CS 598CSC: Topics in Graph Algorithms	Lecture date: 10/15/2024
Instructor: Chandra Chekuri	Scribe: CC

1 Single-Source Shortest Paths with Negative Weights

Dijkstra's algorithm for SSSP in directed graphs with non-negative weights can be implemented to run in $O(m + n \log n)$ time. We will consider SSSP in directed graphs with *negative* weights allowed. The input is a directed graph $G = (V, E)$ with weights $w : E \to \mathbb{R}$ and a source vertex $s \in V$. The goal is to check if G has a negative weight cycle or to output shortest path distances from s to each vertex $v \in V$ (which are well-defined when there is no negative weight cycle). The classical Bellman-Ford algorithm runs in $O(mn)$ time and the Floyd-Warshall runs in $O(n^3)$ time. These are algorithms from 50's and 60's. Goldberg [Gol95] described a scaling algorithm for integer weighted case in 1993 that ran in $O(m\sqrt{n} \log W)$ time where $W = \max\{|w(e)| \mid e \in E, w(e) < 0\}$ is the absolute value of the least negative weight. In a recent breakthrough, Bernstein, Nanongkai and Wulff-Nilsen obtained a *randomized* near-linear time scaling algorithm for the integer weighted case.

Theorem 1 ([BNWN22])**.** *There exists a randomized (Las Vegas) algorithm that takes $O(m \log^8 n \log W)$ time with high probability (and in expectation) for an m-edge input graph $G = (V, E, w)$ and a source $s \in V$. It either returns a shortest path tree from s or returns a negative-weight cycle.*

Building on [BNWN22], Bringmann, Cassis and Fisher [BCF23] further improved the running time to $O(m \log^2 n \log \log n \log(nW))$.

Fineman [Fin24] obtained a strongly poly-time algorithm that beat Bellman-Ford for the first time. His randomized algorithm runs in $\tilde{O}(mn^{8/9})$-time. Huang, Jin and Quanrud [HJQ24] refined the algorithm with additional ideas and obtained a randomized algorithm that runs in $\tilde{O}(mn^{4/5})$-time.

We will discuss the approach [BNWN22] with some insights from [BCF23].

1.1 Background

We will work extensively with the idea of potentials introduced by Johnson in the context of all-pairs-shortest paths. Potentials allow one to reduce the SSSP problem with negative weights to SSSP problem with non-negative weights and one can then use the simpler Dijkstra's algorithm.

Definition 1. *A potential $\phi : V \to \mathbb{R}$ is a simply a weight function on the vertices of $G = (V, E)$. Given $w : E \to \mathbb{R}$ and a potential ϕ the reduced weights $w_\phi : E \to \mathbb{R}$ is defined as follows: for each edge $(u, v) \in E$, $w_\phi(u, v) = \phi(u) + w(u, v) - \phi(v)$.*

Claim 2. *Let ϕ be any potential. Then for any s-t walk P we have $w_\phi(P) = w(P) + \phi(s) - \phi(t)$. Hence for any two s-t walks P and Q, $w(P) - w(Q) = w_\phi(P) - w_\phi(Q)$ which implies that P is a shortest s-t walk respect to w iff it is a shortest s-t walk with respect to w_ϕ. If P and Q are closed walks starting and ending in s (in particular cycles) then $w(P) = w_\phi(Q)$.*

Lemma 3. *G has no negative weight cycle iff there is a potential ϕ such that $w_\phi(e) \geq 0$ for all $e \in E$.*

Definition 2. *We say that a potential ϕ neutralizes a set of negative length edges $S \subseteq E$ if $w_\phi(e) \geq 0$ for all $e \in S$.*

Thus the goal is to either detect that G has a negative cycle (we use this as a shortcut for negative weight cycle) or find a potential that neutralizes all the negative weight edges while not introducing any new negative weight edges. In the rest of these notes we will only be interested in potentials that neutralize some set of negative weight edges while not introducing any new negative weight edges.

Reducing degree: Since we are shooting for a running time that is linear in m we can do a simple reduction that increases the number of vertices from n to $O(m)$ while ensuring that the out-degree of every vertex is $O(1)$. Suppose we have a vertex u with out-edges $(u, v_1), (u, v_2), \ldots, (u, v_\ell)$. For simplicity assume ℓ is a power of 2. We can create a directed binary out-tree with root as u and depth $\log \ell$ where we associate the leaves with $v_1, v_2, \ldots, v_\ell$. The edge incoming to v_i in the binary tree inherits the weight $w(u, v_i)$ and all other edges in the binary tree that are not incident to leaves have weight 0. This increase the number of vertices to $O(m)$.

One can also do a transformation where the number of negative weight edges is $O(n)$ by splitting each vertex u into u^- and u^+ in the standard way and making $w(u^-, u^+)$ equal the most negative weight edges in the out-edges of u (0 if there is no negative weight edge) and adjusting the weights going out of u appropriately (they now all become non-negative). We won't need this reduction but this is useful in [Fin24, HJQ24].

Verification: One of the useful facts about single-source shortest path trees and distances is that we can check in linear time whether a given tree or set of distances are valid or not in linear time. We simply relax all edges and see if any of the distances change - if they do not change then the distances are valid, otherwise they are obviously not. This verification is useful when running randomized algorithms that may work only under the assumption that G has no negative cycle but may still produce some (potentially incorrect) output when G has a negative cycle.

1.2 The high-level scaling approach

The algorithm is based on the following approach. Suppose we have a graph $G = (V, E, w)$ with $w(e) \geq -2B$ for all $e \in E$. The goal is to find a potential $\phi : V \to \mathbb{R}$ such that the new reduced weights with respect to ϕ, denoted by w_ϕ have the property that $w_\phi(e) \geq -B$ for all $e \in E$. We call such a potential a *halving potential*. Suppose we have an algorithm $\mathcal{A}$ that can either output a halving potential or detects a negative cycle in G in (randomized) $T(m, n, W)$ time. One can use $\mathcal{A}$ when weights are integer valued to get an algorithm for SSSP as follows.

Given G with integer weights $w : E \to \mathbb{Z}$. Consider new weights w' where $w'(e) = nw(e)$ for all $e \in E$. We now have $W' = nW$. Let G' be this new graph. Note that G' has a negative cycle iff G has. The important property of G' is the following which is easy to see because we have assumed integer weights for w.

Claim 4. *Let P and Q be any two s-v walks in G'. Then $|w'(P) - w'(Q)| = 0$ or $|w'(P) - w'(Q)| \geq n$. This also holds for the weight function w'_ϕ where ϕ is any potential on V.*

We can use the claimed algorithm $\mathcal{A}$ to detect a negative cycle in which case we stop. (Finding a negative cycle is trickier and we will discuss later how to do it.) We will now focus on finding

shortest path tree from s when there is no negative cycle. We apply $\mathcal{A}$ sequentially to find a potential ϕ to reduce the least negative weight to -1; each invocation reduces the least negative weight by a factor of 2. This takes $O(\log(nW))$ invocations. At the end, we have a potential ϕ such that $w'_\phi(e) \geq -1$ for all $e \in E$. Note that shortest paths in G' are the same as the shortest paths in G' and hence also in G. Now we truncate the negative weights to 0. That is, we let w'' be the weight function where $w''(e) = \max\{0, w'_\phi(e)\}$. Let G'' be the copy of G with weights w''. We compute a shortest path tree from s via Dijkstra's algorithm in G''. We simply output this shortest path tree — note that the shortest path disances in G can be computed from this tree easily.

Claim 5. *If G has no negative cycle then the shortest path tree rooted at s in G'' is a correct shortest path tree for s in G.*

Proof. Let P_v be the shortest path from s to v found in G'' and let Q_v be a shortest path from s to v in G' with weight w'_ϕ. Note that w'' differs from w'_ϕ only in truncating edges with weight -1 to 0. Thus

$$w'_\phi(Q_v) \leq w'_\phi(P_v) \leq w''(P_v) \leq w''(Q_v) \leq w'_\phi(Q_v) + (n-1)$$

since Q_v has at most $n-1$ edges. We claim that $w'_\phi(P_v) = w'_\phi(Q_v)$ which would imply that P_v is a correct shortest path in G'. To see this, if $w'_\phi(P_v) \neq w'_\phi(Q_v)$ then via Claim 4, $|w'_\phi(P_v) - w'_\phi(Q_v)| \geq n$ but this contradicts the preceding set of inequalities. Shortest paths in G' are the same as in G. ■

1.3 Low Diameter Decomposition for Directed Graphs

We saw LDDs in the context of undirected graphs and metrics. Here we are interested in directed graphs and distances. LDDs for directed graphs were implicit in algorithms for cut and flow problems for symmetric demands [KPRT97, Sey95, ENRS00] but the utility of their randomized variants have only recently been explored. Since reachability is asymmetric in directed graphs, in several problems of interest, we are focus on diameter of strongly connected components. Further, we also think of cuts as removing subsets of edges rather than edges leaving a set. With this in mind, a low-diameter decomposition of a directed graph $G = (V, E, w)$ with *non-negative* weights $w : E \to \mathbb{R}_+$ is to remove a set weight subset of edges $E' \subseteq E$ such that the diameter of each strong connected component in $G - E'$ is at most some given parameter D. The diameter of a strong connected component $C \subseteq V$ is $\max_{u,v \in C} d_G(u, v)$ where $d_G(u, v)$ is the distance between u and v in G. Note that this is a notion of "weak-diameter" since we are using distances in G. We can also ask for a strong diameter guarantee where $d_C(u, v) \leq D$ for all $u, v \in C$ where $d_C(u, v)$ is the distance using only edges inside $G[C]$.

We state a theorem from [BNWN22] on existence and fast computation of probabilistic LDDs.

Theorem 6 ([BNWN22])**.** *There is a randomized algorithm for low-diameter-decomposition of directed graphs with the following properties. It takes as input a directed graph $G = (V, E, w)$ with non-negative edge weights $w : E \to \mathbb{R}_+$ and a diameter bound $D > 0$. It outputs a random set of edges $E^{rem} \subseteq E$ such that*

- *The weak diameter of each strong connected component of $G - E^{rem}$ is at most D. More formally, for any $u, v \in V$ such that u and v are in the same strong connected component of $G - E^{rem}$, $d_G(u, v)$ and $d_G(v, u)$ are at most D.*
- *The probability that an edge is cut, that is, $\mathbf{P}[e \in E^{rem}]$ is at most $(O(\log^2 n)\frac{w(e)}{D} + n^{-10})$.*

The algorithm runs in time $O(m \log^2 n + n \log^3 n)$.

The term n^{-10} is negligible and is for technical reasons to obtain a fast algorithm. We will ignore it for the rest of the notes to keep it simpler. We will not describe the details of the above algorithm in these notes and the refer the reader to [BNWN22].

Remark 7. *[BCF23] describes an algorithm that guarantees strong diameter decomposition (where the distances in each strong connected component rather in the original graph are at most D), however the cutting probability increases to $O(\log^3 n)w(e)/D$. [BCF23] also describes a different decomposition procedure which helps them obtain a faster algorithm while not necessarily giving a low-diameter-decomposition.*

2 The main algorithm and its analysis

In this section we describe the main algorithm that forms the basis of the scaling algorithm. We call this algorithm ScaleDown.

Theorem 8. *There is a randomized algorithm* ScaleDown *that takes as input an edge-weighted graph $G = (V, E, w)$ with $w(e) \geq -2B$ for all e and has the following properties.*

- *If the algorithm terminates it outputs a halving potential $\phi : V \to \mathbb{R}$ such that $w_\phi(e) \geq -B$ for all e.*
- *If G has no negative cycle then the algorithm terminates in* expected *time $O(m \log^4 n)$ time.*
- *If G has a negative cycle the algorithm may not terminate but if it does terminate it outputs a halving potential.*

Note that we can check whether a given potential ϕ is a halving potential.

One can use ScaleDown and combine with the scaling approach that we discussed to obtain the following theorem.

Theorem 9. *There is a randomized algorithm* SPMain *that takes as input an edge-weighted graph $G = (V, E, w)$ with integer weights and has the following properties.*

- *If the algorithm has a negative cycle it does not terminate. terminates in* expected *time $O(m \log^4 n \log(nW))$ time and outputs a valid shortest path tree.*

Proof. We simply follow the scaling approach that we discussed in Section 1.2 and use the algorithm ScaleDown. ■

The algorithm ScaleDown is based on a hop-reduction step based on LDD and the use of two easy special cases that we describe next.

2.1 Two easy but important cases

It is easy to find shortest paths in DAGs (directed acyclic graphs) in $O(m+n)$ time via topological sort. This holds even with negative weights. This can be extended easily to obtain the following lemma.

Lemma 10. *Let $G = (V, E, w)$ be a directed graph such that the edges inside each strong connected component are non-negative. Then one can find a potential ϕ that neutralizes all negative weight edges in $O(m + n)$ time.*

Proof. Let $C_1, C_2, \ldots, C_h$ be the strong connected components of G and without loss of generality assume that the numbering of the components is a topological sort of the DAG G^* which is the SCC graph of G. Consider the potential $\phi : V \to \mathbb{R}$ where $\phi(v) = -iW$ for each $v \in C_i$. It is easy to check this potential neutralizes all negative weight edges and does not introduce any new negative weight edges. Computing the strong connected components and the topological sort can be done in linear time. ■

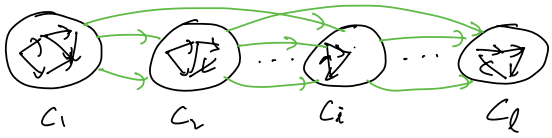

Figure 1: FixDagEdges. Set potential of all vertices in C_i to be $-iW$. Does not affect the edges inside the strong connected components since potential is same for all vertices of each components. Neutralizes all the green dag edges.

We will call the algorithm implied by the preceding lemma as FixDagEdges(G).

A second case is when we are guaranteed that G has s-v shortest paths with at most h negative weight edges for each v. Then one can adapt Bellman-Ford and Dijkstra to obtain shortest paths (equivalently a potential that neutralizes all negative weight edges) in $O(mh)$ time. We need a refinement when we don't have a fixed bound h for each vertex but an average bound.

Lemma 11 ([BNWN22])**.** *Suppose $G = (V, E, w)$ has no negative cycle. Let h_v be the maximum number of negative weight edges in an s-v shortest path. Then there is an algorithm that runs in $O(\log n(n + \sum_{v \in V} |\delta^+(v)|h_v)$ time that returns all the shortest path distances from s. If the guarantee does not hold then the algorithm may not terminate or return incorrect shortest path distance values.*

We leave the proof of the above as an exercise or refer the reader to [BNWN22]. Note that we ensured that the out-degree of each vertex is $O(1)$ which increased n to $O(m)$. Thus, when applied to our setting the running time is $O(m \log n + (\sum_v h_v) \log n)$. We can also find a potential that neutralizes all the negative weight edges in the same time. We call this procedure ElimNegEdges.

2.2 ScaleDown via hop reduction and fixing

We will assume that G has no negative cycle following earlier discussion. The algorithm is based on defining two graphs G^B and $G^B_{\geq 0}$ which basically modify the weights of the edges. We will use w^B and $w^B_{\geq 0}$ to refer to these weights.

We set $w^B(e) = w(e) + B$ if $w(e) < 0$ and $w^B(e) = w(e)$ otherwise. In other words we add B to each negative weight edge. Note that edges that remain negative in G^B are those with weight in the range $[-2B, B)$. We set $w^B_{\geq 0}(e) = \max\{0, w^B(e)\}$ so essentially we are setting all negative

weight edges in w^B to 0 to generate non-negative weights. Note that G^B also does not have neg cycle since G does not have one (and also $G^B_{\geq 0}$ for obvious reasons).

Let $H = (V_H, E_H)$ be any subgraph of G. We will use the notation H^B and $H^B_{\geq 0}$ to the corresponding subgraphs of G^B and $G^B_{\geq 0}$.

Claim 12. *Suppose we find a potential ϕ that neutralizes all negative weight edges in G^B. Then $w_\phi(e) \geq -B$ for all $e \in E$.*

Proof. Fix $e = (u, v)$. If $w(e) \geq 0$ the potential ϕ does not effect it (note that this is our assumption) since $w^B(e) = w(e)$. If $w(e) < 0$ then there are two cases. If $w(e) \geq -B$ then $w^B(e) \geq 0$ and by our assumption $w^B_\phi(e) \geq 0$ which implies that

$$w_\phi(e) = \phi(u) + w(e) - \phi(v) = \phi(u) + w^B(e) - B - \phi(v) = w^B_\phi(e) - B \geq -B.$$

If $w(e) < B$ then e is still negative in G^B and $w^B(e) = w(e) + B$ and $w^B_\phi(e) \geq 0$ and it can be easily verified that $w_\phi(e) \geq -B$. ■

Thus our goal is to find a potential to neutralize all edges in G^B. It is not quite obvious why it is easier to work with G^B but a high-level intuition is that if G does not a negative length cycle then G^B is making the problem easier. [BCF23] uses a slightly different scheme which ensures that the minimum mean cycle length is ≥ 1 which formalizes more explicitly the change in the graph by adding B to the negative weight edges.

To find a potential to neutralize negative weight edges in G^B we add a dummy source s with edges of length 0 to each vertex v and try to find shortest path distances from s. We will use the notation $G^B + s$ to indicate the graph obtained by adding such a dummy vertex.

A key definition is the following.

Definition 3. *Let $\eta(G^B, v)$ denote the maximum number of negative weight edges among all shortest s-v paths in $G^B + s$ and let $\eta(G^B) = \max_{v \in V} \eta(G^B, v)$. For each v let $P(G^B, v)$ be a shortest s-v path that achieves $h(G^B, v)$.*

In other words we are interested in the negative weight hop distance. Note that if $\eta(G^B)$ is small or even if the average $\frac{1}{n}\sum_v \eta(G^B, v)$ is small (say a poly-logarithmic factor) then via Lemma 11 we can solve the problem in near-linear time. Our goal is to reduce the problem to this setting by using LDD, recursion, and DAG edge fixing.

It is useful to consider the structure of shortest paths $P(G^B, v)$ in G^B. Note that since s is connected to each vertex v with edge of weight 0 the shortest path distances from s are at most 0. Thus, if $\eta(G^B, v) > 0$ for a path $P(G^B, v)$ then the weight of any prefix of that path is at most 0. See Fig **??**.

Key lemma on using LDD: The key lemma that enables recursion is to reduce $\eta(G^B, v)$.

Lemma 13. *Suppose $\eta \geq \eta(G^B)$ is an upper bound on $\eta(G^B)$. Suppose we do an LDD via Theorem 6 on $G^B_{\geq 0}$ with diameter parameter $D = \frac{\eta}{2}B$ and let E^{rem} be the edges removed by the LDD procedure. Then,*

- *For any SCC $H = (V_H, E_H)$ of $G - E^{rem}$ we have $\eta(H^B) \leq \eta/2$.*

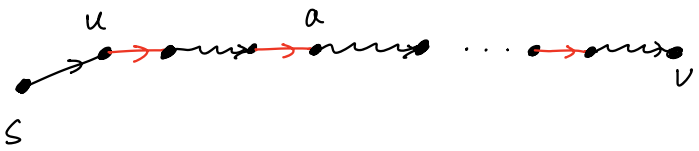

Figure 2: Shortest s-v path in G^B with negative weight edges shown in red. Note that some of the original negative weigh edges in G may no longer be negative in G^B. Since dummy source s has edge of 0 weight to each vertex the weight of any prefix of the path (say to vertex a) has to be at most 0.

- *For any vertex v, $E[|P(G^B, v) \cap E^{rem}|] \leq O(\log^2 n)$.*

Proof. The first property is the trickier one. We refer to Fig 3. We will consider all three graphs $G, G^B, G^B_{\geq 0}$ as well as H so it is important to keep track of the argument closely.

Consider the SCC $H = (V_H, E_H)$ and the graph $H^B + s$. Fix a vertex $v \in V_H$ and a shortest s-v path $P(H^B, v)$ in H^B. This path starts with an edge (s, u) where $u \in V_H$. Let P' be the part of the path from u to v. Note that $u, v \in H$ which is a strong connected component with diameter at most D in $G^B_{\geq 0}$. This implies that there is a path Q from v to u in G such that $w^B_{\geq 0}(Q) \leq D = \eta B/2$. Note that $w(Q) \leq w^B(Q) \leq w^B_{\geq 0}(Q) \leq D$ since edge weights only increased by going from G to $G^B_{\geq 0}$.

We now want to understand $w(P')$. Note that this is the weight of the path in G. Since $P(H^B, v)$ is a shortest path from s to v in H^B and s has 0 weight edges to each vertex, $w^B(P') \leq 0$. Consider any negative weight edge e on P' with respect to w^B. Since $w^B(e) < 0$, it implies that $w(e) = w^B(e) - B$ since we added B to each negative weight edge of G to get G^B. Suppose $P(H^B, v)$ has more than $\eta/2$ negative weight edges. This means that $w(P') \leq w^B(P') - (\eta/2+1)B \leq -(\eta/2+1)B$ since $w^B(P') \leq 0$. But then $w(P') + w(Q) < -(\eta/2+1)B + D < 0$ since $D = \eta B/2$. This implies that we have a closed walk of negative weight in G which implies that G has a negative weight cycle contradicting our assumption.

For the second part we consider $P(G^B, v)$. What is its weight according $w^B_{\geq 0}$? We have $w^B(P(G^B, v)) \leq 0$. Since it has at most η negative weight edges according to w^B, when we make those edges 0 we increase the weight of the path to ηB since each negative weight edge in w^B is $\geq -B$. Thus $w^B_{\geq 0}(P(G^B, v)) \leq \eta B$. An edge of weight x is cut with probability $\frac{x}{D} O(\log^2 n)$. Thus, by linearity of expectation the total number of edges in $P(G^B, v)$ that are cut by the LDD procedure is

$$\leq O(\log^2 n) \frac{w^B_{\geq 0}(P(G^B, v))}{D} \leq \frac{\eta B}{\eta B/2} O(\log^2 n) = O(\log^2 n).$$

■

Remark 14. *A reader may wonder about the following. Consider a vertex v and a path $P(G^B, v)$ with η negative weight edges. Why does that path not exist in H^B since we proved that in H^B the shortest path to v can only contain $\eta/2$ negative weight edges? The point is the following. Suppose $P(G^B, v)$ has (s, u') as its first edge. Then the proof is showing that there cannot be a (v, u') path*

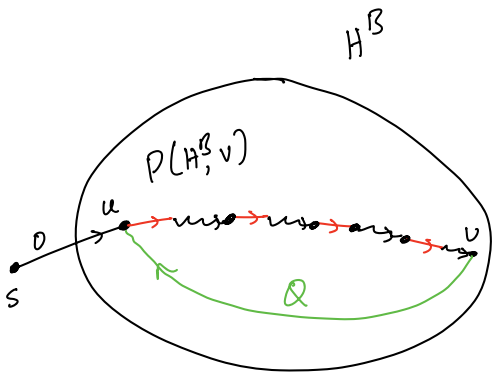

Figure 3: Proof of the key lemma. $w(Q) \leq w^B(Q) \leq w^B_{\leq 0}(Q) \leq D = \eta B/2$. If $P(H^B, v)$ has more $\eta/2$ negative weight edges then $w(P(H^B, v) - u) \leq -(\eta/2 + 1)B$ which implies closed walk in G of negative weight.

in G with weight at most $D = \eta B/2$. Thus u' and v cannot be in the same SCC with diameter bound D. Thus the shortest path to v inside H^B is not going to be able to use u'.

Recursive algorithm: The properties of the key lemma point to a (reasonably natural in retrospect) recursive algorithm that reduces the hop length, recurses and fixes the remaining edges. Recall that the goal is to find a potential ϕ that neutralizes all negative weight edges in G^B.

The algorithm start with an upper bound of $\eta = n$ on $\eta(G^B)$ and computes a set of edges E^{rem} using the LDD procedure with $D = \eta B/2$. For each SCC H in $G - G^{rem}$ we recurse since $\eta(H^B) \leq \eta/2$. This recursively yields a potential ϕ_1^H on vertices of H that neutralizes all the edges inside H. We find potentials separately in each SCC and since the SCCs partition the vertex set we find an overall potential ϕ_1 that neutralizes all edges inside the SCCs. Note that $G - E^{rem}$ consists of other edges which are not inside any SCC. We call these the DAG edges since they induce a DAG on the SCCs. Once all the edges inside SCCs are neutralized via ϕ_1, we can use the procedure FixDAGEdges to neutralize these edges via a potential ψ. Now we consider $\phi_2 = \phi_1 + \psi$. Thus the only negative weight edges in G^B with respect to the potential ϕ_2 are the edges in E^{rem} which have been removed. The crucial property we can exploit is the following which is simply a property of potentials.

Claim 15. *With respect to the potential ϕ_2, $P(G^B, v)$ is a shortest path from s to v.*

The key lemma showed that $E[|P(G^B, v) \cap E^{rem}|] = O(\log^2 n)$. Thus, in the graph G^B with potential ϕ_2, for each vertex v there is a shortest path from s with $O(\log^2 n)$ negative weight edges in expectation. Then we are done since we can use linearity of expectation and the algorithm implied by Lemma 11.

Now we formally state the algorithm. We call it ReduceHopsAndFix(G^B, η) with the guarantee that $\eta(G^B) \leq \eta$ and outputs a potential that neutralizes all negative weight edges.

Algorithm ReduceHopsAndFix(G^B, η)

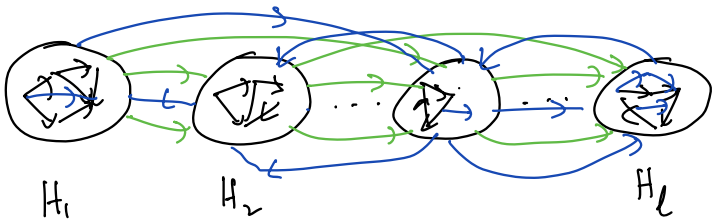

Figure 4: Illustration of hop reduction followed by fixing. Blue edges are E^{rem} removed by LDD procedure. The green edges are the DAG edges in $G^B - E^{rem}$. $H_1, H_2, \ldots, H_\ell$ are the strong connected components of $G^B - E^{rem}$ in topological order.

1. If $(\eta \leq 2)$ then use algorithm implied by Lemma 11 to find potential ϕ and return it
2. E^{rem} is output LDDProcedure$(G^B_{\geq 0}, D = \eta B/2)$
3. Let $H_1, H_2, \ldots, H_\ell$ be the SCCs in $G - E^{rem}$ and let E^{dag} be edges of $G - E^{rem}$ not inside the SCCs
4. For $i = 1$ to ℓ do
   (a) Compute $\phi^i_1 : V_{H_i} \to \mathbb{R}$ on vertices of H_i via recursive call ReduceHopsAndFix$(H^B, \eta/2)$
5. $\phi_1 : V \to \mathbb{R}$ is obtained by amalgamating ϕ^i_1, $i \in [\ell]$.
   // *ϕ_1 neuralizes all edges inside SCCs in $G^B - E^{rem}$*
6. Use FixDagEdges in $(G^B - E^{rem})_{\phi_1}$ to neutralize all DAG edges. Let ψ be the potential computed.
7. $\phi_2 = \phi_1 + \psi$.
   // *Only negative weight edges left in G^B with respect to ϕ_2 are those in E^{rem}.*
8. Use ElimNegEdges on graph $(G^B)_{\phi_2}$ via Algorithm in Lemma 11 to neutralize edges in E^{rem}. Let ϕ_3 be the potential computed in this step.
9. Let $\phi = \phi_1 + \psi + \phi_3$.
10. If $w^B_\phi(e) < 0$ for any edge e then go into an infinite loop
11. Return ϕ

Correctness: Correctness essentially follows from the key lemma, the properties of potentials, the correctness of the algorithms to fix dag edges and to find shortest paths with small number of negative hops, and induction. Note that, under the assumption that G has no negative cycle, each step of the algorithm correctly computes its output and the algorithm becomes a correct Las Vegas algorithm. If G has a negative cycle then the algorithm is not guaranteed to terminate. Since we check validity of ϕ before returning it, we ensure that it returns a valid output if it terminates.

Time analysis: We now analyze the time complexity. Note that the algorithm has recursion depth $O(\log \eta^*)$ where η^* is the parameter used for the first call. We use $\eta^* = n$ because that is a valid upper bound on $\eta(G^B)$. Hence recursion depth is $O(\log n)$. We consider the expected time outside of the recursion. In the base case it is easy to see that the running time is $O(m \log n)$ via Lemma 11. The LDD procedure takes $O(m \log^3 n)$. We use the algorithm implied by Lemma 11 in the last step when neutralizing the negative weigh edges in E^{rem}. Here we use the second property of the key lemma. Recall that $E[|P(G^B, v) \cap E^{rem}|] = O(\log^2 n)$. Thus, by linearity of expectation $E[\sum_v |P(G^B, v) \cap E^{rem}|] = O(n \log^2 n)$. Thus, the expected running time of the algorithm from Lemma 11 in the last step is $O(m \log^3 n)$. Thus, the total time outside of the recursion is $O(m \log^3 n)$. In each recursive call we reduce the parameter η by half and the total sum of the edges inside the SCCs is at most m, thus the total expected time of the algorithm is simply the depth of the recursion times the time outside the recursion. Thus, the expected time of the algorithm is $O(m \log^4 n)$.

ScaleDown(G) is nothing but ReduceHopsAndFix(G^B, n). We had argued the desired properties of this algorithm which proves Theorem 9.

3 Putting things together

We now use ScaleDown and SPMain to obtain a Montecarlo algorithm.

Theorem 16. *There is a randomized algorithm* SPMonteCarlo *that given a graph $G = (V, E, w)$ with integer weights and a source $s \in V$ behaves as follows.*

- *If G contains a negative weight cycle it returns an error message.*
- *If G does not contain a negative weight cycle then the algorithm returns a shortest path tree rooted at s with high probability. It may return an error message even if there is no negative cycle.*

The algorithm runs in $O(m \log^5 n \log(nW)))$ time.

Proof. We first consider the following intermediate algorithm SPIntermediate. It runs SPMain on input G, s; if it terminates before twice its expected run time and returns a shortest path tree then SPIntermediate returns the same output. If SPMain exceeds twice its expected run time then the algorithm is stopped and SPIntermediate returns error. By Markov's inequality, this intermediate algorithm has the following features: (i) if G has no negative cycle it returns a correct a shortest path tree with probability $1/2$ (ii) if G has negative cycle it always returns error and (iii) it may return error even if G has no negative cycle with probability at most $1/2$.

The desired algorithm SPMonteCarlo is simply running SPIntermediate $O(c \log n)$ times independently and returning error if all of the invocations return error. It is easy to check that it has the desired properties. The run-time is $O(c \log n)$ times the expected run-time of SPMain. ■

SPMonteCarlo can be viewed as almost what we would like because it returns a valid shortest path tree with high probability when it exists. But it may return an error with a small probability even when there is no negative cycle. Ideally we would also like an algorithm that can detect and output a negative cycle with high probability when G has one. Suppose we had such an algorithm then we can run the two algorithms. In the next subsection we discuss an algorithm to find a negative cycle.

3.1 Finding a negative cycle

In this section we discuss a scaling algorithm to find a negative cycle assuming we have an algorithm $\mathcal{A}$ with the following properties: $\mathcal{A}$ detects when G has a negative cycle and otherwise outputs a valid potential that neutralizes all negative weight edges. We dont' have a deterministic algorithm of that form but we do have SPMonteCarlo which can act as a substitute. We will not dwell into all the details to make the whole process work with the weaker algorithm and refer the reader to [BNWN22, BCF23].

Finding a negative cycle is not quite as easy as it may appear. First, we assume that $\mathcal{A}$ has detected that G has a negative cycle. Given $G = (V, E, w)$ with integer weights we start scaling all weights by n^3 so we set $w'(e) = n^3 w(e)$. Let G_0 be this new graph.

Claim 17. *If G has a negative cycle then G_0 has a negative cycle with weight $\leq -n^3$.*

Given G and an integer $M \geq 0$ we define a graph G^{+M} as the graph obtained by adding M to each edge weight. Note that we add M to all edges, not just negative weight edges as we did earlier in defining G^B.

Let M^* be the smallest integer such that $G_0^{+M^*}$ does *not* have a negative cycle. It means that $G_0^{+(M^*-1}$ has a negative cycle.

Claim 18. *$M^* \geq n^2$ and $M^* \leq n^3 W$. Given access to $\mathcal{A}$ that can detect a negative cycle using binary search one can find M^* using $O(\log(nW))$ calls to $\mathcal{A}$.*

Proof. Since there is a negative weight cycle C in G_0 of weight $\leq -n^3$, to make it postive we need to add at least n^2 to each edge since $|C| \leq n$. The second part is easy to see. ■

The following claim is easy to see.

Claim 19. *Since $G_0^{+M^*}$ does not have a negative cycle there is a potential $\phi : V \to \mathbb{Z}$ such ϕ neutralizes all negative weight edges in $G_0^{+M^*}$ and this can be computed by $\mathcal{A}$.*

The main lemma that leads to the algorithm is the following.

Lemma 20. *Let ϕ be a potential that neutralizes the negative weight edges in $G_0^{+M^*}$. Let $E' = \{e \mid w_0^{+M^*}(e) > n\}$. Let $G_1 = (V, E - E')$. Then G_1 contains a cycle, and* any *cycle in G_1 is a negative cycle in G.*

Proof. There is a negative cycle C in $G_0^{+(M^*-1)}$ which is also a negative weight cycle in G. How much does its weight increase in $G_0^{+(M^*)}$? Only by $|C|$. Thus the weight of C in $G_0^{+(M^*)}$ is at most n. Note that ϕ neutralizes all negative weights in $G_0^{+(M^*)}$ and thus all edge weights of C become non-negative with respect to ϕ but the total weight of C does not change with respect to ϕ since potentials do not change cycle weights. Thus $(w_0^{+M^*})_\phi(C) \leq n$ and it contains only non-negative edge weights. Thus any edge in C must have weight at most n with respect to the weight function $(w_0^{+M^*})_\phi$. This means $E(C) \cap E' = \emptyset$ and thus all edges of C survives in G_1 which implies that G_1 must have a cycle.

Now we argue that any cycle in G_1 is a negative weight cycle in G. Let C be an arbitrary cycle in G_1. Since C does not have any edges from E' we have $(w_0^{+M^*})_\phi(C) \leq n|C| \leq n^2$. Since potentials do not change cycle weights we have $w_0^{+M^*}(C) \leq n^2$. We have

$$w_0(C) = w_0^{+M^*}(C) - M^*|C| \leq w_0^{+M^*}(C) - 2M^* \leq n^2 - 2n^2 < 0.$$

We used the fact that $|C| \geq 2$ and the fact that $M^* \geq n^2$. Thus the weight of C in G_0 is negative which implies it is negative in G as well. ■

Based on the above discussion we see that the following algorithm outputs a negative cycle assuming access to $\mathcal{A}$ and moreover it invokes $\mathcal{A}$ only $O(\log(nW))$ times.

FindNegCycle(G, w)

1. Let G_0 be a copy of G where $w_0(e) = n^3 w(e)$ for all e
2. Fine smallest M^* using $O(\log(nW))$ calls to $\mathcal{A}$ such that $G_0^{+M^*}$ does not have a negative cycle
3. Use $\mathcal{A}$ to find a potential ϕ that neutralizes all negative weight edges in $G_0^{+M^*}$.
4. Let $E' = \{e \mid w_0^{+M^*}(e) > n\}$
5. Find any cycle C in $G - E'$ and output it.

Note that we can use SPMonteCarlo in place of $\mathcal{A}$. The resulting algorithm may make mistakes and fail but we can detect failure in finding a negative cycle, and thus we can obtain a Monte Carlo algorithm that can find a negative cycle with high probability if G has one.

References

[BCF23] Karl Bringmann, Alejandro Cassis, and Nick Fischer. Negative-weight single-source shortest paths in near-linear time: Now faster! In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 515–538. IEEE, 2023.

[BNWN22] 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.

[ENRS00] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-conquer approximation algorithms via spreading metrics. *Journal of the ACM (JACM)*, 47(4):585–616, 2000.

[Fin24] Jeremy T Fineman. Single-source shortest paths with negative real weights in $\tilde{O}(mn^{8/9})$ time. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 3–14, 2024.

[Gol95] Andrew V Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, 1995.

[HJQ24] Yufan Huang, Peter Jin, and Kent Quanrud. Faster single-source shortest paths with negative real weights via proper hop distance. *arXiv preprint arXiv:2407.04872*, 2024.

[KPRT97] Philip N Klein, Serge A Plotkin, Satish Rao, and Eva Tardos. Approximation algorithms for steiner and directed multicuts. *Journal of Algorithms*, 22(2):241–269, 1997.

[Sey95] Paul D. Seymour. Packing directed circuits fractionally. *Combinatorica*, 15(2):281–288, 1995.