Princeton Univ. F'24 COS 597b: Recent Advances in Graph Algorithms

Lecture 8: Deterministic Decremental SSSP and Approximate Min-Cost Flow in Almost-Linear Time

Lecturer: *Huacheng Yu* Scribe: *Chenxiao Tian*

1 Introduction

This lecture notes starts going through the paper [1] which introduces the first deterministic, almost-linear time data structure for decremental SSSP in undirected graphs.In this lecture notes, we firstly go through some basic notations, goal, settings, main result and background of the problem. Then we review the ES-tree and an approximated shortest paths algorithm [2] based on it. After that, we introduce the concept of Emulator H of a graph. In this lecture note, we finally go through the main building block construction in this paper, which is the Robust Core Data Structure that can lead to an algorithm, the algorithm could keep the Emulator, maintain decremental cores and create new cores.

Settings. *We begin with an undirected weighted graph $G = (V, E, w)$ with n nodes and m edges, where edge weights are non-negative ($w(e) \geq 0$) for all $e \in E$, and a fixed source node $s \in V$. The problem is to find the shortest paths from the fixed s to any other vertex in the graph. As the edges of G are deleted sequentially, we maintain a $(1+\epsilon)$-approximation of the shortest path distances from the source s to every other vertex in the graph.*

Goal. *To minimize the total update time required to maintain these $(1+\epsilon)$-approximate shortest paths as the graph transitions from its initial state G to an empty graph.*

Main Result. *For any constant $\epsilon > \frac{1}{polylog(n)}$, it is possible to maintain $(1+\epsilon)$-approximate shortest paths from the source node s to all vertices in $O(m \cdot n^{o(1)})$ total update time, deterministically.*

Observation 1. *We could assume without loss of generality (WLOG) that the maximum degree of the graph G is at most 3. (The weight in the graph G is strictly bounded by n^3.)*

2 ES-tree Algorithm and Approximated Shortest Paths Algorithm

The ES-tree algorithm (Even Shiloach 1981) in [2] maintains a decremental unweighted graph and computes exact distances from a source node s up to a parameter d. The total time complexity is $O(md)$. Now we review their algorithms in this section.

Algorithm 1.

- *Maintain a BFS tree from s with a depth d.*
- *When a tree edge is deleted:*
  - *If a vertex v is disconnected at level l:*

    - *Go through all neighbors u of v.*
    - *Check if u is in level $l-1$. If such a u exists, reconnect (u,v) in the tree.*
    - *Otherwise, disconnect v from all its children in the tree and move v to level $l+1$.*
  - *Repeat this process for all disconnected vertices in levels $l+1, l+2, \ldots, d$.*

Remark 1. *Time Analysis for the Algorithm 1: For each vertex v and level l, the above steps are performed at most once for every edge (u,v). The total time complexity is no more than $O(md)$. Weighted Graphs Case for Algorithm 1: The same algorithm can be applied to weighted graphs by maintaining the shortest-path (SP) tree up to a distance d, with a total time complexity of $O(md)$.*

Algorithm 2. ***Approximated Shortest Paths:*** *The algorithm can maintain a $(1+\epsilon)$-approximation of SPs that use at most u edges in time $\tilde{O}(mu)$, where $\tilde{O}$ suppresses logarithmic factors.*

- *For $k = 1, 2, \ldots, 2^i$:*
  - *Maintain an ES-tree for distances approximately k, with an additive error $\leq \epsilon \cdot k$.*
  - *Round the edge weights to multiples of $(\epsilon \cdot k)/u$ (additive error of SP $\leq \epsilon \cdot k$).*
  - *The ES-tree maintains SPs up to a distance of $O(k)$.*

Remark 2. *The total time complexity is:*

$$\tilde{O}\left(m \cdot \frac{k}{\epsilon \cdot (k/u)}\right) = \tilde{O}(m \cdot u).$$

3 Emulator H of a graph

In this section, we introduce the definition of the Emulator H of a graph.

Definition 1. *An (h,ϵ)-hop emulator H of a graph G is defined as follows:*

1. *$G \subseteq H$ (The graph G is contained within H).*
2. *For any $u, v \in V$, $dist_G(u,v) \leq dist_H(u,v)$.*
3. *For any $u, v \in V$, there exists a path P in H using $\leq h$ edges such that the length of P in H satisfies:*
$$length(P) \leq (1+\epsilon) \cdot dist_G(u,v).$$

Goal. *Assuming that the shortest paths (SPs) in G have $\leq h$ edges, the goal is to construct and maintain an $(h/\gamma, \epsilon)$-hop emulator H, where we think $\gamma = n^{o(1)}$.*

Plan. *We focus on unweighted graph G. Maintain a collection of cores $\{C_i\}$, such that for any vertex $u \in G$, there exists a core C_i satisfying:*

$$dist(u, C_i) \leq O(diam(C_i)).$$

Additionally, maintain ES-trees from each C_i up to a distance $D_i \geq D$ (referred to as the shell), such that:

$$\epsilon D_i \gg diam(C_i).$$

Definition 2. *(**Definition of Emulator** H)*
The emulator H is defined based on the collection $\{C_i\}$ as follows:

- *For any vertex v that belongs to shell(C_i), add an edge with weight approximately equal to $dist_G(v, C_i)$.*

Claim 1. *The constructed H is an $(O(h/D), O(\epsilon))$-emulator.*

4 The Algorithm maintains decremental cores and creates new cores

The following algorithm in this section maintains decremental cores and also create new cores as needed. We introduce it by making a summary on its goal, key construction, algorithm and lemma 1 in this section.

Goal. *1. Maintain each decreasing core C_i, such that diam(C_i) remains small.*

2. Maintain $\{C_i\}$ such that every u is close to some C_i.

3. Ensure each vertex is in at most $\triangle = n^{o(1)}$ shell(C_i)s.

Definition 3. *(**Definition of Robust Core Data Structure**)*
Given a decremental graph and an initial core C^{init} with diameter $\leq d$, the robust core data structure maintains a decreasing set $C \subseteq C^{init}$ until C is empty, such that:

1. *$diam_G(C) \leq d \cdot str$ (stretch $str = n^{o(1)}$),*
2. *Every v that leaves C (i.e., $v \in C^{init} \setminus C$) satisfies:*

$$|ball(v, 2d) \cap C^{init}| \leq (1 - \delta)|C^{init}| \quad (\delta = n^{-o(1)}).$$

Algorithm 3. ***How to Maintain the Emulator?***
To maintain the emulator, we use the following steps in the algorithm:

- *Maintain cores C_i with initial diameter d_i.*
- *Maintain EStree(C_i) up to distance $D_i = d_i \cdot str/(4\epsilon)$.*
- *While there exists u such that no C_i has dist$(C_i, u) \leq 4d_i$ (checked using the maintained EStrees [2]):*
  1. *Let l be the largest value such that:*

$$|ball(u, d \cdot (str/\epsilon)^l)| \geq n^{l/k}, \quad l \in [0, k].$$

  2. *Let $C^{init} = ball(u, d \cdot (str/\epsilon)^l)$.*
  3. *Set $d^{init} = 2d \cdot (str/\epsilon)^l$.*

4. *Maintain C^{init} using the robust core data structure, where k is picked such that $k = \omega(1)$ and $(str/\epsilon)^k = n^{o(1)}$.*

Lemma 1. *For any u, u can participate in at most $\tilde{O}\left(n^{1/k} \cdot \delta^{-1}\right)$ cores.*

Proof Sketch: For each diameter $d \cdot (\mathrm{str}/\epsilon)^l$, cores are disjoint.

$$|\mathrm{ball}_G(u, 2d \cdot (\mathrm{str}/\epsilon)^l)| \geq n^{(l+1)/k}, \quad \text{when some } u \in C_i \text{ is initialized.}$$

Using (2) in the definition of the robust core:

$$|\mathrm{ball}_G(u, 2d \cdot (\mathrm{str}/\epsilon)^l) \cap C^{\mathrm{init}}| \leq (1-\delta)|C^{\mathrm{init}}|.$$

Thus, at least:

$$\delta \cdot |C^{\mathrm{init}}| \quad \text{leaves } \mathrm{ball}(u, 2d \cdot (\mathrm{str}/\epsilon)^l) \geq \delta \cdot n^{l/k}.$$

Hence, this process can happen at most $n^{1/k} \cdot \delta^{-1}$ times.

References

1. Bernstein, A., Gutenberg, M. P. & Saranurak, T. *Deterministic Decremental SSSP and Approximate Min-Cost Flow in Almost-Linear Time* in *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)* (2022), 1000–1008.
2. Even, S. & Shiloach, Y. An on-line edge-deletion problem. *Journal of the ACM* **28,** 1–4 (Jan. 1981).