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

Lecture 2: Negative-Weight Single-Source Shortest Paths in Near-linear Time [BNW22]

Lecturer: *Huacheng Yu* Scribe: *Ilya Maier*

1 Overview

Last week, we started presenting the main framework of the algorithm. The setup was that we have a graph G with weights $w(e) \geq -2$ and a modified graph G^{+1} where $w^{+1}(e) = w(e)+1$ and we were trying to compute a good price function ϕ for G s.t. $w(u,v)+\phi(u)-\phi(v) \geq -1$. The main framework consists of computing the LDD of G^{+1} and partition the edges into 3 sets:

- SCC-internal edges
- "forward" edges: if we contract the SCCs, the resulting graph is a Directed Acyclic Graph (DAG) as we will see later, so we can indeed find a topological ordering of the SCCs
- "backward" edges, i.e. E^{rem}

We also add a dummy source s which has edges to all other vertices with weight 0. Then, we compute the price functions ϕ_1, ϕ_2, ϕ_3 for G with only the first set of edges, the first two sets of edges and all edges respectively. We already saw how to compute ϕ_1 and ϕ_2 and the plan for today is to compute ϕ_3 with which we will have an $\tilde{O}(m\sqrt{n})$ algorithm, then see how we can improve the runtime to $\tilde{O}(m)$ and finally present an algorithm and a proof sketch for computing LDDs.

2 An $\tilde{O}(m\sqrt{n})$ algorithm

To compute ϕ_3, we want to just compute the exact shortest distances in $G^{+1}_{\phi_1+\phi_2}$, from which we straightforwardly obtain ϕ_3. However, $G^{+1}_{\phi_1+\phi_2}$ can still contain negative edges, so using just Bellman-Ford will result in a bad runtime. Instead, we want to use a combination of Dijkstra + Bellman-Ford s.t. computing SSSP takes only $\tilde{O}(mk)$ time where k is the maximum number of negative edges on a shortest path. Hence, we want to show that k is indeed small.

The main intuition is that because $\Pr[e \in E^{\text{rem}}] \leq \tilde{O}(\frac{\max\{0,w(e)\}}{D} + n^{-10})$, there should be only a few "backward" edges and hence also a small number of negative edges (as we have already resolved all other edges).

Lemma 1. *The expected number of negative edges on a shortest path $s \to v$ in G^{+1} is $\tilde{O}(n/D + n^{-10})$.*

Proof. Let P be any SP in G^{+1}. Then,

$$\mathbb{E}[\#e \in P \text{ s.t. } e \in E^{\text{rem}}] = \sum_{e \in P} \Pr[e \in E^{\text{rem}}] \tag{1}$$

$$\leq \sum_{e \in P} \tilde{O}\left(\frac{\max\{0, w(e)\}}{D} + n^{-10}\right) \tag{2}$$

$$\leq \sum_{e \in P} \tilde{O}\left(\frac{w(e)+1}{D} + n^{-10}\right) \tag{3}$$

$$\leq \tilde{O}\left(\frac{w(e)+|P|}{D} + n^{-10}\right) \tag{4}$$

$$\leq \tilde{O}\left(\frac{0+n}{D} + n^{-10}\right) \tag{5}$$

where in (3) we used that $w^{+1}(e) \geq -1$ and in (5) that there is always a path from s to v for all v with weight 0 as s is a dummy source. □

With that we are ready to present how to compute ϕ_3. Remember that we transformed G s.t. every vertex has a constant out-degree. This property will be needed in the following lemma. One way to construct such graph is by replacing every vertex v with a directed cycle on $\deg_{\text{in}}(v) + \deg_{\text{out}}(v)$ vertices and edge weights set to 0, s.t. each in-/out-edge is attached to a corresponding vertex. In this case, the number of vertices n' becomes $O(m)$.

Lemma 2. *Let l_v be the number of negative edges on the SP $s \to v$, then SSSP can be solved in $\tilde{O}(\sum_v l_v \cdot \deg_{\text{out}}(v) + m)$.*

Proof sketch: we initialize a priority queue for Dijkstra and run it ignoring the non-negative edges and marking every visited vertex. Then for each marked vertex, we unmark it and update the distances of its out-neighbors using Bellman-Ford, adding vertices for which we changed the distance estimates back to the queue. Then we repeat the process until the queue is empty.

We can show that the following invariant holds: after k iterations, the distance estimate for any vertex v with $l_v < k$ is correct. For runtime, notice that the initial run of Dijkstra takes $\tilde{O}(m)$ time and after this the only vertices we add to the queue are the ones for which the distance estimates have changes because there is a shorter path using a negative edge. As the distance estimate can change at most $O(l_v)$ times and each time we will only consider the const. number of out-neighbors of v, the claim follows.

Now, we get that by setting $D = \tilde{O}(\sqrt{n})$, l_v is expected to be $\tilde{O}(\sqrt{n})$, so computing ϕ_3 also takes $\tilde{O}(m\sqrt{n})$. Finally, let's summarize what we have achieved so far. ϕ_1 ensures that all SCC-internal edges have a non-negative weight. ϕ_2 ensures that for forward edges and ϕ_3 for backward edges. Hence, $\phi_1 + \phi_2 + \phi_3$ achieves $w(e) \geq -1$ in G as the edge weights differ by exactly one between G and G^{+1}, so we achieved what we were seeking.

3 Improving the runtime to $\tilde{O}(m)$

For improving the runtime, notice that computing ϕ_2 already only takes $O(m)$ time and that computing ϕ_3 would also only take $\tilde{O}(m)$ time if D was $\Theta(n)$. However, raising D to

be $\Theta(n)$ changes the runtime for stage 1 to become $\tilde{O}(mn)$ as an SP can now contain D negative edges. Luckily, this can be resolved by applying the algorithm recursively on the graph containing only the SCC-internal edges. We start by initializing D to be n. With each step of recursion we update D_{next} to be $D/2$, so we will only need $O(\log n)$ recursion steps until $D \in O(1)$. Once it becomes smaller than say 2, we can just run the mix of Dijkstra + Bellman-Ford as our base case. As such, each iteration will only take $\tilde{O}(m)$ time and hence the total runtime is $\tilde{O}(m)$. Here is the simplified pseudo-code for overview:

Algorithm 1 ComputePriceFunction(G^{+1}, D)

1: Compute $LDD(G^{+1}, D)$
2: Let E_1 be the SCC-internal, E_2 the forward and E_3 the backward edges
3: $\phi_1 \leftarrow$ recurse on (V, E_1)
4: $\phi_2 \leftarrow$ compute the topological ordering of $G[E_2]$ and set $\phi_2(v) = -iW$ for all v that are i-th in the order
5: $\phi_3 \leftarrow$ run the mix of Dijkstra + Bellman-Ford on $G^{+1}_{\phi_1+\phi_2}$
6: **return** $\phi_1 + \phi_2 + \phi_3$

With this, we obtain the main theorem:

Theorem 3 ([BNW22])**.** *There exists a randomized algorithm which either detects a negative weight cycle or outputs the shortest-path tree from s in $\tilde{O}(m \log W)$ time.*

4 Low-Diameter Decomposition

In this section we want to present the main ideas of how to compute the LDD, which is a major component of the main algorithm. Here is a reminder of the guarantees that we want to have:

Theorem 4. *There exists an algorithm LDD(G,D) that computes a partition $V_1, \ldots, V_k$ of $V(G)$ and a set of edges E^{rem} s.t.*

- *$G[V_i]$ is a Strongly-Connected-Component (SCC) with weak diameter of at most D*
- *Contracting each SCC V_i results in a DAG*
- $\Pr[e \in E^{\text{rem}}] \leq \tilde{O}(\frac{\max\{0,w(e)\}}{D} + n^{-10})$
- *Runtime of $LDD(G, D)$ is $\tilde{O}(m)$*

4.1 Undirected case

To introduce the concept, let's first see how to compute LDD in undirected graphs, without considering the runtime yet. First, WLOG assume that the weights are non-negative (we can just set the negative weights to be 0, which will result in the following new weight function $w'(e) = \max\{0, w(e)\}$). For a set of vertices B let $\partial B = \{e \in E \mid e \cap B = 1\}$, i.e. the set of incident edges to B that only have one endpoint in B. Then, we will iteratively

Algorithm 2 LDD(G, D)

while $G \neq \emptyset$ **do**
  let v be any vertex
  sample $d \sim \text{Geo}(\frac{22 \log n}{D})$
  let $B = \{u \in V(G) \mid \text{dist}_G(v, u) \leq d\}$ ▷ ball around v with radius d
  add ∂B to E^{rem}
  remove B from G (B becomes one of the SCCs/connected components)

pick a vertex v, sample a random radius smaller than $D/2$ and carve out a ball around v from the graph.

To not violate the diameter guarantee, we will additionally check that $d \leq D/2$ and if that's not the case, we will simply put all edges in E^{rem}. Let's analyze the output properties of $LDD(G, D)$.

Lemma 5. *Each $G[V_i]$ is a connected component of diameter at most D.*

Proof. It is clear that $G[V_i]$ is a connected component, as V_i is a ball around v and so each vertex is connected to it. Furthermore, as we have checked that $d \leq D/2$, for any $u, w \in V_i$ there is a walk $u \to v \to w$ of length at most D, so the diameter guarantee also holds. □

Lemma 6. $\Pr[e \in E^{\text{rem}}] \leq \tilde{O}(\frac{\max\{0, w(e)\}}{D} + n^{-10})$

Proof sketch: First, note that $\Pr[d > D/2] \leq (1 - \frac{22 \log n}{D})^{D/2} \leq e^{-11 \log n} \leq n^{-11}$. By union bound over all vertices, the probability that d is ever larger than $D/2$ (and hence that all edges are in E^{rem}) is at most n^{-10}.

Otherwise $d \leq D/2$ and we can view ball growing as a random process. We start with a ball of radius 0 and at each step either increase the radius by 1 or terminate with probability $\frac{22 \log n}{D}$. Now, consider an edge $e = (u, v)$ and the first time either u or v are in some ball that we are about to carve out. WLOG, let it be u. For v to be in the same ball and not get removed, the above process must not terminate $w(e)$ many times. This happens with probability at least $(1 - \frac{22 \log n}{D})^{w(e)} \geq 1 - \frac{22 \log n}{D} \cdot w(e) = 1 - \tilde{O}(\frac{w(e)}{D})$ as wanted.

4.2 Directed LDD

For directed graphs, instead of a ball we can either carve out an in- or an out-ball around v (where in an in-ball each vertex can reach v and the incident edges ∂B are all edges that go into the ball). Because of that we will then have to recurse on the carved out ball, as we do not immediately get an SCC. We will repeat the following:

sample $d \sim \text{Geo}(\frac{22 \log n}{D})$
find either $B = B_{in}(v, d)$ or $B = B_{out}(v, d)$ with $|B| \leq n/2$
add ∂B to E^{rem}
remove B from G
recurse on B

until every vertex left fulfills that both their in- and out-ball are of size at least $n/2$ (where n is the current size of the graph). We will then add these vertices to their own SCC.

To compute the in- or out-ball, we will originally sample $O(\log n)$ vertices and compute their in- and out-balls using Dijkstra. We then will estimate the size of $B_{in/out}(v)$ for any v by the number of sample vertices that contain v in their in-/out-balls. Using Chernoff bounds, we can show that this gives a good approximation. Everything else can also be done in $\tilde{O}(m)$ with efficient book-keeping, for more details, I recommend seeing the notes of Danupon Nanongkai (one of the paper's authors): link.

Let's again analyze the output properties:

Lemma 7. *$G[V_i]$ is a Strongly-Connected-Component (SCC) with weak diameter of at most D.*

Proof. Note that the only vertices we add to an SCC are the ones that reach at least $n/2$ other vertices in their in- and out-ball. Hence, to find a path from any u to some v, we just consider $w \in (B_{out}(V) \cap B_i n(v))$ and construct the walk $u \to w \to v$ of length at most $2 \cdot D/2 \leq D$. □

Lemma 8. *Contracting each SCC V_i results in a DAG*

Proof. Note that for any ball we carve out, we always remove either its in- or out-edges, so a carved out ball can never participate in a cycle. Since each SCC is formed of vertices that are left after we carved out all balls from the current graph, the claim follows. □

Analyzing the probabilities of edges being removed is done similarly as in the undirected case, just that the probability increases by a factor of $\log n$ because of recursion.

References

[BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *63rd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2022, Denver, CO, USA, October 31 - November 3, 2022*, pages 600–611. IEEE, 2022.