

CS 31: Algorithms (Spring 2019): Lecture 14


Date: 9th May, 2019

Topic: Graph Algorithms 4: Directed graphs with negative cost cycles

Disclaimer: These notes have not gone through scrutiny and in all probability contain errors.

Please notify errors on Piazza/by email to deeparnab@dartmouth.edu.

1 Directed Graphs without Negative Cost Cycles

In this section, we look at *directed*¹ graphs with negative costs for which we can find the shortest paths efficiently. These are ones which have no negative cost cycles. That is, for every cycle C in the graph, we have $\sum_{e \in C} c(e) \geq 0$. We will be looking at directed graphs. 

Exercise: *Prove that a graph has no negative cost cycles if and only if it has no negative cost circuits.*

Instead of solving the shortest path problem in G , we solve a different problem. Recall the definitions of paths and 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 $v \in V$.

The following lemma shows the connection to shortest paths and negative cost cycles.

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. \square

Remark: *The above lemma is false when there are negative cost cycles. Find such examples.*

¹This is one example of a problem where the undirected problem is significantly more difficult than the directed version. The reason is a cycle in an undirected graph has to be of length 3 or more, but in a directed graph a pair of antiparallel edges is also a cycle.

Lemma 1 allows us to focus on the Length Bounded Shortest Walk problem. We solve this problem with any costs using the technique that you all have learned and reverse : dynamic programming.

Now that I whispered dynamic programming in your ears, let us consider the minimum cost walk $w = (s, e_0, v_1, e_1, \dots, v_{\ell-1}, e_{\ell-1}, v)$ from s to v of length $\ell \leq k$. Can we break this solution into pieces?

Well, what can we say about the walk $w' = (s, e_0, \dots, v_{\ell-1})$? 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.

- *Definition.* We define $\text{dist}[v, i]$ for all $v \in V$ and $0 \leq i \leq k$ as the cost of the minimum cost walk from s to v of length $\leq i$.
- *Recurrence.* The recurrence is this

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

- *Base Case.* $\text{dist}[s, 0] = 0$; $\text{dist}[v, 0] = \infty$ for all $v \neq s$.
- *Proof.* Let's elaborate on the English explanation above. Let $\text{Cand}(v, i)$ denote the set of walks from s to v of length at most i . Thus,

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

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

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


(\geq) On the other hand, fix $w \in \text{Cand}(v, i)$ of length $\leq i$ and cost $\text{dist}[v, i]$. If the walk is not of length i , then $w \in \text{Cand}(v, i - 1)$ and thus $\text{dist}[v, i - 1] \leq \text{dist}[v, i]$. Otherwise, the walk is of length $i \geq 1$, 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 $\text{Cand}(u, i - 1)$. Since $c(w') = c(w) - c(u, v)$, we get $\text{dist}[u, i - 1] \leq \text{dist}[v, i] - c(u, v)$.

- *Pseudocode.*

```

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

```

- Running Time and Space. The space required is $\Theta(nk)$. The running time is $\Theta(km)$. 

Exercise: Try out the above DP on an example graph.

Lemma 1 states that if we are promised that G has no negative cost cycle, then $\text{LBSW}(G, s, n-1)$ will indeed return $\text{dist}[v, n-1]$ 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 $\text{dist}[v, n-1] = \text{dist}[v, n]$ 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 is not true for all v .

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 $\text{dist}[v, n] < \text{dist}[v, n-1]$.

Proof. Let $C = (x_1, \dots, x_k, x_1)$ 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

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

Adding these up, we get

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

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

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

```

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

```

Theorem 1. If there is a negative cost cycle reachable from s , BELLMAN-FORD returns one, or it returns a shortest path tree T to all vertices reachable from s .



Exercise: Complete the proof of the above theorem.

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 can be done "edge-by-edge" instead of "vertex-by-vertex". That is, we could replace the for-loop (Line 7) as follows:

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

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

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 $\text{dist}[v, i]$. Instead, we can just store $\text{dist}[v]$ for each vertex. We still need to run the loop for $i = 1$ to $n - 1$, and then check if running one extra loop leads to a drop in $\text{dist}[v]$ for

any vertex. If it doesn't, then there is no negative cost cycle and we have the answer is $\text{dist}[v]$'s; if it does, well then there is a negative cost cycle.

With these two observations, we get the following algorithm (which is almost identical to the ones given in the textbooks). However, the fact that the algorithm below gives the correct behavior as expected needs a proof which can be teased out of the proof we gave above for LBSW and BELLMAN-FORD.

```
1: procedure BELLMAN-FORD-TXTBK( $G, s$ ):
2:    $\text{dist}[s] = 0$  for all  $j$ ;  $\text{dist}[v] = \infty$  for all  $v \neq s$ .  $\triangleright$  Base Case
3:   for  $i = 1$  to  $n - 1$  do:
4:     for  $(v, u) \in E$  do:
5:        $\text{dist}[v] = \min(\text{dist}[u] + c(u, v), \text{dist}[v])$ .
6:        $\triangleright$  We abuse notation and say  $\infty + c = \infty$  for any finite  $c$ .
7:       If  $\text{dist}[v]$  modified, set  $\text{parent}[v] \leftarrow u$ .
8:   for  $(u, v) \in E$  do:
9:     if  $\text{dist}[v] > \text{dist}[u] + c(u, v)$  then:
10:      return NEGATIVE CYCLE
```



Exercise: Prove that BELLMAN-FORD-TXTBK is correct.