CS 466/666: Algorithm Design and Analysis **University of Waterloo: Fall 2025**

Lecture 24

December 2, 2025

Instructor: Sepehr Assadi

Topics of this Lecture

We finish this course by going over the algorithm of [BNW22] for the negative weight shortest path. In particular, we will see a proof of the following theorem.

Theorem 1 ([BNW22])**.** *There is a randomized algorithm that for any graph $G = (V, E, w)$ with no negative cycle, finds single-source shortest paths from a given vertex $s \in V$ in $O(m \log^4 n \cdot \log (nW))$ expected time where $W := \max_e |w(e)|$, namely, the largest absolute value of any edge weight.*

1 Background Tools from the Last Lecture

Let us recap the following two main lemmas from the last lecture.

Lemma 2. *There is an algorithm for solving SSSP in any graph in $O(m \log n \cdot \overline{b}_s)$ time, where $\overline{b}_s$ is the average negative-hop distance of all vertices from s (check Lecture 23 for the definitions).*

Lemma 3 (Directed Low Diameter Decomposition [BNW22])**.** *There is a randomized algorithm that given any directed graph $G = (V, E, w)$ with non-negative integer weights and an integer $D \geqslant 1$, outputs a set of edges E_{rem} with the following properties:*

- *Let C be any* strongly connected component (SCC) *of $G \setminus E_{\text{rem}}$. Then, C has a "weak diameter" at most D:*
$$\forall u, v \in C \quad \mathit{dist}_G(u, v) \leqslant D \quad \text{and} \quad \mathit{dist}_G(v, u) \leqslant D.$$
- *For any edge $e \in E$,*
$$\Pr(e \in E_{\text{rem}}) = \frac{O(\log^2 n)}{D} + n^{-10}.$$

The algorithm runs in $O(m \log^3 n)$ time deterministically (in fact, the runtime is $O(m \log^2 n + n \log^3 n)$ but the distinction is not important for us in this lecture).

2 An $\widetilde{O}(m\sqrt{n})$ Time Algorithm for SSSP

We start by proving a weaker version of Theorem 1 which has many of the key ideas. We then sketch how we can extend this algorithm to prove Theorem 1 also. In particular, our goal now is to prove the following simpler theorem which gives an $\widetilde{O}(m\sqrt{n})$ runtime[1] for SSSP.

Theorem 4 (Weaker version of Theorem 1 [BNW22])**.** *There is a randomized algorithm that for any graph $G = (V, E, w)$ with no negative cycle, finds single-source shortest paths from a given vertex $s \in V$ in $O(m\sqrt{n}\log^2 n \cdot \log(nW))$ expected time.*

The general framework of the proof is as follows: suppose we start with G and a weight function w such that $w(e) \geqslant -2B$ for all $e \in E$ and some integer $B \geqslant 1$. We will find a price function ϕ such that after applying it, we will have $w_\phi(e) \geqslant -B$ for all $e \in E$; i.e., the most negative-weight edge is now at most half as negative as before. We then show that repeating this step for $O(\log W)$ iterations is enough to obtain weights that can be thought of as essentially non-negative. This approach is often called **scaling** and is a classical technique in algorithm design. Let us now formalize this further.

Lemma 5 (Scaling Lemma)**.** *There is a randomized algorithm that given any graph $G = (V, E, w)$ where $w(e) \geqslant -2B$ for every edge $e \in E$, outputs a price function function ϕ such that $w_\phi(e) \geqslant -B$ for every edge $e \in E$. The algorithm has expected $O(m\sqrt{n}\log^2 n)$ time.*

Let us see how to use this lemma now to obtain an $\widetilde{O}(m\sqrt{n})$ time algorithm for SSSP.

Proof of Theorem 4 using Lemma 5. Given $G = (V, E, w)$ with $W := \max_e |w(e)|$, we first update the weight function w so that $w(e) \leftarrow n \cdot w(e)$. Clearly, this does not change the shortest path structures (nor change positivity/negativity of any edge). We then run Lemma 5 repeatedly for $t := \log(nW)$ iterations on the graph G to obtain price functions $\phi_1, \phi_2, \ldots, \phi_t$, where

$$w(e) \geqslant -nW \implies w_{\phi_1}(e) \geqslant -\frac{nW}{2} \implies w_{\phi_2}(e) \geqslant -\frac{nW}{2^2} \implies \cdots \implies w_{\phi_t} \geqslant -\frac{nW}{2^t} = -1,$$

where each '$\implies$' corresponds to running the algorithm of Lemma 5 once. Note that here, each price function ϕ_i is applied on top of the price function ϕ_{i-1}, i.e., is obtained by adding the price function of Lemma 5 to the price function ϕ_i.

At this point, we define a new weight function w' wherein $w'(e) = w_{\phi_t}(e) + 1$. In principle, this can potentially change the shortest path structure. Nevertheless, we prove in our special case, this cannot happen. Suppose P and Q are two s-v paths in G (under the original weight function w) and that $w(P) < w(Q)$. We argue that $w'(P) < w'(Q)$ also. We have,

$$w'(P) = w_{\phi_t}(P) + |P| = w(P) + \phi_t(s) - \phi_t(v) + |P| \leqslant w(P) + (n-1) + \phi_t(s) - \phi_t(v),$$

since P can have $n-1$ edges at most. On the other hand

$$w'(Q) = w_{\phi_t}(Q) + |Q| = w(Q) + \phi_t(s) - \phi_t(v) + |Q| \geqslant w(Q) + \phi_t(s) - \phi_t(v).$$

But recall that since we updated the weights w by multiplying them by n, if $w(P) < w(Q)$, then in fact, $w(P) \leqslant w(Q) - n$ even. Thus, we continue to have $w'(P) < w'(Q)$ also as desired.

Since w' is a non-negative weight function, we can simply run Dijkstra's algorithm on G, w' and solve SSSP in $O(m \log n)$ time at this point (and by the previous argument and correctness of price functions, we get the solution is correct). Thus, the runtime of the algorithm is $O(m\sqrt{n} \cdot \log^2 n \cdot \log(nW))$ as desired. □

[1] Recall that $\widetilde{O}(f) := O(f \cdot \text{poly}\log(f))$.

2.1 Proof of Lemma 5: The Scaling Lemma

We update the graph by adding a vertex s^* that is connected to every vertex with an edge of weight $-B$. Throughout, we use the same set of vertices $\{s^*\} \cup V$ but with different subset of edges (subsets of E which is now updated to include (s^*, v)-edges as well) and different weight functions. In particular, define the following two additional weight functions:

$$w^{2B} : E \to \mathbb{Z} \quad \text{wherein } w^{2B}(e) = w(e) + 2B \text{ for all edges } e \in E;$$
$$w^{B} : E \to \mathbb{Z} \quad \text{wherein } w^{B}(e) = w(e) + B \text{ for all edges } e \in E.$$

Note that both these weight functions can potentially destroy the shortest path structure (but that will not be a concern for us because we will only use them to compute a price function). Moreover w^{2B} is now a non-negative weight function. The algorithm, at a high level, is as follows.

Algorithm 1. The high-level description of the algorithm of Lemma 5. The parameter d below will be set later. The steps of the algorithm will be explained in more detail later.

1. Compute a LDD of $G^{2B} = (\{s^*\} \cup V, E, w^{2B})$ with diameter $D = dB$ using Lemma 3. Let $C_1, \ldots, C_k$ be the SCCs and E_{rem} be the removed edges of the LDD.
2. Use the weights w^B (and not w^{2B}) to find a price function ϕ_1 that makes all edges inside C_i's non-negative in $w^B_{\phi_1}$.
3. Use the updated weights $w^B_{\phi_1}$ to find a price function ϕ_2 that additionally makes the DAG edges of $G \setminus E_{\text{rem}}$ non-negative in $w^B_{\phi_2}$.
4. Use the updated weights $w^B_{\phi_2}$ to find a price function ϕ_3 that additionally makes the edges in E_{rem} non-negative in $w^B_{\phi_3}$.
5. Return w_{ϕ_3} (and not $w^B_{\phi_3}$) as a weight function that satisfy $w_{\phi_3}(e) \geqslant -B$ for all $e \in E$.

We will now go over different steps of this algorithm in detail.

Step 1: LDD computation. Recall that an LDD is only defined for graphs with non-negative weights. Since $w(e) \geqslant -2B$ by assumption and $w^{2B}(e) = w(e) + 2B$ by definition, we have w^{2B} is a non-negative weight function. As such, it is valid to apply Lemma 3 and obtain a set E_{rem} of edges such that any SCC C of $G \setminus E_{\text{rem}}$ satisfies:

$$\forall u, v \in C \quad dist_{G^{2B}}(u, v) \leqslant dB \quad \text{and} \quad dist_{G^{2B}}(v, u) \leqslant dB, \tag{1}$$

and for any edge in G

$$\Pr(e \in E_{\text{rem}}) = \frac{O(\log^2 n)}{dB} \cdot w^{2B}(e). \tag{2}$$

This step takes $O(m \log^3 n)$ time.

Step 2: Fixing SCC edges. Consider the graph $G_1 = (\{s^*\} \cup V, E_1, w^B)$ with weight function w^B where $E_1 \subseteq E$ only contains the edges between SCCs of $G \setminus E_{\text{rem}}$ (i.e., remaining edges after removing E_{rem} and DAG edges). Note that the edges of s^* to all other vertices have weight 0 under w^B (as they had weight $-B$ under w). We claim that the shortest paths from s^* in this graph have "few" negative edges.

Claim 6. *For any $v \in V$, the hop distance of s^* to v in G_1 is less than d.*

Proof. Consider a shortest path P_{s^*v} in G_1 from s^* to v in G_1. We know that $w^B(P_{s^*v}) \leqslant 0$ as s^* is connected to v by an edge of weight 0. If $w^B(P_{s^*v}) = 0$, the hop distance of s^* to v will simply be one by taking the (s^*, v) edge directly. Otherwise, we have $w^B(P_{s^*v}) < 0$ and thus P_{s^*v} starts by going from s^* to some vertex u in the same SCC as v and then taking the shortest path P_{uv} inside this SCC (recall that the only edges of G_1 are the ones inside SCCs). We will argue that P_{uv} can have $< d$ edges.

Suppose towards a contradiction that P_{uv} contains at least d edges in G_1. Then, under the original weight function w, we have,

$$w(P_{uv}) = w^B(P_{uv}) - |P_{uv}| \cdot B < 0 - dB = -dB,$$

since $|P_{uv}| \geqslant d$ by our assumption. On the other hand, since u and v are both inside the same SCC of $G \setminus E_{\text{rem}}$, by Eq (3), we have

$$dist_{G^{2B}}(v, u) \leqslant dB.$$

This implies that there exists some path Q_{vu} in G (and not necessarily G_1) such that

$$w(Q_{vu}) = w^{2B}(v, u) - |Q_{vu}| \cdot 2B \leqslant dB - 1.$$

Putting these two implies that in the original graph G, we can go from u to v with a path of weight $< -dB$ and from v to u with a path of weight $< dB$. This implies that we can start from u and return to it by paying a total weight $< -dB + dB < 0$, implying that there must be a negative cycle in G. But this is a contradiction with the statement of Theorem 4 that implied there is no negative cycle in G. ☐

Combining Claim 6 with Lemma 2 implies that we can find s^*-shortest paths in G_1 in $O(m \log n \cdot d)$ time. We will then define, for any $v \in V$,

$$\phi_1(v) = dist_{G_1}(s^*, v).$$

This implies that for any edge $(u, v) \in G_1$,

$$w^B_{\phi_1}(u, v) = w^B(u, v) + \phi_1(u) - \phi_1(v) = w^B(u, v) + dist_{G_1}(s^*, u) - dist_{G_1}(s^*, v) \geqslant 0,$$

where the last inequality is by triangle inequality since (u, v) is an edge of G_1. Thus, we made all SCC edges non-negative under $w^B_{\phi_1}$.

Step 3: Fixing DAG edges. This step is quite straightforward: we compute a topological ordering of the SCCs of the graph $G \setminus E_{\text{rem}}$ in $O(m + n)$ and denote them by $C_1, \ldots, C_k$. For any $v \in C_i$, we define

$$\phi_2(v) = \phi_1(v) + (k - i) \cdot \max_{e \in E} |w^B_{\phi_1}(e)|.$$

Note that for any edge (u, v) inside the same cluster C_i, we have

$$w^B_{\phi_2}(u, v) = w^B(u, v) + \phi_2(u) - \phi_2(v) = w^B(u, v) + \phi_1(u) - \phi_1(v) = w^B_{\phi_1}(u, v) \geqslant 0,$$

as we proved in the last part. For any edge $u \in C_i$ and $v \in C_j$ for $j > i$,

$$w^B_{\phi_2}(u, v) = w^B(u, v) + \phi_2(u) - \phi_2(v) = w^B(u, v) + \phi_1(u) - \phi_1(v) + \max_{e \in E} |w^B_{\phi_1}(e)| = w^B_{\phi_1}(u, v) + \max_{e \in E} |w^B_{\phi_1}(e)| \geqslant 0.$$

Finally, since we are working with a topological ordering of a DAG there are no other edges, and thus all edges, except for E_{rem}, have become non-negative in $w^B_{\phi_2}$.

Step 4: Fixing E_{rem} edges. We now consider s^*-shortest paths in the entire graph G under the updated weight function $w^B_{\phi_2}$. We claim that these shortest paths also contain only a "few" negative edges on average.

Claim 7. *For any $v \in V$, the expected negative hop distance of s^* to v in G under the weight function $w^B_{\phi_2}$ is less than $\dfrac{O(h_{s^*}(v) \cdot \log^2 n)}{d}$, where $h_s(v)$ is the hop distance of s^* to v under the weight function w^B.*

Proof. As in Claim 6, we consider a path P_{s^*v} that goes from s^* to some vertex u and then take P_{uv} with $w^B_{\phi_2}(P_{uv}) < 0$ (otherwise, the hop distance of s^* to v will be one). Note that P_{uv} is also the shortest path from u to v in w^B itself also since price functions do change the shortest path structure. But under w^B, we could have again gone from s^* to v with a weight of 0, and thus $w^B(P_{uv}) < 0$. This implies that

$$w^{2B}(P_{uv}) = w^B(P_{uv}) + |P_{uv}| \cdot B \leqslant h_{s^*}(v) \cdot B.$$

At the same time, the number of negative edges in P_{uv} under $w^B_{\phi_2}$ is at most equal to $|P_{uv} \cap E_{\text{rem}}|$ as previous steps made sure all other edges are non-negative. Thus,

$$\begin{aligned}
\mathbb{E}\left[\text{negative hop distance of } s^* \text{ to } v \text{ in } w^B_{\phi_2}\right] &\leqslant \mathbb{E}\,|P_{uv} \cap E_{\text{rem}}| \\
&= \sum_{e \in P_{uv}} \Pr\left(e \in E_{\text{rem}}\right) && \text{(by linearity of expectation)} \\
&= \sum_{e \in P_{uv}} \frac{O(\log^2 n)}{dB} w^{2B}(e) && \text{(by Eq (4))} \\
&= \frac{O(\log^2 n)}{dB} \cdot h_{s^*}(v) \cdot B && \text{(by the above bound on } w^{2B}(P_{uv})) \\
&= \frac{O(h_{s^*}(v) \cdot \log^2 n)}{d},
\end{aligned}$$

as desired. □

Since the hop distances are always at most $n-1$, by combining Claim 7 and Lemma 2, we can find s^*-shortest path in the entire G under the weight function $w^B_{\phi_2}$ in $O(m \log^3 n \cdot \frac{n}{d})$ expected time. By setting

$$\phi_3(v) = \phi_2(v) + dist_{w^B_{\phi_2}}(s^*, v),$$

for all $v \in V$, we can make all edges of G non-negative under the weight function $w^B_{\phi_3}$. Finally, this implies that under the original weight function w but with the price function ϕ_3, for any $e \in E$, we have

$$w_{\phi_3}(e) = w^B_{\phi_3}(e) - B \geqslant -B.$$

The expected runtime of the algorithm is now

$$O(m \log^3 n + m \log n \cdot d + m \log^3 n \cdot \frac{n}{d}),$$

and thus by setting $d = \sqrt{n} \log n$, we obtain the expected runtime of

$$O(m\sqrt{n} \log^2 n),$$

concluding the proof of Lemma 5.

3 The Final $\widetilde{O}(m)$ Time Algorithm

The algorithm in Theorem 1 can also be obtained in a very similar manner, using the following improved scaling lemma.

Lemma 8 (Improved Scaling Lemma)**.** *There is a randomized algorithm that given any graph $G = (V, E, w)$ where $w(e) \geqslant -2B$ for every edge $e \in E$, outputs a price function function ϕ such that $w_\phi(e) \geqslant -B$ for every edge $e \in E$. The algorithm has expected $O(m \log^4 n)$ time.*

Theorem 1 follows from Lemma 8 the same exact way Theorem 4 followed from Lemma 5. We now show how to prove Lemma 8.

The idea behind the proof of Lemma 8 is to introduce one more level of recursion: instead of balancing the time took in Step 2 and 4 of Algorithm 1 that led to an $\widetilde{O}(m\sqrt{n})$ time, we will make Step 4 much faster and then recurse on the graphs of Step 2. The improvement obtained in Step 2 comes from another metric: the negative hop distances of the SCCs still drop by a factor of two, and thus the recursion depth will only be $O(\log n)$ which makes our algorithm fast enough.

More formally, the algorithm is as follows. Note that we again use the weight functions w^{2B} and w^B and also add the vertex s^* to the graph as before.

Algorithm 2. The high-level description of the algorithm of Lemma 8. The input is a graph $G = (\{s^*\} \cup V, E, w)$ with an additional parameter Δ promised to be an upper bound on the hop distances between all pairs of reachable vertices in V (ignoring s^*) under the weight function w^B. The algorithm returns a price function ϕ such that $w^B_\phi(e) \geqslant 0$ for all $e \in E$. The steps of the algorithm are also explained in more detail later.

1. If $\Delta \leqslant 1$, run a base case algorithm (explained below) and return.
2. Compute a LDD of $G^{2B} = (\{s^*\} \cup V, E, w^{2B})$ with diameter $D = d \cdot B$ using Lemma 3 for a parameter $d = \Delta/2$ to be fixed explicitly later. Let $C_1, \dots, C_k$ be the SCCs and E_{rem} be the removed edges of the LDD.
3. Use the weights w^B (and not w^{2B}) and recurse on the graphs $G_i = (\{s^*\} \cup C_i, E[s^* \cup C_i], w^B)$ with parameter $\Delta/2$ to find a price function ϕ_1 that makes edges inside C_i's non-negative in $w^B_{\phi_1}$.
4. Use the updated weights $w^B_{\phi_1}$ to find a price function ϕ_2 that additionally makes the DAG edges of $G \setminus E_{\text{rem}}$ non-negative in $w^B_{\phi_2}$.
5. Use the updated weights $w^B_{\phi_2}$ to find a price function ϕ_3 that additionally makes the edges in E_{rem} non-negative in $w^B_{\phi_3}$.
6. Return $w^B_{\phi_3}$ as a weight function that satisfy $w^B_{\phi_3}(e) \geqslant 0$ for all $e \in E$.

Step 1: Base case. For the base case, we simply need to run Lemma 2 to find s^*-shortest paths and set

$$\phi(v) = dist_{G^B}(s^*, v).$$

The correctness follows as before as these distances make w^B non-negative and thus $w(e) \geqslant -B$ for all $e \in E$. Moreover, the runtime is only $O(m \log n)$ by Lemma 2 and the promise that the shortest path between every pair of vertices inside V uses at most one hop. (Technically speaking, we could have just run one iteration of the Bellman-Ford algorithm and solve the problem in $O(m)$ time but the difference is inconsequential).

Step 2: LDD computation. As before, w^{2B} is non-negative and so t is valid to apply Lemma 3 and obtain a set E_{rem} of edges such that any SCC C of $G \setminus E_{\text{rem}}$ satisfies:

$$\forall u, v \in C \quad dist_{G^{2B}}(u, v) \leqslant dB \quad \text{and} \quad dist_{G^{2B}}(v, u) \leqslant dB, \tag{3}$$

and for any edge in G

$$\Pr\left(e \in E_{\text{rem}}\right) = \frac{O(\log^2 n)}{dB} \cdot w^{2B}(e) = \frac{O(\log^2 n)}{\Delta \cdot B} \cdot w^{2B}(e). \tag{4}$$

This step takes $O(m \log^3 n)$ time.

Step 3: Fixing SCC edges. We do exactly as in Algorithm 1 and by Claim 6, have that under w^B, the negative hop diameter of every C_i will be $d = \Delta/2$. This means that the recursive call in this step is run with a correct parameter and thus by induction, we will find a price function ϕ_1 that ensures $w^B_{\phi_1}(e) \geqslant 0$ for all e in the SCCs.

Step 4: Fixing DAG edges. This step is exactly as before and can be done in $O(m+n)$ time.

Step 5: Fixing E_{rem} edges. Again, we exactly as in Algorithm 1 and by Claim 7, have that under $w^B_{\phi_2}$, the negative hop distance of any vertex in expectation is

$$O(\log^2 n \cdot \frac{h_{s^*}(v)}{d}) = O(\log^2 n) \cdot \frac{\Delta}{\Delta/2} = O(\log^2 n),$$

using the fact that under w^B (by our initial assumption in the recursion), hop diameter of the graph is Δ and since we set $d = \Delta/2$. This means in that this step can now be implemented in $O(m \log^3 n)$ expected time using Lemma 2.

In conclusion, the algorithm correctly finds a price function ϕ such that w^B_ϕ is non-negative and thus for every $e \in E$, $w_\phi(e) \geqslant -B$ as desired.

For the runtime analysis, the algorithm reduces the value of Δ by a factor of two each time, and we would be calling it with $\Delta = n-1$ at the beginning since any pair of reachable vertices can have at most $n-1$ hops between them. This means there are $O(\log n)$ level of recursion. Each level also takes $O(m \log^3 n)$ expected time at most, leading to a total of $O(m \log^4 n)$ expected time. This concludes the proof of Lemma 8 and the entire proof of Theorem 1.

References

[BNW22] Aaron Bernstein, Danupon Nanongkai, and Christian Wulff-Nilsen. Negative-weight single-source shortest paths in near-linear time. In *2022 IEEE 63rd annual symposium on foundations of computer science (FOCS)*, pages 600–611. IEEE, 2022. 1, 2