen, Travis Gagie, and Rayan Chikhi. Sparse dynamic programming on dags with small width. *ACM Trans. Algorithms*, 15(2):29:1–29:21, 2019. doi:10.1145/3301312.

32 Liam Roditty. A faster and simpler fully dynamic transitive closure. *ACM Trans. Algorithms*, 4(1):6:1–6:16, 2008. doi:10.1145/1328911.1328917.

33 Liam Roditty and Uri Zwick. Improved dynamic reachability algorithms for directed graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

34 Liam Roditty and Uri Zwick. On dynamic shortest paths problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

35 Piotr Sankowski. Dynamic transitive closure via dynamic matrix inverse (extended abstract). In *45th Symposium on Foundations of Computer Science FOCS 2004*, pages 509–517. IEEE Computer Society, 2004. doi:10.1109/FOCS.2004.25.

36 Mikkel Thorup. Fully-dynamic all-pairs shortest paths: Faster and allowing negative cycles. In *Algorithm Theory – SWAT 2004, 9th Scandinavian Workshop on Algorithm Theory, Proceedings*, volume 3111 of *Lecture Notes in Computer Science*, pages 384–396. Springer, 2004. doi:10.1007/978-3-540-27810-8_33.

37 Jeffrey D. Ullman and Mihalis Yannakakis. High-probability parallel transitive-closure algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. doi:10.1137/0220006.

38 Jan van den Brand, Sebastian Forster, and Yasamin Nazari. Fast deterministic fully dynamic distance approximation. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022*, pages 1011–1022. IEEE, 2022. doi:10.1109/FOCS54457.2022.00099.

39 Jan van den Brand and Danupon Nanongkai. Dynamic approximate shortest paths and beyond: Subquadratic and worst-case update time. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 436–455. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00035.

40 Jan van den Brand, Danupon Nanongkai, and Thatchaphol Saranurak. Dynamic matrix inverse: Improved algorithms and matching conditional lower bounds. In *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019*, pages 456–480. IEEE Computer Society, 2019. doi:10.1109/FOCS.2019.00036.

41 Douglas H. Wiedemann. Solving sparse linear equations over finite fields. *IEEE Trans. Inf. Theory*, 32(1):54–62, 1986. doi:10.1109/TIT.1986.1057137.

42 Uri Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002. doi:10.1145/567112.567114.

A Further variants of the fully dynamic shortest paths data structure

A.1 Unweighted digraphs

Similarly as in the case of previous fully dynamic APSP data structures [2, 24], improved bounds can be obtained if the graph G is unweighted. This is simply because the preprocessing of Lemma 3 can be completed in $O(mn)$ time instead of $O(mnh)$ time. Indeed, in an unweighted graph, the shortest h-hop-bounded s,t path, if exists, coincides with the (globally) shortest s,t path. As a result, the Bellman-Ford-based computation can be replaced with breadth-first search. Similarly, the collection of paths Π can be represented using n BFS trees and thus one can achieve quadratic space without resorting to Lemma 10.

For unweighted graphs, the update bound becomes $\widetilde{O}(m\Delta+mn/\Delta+mn/h+nm^2h/\tau+\Delta\tau)$, whereas the query time remains $\widetilde{O}(\Delta+nmh/\tau+n/h)$. For $\Delta=h=n^{1/4}$ and $\tau=mn^{1/2}$ the update and query time bounds become $\widetilde{O}(mn^{3/4})$ and $\widetilde{O}(n^{3/4})$, respectively.

A.2 A slight tradeoff

In the basic variant of the data structure, it is not clear whether pushing the update time below $\widetilde{O}(n^{4/5})$ is possible even at the cost of increasing the query time. Here, we sketch that a slight tradeoff is indeed possible with another trick of [24, Section 4.1]: to delegate handling paths through the congested set to the data structure of [2, Section 3]. For simplicity, assume again that the edge weights are non-negative. Since that data structure, in turn, is tailored to dense graphs, we instead use the following sparse variant implicit in [27].

▶ **Lemma 15.** [2, 27] *Let $G=(V,E)$ be a directed graph and let $C\subseteq V$. Let $h\in[1,n]$. In $\widetilde{O}(|C|mh)$ time one can build a data structure supporting the following.*

For any query set $D\subseteq V$, update the data structure so that it supports queries computing the length of some $s\to t$ path of length at most $\min_{c\in C}\{\delta^h_{G-D}(s,c)+\delta^h_{G-D}(c,t)\}$ for any $s,t\in V$. The worst-case update time is $\widetilde{O}(|D|mh)$ and the query time is $O(|C|)$.

Proof sketch. For at most $2|C|$ *centers* $c_1,\ldots,c_\ell$, repeatedly find shortest $2h$-hop-bounded paths from/to c_i in $G-\{c_1,\ldots,c_{i-1}\}$. While this computation proceeds, maintain vertex congestions $\alpha(\cdot)$ as in Lemma 3. When choosing the subsequent centers c_i, alternate between picking an unused vertex from C and the most congested vertex of $V\setminus\{c_1,\ldots,c_{i-1}\}$, until all vertices of C are used. This preprocessing costs $O(\ell mh)=O(|C|mh)$ time.

Given the above preprocessing, one can prove that by proceeding as in Lemma 4, in $\widetilde{O}(|D|mh)$ time one can recompute a representation of paths $s\to c_i$ of length at most $\delta^{2h}_{G-(D\cup\{c_1,\ldots,c_{i-1}\})}(s,c_i)$ and analogous paths $c_i\to t$, for all i and $s,t\in V$.

Upon a query (s,t), in $O(\ell)=O(|C|)$ time we can find an $s\to t$ path of length at most $y^*=\min_{i=1}^{\ell}\{y_i\}$, where $y_i:=\delta^{2h}_{G-(D\cup\{c_1,\ldots,c_{i-1}\})}(s,c_i)+\delta^{2h}_{G-(D\cup\{c_1,\ldots,c_{i-1}\})}(c_i,t)$. To see that this is enough, let $c^*\in C$ be such that $\delta^h_{G-D}(s,c^*)+\delta^h_{G-D}(c^*,t)$ is minimum. Let j be minimum index such that the corresponding $\le 2h$-hop path $Q=s\to c^*\to t$ contains the center c_j. Then we have $Q\subseteq G-(D\cup\{c_1,\ldots,c_{j-1}\})$ and thus $y^*\le y_j\le\ell(Q)$. ◀

Note that by computing shortest-paths trees from and to a randomly sampled $\widetilde{O}(n/h)$-sized hitting set H we can in fact handle "long" shortest paths in the current graph G, and not only in $G-(C\cup D)$. As a result, we don't need to recompute full shortest paths trees from C – instead, it would be enough to consider short paths in $G-D$ through C upon query. This is what we use Lemma 15 for. Every Δ updates, when a new phase starts, a fresh congested set C is computed. We additionally initialize the data structure of Lemma 15 for the current graph G and the congested set C. This way, that data structure is always off from the current G by at most Δ updates, and thus can be updated in $\widetilde{O}(\Delta mh)$ time. Again, the data structure of Lemma 15 can be reinitialized in such a way that the additional worst-case cost incurred is $\widetilde{O}(|C|mh/\Delta)$. The full worst-case update time becomes:

$$\widetilde{O}(m\Delta+mnh/\Delta+mn/h+\Delta\tau+m^2nh^2/(\tau\Delta)+\Delta hm).$$

Balancing as before, for $\Delta=h^2$ and $\tau=mn/h^3$, we obtain the update bound $\widetilde{O}(mn/h+mh^3)$. Note that this bound is $\Omega(mn^{3/4})$ for any h.

The query bound unfortunately remains $\widetilde{O}(\Delta+|H|+mnh/\tau)=\widetilde{O}(n/h+h^4)$. If we aim at serving $\Theta(n)$ queries per update and the graph is sparse, then we get no improvement over the basic approach. However, for a desired query time of $\widetilde{O}(t)$, where $t\in[n^{4/5},n]$, we can achieve $\widetilde{O}(mn/t^{1/4})$ worst-case update time this way.