A Deterministic "Folklore" Algorithm for Decremental APSP in Weighted, Directed Graphs

Mara Grilnberger

February 14, 2022

Abstract

In this paper, we give a detailed description to a well known deterministic "folklore" algorithm which maintains approximate all-pairs shortest paths in the partially dynamic decremental setting allowing edge deletions and weight increases. This algorithm originates from a combination of algorithms by King [1] and Bernstein [2] and is used as a component in other algorithms [3] [4]. This paper also gives a more detailed analysis of the total update time $\mathrm{O}\left(n^3 \log^3 n \log(nW)/\epsilon\right)$.

1 Introduction

Maintaining all-pairs shortest paths under dynamicity is a well studied area of research. For the maintenance of an exact all-pairs shortest paths for fully dynamic graphs where both edge insertions and deletions are allowed, there is a deterministic algorithm by Demetrescu and Italiano [5]. In the partially dynamic setting, there exists a deterministic algorithm by Ausiello et al. [6] for the incremental and the randomized algorithm by Baswana et al. [7] for the decremental case. These two algorithms have a total update time of $\mathrm{O}(n^3W)$ where W denotes the highest weight in the graph. This paper focuses on the description of a $(1+\epsilon)$-approximate all pairs shortest paths algorithm with a total update time of $\mathrm{O}(n^3 \log(W)/\epsilon)$. This algorithm is considered "folklore" [3] and employs a kown method by King [1] to recursively build a Graph containing edges for paths that previously needed 2 hops. Then an algorithm of Bernstein[2], that maintains a $SSSP$-Tree for each vertex in the graph is used to estimate the lengths of the paths. This paper focuses on a detailed description and analysis of this folklore algorithm. First we give a few preliminaries, follwed by a description of the base algorithms and concepts. Then we give an explaination of the algorithm followed by a more detailed analysis following the description in [3].

2 Preliminaries

In this paper we will consider weighted directed Graphs $G = (V, E)$ where n denotes the number of vertices. We only consider the decremental setting in this paper. The weight of an edge (u, v) in G is given by $w_G(u, v)$ and $w \in \mathbb{N}$ is the maximum weight of an edge $(u, v) \in E$ at any point in time during the update sequence. Similarly, let $W \in \mathbb{N}$ be the maximum edge weight and the ratio R is then given by W/w. For any pair of vertices $u, v \in V$ $\delta(u, v)$ describes the distance from u to v in G. The h-hop distance is defined as follows

Definition 1 (h-hop distance $\delta^{\mathrm{h}}_{\mathrm{G}}(u,v)$)**.** *For a graph $G=(V,E)$ and for a pair of nodes $u,v \in V$, $\delta^{\mathrm{h}}_{\mathrm{G}}(u,v,)$ is the length of the shortest path from u to v in G that contains at most h edges.*

3 Algorithm

3.1 Bernsteins h-$SSSP$ Algorithm

First let us describe the exact $SSSP$ algorithm by King [1], where distances are maintained up to a distance d.The algorithm maintains a shortest path tree from s up to a distance d and is used in the h-$SSSP$ algorithm to be described later. Each vertex in the graph can be in one of three states at any given time:

- *uncertain* when the distance to the startnode s might have changed
- *changed* once it is certain that the distance increased but the new distance has not been computed yet
- *settled* once the distance has been updated and the shortest path tree to this node has been repaired similarly to Dijkstra's $SSSP$ algorithm

For each vertex v, $l(v)$ denotes the distance from the source node s to v at the time the shortest path tree was restored after the last update. Once this distance becomes larger than d, it is set to ∞. Further, we maintain the set $Pred_v$ of neigbours w such that the edge (v,w) is not in the shortest path tree and the status of w is *settled* or *uncertain*. The distance of a prospective new shortest path to v via such an alternate edge $d(v) = min_{w \in Pred_v} l(w) + w_G(w,v)$ is computed and saved for each vertex as well.
The algorithm maintains a min-heap H containing vertices that need to be examined after an update. Once an edge (u,v) is deleted, the node v becomes *uncertain* and is added to H with the old distance to s, $l(v)$ as the key. Then until the heap is empty, the node w with the smallest distance-estimate as key is retreived. If the estimate is larger than d, the estimate for all other nodes still in H is even larger and can therefore be set to ∞ and the algorithm is done. Otherwise, if w is *uncertain*, we check if $key(w) = d(w)$ i.e. there is an alternate edge connecting w to the shortest path tree so that the distance from s to w remains unchanged, the edge to the vertex v that minimizes $d(w)$ is added to the tree. As a result v is removed from $Pred_w$ and w is added to the set $Pred_z$ of all its neighbours z. If there is no such alternate edge, w is removed from $Pred_z$ for each neighbour z such that (w,z) is an edge in the shortest path tree and w is reinserted to H with the updated distance-estimate $d(w)$. Finally, the edges to all children of w in the tree are removed and they are made *uncertain* and also added to the heap to be examined in the future. When a *changed* node w is retreived from the heap, the new edge that minimizes the nodes distance to s is reinserted to the tree, the edge is removed from $Pred_w$ then the neighbours $Pred$-sets are updated and finally the estimate $l(w)$ is updated to the value of $d(w)$. In this way the shortest path tree is either repaired when an alternate path to the nearest non-tree node exists that maintains the distance or it is rebuilt step-by-step from the nearest node that is no longer in the tree following the deletion.
If the weight of an edge (u,v) is increased, the same method is used but first u is inserted into $Pred_v$, so that the edge with the new weight is considered again when finding the new shortest path to v. An exact description and pseudocode can be found in [1]

Next let us describe the concept of the h-$SSSP$ algorithm from [2] that is used as a building block in the "folklore" algorithm. To achieve the run times mentioned in the this paper a more complicated version of the algorithm needs to be used. It maintains distances from a start node s to every $v \in V$ in a decremental setting for paths with at most h hops. The algorithm maintains a set of Datastructures h-$SSSP_k$, where h-$SSSP_k$ for a fixed k contains a $(1+\epsilon)$-approximation for those distances between vertices u, v where $2^k \leq \delta^h(u,v) \leq 2^{k+1}$, for other distances the approximation can be worse. By maintaining such a datastructure for every k from $\lfloor \log(w) \rfloor$ to $\lceil \log(nW) \rceil$ it can be guaranteed that a good approximation for each distance in the graph is maintained for some k. Each h-$SSSP_k$ is maintained by first creating a new graph $G_k = (V, E_k)$ from the input graph G:

$$E_k = \{(u,v) \in E | w_G(u,v) \leq 2^{k+1}\}$$

and scaling the weights for an edge $(u,v) \in E_k$

$$w_{G_k}(u,v) = \left\lceil \frac{w_G(u,v)}{\alpha} \right\rceil \text{ where } \alpha = \frac{\epsilon 2^k}{h}$$

On this new graph with smaller weights the algorithm by King [1] as described earlier is used, computing $SSSP$ up to a certain distance d where d is set to $\lceil \frac{4h}{\epsilon} \rceil$. Multiplying the results by α gives the distance estimates for a fixed k. Let $\delta'_k(v)$ be the distance estimate for $\delta^h_G(s,v)$ maintained by h-$SSSP_k$. The minimum over all such $\delta'_k(v)$ for $\lfloor \log(w) \rfloor \leq k \leq \lceil \log(nW) \rceil$ is the overall distance estimated given by the h-$SSSP$ algorithm.
Since W is the largest weight of an edge at any point in the update sequence and therefore not known from the start, it suffices to use the largest weight W' at the current time and update this value throughout the update sequence [2]. Bernstein states:

Theorem 1 (h-$SSSP_k$ [2])**.** *For a graph $G = (V, E)$, a source node $s \in V$ and $k \in \mathbb{N}$ the algorithm decrementally maintains the distance estimate $\delta(s,v) \leq \delta'_k(v)$ such that:*

$$\delta_G(s,v) \leq \delta'_k(v) \leq (1+\epsilon)\delta^h_G(s,v) \text{ if } 2^k \leq \delta^h_G(s,v) \leq 2^{k+1}$$

Any such h-$SSSP_k$ can be maintained in $O(mh/\epsilon + \Delta)$ total update time for an ϵ with $0 \leq \epsilon \leq 1$

In the following we use the h-$SSSP$ datastructure D^i_v to refer to the set of h-$SSSP_k$ for a start vertex $v \in V$ on a graph G_i, where $\lfloor \log(w) \rfloor \leq k \leq \lceil \log(nW) \rceil$ and the minimum distance estimate over all h-$SSSP_k$ gives the distance estimate $\delta'_i(v)$.

3.2 Combined Folklore Algorithm

The algorithm as described in [3] starts by building a set of new Graphs $G^1, ..., G^{\lceil \log n \rceil}$ from the input Graph G similar to the transitive closure algorithm for unweighted graphs in [1]. G^1 is simply the same as the original graph G. Any other $G^i = (V, E^i)$ is constructed depending on the paths with at most 2 hops in the previous Graph G^{i-1}.

$$E^i = \{(u,v) \in E^{i-1} | \delta^2_{i-1}(u,v) < \infty\} \text{ where } \delta^h_i \text{ is the } h\text{-hop distance in } G_i$$

G^i contains an edge (u,v) for those vertices u, v that are connected by a path with at most two edges

in G^{i-1} and at most 2^i edges in the original graph G. Now n instances of the h-$SSSP$ algorithm (with $h = 2$) and corresponding datastructures D_v^i, one for every vertex $v \in V$ are used to maintain the distances between nodes that are connected by no more than 2 edges. These distance estimates are now used as weights for the new edges in G^{i+1}, where the quality of the approximation decreases by a factor of $(1 + \epsilon)$ with every increment of i.

Algorithm 1 Algorithm using transitive closure and h-$SSSP$

$\epsilon \leftarrow \frac{\epsilon}{2\lceil \log n \rceil}$
Require:
$G = G^1, ..., G^{\lceil \log n \rceil}$ were initialized as described above if G initially has edges
Let $D_v^1, ..., D_v^{\lceil \log n \rceil}$ be corresponding h-$SSSP$ datastructures with $h = 2$ for each vertex v, also initialized
function UPDATE(w, z, i)
    Update edge (w, z) in G^i ▷ either increase weight of edge (w, z) or delete it
    for every vertex v in V **do**
        Run h-SSSP update for D_v^i where:
        if $i < \lceil \log n \rceil$ **then**
            if $\delta_i'(u)$ in D_v^i changed **then**
                if $\delta_i'(u)$ has become larger than ∞ **then**
                    remove edge (v, u) from E^{i+1} by calling UPDATE(v,u,i+1)
                else
                    Set weight $w_{G^{i+1}}(v, u)$ to distance estimate $\delta_i'(u)$ by calling UPDATE(v,u,i+1)
                end if
            end if
        end if
    end for
end function

To initialize the datastructures one can initialize the distance estimates with 0 and then use the *Update* function to increase the edgeweights or respectively delete the edges in an arbitrary order. To update the distance estimates the function should be called with the following parameters: *Update*(u,v,1). Eventually the recursion will terminate since an Update to the "highest" Graph $G^{\lceil \log n \rceil}$ does not change any other graph. Then $D_{\log n}$ contains $(1 + \epsilon)$ approximate distance estimates for the distances in G after the last update.

3.3 Resulting Approximation

First the distance estimate $\delta'_{\lceil \log n \rceil}(u, v)$ in $D_u^{\lceil \log n \rceil}$ after the update procedure completes, is indeed a $(1 + \epsilon)$-approximation of $\delta_G(u, v)$. For any pair of nodes $u, v \in V$ D_u^i for $v \in V$ maintains a distance estimate from u to v $\delta_i'(v)$ (also described as $\delta_i'(u, v)$ in the following) such that:

$$\delta_G(u, v) \leq \delta_i'(u, v) \leq (1 + \epsilon')^i \delta_G^{2^i}(u, v) \tag{1}$$

The proof is by induction on i. First we consider the base case where $i = 1$. Let $u, v \in G = G^1$, $\delta_i'(u, v)$ is given by the minimum $\delta'_{i,k}(v)$ of all h-$SSSP_k$ datastructures in D_u^1 (where $\lfloor \log(w) \rfloor \leq k \leq$

$\lceil \log(nW) \rceil$). Hence by Theorem 1 we directly get

$$\delta_G(u,v) \leq \delta'_1(u,v) \leq (1+\epsilon')\delta^2_G(u,v).$$

Now consider the induction step where $i \geq 2$:
Assume that $\delta'_i(u,v) \leq \infty$ otherwise the claim holds since then there is no path of length at most 2 in G_i from u to v. Hence there is no vertex w such that $\delta'_{i-1}(u,w) \leq \infty$ and $\delta'_{i-1}(w,v) \leq \infty$ also $\delta'_{i-1}(u,v) = \infty$ and by the induction Hypothesis the right side of the inequality follows. Then there exists a path from u to v with at most 2^i edges in G. Let q be an arbitrary but fixed u - v path with $\leq 2^i$ edges in G. Then q consists of two subpaths q_1 from u to some vertex w and q_2 from w to v, where q_1 and q_2 consist of $\leq 2^{i-1}$ edges each. By the induction hypothesis we get

$$\delta_G(u,w) \leq \delta'_{i-1}(u,w) \leq (1+\epsilon')^{i-1}\delta^{2^{i-1}}_G(u,w)$$

and similarly

$$\delta_G(w,v) \leq \delta'_{i-1}(w,v) \leq (1+\epsilon')^{i-1}\delta^{2^{i-1}}_G(w,v).$$

Hence from Theorem 1 we get

$$\begin{aligned}
\delta'_i(u,v) &\leq (1+\epsilon)\left(\delta_{G_i}(u,v)\right) \\
&\leq (1+\epsilon)\left(\delta'_{i-1}(u,w) + \delta'_{i-1}(w,v)\right) \leq (1+\epsilon')^i\left(\delta^{2^{i-1}}_G(u,w) + \delta^{2^{i-1}}_G(w,v)\right) \\
&\leq (1+\epsilon')^i \operatorname{length}_G(q)
\end{aligned}$$

Since this also holds for the shortest path from u to v the right inequality of (1) holds. Distances in the graph are also never underestimated by a $\delta'_i(u,v)$ since for $1 < i \leq \lceil \log n \rceil$ all edge weights in the graph G^i are given by $(1+\epsilon)$-approximations of 2-hop distances (that are always larger or equal to the actual distances in G^i) from u to v in the previous graph G^{i-1} by Theorem 1. Since any shortest path in G can have at most n hops, for a distance estimate $\delta'_{\lceil \log n \rceil}(u,v)$ produced by the h-$SSSP$ datastructure $D^{\lceil \log n \rceil}_u$ on $G^{\lceil \log n \rceil}$ we have

$$\begin{aligned}
\delta_G(u,v) \leq \delta'_{\lceil \log n \rceil}(u,v) &\leq (1+\epsilon')^{\lceil \log n \rceil}\delta^{2^{\lceil \log n \rceil}}_G(u,v) \\
&= (1+\epsilon')^{\lceil \log n \rceil}\delta_G(u,v).
\end{aligned}$$

Since ϵ' was chosen as $\frac{\epsilon}{2\lceil \log n \rceil}$ similarly to [8] it follows

$$\delta'_{\lceil \log n \rceil}(u,v) \leq e^{\epsilon/2}\delta_G(u,v).$$

Because $\epsilon \in (0,1)$ we get

$$\delta'_{\lceil \log n \rceil}(u,v) \leq (1+\epsilon)\delta_G(u,v).$$

Therefore the distance estimate maintained by the algorithm is indeed a $(1+\epsilon)$ approximation.

3.4 Analysis

Now consider the total update time (as described in [3]) of this algorithm for a number Δ_1 updates that are direct changes to G and its n Datastructures D_v^1 for $v \in V$. First we need to determine how often a change to a given $h\text{-}SSSP_k$ that is part of some Datastructure D_v^i is made. Since the $h\text{-}SSSP$ algorithm runs on the graph $G^{i'}$ with the scaled down weights and as described only distances up to $\lceil 4h/\epsilon' \rceil$ where $h = 2$ are maintained by the datastructure. Because distances can only increase and edge weights are positive integers a distance estimation $\delta'_k(u)$ resulting from $h\text{-}SSSP_k$ in D_v^i can change at most $\lceil 8/\epsilon' \rceil$ times. Since k takes on $\mathrm{O}(\log(nR))$ values the distance estimates from the source node v to any other node, maintained in D_v^i, change $\mathrm{O}\left(n \log(nR)/\epsilon'\right)$ times in total.
Since we maintain such a datastructure for every graph G^i where $1 \leq i \leq \lceil \log n \rceil$ and any vertex $v \in V$ the number of changes in a gaph G^1 where $i > 1$ to some distance estimation in the algorithm is bounded by $\mathrm{O}\left(n^2 \log^2 n \log(nR)/\epsilon'\right)$.
Changes in the distance in one Datastructure D_v^i cause updates in the Datastructures for the next Graph G^{i+1}. For dense graphs the $h\text{-}SSSP$ algorithm has a total update time of

$$\mathrm{O}\left(n^2 h \log n \log(nW)/\epsilon + \Delta\right) \text{ [2]}$$

since the number of edges in all G^i is bounded by n^2 (Δ here is the total number of updates to the graph $h\text{-}SSSP$ is run on). We run this algorithm for every Graph G^i and every vertex in each of those graphs where the total number of updates for all of those algorithm instances is bound by $n\Delta_1 + \mathrm{O}\left(n^2 \log^2 n \log(nR)/\epsilon'\right)$. If we assume that $\Delta_1 \in \mathrm{O}(n^2 \log(W)/\epsilon))$ we get that all updates run in time

$$\mathrm{O}\left(n^3 \log^2 n \log(nW)/\epsilon'\right).$$

By the definition of ϵ' the total update time stated in [8] is given by:

$$\mathrm{O}\left(n^3 \log^3 n \log(nW)/\epsilon\right).$$

3.5 Conclusion

In this paper, we have given a more detailed description and analysis of the $(1+\epsilon)$-approximate partially dynamic all-pairs shortest path folklore algorithm for the decremental setting. The algorithm works by first taking the input graph G and creating a set of graphs incrementally by adding edges to a graph, with the same vertices, that connect those vertices that have a path of length at most 2 between them in the previous graph. The last graph then contains an edge from every vertex to every other in the original graph reachable vertex. The distances in the last graph are maintained by running the $h\text{-}SSSP$ with $h = 2$ algorithm for every node on all these Graphs, updating the weights in the next graph accordingly. The total update time of the algorithm is in $\mathrm{O}\left(n^3 \log^3 n \log(nW)/\epsilon\right)$.

References

[1] V. King, "Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs," *Foundations of Computer Science, 1975., 16th Annual Symposium on*, 09 1999.

[2] A. Bernstein, "Maintaining shortest paths under deletions in weighted directed graphs," in *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 725–734.

[3] A. Karczmarz and J. Łącki, "Reliable hubs for partially-dynamic all-pairs shortest paths in directed graphs," *ArXiv*, vol. abs/1907.02266, 2019.

[4] A. Karczmarz, "Decrementai transitive closure and shortest paths for planar digraphs and beyond," in *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, A. Czumaj, Ed. SIAM, 2018, pp. 73–92.

[5] C. Demetrescu and G. Italiano, "A new approach to dynamic all pairs shortest paths," *J. ACM*, vol. 51, pp. 968–992, 01 2004.

[6] G. Ausiello, G. Italiano, A. Marchetti-Spaccamela, and U. Nanni, "Incremental algorithms for minimal length paths." *Journal of Algorithms*, vol. 12, pp. 615–638, 12 1991.

[7] S. Baswana, R. Hariharan, and S. Sen, "Improved decremental algorithms for maintaining transitive closure and all-pairs shortest paths," *Journal of Algorithms*, vol. 62, pp. 74–92, 04 2007.

[8] A. Karczmarz and J. Łącki, "Simple label-correcting algorithms for partially-dynamic approximate shortest paths in directed graphs," in *SOSA 2020*, 2020.