Graphs : Shortest Paths : Bellman-Ford[1]

We have now seen how to find shortest paths in graphs when costs are positive, and also seen how Dijkstra's algorithm fails when some edges are negative. But in some applications, some edges can indeed have negative costs. Or to put it differently, sometimes certain edges can have "profits". Can you think of such a situation? (Hint: the "Ferryman" problem in PSet 0?) Can we think of an algorithm in this case?

Turns out that we can ***if*** the graph has no negative-cost cycles. That is, a cycle whose sum of costs is negative. Recall, when graphs do have negative cost cycles, valid distance labels cannot exist; so any algorithm trying to find shortest paths must use some other kind of certificate (and none are known). There is another reason why the absence of negative cost cycles makes life algorithmically easier: we can focus on "easier" object than paths. Recall a walk in a graph is an alternating sequence of vertices and edges which can repeat (while a path can't). As we will see, finding short walks are easy. And when there are no negative cost cycles, the shortest cost walk from s to v can be no smaller than the shortest cost path: after all, a walk is nothing but a path + some cycles, and all cycles have non-negative cost. That is the main idea of the algorithm we will see today. At this point I should mention there is a different algorithm which we have also alluded to in the class which solves this problem — this is explored in the problem sets. Read it.

1 Directed Graphs without Negative Cost Cycles

As mentioned above, we will be solving the shortest walk problem. In particular, we solve the *length bounded shortest cost walks*.

LENGTH BOUNDED SHORTEST WALK
Input: Directed graph $G = (V, E)$ with costs $c(e)$ on edges which could be negative; a source vertex s; a parameter k.
Output: Minimum Cost *walk* w from s to v of length $\leq k$, for all $v \in V$.

Lemma 1. If G has no negative cost cycles, then the minimum cost walk from s to v of length $\leq n-1$ is the shortest path from s to v.

Proof. Clearly a path from s to v is a walk of length $\leq n-1$, so what we really need to prove is the reverse direction. That is, given a walk of length w from s to v of length $\leq n-1$, there is a path of the same or less cost. But note that we proved in graph-basics.pdf that any walk contains a subset of edges which is a path. If we recall that proof, we see that the edges left out constitutes a bunch of circuits. Since no circuit is of negative cost, we get that the cost of the path is at most the cost of the walk. □

We use dynamic programming to solve LBSW. Consider the minimum cost walk $w = (s, e_0, v_1, e_1, \ldots, v_{\ell-1}, e_{\ell-1}, v)$ from s to v of length $\ell \leq k$. Can we break this solution into pieces? Well, what can we say about the walk $w' = (s, e_0, \ldots, v_{\ell-1})$? We claim that this is the minimum cost walk from s to $v_{\ell-1}$ of length at most $k-1$. Why? Suppose not, and there was another walk w'' of strictly smaller cost. Then adding the edge $e_{\ell-1}$ to w'' would give a length $\leq k$ walk from s to v of strictly smaller cost than w. Furthermore, if we had the smallest cost length $\leq k-1$ length walks from s to x for all vertices x which have an edge to v, we should be able to find the smallest cost length $\leq k$ walk from s to v. Time to write the DP methodically now.

[1]*Lecture notes by Deeparnab Chakrabarty. Last modified : 20th Feb, 2024*
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

- ***Definition.*** We define $\mathsf{dist}[v,i]$ for all $v \in V$ and $0 \le i \le k$ as the cost of the minimum cost walk from s to v of length $\le i$. We are interested in $\mathsf{dist}[v,k]$ for all $v \in V$.

- ***Base Case.*** $\mathsf{dist}[s,0] = 0$; $\mathsf{dist}[v,0] = \infty$ for all $v \neq s$.

- ***Recurrence.*** The recurrence is this

$$\forall v, i > 0, \quad \mathsf{dist}[v,i] = \min\left(\mathsf{dist}[v,i-1], \min_{u:(u,v)\in E} \left(c(u,v) + \mathsf{dist}[u,i-1]\right)\right) \tag{1}$$

- ***Proof.*** Let's elaborate on the English explanation above. Let $\mathsf{Cand}(v,i)$ denote the set of walks from s to v of length at most i. Thus,

$$\mathsf{dist}[v,i] = \min_{w\in\mathsf{Cand}(v,i)} c(w)$$

where $c(w) = \sum_{e\in w} c(e)$.

($\le$) Clearly, $\mathsf{Cand}(v,i-1) \subseteq \mathsf{Cand}(v,i)$ and so $d[v,i] \le d[v,i-1]$. Now fix any u with $(u,v) \in E$, and let w' be the walk in $\mathsf{Cand}(u,i-1)$ of cost $\mathsf{dist}[u,i-1]$. Then $w' \circ (u,(u,v),v)$ is a walk in $\mathsf{Cand}(v,i)$ of cost $\mathsf{dist}[u,i-1] + c(u,v)$. This takes care of this case.

Remark: *Do you see how the proof above **fails** if we said paths instead of walks? It is important that you do. w' could already have v...*

($\ge$) On the other hand, fix $w \in \mathsf{Cand}(v,i)$ of length $\le i$ and cost $\mathsf{dist}[v,i]$. If the walk is not of length i, then $w \in \mathsf{Cand}(v,i-1)$ and thus $\mathsf{dist}[v,i-1] \le \mathsf{dist}[v,i]$. Otherwise, the walk is of length $i \ge 1$, and so let u be the second-to-last vertex in this walk. But then the subset w' of the walk w ending at u is a walk in $\mathsf{Cand}(u,i-1)$. Since $c(w') = c(w) - c(u,v)$, we get $\mathsf{dist}[u,i-1] \le \mathsf{dist}[v,i] - c(u,v)$.

- ***Pseudocode.***

```
1:  procedure LBSW($G, s, k, c$):
2:      ▷ Returns $\mathsf{dist}[v,i]$ for every $v$ and $0 \le i \le k$.
3:      ▷ Also returns $\mathsf{parent}[v,i]$ for every $v$ and $0 \le i \le k$.
4:      $\mathsf{dist}[s,0] \leftarrow 0$; $\mathsf{dist}[v,0] \leftarrow \infty$ for all $v \neq s$. ▷ Base Case
5:      $\mathsf{parent}[s,0] \leftarrow \bot$; $\mathsf{parent}[v,0] \leftarrow \bot$ for all $v \neq s$. ▷ Base Case
6:      for $i = 1$ to $k$ do:
7:          for $v \in V$ do:
8:              $\mathsf{dist}[v,i] \leftarrow \min\left(\mathsf{dist}[v,i-1], \min_{u:(u,v)\in E}(\mathsf{dist}[u,i-1] + c(u,v))\right)$.
9:              ▷ Takes $\deg^-(v)$ time
10:             ▷ We abuse notation and say $\infty + c = \infty$ for any finite $c$.
11:             If $u$ is the vertex that minimizes, $\mathsf{parent}[v,i] \leftarrow u$.
12:     ▷ At this point $\mathsf{dist}[v,k]$ has the minimum cost walk of length at most $k$
13:     ▷ The $\mathsf{dist}[v,k]$'s can be used to recover the paths as well; parents can also be used.
```

- ***Running Time and Space.*** The space required is $O(nk)$. The running time is $O(k(n+m))$.

Lemma 1 states that if we are promised that G has no negative cost cycle, then LBSW$(G, s, n-1)$ will indeed return $\mathsf{dist}[v, n-1]$ for all vertices which are the shortest path lengths. However, this does raise the following question: how would we ***know*** whether or not G has a negative cost cycle? Is there an algorithm that can detect it? It so happens that the previous algorithm is powerful enough to do this as well. Consider running LBSW on G, s, k with $k = n$. Then note that if G has no negative cost cycle C reachable from s, then $\mathsf{dist}[v, n-1] = \mathsf{dist}[v, n]$ for all v; the minimum cost walk of length at most $n-1$ is the shortest path which is also the minimum cost walk of length at most n. The following lemma shows that if G has a negative cost cycle C reachable from s, then the above ***cannot be true*** for all v. And therefore, just checking this for all v would help us detect a negative cost cycle.

Lemma 2. Suppose there is a negative cost cycle C in G and suppose C has a vertex reachable from s. Consider running LBSW on G, s, k with $k = n$. Then, there exists a vertex $v \in C$ such that $\mathsf{dist}[v, n] < \mathsf{dist}[v, n-1]$.

Proof. Let $C = (x_1, \ldots, x_k, x_1)$ be the negative cost cycle reachable for s. Since (x_i, x_{i+1}) is an edge for all $1 \leq i \leq k$ (with the abuse $k+1 = 1$), using (1) we get

$$\mathsf{dist}[x_{i+1}, n] \leq \mathsf{dist}[x_i, n-1] + c(x_i, x_{i+1}), \quad \forall 1 \leq i \leq k$$

Adding these up, we get

$$\sum_{v \in C} \mathsf{dist}[v, n] \leq \sum_{v \in C} \mathsf{dist}[v, n-1] + c(C) < \sum_{v \in C} \mathsf{dist}[v, n-1]$$

since $c(C) < 0$. This means $\mathsf{dist}[v, n] < \mathsf{dist}[v, n-1]$ for some $v \in C$. □

```
1:  procedure BELLMAN-FORD($G, s$):
2:      Run LBSW($G, s, n$) to obtain dist$[v, i]$ and parent$[v, i]$ for all $v$ and $0 \leq i \leq n$.
3:      if there exists $v$ with dist$[v, n]$ < dist$[v, n-1]$ then:
4:          ▷ There is a negative cost cycle containing $v$.
5:          ▷ The following code uses the parent pointers to recover it.
6:          $x = v; j = n$
7:          while $x \neq v$ or $j = n$ do:
8:              Add (parent$[x, j]$, $x$) to $C$.
9:              $x$ = parent$[x, j]$.
10:             $j = j - 1$.
11:         return C
12:     else:
13:         ▷ dist$[v, n]$'s are indeed shortest path lengths, and construct the shortest path tree
14:         Add all vertices with dist$[v, n] < \infty$ to $T$.
15:         for $v \in T$ do:
16:             Add (parent$[v, n]$, $v$) to $T$.
17:         return $T$.
```

Therefore, given any graph G and a source s, if we run LBSW on G, s with $k = n$, then either some $\mathsf{dist}[v, n] < \mathsf{dist}[v, n-1]$ for some v implying a negative cost cycle reachable from s, or $\mathsf{dist}[v, n-1]$ contains the cost of the shortest paths from s to v. This is the BELLMAN-FORD algorithm.

1.1 A Space Saver and a Better(?) Presentation

Our presentation of the BELLMAN-FORD algorithm above is arguably a bit circuitous than what you would see in the textbooks. But it is actually the *same* algorithm. Let's try to see now how to get to these "textbook" presentations.

First thing we notice is that Line 8 of LBSW can be done "edge-by-edge" instead of "vertex-by-vertex". That is, we could replace the for-loop (Line 7 in LBSW) as follows:

$$\text{For all edges}(u,v) \in E : \mathsf{dist}[v,i] = \min\left(\mathsf{dist}[v,i-1], \mathsf{dist}[u,i-1] + c(u,v)\right))$$

That is, instead of minimizing over all out-edges in Line 8 of LBSW, we can stagger the minimizations.

The second observation is that since the BELLMAN-FORD algorithm is only interested in the values of $k = n-1$ and $k = n$, we really don't need to take care of the parameter i in $\mathsf{dist}[v,i]$ for all the other i's. Of course, the way the DP builds up, it needs the smaller i's to get to the final $n-1$ and n. But note that the DP *can also forget* (we sort of saw this in FIBONACCI: instead of the array, one could just keep the "last two" values). That is, we just store $\mathsf{dist}[v]$ for each vertex, and in the ith loop, we set $\mathsf{dist}_{\mathsf{new}}[v]$ for each vertex v, and once all the changes are done, we reset $\mathsf{dist}_{\mathsf{new}}$'s to become dist's.

```
1: procedure BELLMAN-FORD-LOW-SPACE($G, s$):
2:     $\mathsf{dist}[s] = 0$ for all $j$; $\mathsf{dist}[v] = \infty$ for all $v \neq s$. ▷ Base Case
3:     for $i = 1$ to $n-1$ do:
4:         for $(u,v) \in E$ do:
5:             if $\mathsf{dist}[v] > \mathsf{dist}[u] + c(u,v)$ then:
6:                 Set $\mathsf{dist}_{\mathsf{new}}[v] \leftarrow \mathsf{dist}[u] + c(u,v)$
7:                 Set $\mathsf{parent}[v] \leftarrow u$.
8:         ▷ At this point, after ith for loop, $\mathsf{dist}_{\mathsf{new}}[v]$ has what the $\mathsf{dist}[v,i]$'s have from LBSW
9:         ▷ Now revert all $\mathsf{dist}_{\mathsf{new}}$'s to dist's
10:        Set $\mathsf{dist}[v] \leftarrow \mathsf{dist}_{\mathsf{new}}[v]$ for all $v$.
11:    ▷ At this point, $\mathsf{dist}[v]$ is really $\mathsf{dist}[v, n-1]$. We do a final check for negative cost cycles.
12:    for $(u,v) \in E$ do:
13:        if $\mathsf{dist}[v] > \mathsf{dist}[u] + c(u,v)$ then:
14:            return NEGATIVE CYCLE
```

We will end this lecture with one final subtlety. One doesn't keep an extra $\mathsf{dist}_{\mathsf{new}}[v]$ for each vertex v, but goes ahead and changes the *same* variable. This could lead a change percolate "faster" in the same iteration i. We can no longer say, though, that at the end of the ith iteration, $\mathsf{dist}[v]$ contains the shortest cost walk of length $\leq i$, but one *can*, there is a walk of cost equal to $\mathsf{dist}[v]$ and so, if there are no negative cost cycles, this is at most $\mathsf{dist}[v, n-1]$. So the "$(n-1)$th column" behavior remains the same. More precisely, one can prove after iteration i, $\mathsf{dist}[v] \leq \mathsf{dist}[v,i]$ as in LBSW, and furthermore since there is a walk of cost $\mathsf{dist}[v]$, $\mathsf{dist}[v] \geq \mathsf{dist}[v, n-1]$. So, at the end of the $(n-1)$th iteration, $\mathsf{dist}[v]$ equals $\mathsf{dist}[v, n-1]$. One may wonder if the "negative cycle catching" is jeopardized. Well, if the check over all edges described in Lines 12 and 14, then the proof of Lemma 2 would still show what we need to show: if there is a negative cost cycle reachable from s, then one of the distance labels *must* be modified. So, the following algo, and which is one which you may find in textbooks and other online sources, is indeed correct (though you can't stop at any arbitrary iteration and get LBSWs).

```
procedure BELLMAN-FORD-TXTBK($G, s$):
    dist$[s] = 0$ for all $j$; dist$[v] = \infty$ for all $v \neq s$. ▷ Base Case
    for $i = 1$ to $n - 1$ do:
        for $(u, v) \in E$ do:
            dist$[v] = \min($dist$[u] + c(u, v)$, dist$[v])$.
            ▷ We abuse notation and say $\infty + c = \infty$ for any finite $c$.
            If dist$[v]$ modified, set parent$[v] \leftarrow u$.
    ▷ At this point, dist$[v]$ is really dist$[v, n - 1]$. We do a final check for negative cost cycles.
    for $(u, v) \in E$ do:
        if dist$[v] >$ dist$[u] + c(u, v)$ then:
            return NEGATIVE CYCLE
```

Remark: *We have been talking about directed graphs so far. What about the same problem in* ***undirected graphs****? More precisely, can we detect if an undirected graph has a negative cost cycle? Turns out this is tricky. Why? The reason is that a cycle in an undirected graph has to be of length 3 or more. That is, $\{(u, v), (v, u)\}$ is not a cycle in an undirected graph. However, in a directed graph a pair of antiparallel edges is indeed a cycle. So, how will you go about finding whether an undirected graph has a negative cost cycle?*

Remark: *Ok, what about shortest* ***paths*** *when there are negative cost edges* and *negative cost cycles? Paths are not allowed to repeat vertices, so this problem is indeed well-defined. How will we solve this one? Short answer: no one knows, and many people* ***believe*** *that no good algorithm exists! For those who know the jargon, it is NP-hard.*