Reliable Hubs for Partially-Dynamic All-Pairs Shortest Paths in Directed Graphs

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

Jakub Łącki
Google Research, New York, USA
jlacki@google.com

Abstract

We give new partially-dynamic algorithms for the all-pairs shortest paths problem in weighted directed graphs. Most importantly, we give a new *deterministic* incremental algorithm for the problem that handles updates in $\widetilde{O}(mn^{4/3}\log W/\epsilon)$ total time (where the edge weights are from $[1, W]$) and explicitly maintains a $(1+\epsilon)$-approximate distance matrix. For a fixed $\epsilon > 0$, this is the first deterministic partially dynamic algorithm for all-pairs shortest paths in directed graphs, whose update time is $o(n^2)$ regardless of the number of edges. Furthermore, we also show how to improve the state-of-the-art partially dynamic *randomized* algorithms for all-pairs shortest paths [Baswana et al. STOC'02, Bernstein STOC'13] from Monte Carlo randomized to Las Vegas randomized without increasing the running time bounds (with respect to the $\widetilde{O}(\cdot)$ notation).

Our results are obtained by giving new algorithms for the problem of dynamically maintaining *hubs*, that is a set of $\widetilde{O}(n/d)$ vertices which hit a shortest path between each pair of vertices, provided it has hop-length $\Omega(d)$. We give new subquadratic deterministic and Las Vegas algorithms for maintenance of hubs under either edge insertions or deletions.

2012 ACM Subject Classification Theory of computation → Dynamic graph algorithms

Keywords and phrases shortest paths, dynamic, incremental, decremental, directed graphs, hubs

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.65

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.02266`.

Funding *Adam Karczmarz*: Supported by ERC Consolidator Grant 772346 TUgbOAT and the Polish National Science Centre 2018/29/N/ST6/00757 grant.

1 Introduction

The sampling scheme of Ullman and Yannakakis [27] is a fundamental tool in designing dynamic algorithms for maintaining shortest path distances. Roughly speaking, the main idea is that if each vertex of the graph is sampled independently with probability $\Omega(\frac{d\ln n}{n})$, then with high probability[1] the set of the sampled vertices has the following property. If the shortest path between some vertices u and v contains more than d edges, then this shortest path contains a sampled vertex[2]. We call each set having this property a set of *hubs*[3] of that graph.

[1] We say that a probabilistic statement holds *with high (low) probability.* abbreviated w.h.p., if it holds with probability at least $1-n^{-\beta}$ (at most $n^{-\beta}$, resp.), where β is a constant that can be fixed arbitrarily.
[2] For simplicity, in the introduction we assume that the shortest paths are unique.
[3] Zwick [29] uses the name *bridging set* for an analogous concept. Some works also use the term *hitting set*, but hitting set is a more general notion, which in our paper is used in multiple different contexts.

The fact that one can easily obtain a set of hubs by random sampling is particularly useful for dynamic graph algorithms, since, by tuning constants in the sampling probability, one can assure that the set of hubs remains valid at each step (with high probability), while the graph is undergoing edge insertions and deletions, assuming the total number of updates is polynomial. This property has been successfully exploited to give a number of dynamic graph algorithms, e.g. [2, 4, 5, 9, 12, 11, 13, 20, 21, 22]. At the same time, the sampling approach also suffers from two drawbacks. First, it yields Monte Carlo algorithms, which with some nonzero probability can return incorrect answers. Second, it relies on the *oblivious adversary* assumption, that is, it requires that the updates to the graph are independent of the randomness used for sampling hubs. This becomes a substantial issue for problems where the answer to a query is not unique, e.g., for maintaining $(1+\epsilon)$-approximate distances or maintaining the shortest paths themselves (i.e. not just their lengths). In a typical case, the choice of the specific answer to a query depends on the randomness used for vertex sampling, which in turn means that in each answer to a query the data structure is revealing its randomness. Hence, if the following updates to the data structure depend on the specific values returned by the previous queries, the oblivious adversary assumption is not met.

In this paper we attempt to address both these issues. We study the dynamic maintenance of *reliable* hubs, that is we show how to maintain hubs using an algorithm that does not err, even with small probability. In addition, in the incremental setting we give an algorithm that maintains hubs deterministically. While the algorithms are relatively straightforward for unweighted graphs, making them also work in the weighted setting is a major challenge, which we manage to overcome. We then show how to take advantage of our results on reliable hubs to obtain improved algorithms for the problem of maintaining all-pairs shortest paths in directed graphs. In particular, we give a faster deterministic incremental algorithm and show how to improve the state-of-the-art decremental algorithms from Monte Carlo to Las Vegas.

1.1 Our Contribution

We study the problem of maintaining *reliable* hub sets in the partially dynamic setting. For the description, let us first assume the case when the graph is unweighted. Our first observation is that one can deterministically maintain the set of hubs H_d under edge insertions in $\widetilde{O}(nmd)$ total time. To that end, we observe that after an edge uw is inserted, we may ensure the set of hubs H_d is valid by extending it with both u and w. This increases the size of H_d, and hence we have to periodically discard all the hubs and recompute them from scratch.

The deterministic computation of hubs has been studied before. For unweighted digraphs, King [19] showed how to compute a hub set H_d of size $\widetilde{O}\left(\frac{n}{d}\right)$ in $\tilde{O}(n^2)$ time. The algorithm, given shortest path trees up to depth d from all vertices $v \in V$, computes a *blocker-set* [19] of these trees. (A blocker-set S of a rooted tree is a set such that, for each path from the root to a leaf of length d, that path contains a vertex of S distinct from the root.) Hence, if we work on unweighted graphs, in order to keep the set H_d valid and relatively small, we can maintain shortest path trees up to depth d from all vertices using the Even-Shiloach algorithm [10] in $O(nmd)$ total time, and recompute H_d using King's algorithm every $\widetilde{O}(\frac{n}{d})$ insertions. The total time needed for maintaining the hubs is therefore $\widetilde{O}(nmd)$.

Furthermore, we also show how to maintain reliable hubs in a decremental setting. Suppose our goal is to compute a set of hubs that is guaranteed to be valid, which clearly is not the case for the sampled hubs of [27]. We show that if shortest path trees up to depth d are maintained using *dynamic tree* data structures [24, 25], one can recompute a certainly-valid set H_d in $\widetilde{O}\left(\frac{n^2}{d}\right)$ time using a Las Vegas algorithm. To this end observe that one can deterministically *verify* if a set $B \subseteq V$ is a blocker-set of n shortest path trees up to depth d in $\widetilde{O}(n \cdot |B|)$ time. Therefore, a hub set H_d can be found by combining the approaches of [27] and [19]: we may sample candidate hub sets of size $\widetilde{O}(\frac{n}{d})$ until a blocker-set of the trees is found. The number of trials is clearly constant with high probability.

We further extend this idea and show that the information whether B is a blocker-set of a collection of n shortest path trees up to depth d can be maintained subject to the changes to these trees with only polylogarithmic overhead. Consequently, we can detect when the sampled hub set H_d (for any d) ceases to be a valid hub set in $\widetilde{O}(nmd)$ total time. The algorithm may make one-sided error (i.e., say that H_d is no longer a valid hub set when it is actually still good), but the probability of an error is low if we assume that the update sequence does not depend on our random bits. Subsequently we show how to extend this idea to improve the total update time to $\widetilde{O}(nm)$. Assume we are given a valid d-hub set H_d. We prove that in order to verify whether H_{6d} is a valid $6d$-hub set, it suffices to check whether it hits sufficiently long paths between the elements of H_d. We use this observation to maintain a family of reliable hub sets $H_1, H_6, \ldots, H_{6^i}, \ldots, H_{6^k}$ (where $6^k \leq n$) under edge deletions (or under edge insertions) in $\widetilde{O}(nm)$ total time. Using that, we immediately improve the state-of-the-art decremental APSP algorithms of Baswana et al. [4] (for the exact unweighted case) and Bernstein [5] (for the $(1+\epsilon)$-approximate case) from Monte Carlo to Las Vegas (but still assuming an oblivious adversary) by only adding a polylogarithmic factor to the total update time bound.

Generalization to weighted digraphs. Adapting the reliable hub sets maintenance (for both described approaches: the incremental one and sample/verify) to *weighted* digraphs turns out to be far from trivial. This is much different from the sampling approach of Ullman and Yannakakis [27], which works regardless of whether the input graph is weighted or not. The primary difficulty is maintaining all shortest paths consisting of up to d edges. While in the unweighted case the length of a path is equal to the number of edges on this path, this is no longer true in the weighted case.

To bypass this problem we first relax our definition of hubs. For each $u, v \in V$ we require that some $(1+\epsilon)$-approximate shortest $u \to v$ path contains a hub on each subpath consisting of at least $d+1$ edges. Next, we show that running King's blocker-set algorithm on a set of $(1+\epsilon)$-approximate shortest path trees up to depth[4] d from all vertices of the graph yields a hub set that hits paths approximating the true shortest paths within a factor of $(1+\epsilon)^{\Theta(\log n)}$. Note that a collection of such trees can be maintained in $\widetilde{O}(nmd \log W/\epsilon)$ total time subject to edge insertions, using Bernstein's h-SSSP algorithm [5] with $h = d$.

The $\Theta(\log n)$ exponent in the approximation ratio comes from the following difference between the weighted and unweighted case. In a $(1+\epsilon)$-approximate shortest path tree up to depth d, the length of a $u \to v$ path is no more than $(1+\epsilon)$-times the length of the shortest $u \to v$ path in G that uses at most d edges. However, the $u \to v$ path in the tree might consist of any number of edges, in particular very few. Pessimistically, all these trees have depth $o(d)$ and their blocker-set is empty, as there is no path of hop-length $\Omega(d)$ that we need to hit. Note that this is an inherent problem, as the fact that we can find a small blocker-set in the unweighted case relies on the property that we want it to hit paths of $\Omega(d)$ edges.

[4] In such a tree (see Definition 23), which is a subgraph of G, for all $v \in V$, the path from the source s to v has length not exceeding $(1+\epsilon)$ times the length of a shortest out of $s \to v$ paths in G that use no more than d edges; however the tree path itself can have arbitrary number of edges.

Luckily, a deeper analysis shows that our algorithm can still approximate the length of a $s \to v$ path. Roughly speaking, we split the $s \to v$ path P into two subpaths of $d/2$ edges. If each of these two subpaths are approximated in the h-SSSP data structures by paths of less than $d/4$ edges, we replace the P by the concatenation of the two approximate paths from the h-SSSP data structures. This way, we get a path that can be longer by a factor of $(1+\epsilon)$, but whose hop-length is twice smaller. By repeating this process $O(\log n)$ times we obtain a path of constant hop-length whose length is at most $(1+\epsilon)^{\Theta(\log n)}$ larger than the length of P. The overall approximation ratio is reduced to $(1+\epsilon)$ by scaling ϵ by a factor of $\Theta(\log n)$.

Deterministic incremental all-pairs shortest paths. We now show how to apply our results on reliable hubs to obtain an improved algorithm for incremental all-pairs shortest paths problem in weighted digraphs. We give a deterministic incremental algorithm maintaining all-pairs $(1+\epsilon)$-approximate distance estimates in $\widetilde{O}(mn^{4/3}\log W/\epsilon)$ total time.

Let us now give a brief overview of our algorithm in the unweighted case. First, we maintain the set of hubs H_d under edge insertions as described above in $\widetilde{O}(nmd)$ total time. Second, since the set H_d changes and each vertex of the graph may eventually end up in H_d, we cannot afford maintaining shortest path trees from all the hubs (which is done in most algorithms that use hubs). Instead, we use the folklore $\widetilde{O}(n^3/\epsilon)$ total time incremental $(1+\epsilon)$-approximate APSP algorithm [5, 19] to compute distances between the hubs. Specifically, we run it on a graph whose vertex set is H_d and whose edges represent shortest paths between hubs of hop-lengths at most d. These shortest paths are taken from the shortest path trees up to depth d from all $v \in V$ that are required for the hub set maintenance. We reinitialize the algorithm each time the set H_d is recomputed. This allows us to maintain approximate pairwise distances between the hubs at all times in $\widetilde{O}\left(m(n/d)^2/\epsilon\right)$ total time.

Finally, we show how to run a dynamic algorithm on top of a *changing* set of hubs by adapting the shortcut edges technique of Bernstein [5]. Roughly speaking, the final estimates are maintained using $(1+\epsilon)$-approximate shortest path trees [5] up to depth $O(d)$ from all vertices v on graph G augmented with shortcuts from v to H_d and from H_d to v. This poses some technical difficulties as the set of shortcuts is undergoing both insertions (when a hub is added) and deletions (when the entire set of hubs is recomputed from scratch). However, one can note that in the incremental setting the shortcuts that no longer approximate the distances between their endpoints do not break the approximation guarantee of our algorithm. Eventually, we use shortcuts between all pairs of vertices of G but only some of them are guaranteed (and sufficient) to be up to date at any time. The total time cost of maintaining this component is $\widetilde{O}(nmd/\epsilon)$. Setting $d = \widetilde{O}(n^{1/3})$ gives the best update time.

It is natural to wonder if this approach could be made to work in the decremental setting. There are two major obstacles. First, it is unclear whether one can deterministically maintain a valid set of hubs under deletions so that only $O(1)$ vertices (in amortized sense) are added to the hub set after each edge deletion. Note that in extreme cases, after a single edge deletion the set of hubs may have to be extended with polynomially many new vertices. Second, all algorithms using the above approach of introducing shortcuts from and to hubs also maintain a decremental shortest path data structure on a graph consisting of the edges of the original graph and shortcut edges representing distances between the hubs. If hubs were to be added, the graph maintained by the data structure would undergo both insertions (of shortcuts) and deletions (of edges of the original graph) which would make this a much harder, fully dynamic problem. Some earlier works dealt with a similar issue by ignoring some "inconvenient" edge insertions [14] or showing that the insertions are well-behaved [6]. However, these approaches crucially depended on the graph being undirected.

1.2 Related Work

The dynamic graph problems on digraphs are considerably harder than their counterparts on undirected graphs. An extreme example is the dynamic reachability problem, that is, transitive closure on directed graphs, and connectivity on undirected graphs. While there exist algorithms for undirected graphs with polylogarithmic query and update times [17, 28, 26, 16, 18], in the case of directed graphs the best known algorithm with polylogarithmic query time has an update time of $O(n^2)$ [23, 7, 20]. In addition, a combinatorial algorithm with an update time of $O(n^{2-\epsilon})$ is ruled out under Boolean matrix multiplication conjecture [1].

In 2003, in a breakthrough result Demetrescu and Italiano gave a fully dynamic, exact and deterministic algorithm for APSP in weighted directed graphs [8]. The algorithm handles updates in $\widetilde{O}(n^2)$ amortized time and maintains the distance matrix explicitly. The bound of $O(n^2)$ is a natural barrier as a single edge insertion or deletion may change up to $\Omega(n^2)$ entries in the distance matrix. For dynamic APSP in digraphs there exists faster algorithms with polylogarithmic query time, all of which work in incremental or decremental setting:

- Ausiello et al. [3] gave a deterministic incremental algorithm for exact distances in unweighted digraphs that handles updates in $\widetilde{O}(n^3)$ total time.
- Baswana et al. [4] solved the same problem in the decremental setting with a Monte Carlo algorithm with $\widetilde{O}(n^3)$ total update time.
- Bernstein [5] gave a Monte Carlo algorithm for $(1+\epsilon)$-approximate distances in weighted graphs (with weights in $[1, W]$) with $\widetilde{O}(nm \log W/\epsilon)$ total update time. The algorithm works both in the incremental and decremental setting.
- Finally, deterministic partially-dynamic (both incremental and decremental) algorithms for APSP in directed graphs with $\widetilde{O}(n^3 \log W/\epsilon)$ total update time can be obtained by combining the results of [19] and [5].

The algorithms of Baswana et al. [4] and Bernstein [5] both use sampled hubs and thus require the *oblivious adversary* assumption. We highlight that in the class of deterministic algorithms, the best known results have total update time $\widetilde{O}(n^3)$ [3, 5], even if we only consider sparse unweighted graphs in incremental or decremental setting and allow $(1+\epsilon)$ approximation. In the incremental setting, for not very dense graphs, when $m = O(n^{5/3-\epsilon})$, our algorithm improves this bound to $\widetilde{O}(mn^{4/3})$.

Organization of the paper. In Section 2 we fix notation, review some of the existing tools that we use and give a formal definition of hubs. Section 3 describes the hub set maintenance for incremental unweighted digraphs and our $(1+\epsilon)$-approximate incremental algorithm for sparse graphs. In Section 4 we show a faster Las Vegas algorithm for computing reliable hubs and further extend it to maintain reliable hub sets in the partially dynamic setting. There we also sketch how to use it in order to to improve the state-of-the-art decremental APSP algorithms from Monte Carlo to Las Vegas randomized. Finally, in Section 5 we sketch how to adapt the hub set maintenance algorithms of Sections 3 and 4, so that they work on weighted graphs. Due to limited space, many proofs and details can only be found in the full version of this paper.

2 Preliminaries

In this paper we deal with *directed* graphs. We write $uv \in E(G)$ when referring to edges of G and use $w_G(uv)$ to denote the weight of uv. If G is unweighted, then $w_G(e) = 1$ for each $e \in E$. For weighted graphs, $w_G(e)$ can be any real number from the interval $[1, W]$. For simplicity, in this paper we assume that W is an input parameter given beforehand. If $uv \notin E$, we assume

$w_G(uv)=\infty$. We define the union $G\cup H$ to be the graph $(V(G)\cup V(H),E(G)\cup E(H))$ with weights $w_{G\cup H}(uv)=\min(w_G(uv),w_H(uv))$ for each $uv\in E(G\cup H)$. For an edge $e=uv$, we write $G+e$ to denote $(V(G)\cup\{u,v\},E(G)\cup\{e\})$. The *reverse graph* G^{R} is defined as $(V(G),\{xy:yx\in E(G)\})$ and $w_{G^{\mathrm{R}}}(xy)=w_G(yx)$.

A sequence of edges $P=e_1\dots e_k$, where $k\geq 1$ and $e_i=u_iv_i\in E(G)$, is called a $u\to v$ path in G if $u=u_1$, $v_k=v$ and $v_{i-1}=u_i$ for each $i=2,\dots,k$. We sometimes view a path P in G as a subgraph of G with vertices $\{u_1,\dots,u_k,v\}$ and edges $\{e_1,\dots,e_k\}$ and write $P\subseteq G$. The *hop-length* $|P|$ is defined as $|P|=k$. The *length* of the path $\ell(P)$ is defined as $\ell(P)=\sum_{i=1}^{k}w_G(e_i)$. If G is unweighted, then we clearly have $|P|=\ell(P)$. For convenience, we sometimes consider a single edge uv a path of hop-length 1. It is also useful to define a length-0 $u\to u$ path to be the graph $(\{u\},\emptyset)$. If P_1 is a $u\to v$ path and P_2 is a $v\to w$ path, we denote by $P_1\cdot P_2$ (or simply P_1P_2) a path $P_1\cup P_2$ obtained by concatenating P_1 with P_2.

A digraph T is called an *out-tree over V rooted in r* if $v\in V(T)\subseteq V$, $|E(T)|=|V(T)|-1$ and for all $v\in V(T)$ there is a unique path $T[v]$ from r to v. The depth $\mathrm{dep}_T(v)$ of a vertex $v\in V(T)$ is defined as $|T[v]|$. The depth of T is defined as $\max_{v\in V(T)}\{\mathrm{dep}_T(v)\}$. Each non-root vertex of an out-tree has exactly one incoming edge. For $v\in V(T)\setminus\{r\}$ we call the other endpoint of the incoming edge of v the *parent* v and write $\mathrm{par}_T(v)$ when referring to it.

The *distance* $\delta_G(u,v)$ between the vertices $u,v\in V(G)$ is the length of the shortest $u\to v$ path in G, or ∞, if no $u\to v$ path exists in G. We define $\delta^k_G(u,v)$ to be the length of the shortest path from u to v among paths of at most k edges. Formally, $\delta^k_G(u,v)=\min\{\ell(P): u\to v=P\subseteq G \text{ and } |P|\leq k\}$. We sometimes omit the subscript G and write $w(uv)$, $\delta(u,v)$, $\delta^k(u,v)$ etc. instead of $w_G(u,v)$, $\delta_G(u,v)$, $\delta^k_G(u,v)$, etc., respectively.

We say that a graph G is *incremental*, if it only undergoes edge insertions and edge weight decreases. Similarly, we say that G is *decremental* if it undergoes only edge deletions and edge weight increases. We say that G is *partially dynamic* if it is either incremental or decremental. For a dynamic graph G we denote by n the maximum value of $|V|$ and by m the maximum value of $|E|$ throughout the whole sequence of updates.

We denote by Δ the total number of updates a dynamic graph G is subject to. If G is unweighted, then clearly $\Delta\leq m$ and in fact we assume $\Delta=m$, which allows us to simplify the analyses. For weighted digraphs, on the other hand, since the total number of weight increases/decreases that an edge is subject to is unlimited, Δ may be much larger than m. As a result, it has to be taken into account when analyzing the efficiency of our algorithms.

We call a partially-dynamic $(1+\epsilon)$-approximate APSP problem on weighted graphs *restricted* if the edge weights of G are of the form $(1+\epsilon)^i$ for $i\in[0,\lceil\log_{1+\epsilon}W\rceil]\cap\mathbb{N}$ at all times and additionally each update is required to actually change the edge set or change the weight of some existing edge. Consequently, observe that in the restricted problem we have $\Delta\leq m\cdot(\lceil\log_{1+\epsilon}W\rceil+2)$. In the following we concentrate on the restricted problem. This is without much loss of generality – we provide a reduction in the full version of the paper.

Partially-dynamic single-source shortest path trees.

▶ **Definition 1.** *Let $G=(V,E)$ be an unweighted digraph and let $s\in V$. Let $d>0$ be an integer. We call an out-tree $T\subseteq G$ rooted in s a shortest path tree from s up to depth d if: (a) for any $v\in V$, $v\in V(T)$ iff $\delta_G(s,v)\leq d$, and (b) for any $v\in V(T)$, $\delta_T(s,v)=\delta_G(s,v)$.*

▶ **Theorem 2** (Even-Shiloach tree [10, 15])**.** *Let $G=(V,E)$ be an unweighted graph subject to partially dynamic edge updates. Let $s\in V$ and let $d\geq 1$ be an integer. Then, a shortest path tree from s up to depth d can be explicitly maintained*[5] *in $O(md)$ total time.*

[5] By this we mean that the algorithm outputs all changes to the edge set of the maintained tree.

▶ **Theorem 3** (h-SSSP [5]). *Let $G = (V, E)$ be a weighted digraph. Let $s \in V$ and let $h \geq 1$ be an integer. There exists a partially dynamic algorithm explicitly maintaining $(1+\epsilon)$-approximate distance estimates $\delta'(s, v)$ satisfying $\delta_G(s, v) \leq \delta'(s, v) \leq (1+\epsilon)\delta^h_G(s, v)$ for all $v \in V$. The total update time of the algorithm $O(mh \log n \log(nW)/\epsilon + \Delta)$.*

Hubs and how to compute them.

▶ **Definition 4.** *Let $G = (V, E)$ be a directed graph. Let $B \subseteq V$ and let $d > 0$ be an integer. We say that a path P in G is* (B, d)-covered *if it can be expressed as $P = P_1 \dots P_k$, where $P_i = u_i \to v_i$, $|P_i| \leq d$ for each $i = 1, \dots, k$, and $u_i \in B$ for each $i = 2, \dots, k$.*

We now define a blocker set, slightly modifying a definition by King [19].

▶ **Definition 5.** *Let V be a vertex set and let d be a positive integer. Let $B \subseteq V$ and let T be a rooted tree over V of depth no more than d. We call B a (T, d)-blocker set if for each $v \in V(T)$ such that $\text{dep}_T(v) = d$, either v or one of its ancestors in T belongs to B.*

Let $\mathcal{T}$ be a collection of rooted trees over V of depth no more than d. We call B a $(\mathcal{T}, d)$-blocker set if B is a (T, d)-blocker set for each $T \in \mathcal{T}$.

▶ **Lemma 6.** *Let V be a vertex set of size n. Let d be a positive integer. Let $\mathcal{T}$ be a collection of rooted trees over V of depth at most d. Then, a $(\mathcal{T}, d)$-blocker set of size $O\left(\frac{n}{d} \log n\right)$ can be computed deterministically in $O(n \cdot (|\mathcal{T}| + n) \log n)$ time.*

▶ **Definition 7.** *Let $G = (V, E)$ be a directed graph and let $d > 0$ be an integer. A set $H_d \subseteq V$ is called a* d-hub set *of G if for every $u, v \in V$ such that $\delta_G(u, v) < \infty$, there exists some shortest $u \to v$ path that is (H_d, d)-covered.*

▶ **Lemma 8.** *Let $G = (V, E)$ be a directed* unweighted *graph and let $d > 0$ be an integer. Suppose we are given a collection $\mathcal{T} = \{T_v : v \in V\}$ of shortest path trees up to depth d from all vertices of G. Let B be a $(\mathcal{T}, d)$-blocker set. Then B is a $2d$-hub set of G.*

Deterministic incremental algorithm for dense graphs.

▶ **Theorem 9** ([19]+[5]). *There exist an incremental algorithm maintaining all-pairs $(1+\epsilon)$-approximate distance estimates of a digraph in $\widetilde{O}(n^3 \log(W)/\epsilon) + O(\Delta)$ total time.*

As mentioned before, the above theorem basically follows by combining the partially dynamic transitive closure algorithm of King [19] with Bernstein's h-SSSP algorithm (Theorem 3) for $h = 2$.

3 Deterministic Incremental Algorithm for Sparse Graphs

In this section we present our deterministic incremental algorithm with $\widetilde{O}(mn^{4/3}/\epsilon)$ total update time. We first observe that whenever an edge xy is added, the set of hubs may be "fixed" by extending it with both x and y.

▶ **Lemma 10.** *Let $G = (V, E)$ be a directed unweighted graph. Let H_d be a d-hub set of G. Let $x, y \in V$ be such that $xy \notin E$. Then $H'_d = H_d \cup \{x, y\}$ is a d-hub set of $G' = G + xy$.*

Let $d > 1$ be an even integer and let ϵ_1, $0 < \epsilon_1 < \epsilon$ be a real number, both to be set later. Our data structure consists of several components. Each subsequent component builds upon the previously defined components only.

Exact shortest paths between nearby vertices. The data structure maintains two collections $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in V\}$ and $\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in V\}$ of shortest path trees up to depth $\frac{d}{2}$ in G and G^{R}, resp. By Theorem 2, each tree of $\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}$ can be maintained under edge insertions in $O(md)$ total time. The total time spent in this component is hence $O(nmd)$.

The hubs. A d-hub set H_d of both G and G^{R} such that $|H_d| = O\big(\frac{n}{d}\log n\big)$ is maintained at all times, as follows. Initially, H_d is computed in $O(n^2 \log n)$ time using Lemma 6 and the trees of $\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}$ (see Lemma 8). Next, the data structure operates in phases. Each phase spans $f = \Theta(\frac{n}{d}\log n)$ consecutive edge insertions. When an edge xy is inserted, its endpoints are inserted into H_d. By Lemma 10, this guarantees that H_d remains a d-hub set of both G and G^{R} after the edge insertion. Once f edges are inserted in the current phase, the phase ends and the hub set H_d is recomputed from scratch, again using Lemma 6. Observe that the size of $|H_d|$ may at most triple within each phase.

The total time spent on maintaining the set H_d is clearly $O\Big(\frac{m}{f} \cdot n^2 \log n\Big) = O(nmd)$.

Approximate shortest paths between the hubs. In each phase, we maintain a *weighted* graph $A = (H_d, E_A)$, where $E_A = \{uv : u, v \in H_d, \delta_{G^{\text{R}}}(u,v) \le d\}$ and $w_A(uv) = \delta_{T_u^{\text{to}}}(u,v) = \delta_{G^{\text{R}}}(u,v) \le d$. Observe that during each phase, the graph A is in fact incremental. We can thus maintain $(1+\epsilon_1)$-approximate distance estimates $\delta'_A(u,v)$ for all $u, v \in H_d$ in $\widetilde{O}(|H_d|^3/\epsilon_1) = \widetilde{O}\Big(\big(\frac{n}{d}\big)^3/\epsilon_1\Big)$ total time per phase, using a data structure $\mathcal{D}_A$ of Theorem 9.[6]

Summing over all phases, the total time spent on maintaining the $(1+\epsilon')$-approximate distances estimates of the graph A is $\widetilde{O}\Big(m\big(\frac{n}{d}\big)^2/\epsilon_1\Big)$.

▶ **Lemma 11.** *For any $u, v \in H_d$, $\delta_{G^{\text{R}}}(u,v) = \delta_A(u,v)$.*

By the above lemma, for each $u, v \in H_d$ we actually have $\delta'_A(u,v) \le (1+\epsilon_1)\delta_{G^{\text{R}}}(u,v)$.

Shortcuts to hubs. For each $u \in V$, let S_u be a graph on V with exactly n edges $\{uv : v \in V\}$ satisfying $w_{S_u}(u,v) \ge \delta_{G^{\text{R}}}(u,v)$ for all $v \in V$, and additionally $w_{S_u}(u,v) \le (1+\epsilon_1)\delta_{G^{\text{R}}}(u,v)$ if u, v both currently belong to H_d. The edges between vertices of H_d are the only ones that our algorithm needs to compute approximate distances. For other edges we only need to make sure they will not cause the algorithm to underestimate the distances.

Observe that the graphs S_u can be maintained using the previously defined components as follows. First, they are initialized so that their edges are all infinite-weight. Whenever the data structure $\mathcal{D}_A$ changes (or initializes) some of its estimates $\delta'_A(u,v) \le (1+\epsilon_1)\delta_{G^{\text{R}}}(u,v)$, we perform $w_{S_u}(u,v) := \min(w_{S_u}(u,v), \delta'_A(u,v))$. This guarantees that the invariants posed on S_u are always satisfied and S_u is incremental. The total number of updates to all graphs S_u is equal to the number of estimate updates made by $\mathcal{D}_A$ and thus can be neglected.

For $u \in V$, we set up a h-SSSP data structure $\mathcal{D}_u$ of Theorem 3 for the graph $G^{\text{R}} \cup S_u$ with source vertex u and $h = d+1$. Hence, $\mathcal{D}_u$ maintains distance estimates $\delta'(u,v)$ such that $\delta'(u,v) \le (1+\epsilon')\delta^{d+1}_{G^{\text{R}} \cup S_u}(u,v)$. As the graph $G^{\text{R}} \cup S_u$ is incremental and has $O(m)$ edges, the total time that $\mathcal{D}_u$ needs to operate is $\widetilde{O}(md/\epsilon_1) + O(\Delta_u)$, where Δ_u is the total number of updates to $G^{\text{R}} \cup S_u$. Summing the update times for all data structures $\mathcal{D}_u$, we obtain

[6] Technically speaking, the total update time of the data structure of Theorem 9 is $\widetilde{O}(n^3/\epsilon') + O(\Delta)$. However, all updates to $\mathcal{D}_A$ arise when some previous component updates its explicitly maintained estimates, so the Δ term is asymptotically no more than the total update time of the previously defined components and can be charged to those. In the following, we omit Δ terms like this without warnings.

$\widetilde{O}(nmd/\epsilon_1) + O(\sum_{v\in V} \Delta_u)$ total time. Note that $\sum_{u\in V} \Delta_u$ equals nm plus the number of updates to the graphs S_u, which can be charged to the operating cost of data structure $\mathcal{D}_A$, as argued before. We conclude that the total update time of all $\mathcal{D}_u$ is $\widetilde{O}(nmd/\epsilon_1)$.

Observe that a shortest $u \to v$ path in G^{R}, where $u \in H_d$ and $v \in V$ is approximated by a path in $G^{\mathrm{R}} \cup S_u$ consisting of at most $d+1$ edges. The first edge belongs to S_u and "jumps" to some hub. The latter (at most d) edges belong to G^{R}. This is formalized as follows.

▶ **Lemma 12.** *Let $u \in H_d$ and $v \in V$. Then $\delta^{d+1}_{G^{\mathrm{R}}\cup S_u}(u,v) \le (1+\epsilon_1)\delta_{G^{\mathrm{R}}}(u,v)$.*

By the above lemma, we conclude that for $u \in H_d$, $v \in V$, the estimate $\delta'(u,v)$ produced by the data structure $\mathcal{D}_v$ satisfies $\delta'(u,v) \le (1+\epsilon_1)\delta^{d+1}_{G^{\mathrm{R}}\cup S_u}(u,v) \le (1+\epsilon_1)^2\delta_{G^{\mathrm{R}}}(u,v)$.

All-pairs approximate shortest paths. We maintain another set of shortcut graphs R_u, for $u \in V$. Again R_u has exactly n edges $\{uv : v \in V\}$ whose weights satisfy $w_{R_u}(uv) \ge \delta_G(u,v)$ for all v and $w_{R_u}(uv) \le (1+\epsilon_1)^2\delta_G(u,v)$ if $v \in H_d$. Each graph R_u is maintained using the previously defined data structures $\mathcal{D}_v$. Initially all weights of R_u are infinite. Whenever some $\mathcal{D}_v$ changes the estimate $\delta'(v,u)$, we set $w_{R_u}(uv) := \min(w_{R_u}(uv), \delta'(v,u))$. Since for $v \in H_d$ we have $\delta'(v,u) \le (1+\epsilon_1)^2\delta_{G^{\mathrm{R}}}(v,u)$, equivalently, $\delta'(v,u) \le (1+\epsilon_1)^2\delta_G(u,v)$ and we obtain $w_{R_u}(uv) \le (1+\epsilon_1)^2\delta_G(u,v)$. Therefore, the graphs R_u are all incremental and the total number of changes they are subject to is no more than the total number of estimate changes made by the data structures $\mathcal{D}_v$, $v \in V$. Thus, we may neglect the cost of actually performing these changes.

Finally, for each $u \in V$ we set up a h-SSSP data structure $\mathcal{D}'_u$ of Theorem 3 on graph $G \cup R_u$ with source u and $h = d+1$, maintaining $(1+\epsilon_1)$-approximate estimates of the values $\delta^{d+1}_{G\cup R_u}(u,\cdot)$. Similarly as was the case for the data structures $\mathcal{D}_u$ of the previous component, as the graphs $G \cup R_u$ are incremental, the total operating time of the h-SSSP instances running on the graphs $G \cup R_u$ is $\widetilde{O}(nmd/\epsilon_1)$.

▶ **Lemma 13.** *Let $u, v \in V$. Then $\delta^{d+1}_{G\cup R_u}(u,v) \le (1+\epsilon_1)^2\delta_G(u,v)$.*

By the above lemma, the the distance estimates $\delta''(u,v)$ maintained by the data structure $\mathcal{D}'_u$, approximate the corresponding distances $\delta_G(u,v)$ within a factor of $(1+\epsilon_1)^3$.

▶ **Theorem 14.** *Let G be a directed unweighted graph. There exists a deterministic incremental algorithm maintaining $(1+\epsilon)$-approximate distance estimates between all pairs of vertices of G in $\widetilde{O}(mn^{4/3}/\epsilon)$ total time.*

4 Partially-Dynamic Verification of a Sampled Hub Set

In this section we show how to maintain the information whether a sampled set remains a hub set of an unweighted digraph G subject to partially dynamic updates. For simplicity, assume that G is decremental (the incremental case, being somewhat easier, can be handled similarly). We start by showing how a reliable hub set can be found if we are given shortest path trees up to depth d from all vertices of G, stored in dynamic tree data structures.

▶ **Lemma 15** ([27, 29]). *Let V be a vertex set of size n and let $d > 0$ be an integer. Let $\mathcal{T}$ be a collection of rooted trees of depth no more than d, whose vertex sets are subsets of V.*

Let $c > 1$ be some constant. Let B be a random subset of V of size $\min\left(\lceil c\frac{n}{d}\ln n\rceil, n\right)$. Then, B is a $(\mathcal{T}, d)$-blocker set with probability at least $\max(0, 1 - |\mathcal{T}|/n^{c-1})$.

The following data structure is an easy application of the data structure of Tarjan [25].

▶ **Lemma 16.** *Let V be some set of n vertices. Let F be a forest of (initially single-vertex) rooted out-trees over V such that the vertex sets of the individual trees of F form a partition of V. For $v \in V$, let $T_v \in F$ denote the unique tree of F containing v.*

There exists a data structure for dynamically maintaining F (initially consisting of n 1-vertex trees) and supporting the following operations in $O(\log n)$ time each:

1. `parent`(v)*: if v is not the root of T_v, return its parent. Otherwise return* **nil**.
2. `link`(u, v)*: assuming $T_u \neq T_v$ and that u is the root of T_u, make u a child of v in T_v.*
3. `cut`(v)*: assuming v is not the root of T_v, split T_v into two trees by removing the edge between v and its parent.*
4. `depth`(v)*: return the depth of the tree T_v.*

▶ **Lemma 17.** *Let V be a vertex set, $n = |V|$, and let $d > 0$ be integral. Let $\mathcal{T}$ be a collection of rooted trees over V of depth no more than d, where $|\mathcal{T}| = O(\text{poly}\, n)$. Suppose each $T \in \mathcal{T}$ is given as a separate data structure of Lemma 16 and for each $T \in \mathcal{T}$, $\text{root}(T)$ is known.*

Then, there exists a Las Vegas randomized algorithm computing a $(\mathcal{T}, d)$-blocker set B of size $O\left(\frac{n}{d}\log n\right)$ in $O(|\mathcal{T}| \cdot \frac{n}{d} \cdot \log^2 n)$ time with high probability.

Proof. Let $|\mathcal{T}| = O(n^\alpha)$ for some $\alpha > 0$. The algorithm is to simply repeatedly pick random subsets B of V of size $\min(\lceil(\alpha + 2)\frac{n}{d}\ln n\rceil, n)$ until B succeeds in being a $(\mathcal{T}, d)$-blocker set. By Lemma 15, for a random B, the probability that this is not the case is at most $\frac{1}{n}$. Hence, the probability that we fail finding a $(\mathcal{T}, d)$-blocker set after $k = O(1)$ trials is at most $1/n^k$.

We thus only need to show how to verify whether a set B is actually a $(\mathcal{T}, d)$-blocker set in $O(|\mathcal{T}| \cdot |B| \log n)$ time. Recall that for a single $T \in \mathcal{T}$, if the depth of T is no more than d, then B is a (T, d)-blocker set if the tree T' obtained from T by removing all subtrees rooted in vertices of B, has depth less than d. Consequently, to verify whether B is a (T, d)-blocker set, we take advantage of the fact that T is stored in a data structure of Lemma 16.

We first check whether $r = \text{root}(T) \in B$. If this is the case, B is a (T, d)-blocker set in a trivial way. Otherwise, for each $b \in B$, we store $p_b =$ `parent`(b) and perform `cut`(b). Afterwards, one can see that B is a (T, d)-blocker set if and only if `depth`$(r) < d$. Finally, we revert all the performed `cut` operations by running `link`(b, p_b) for all $b \in B$.

Clearly, the time needed to verify whether B is a (T, d)-blocker set for any $T \in \mathcal{T}$, is $O(|B| \log n)$. Hence, one can check whether B is a $(\mathcal{T}, d)$-blocker set in $O(|\mathcal{T}| \cdot |B| \log n)$ time. ◀

Now we move on to the problem of detecting when a sampled set ceases to be a valid hub set of G. In fact, our algorithm will solve a bit more general problem (which is anyway needed for applications, as we will see later), as follows.

Let $|V| = n = a_0 > a_1 > \ldots > a_q = 1$ be some sequence of integers such that $a_i \mid a_{i-1}$. For each $i = 0, \ldots, q$, let A_i be a random a_i-subset (a subset of size a_i) of V. By Lemmas 8 and 15, each A_i is in fact an $\Theta((n/a_i)\ln n)$-hub set of G with high probability.

We would like to detect when some A_i ceases to be an $\Theta((n/a_i)\ln n)$-hub set of G while G undergoes edge deletions. Using this terminology, both state-of-the-art Monte-Carlo randomized algorithms for decremental exact shortest paths [4] and partially-dynamic $(1 + \epsilon)$-approximate shortest paths [5] (for unweighted digraphs) use randomness only for constructing hub sets $A_0, \ldots, A_q$ (they use $a_i = 2^{q-i}$, but in fact any $a_i = c^{q-i}$, where c is a positive integer, would be sufficient for these algorithms to work), valid simultaneously for all versions of the input graph with high probability (the sets A_i satisfy this, as we will later show).

Without loss of generality, we can assume that given the sets $A_0, \ldots, A_q$, the algorithms of [4, 5] proceed deterministically. Suppose we develop an efficient partially dynamic algorithm $\mathcal{A}$ *verifying* whether each A_i remains a $\Theta((n/a_i)\ln n)$-hub set of G (i.e., $\mathcal{A}$ is supposed to detect that some A_i ceases to be a $\Theta((n/a_i)\ln n)$-hub set immediately after this happens) and producing *false negatives* with low probability (the algorithm is guaranteed to be correct if it says that all A_i have the desired property but might be wrong saying that some A_i is no longer a hub set). Then, we could use $\mathcal{A}$ to convert the algorithms of [4, 5] into *Las Vegas* algorithms by drawing new sets $A_0, \ldots, A_q$ and restarting the respective algorithms whenever $\mathcal{A}$ detects (possibly incorrectly) that any of these sets ceases to be a hub set. As this does not happen w.h.p., with high probability the overall asymptotic running time remains unchanged. The remainder of this section is devoted to describing such an algorithm $\mathcal{A}$.

▶ **Lemma 18.** *Let $d > 0$ be an integer. Let F be a forest of out-trees of depth no more than d over V. Denote by T_v the unique tree of F containing $v \in V$. Let $B \subseteq V$ be fixed.*

There exists a data structure with update time $O(\log n)$, maintaining the information whether B is a (F, d)-blocker set, subject to updates to F of the following types:

- *cut the subtree rooted in v out of T_v where $v \in V$ and v is not the root of T_v,*
- *make the tree T_r a child of $v \in T_v$ where $r \in V$ is the root of T_r and $v \notin T_r$,*

The following lemma says that in order to test whether a given set of vertices is a $6d$-hub set it suffices to test the hub set property for paths starting in vertices of a d-hub set.

▶ **Lemma 19.** *Let $G = (V, E)$ be a directed unweighted graph. Let H_d be a d-hub set of G. Suppose we are given two collections $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in H_d\}$, $\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in H_d\}$ of shortest path trees up to depth d from all vertices of H_d in G and G^{R}, respectively.*

Let B be a $(\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}, d)$-blocker set. Then B is a $6d$-hub set of G.

Observe that by Lemma 15, there exists an integral constant $z > 0$, such that for any fixed collection of trees $\mathcal{T}$ of depth no more than $z \cdot \frac{n}{a_i}\lceil \ln n \rceil$, where $|\mathcal{T}| = O(n^3)$, A_i is a $\left(\mathcal{T}, z \cdot \frac{n}{a_i}\lceil \ln n \rceil\right)$-blocker set with high probability. For $i = 0, \ldots, q$, set $d_i = z \cdot \frac{n}{a_{i+1}}\lceil \ln n \rceil$ where $a_{q+1} = 1$. Suppose G undergoes partially dynamic updates. For each $i = 1, \ldots, q$, and $v \in V$ let $T_{i,v}^{\text{from}}$ ($T_{i,v}^{\text{to}}$) denote the shortest path tree that the algorithm of Theorem 2 would maintain for $d = d_{i-1}$ and source v in G (in G^{R}, respectively). Note that how the trees $T_{i,v}^{\text{from}}$ and $T_{i,v}^{\text{to}}$ evolve depends only on the sequence of updates to G (which, by the oblivious adversary assumption, does not depend on sets $A_0, \ldots, A_q$ in any way) and the details of the deterministic algorithm of Theorem 2. Since only $O(mn) = O(n^3)$ different trees appear in $\{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}$ throughout all updates, A_i remains a $(\{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}, d_{i-1})$-blocker set throughout the whole sequence of updates with high probability, by Lemma 15.

Let $\mathcal{T}_i^{\text{from}} = \{T_{i,v}^{\text{from}} : v \in A_{i-1}\}$ and $\mathcal{T}_i^{\text{to}} = \{T_{i,v}^{\text{to}} : v \in A_{i-1}\}$, i.e., $\mathcal{T}_i^{\text{from}}$ ($\mathcal{T}_i^{\text{to}}$) contains only trees with roots from a subset $A_{i-1} \subseteq V$. However A_i being a blocker set of such a collection of trees will turn out sufficient for our needs. Clearly, since we have $\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}} \subseteq \{T_{i,v}^{\text{from}} : v \in V\} \cup \{T_{i,v}^{\text{to}} : v \in V\}$, by the above claim, A_i in fact remains a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$-blocker set throughout the whole sequence of updates with high probability.

Now, let $q = \lceil \log_6 n \rceil$ and for $i = 1, \ldots, q$ set $a_i = 6^{q-i}$. To verify whether each A_i remains a d_i-hub set subject to partially dynamic updates to G, we proceed as follows. We deterministically maintain the trees $\bigcup_{i=1}^{q}(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}})$ subject to partially dynamic updates to G using Theorem 2. The total number of changes these trees are subject to throughout the whole sequence of updates is $O(\sum_{i=1}^{q} a_{i-1} \cdot m \cdot d_{i-1}) = O\left(\sum_{i=1}^{q} a_{i-1} \cdot m \cdot \frac{n}{a_i}\ln n\right) = O\left(nm\log n \cdot \sum_{i=1}^{q} \frac{a_{i-1}}{a_i}\right) = \widetilde{O}(nm)$.

We additionally store each tree $T_{i,v}^{\text{from}}$ (and $T_{i,v}^{\text{to}}$), for $v \in A_{i-1}$, in a data structure of Lemma 18 with $B = A_i$. Whenever the data structure of Theorem 2 updates some tree, the update is repeated in the corresponding data structure of Lemma 18. Consequently, the total time needed to maintain these additional data structures is $\widetilde{O}\left(nm \cdot \sum_{i=1}^{q} \frac{a_{i-1}}{a_i}\right) = \widetilde{O}(nm)$.

After each update we can detect whether each A_i is still a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$-blocker set in $O(\sum_i^q |A_{i-1}| \log n) = \widetilde{O}(n)$ time by querying the relevant data structures of Lemma 18[7] storing $\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}$. By Lemma 19, a simple inductive argument shows that if this is the case, each A_i is a d_i-hub set of both G and G^{R}. Hence, verifying all $A_1, \ldots, A_q$ while G evolves takes $\widetilde{O}(mn)$ total time. The algorithm terminates when it turns out that some A_i is no longer a $(\mathcal{T}_i^{\text{from}} \cup \mathcal{T}_i^{\text{to}}, d_{i-1})$-blocker set. However, recall that this happens only with low probability, regardless of whether A_i actually ceases to be a d_i hub set or not. We have proved the following.

▶ **Theorem 20.** *Let G be an unweighted digraph. Let $q = \lceil \log_6 n \rceil$. For $i = 0, \ldots, q$, let A_i be a random 6^{q-i}-subset of V. One can maintain the information whether each A_i is a $\Theta(6^i \ln n)$-hub set of G, subject edge deletions issued to G, in $\widetilde{O}(nm)$ total time.*

The algorithm might produce false negatives with low probability.

By plugging in the hubs of Theorem 20 into the algorithms of [4, 5], we obtain the following.

▶ **Corollary 21.** *Let G be an unweighted digraph. There exists a Las Vegas randomized decremental algorithm maintaining exact distance between all pairs of vertices of G with $\widetilde{O}(n^3)$ total update time w.h.p. It assumes an adversary oblivious to the random bits used.*

▶ **Corollary 22.** *Let G be an unweighted digraph. There exists a Las Vegas randomized decremental algorithm maintaining $(1+\epsilon)$-approximate distance estimates between all pairs of vertices of G in $\widetilde{O}(nm/\epsilon)$ total time w.h.p. The algorithm assumes an oblivious adversary.*

5 Approximate Shortest Paths for Weighted Graphs

In this section we give key definitions used to generalize the reliable hub maintenance algorithms to weighted graphs, at the cost of $(1+\epsilon)$-approximation. Then, we state the main theorem (Theorem 26) relating blocker sets in $(1+\epsilon)$-approximate shortest path trees to the approximate hub sets. Finally, we explain briefly how to incorporate these tools into our improved dynamic APSP algorithms in order to generalize then to weighted graphs.

▶ **Definition 23.** *Let $G = (V, E)$ be a weighted digraph and let $s \in V$ be a source vertex. Let d be a positive integer. An out-tree $T \subseteq G$ is called a $(1+\epsilon)$-approximate shortest path tree from s up to depth d, if T is rooted at s and for any $v \in V$ such that $\delta_G^d(s,v) < \infty$, we have $v \in V(T)$ and $\delta_T(s,v) \leq (1+\epsilon)\delta_G^d(s,v)$.*

▶ **Definition 24.** *Let $G = (V, E)$ be directed and let $d > 0$ be an integer. A set $H_d^\epsilon \subseteq V$ is called an* $(1+\epsilon)$-approximate d-hub set *of G if for every $u, v \in V$ such that $\delta_G(u,v) < \infty$, there exists a path $P = u \to v$ in G such that $\ell(P) \leq (1+\epsilon)\delta_G(u,v)$ and P is (H_d^ϵ, d)-covered.*

We also extend the definition of a (T, d)-blocker set to trees of depth more than d.

▶ **Definition 25.** *Let V be a vertex set and let $d > 0$ be an integer. Let T be a rooted tree over V. Define T^d to be the set of all maximal subtrees of T of depth no more than d, rooted in non-leaf vertices $x \in V(T)$ satisfying $d \mid \mathrm{dep}_T(x)$.*

[7] The Even-Shiloach algorithm (Theorem 2), apart from maintaining distance labels for all $v \in V$, moves around entire subtrees of the maintained tree T. Hence, in order to ensure that some set B remains a blocker-set of T, it is not sufficient to simply check whether $B \cap V(T[v])$ whenever the Even-Shiloach algorithm changes the distance label of v to d (and, consequently, use a data structure much simpler than that given in Lemma 18).

Then, B is a (T,d)-blocker set if and only if it is a (T^d,d)-blocker set (in terms of Definition 5). Let $\mathcal{T}$ be a collection of rooted trees over V. We call B a $(\mathcal{T},d)$-blocker set if and only if B is a (T,d)-blocker for each $T \in \mathcal{T}$.

▶ **Theorem 26.** *Let $G=(V,E)$ be a directed graph and let $d<n$ be an even integer. Let $\mathcal{T}^{\text{from}} = \{T_v^{\text{from}} : v \in V\}$ ($\mathcal{T}^{\text{to}} = \{T_v^{\text{to}} : v \in V\}$) be a collection of $(1+\epsilon)$-approximate shortest path trees up to depth-3d from all vertices in G (in G^{R}, resp.).*

Let $B \subseteq V$ be a $(\mathcal{T}^{\text{from}} \cup \mathcal{T}^{\text{to}}, \frac{d}{2})$-blocker set. Then B is a $(1+\epsilon)^p$-approximate $2dp$-hub set of G, where $p = \lceil \log_2 n \rceil + 1$.

Recall that our reliable hubs maintenance algorithms for unweighted graphs essentially maintained some shortest path trees up to depth d and either computed their blocker sets using King's algorithm, or dynamically verified whether the sampled hub sets remain blocker sets of the shortest path trees.

We first replace all shortest path trees up to depth d with $(1+\epsilon)$-approximate shortest path trees up to depth d. We use the following extension of Bernstein's h-SSSP algorithm.

▶ **Lemma 27.** *The h-SSSP algorithm of Theorem 3 can be extended so that it maintains a $(1+\epsilon)$-approximate shortest path tree up to depth h from s within the same time bound.*

By Theorem 26, by finding blocker sets of approximate shortest path trees (as in Definition 25), we can compute/verify $(1+\epsilon')^{\Theta(\log n)}$-approximate $\Theta(d \log n)$-hub sets as before.

Given appropriate hub sets, all that both our deterministic incremental $(1+\epsilon)$-approximate APSP algorithm, and Bernstein's randomized $(1+\epsilon)$-approximate partially dynamic APSP algorithm do, is essentially set up and maintain a "circuit" (i.e., a collection of data structures whose outputs constitute the inputs of other structures) of h-SSSP data structures from the hubs with different parameters h and appropriately set ϵ'. In order to make these algorithms work with our reliable approximate hub sets, we basically need to play with the parameters: increase all h's by a polylogarithmic factor, and decrease ϵ' by a polylogarithmic factor.

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, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 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 Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 440–452. SIAM, 2017. doi:10.1137/1.9781611974782.28.

3 Giorgio Ausiello, Giuseppe F. Italiano, Alberto Marchetti-Spaccamela, and Umberto Nanni. Incremental Algorithms for Minimal Length Paths. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 22-24 January 1990, San Francisco, California, USA.*, pages 12–21, 1990. URL: http://dl.acm.org/citation.cfm?id=320176.320178.

4 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.

5 Aaron Bernstein. Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs. *SIAM J. Comput.*, 45(2):548–574, 2016. doi:10.1137/130938670.

6 Aaron Bernstein and Liam Roditty. Improved Dynamic Algorithms for Maintaining Approximate Shortest Paths Under Deletions. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1355–1365, 2011. doi:10.1137/1.9781611973082.104.

7 Camil Demetrescu and Giuseppe F. Italiano. Fully Dynamic Transitive Closure: Breaking Through the $O(n^2)$ Barrier. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 381–389, 2000. doi:10.1109/SFCS.2000.892126.

8 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.

9 Camil Demetrescu and Giuseppe F. Italiano. Fully dynamic all pairs shortest paths with real edge weights. *J. Comput. Syst. Sci.*, 72(5):813–837, 2006. doi:10.1016/j.jcss.2005.05.005.

10 Shimon Even and Yossi Shiloach. An On-Line Edge-Deletion Problem. *J. ACM*, 28(1):1–4, 1981. doi:10.1145/322234.322235.

11 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. A Subquadratic-Time Algorithm for Decremental Single-Source Shortest Paths. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1053–1072, 2014. doi:10.1137/1.9781611973402.79.

12 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Sublinear-time decremental algorithms for single-source reachability and shortest paths on directed graphs. In *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 674–683, 2014. doi:10.1145/2591796.2591869.

13 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Improved Algorithms for Decremental Single-Source Reachability on Directed Graphs. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 725–736, 2015. doi:10.1007/978-3-662-47672-7_59.

14 Monika Henzinger, Sebastian Krinninger, and Danupon Nanongkai. Dynamic Approximate All-Pairs Shortest Paths: Breaking the $O(mn)$ Barrier and Derandomization. *SIAM J. Comput.*, 45(3):947–1006, 2016. doi:10.1137/140957299.

15 Monika Rauch Henzinger and Valerie King. Fully Dynamic Biconnectivity and Transitive Closure. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, USA, 23-25 October 1995*, pages 664–672, 1995. doi:10.1109/SFCS.1995.492668.

16 Monika Rauch Henzinger and Valerie King. Randomized dynamic graph algorithms with polylogarithmic time per operation. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*, pages 519–527, 1995. doi:10.1145/225058.225269.

17 Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001. doi:10.1145/502090.502095.

18 Bruce M. Kapron, Valerie King, and Ben Mountjoy. Dynamic graph connectivity in polylogarithmic worst case time. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1131–1142, 2013. doi:10.1137/1.9781611973105.81.

19 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 '99, 17-18 October, 1999, New York, NY, USA*, pages 81–91, 1999. doi:10.1109/SFFCS.1999.814580.

20 Liam Roditty and Uri Zwick. Improved Dynamic Reachability Algorithms for Directed Graphs. *SIAM J. Comput.*, 37(5):1455–1471, 2008. doi:10.1137/060650271.

21 Liam Roditty and Uri Zwick. On Dynamic Shortest Paths Problems. *Algorithmica*, 61(2):389–401, 2011. doi:10.1007/s00453-010-9401-5.

22 Liam Roditty and Uri Zwick. Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. *SIAM J. Comput.*, 41(3):670–683, 2012. doi:10.1137/090776573.

23 Piotr Sankowski. Dynamic Transitive Closure via Dynamic Matrix Inverse (Extended Abstract). In *45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings*, pages 509–517, 2004. doi:10.1109/FOCS.2004.25.

24 Daniel Dominic Sleator and Robert Endre Tarjan. A Data Structure for Dynamic Trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. `doi:10.1016/0022-0000(83)90006-5`.

25 Robert Endre Tarjan. Dynamic trees as search trees via Euler tours, applied to the network simplex algorithm. *Math. Program.*, 77:169–177, 1997. `doi:10.1007/BF02614369`.

26 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing, May 21-23, 2000, Portland, OR, USA*, pages 343–350, 2000. `doi:10.1145/335305.335345`.

27 Jeffrey D. Ullman and Mihalis Yannakakis. High-Probability Parallel Transitive-Closure Algorithms. *SIAM J. Comput.*, 20(1):100–125, 1991. `doi:10.1137/0220006`.

28 Christian Wulff-Nilsen. Faster Deterministic Fully-Dynamic Graph Connectivity. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1757–1769, 2013. `doi:10.1137/1.9781611973105.126`.

29 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`.