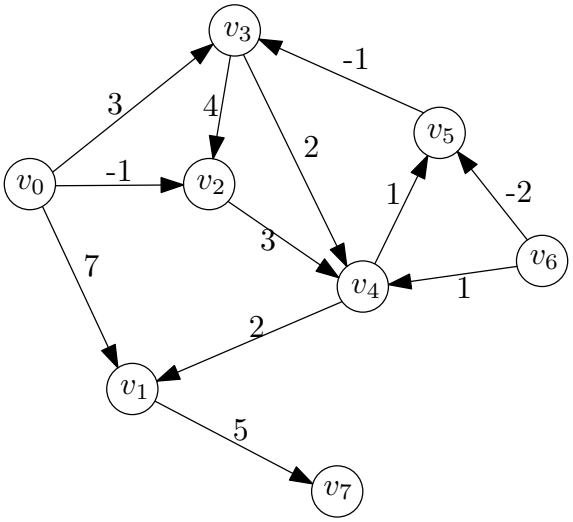
École Polytechnique Fédérale de Lausanne
Prof. Marcus May 19, 2022

Math 261 – Discrete Optimization (Spring 2022)

Assignment 11

Problem 1

In the following network, compute the minimum cost path from v_0 to all other vertices using the Bellman-Ford algorithm. Note the vector of shortest lengths for each iteration t.



Solution:

This is the `ONE-TO-ALL` problem, which we can turn into `ALL-TO-ONE` by reversing the edges. We can then use Bellman-Ford, where at each time t we write down the distance labels as a $[B_0(t), B_1(t), \ldots, B_7(t)]$.

time	Minimum Cost Walk vector							
t	$B_0(t)$	$B_1(t)$	$B_2(t)$	$B_3(t)$	$B_4(t)$	$B_5(t)$	$B_6(t)$	$B_7(t)$
0	0	∞	∞	∞	∞	∞	∞	∞
1	0	7	-1	3	∞	∞	∞	∞
2	0	7	-1	3	2	∞	∞	12
3	0	4	-1	3	2	3	∞	12
4	0	4	-1	2	2	3	∞	9
5	0	4	-1	2	2	3	∞	9
6	0	4	-1	2	2	3	∞	9

Problem 2

Let D be a network with no negative cost cycles and let $\mathbf{B} \in \mathbb{R}^V$ be a vector with the values of the minimum cost walks from v_i to v_n. Show how you can use $\mathbf{B}$ to construct the actual minimum costs walks.

Solution:

Since we know that there exists a solution that is an inrooted tree, the goal will be to simply find that tree. One property of a tree is the fact that it is connected — hence there must be some v_i for which the walk (v_i, v_n) is a minimal cost walk. Such a walk will satisfy $c(v_i, v_n) = B_i$ and so we can simply look for any vertex that satisfies this equality and draw the path (of length 1). Now we can proceed inductively. Assume we have built a tree T so far, but still have some vertices not yet put in the tree. It should be clear that there must be some vertex v_j which can reach T' in one step — that is, there exists a vertex v_j that satisfies

$$B_j = B_k + c(v_j, v_k) \tag{1}$$

for some vertex $v_k \in T$. Again, we simply look for some vertex that satisfies this equality and add it to the tree. In the end, we have an inrooted tree, and so for each v_i we can simply list the path from v_i to v_n in that tree, and this will be a valid list of paths.

Problem 3

Given a directed graph $D = (V, W)$, the Bellman-Ford algorithm defines a collection of vectors $\mathbf{B}(t)$ for $t = 0, \ldots, n = |V|$ using

$$B_i(t) = \{\text{the smallest cost walk that one can have from } v_i \text{ to } v_n \text{ using at most } t \text{ edges }\}$$

with $B_i(t) = \infty$ if no walk of length at most t exists. Assume all edge costs are finite.

1. Show that $B_i(n-1) < \infty$ if and only if there exists a directed walk from v_i to v_n in D.

   Solution:

   Certainly if there does not exist a directed walk from v_i to v_n is D, then $B_i(t) = \infty$ for all t. To show the other direction, assume there exists a walk w from v_i to v_n — we need to show that this means there exists a walk from v_i to v_n using only $n-1$ edges. Without loss of generality, we can assume w is the walk with the fewest possible edges, and then we will assume (for contradiction) that w contains more then $n-1$ edges. Pretend you are performing w and record the vertices that you see as you do it. Since w has length more than $n-1$, you will see more than n vertices. But there are only n vertices in D, so you must see some vertex twice. Hence your vertex list must look like either

   $$v_i \to \cdots \to v_k \to \cdots \to v_n$$

   where $k \in \{i, n\}$ or

   $$v_i \to \cdots \to v_k \to \cdots \to v_k \to \cdots \to v_n$$

   where $v_k \notin \{i, n\}$. In the first case it should be clear that there is a contradiction because one gets a shorter walk than w by either starting at v_k (if $k = i$) or ending at v_k (if $k = n$). In the second case, the same is true — we simply remove the middle part (between the vertices v_k) to get

   $$v_i \to \cdots \to v_k \to \cdots \to v_n$$

   and this will again be a shorter path (a contradiction).

2. Assume that $B_i(n-1) < \infty$ for all i. Show that $\mathbf{B}(n) = \mathbf{B}(n-1)$ if and only if there is no negative cost cycle.

Solution:

$\Rightarrow$: Assume $B_i(n-1) < \infty$ but there exists a walk

$$v_i \to \cdots \to \underbrace{v_k \to \cdots \to v_k}_{C} \to \cdots \to v_n$$

where C is a negative cost cycle. We need to show that $\mathbf{B}(n) \neq \mathbf{B}(n-1)$ — so assume (for contradiction) that they are equal. Let $v_{j_0}, \ldots, v_{j_\ell}$ be the list of vertices in C (where $j_0 = j_\ell = k$). The key idea is to find a contradiction using these vertices, where the contradiction will need to come from the fact that C is a negative length cycle. So let us calculate the differences across each edge in C

$$X = \sum_{s=1}^{\ell} B_{j_s}(n) - B_{j_{s-1}}(n-1).$$

On one hand, we have the inequality

$$B_{j_s}(n) \leq B_{j_{s-1}}(n-1) + c(j_{s-1}, j_s)$$

which holds by the minimality of $\mathbf{B}(n)$. Hence

$$X \leq c(j_{s-1}, j_s) = c(C) < 0$$

where $c(C)$ is the cost of the cycle (which is less than 0 by hypothesis). On the other hand, we have assumed $\mathbf{B}(n) = \mathbf{B}(n-1)$, so

$$X = \sum_{s=1}^{\ell} B_{j_s}(n) - B_{j_{s-1}}(n) = 0$$

since we are summing over the same vertices. This gives us the contradiction. $\Leftarrow$: Assume that there exists an i for which $B_i(n) \neq B_i(n-1)$. Since it is impossible for $B_i(n) > B_i(n-1)$, it means we must have

$$B_i(n) < B_i(n-1)$$

In other words, there is a walk w from v_i to v_n which touches exactly n edges (and so $n+1$ vertices) that has total cost $B_i(n)$. As we saw in part (a), any such walk must have a cycle C, and removing that cycle gives us another walk w' which is strictly shorter — say it has length k. Again, it is impossible for $B_i(k) \geq B_i(n-1)$ so we must have the strict inequality

$$c(w') \geq B_i(k) > B_i(n) = c(w).$$

Thus the edges we removed from w to get w' must have lowered the cost. Since these edges form a cycle, we have thus found a negative cost cycle.

Problem 4

Due to the decentralized nature of the global currency market, it might be the case that an individual or an organized group makes a large profit without risk. Arbitrage is a phenomenon that refers to cases when it is possible to convert one unit of a currency into more than one unit of the same currency by using discrepancy in exchange rates. For example, consider the case that 1 CHF buys 60 RUB, 1 RUB buys 0.019 USD and 1 USD buys 0.93 CHF. This means that a trader can transform 1 CHF into $60 \cdot 0.019 \cdot 0.93 = 1.0602$ CHF gaining a profit of 6.02%.

1. Given a list of currencies $r_1, \ldots, r_n$ and a matrix $E \in \mathbb{R}_{>0}^{n \times n}$ where $E_{i,j}$ denotes the amount of currency r_j that one can buy for 1 unit of r_i (the exchange rate between currencies r_i and r_j), design an algorithm to test if there is a possibility of arbitrage.

   Solution:

   Observe that in the case of arbitrage one makes a circular sequence of trades $t \in T$. The profit has been obtained in the case when the product of the corresponding exchange rates

   $$\prod_{t \in T} E_{i(t),j(t)} > 1 \iff \prod_{t \in T} \frac{1}{E_{i(t),j(t)}} < 1 \iff \sum_{t \in T} \log\left(\frac{1}{E_{i(t),j(t)}}\right) < 0,$$

   where $i(t)$ indicates the index of the currency sold in transaction t and $j(t)$ is the index of the currency bought. This suggests the following construction: Let $G = (V, E)$:

   Let D be a complete directed graph (edges going both ways between all pairs of vertices) with the set of nodes $V = r_1, \ldots, r_n$. For each arc $(r_i, r_j) \in E$ set the weight $w_{i,j} = \log\left(\frac{1}{E_{i,j}}\right)$. An arbitrage opportunity is (by definition) a negative cost cycle in this graph.

   To determine whether such an opportunity exists, pick a "destination" vertex v_n arbitrarily — since the graph is complete, any negative cost cycle will be able to reach v_n — and run Bellman-Ford. As we showed in Problem 3, we can simply check whether $\mathbf{B}(n) = \mathbf{B}(n-1)$ — if it does, then there is no negative cost cycle. If it doesn't there is one (and therefore an arbitrage opportunity).

   Note, that our solution uses the fact that we started with a complete graph, since (in general) it may be the case that a negative cost cycle exists but (depending on the choice of v_n) $B_i(n) = \infty$ for all vertices in the cycle.

2. Show how you can use your algorithm to find the arbitrage (not just show it exists).

   Hint: First find the length of the shortest walk (in terms of number of edges) that ends at v_n and contains a negative cost cycle.

   Solution:

   First some notation: we will say a walk is "bad" if (a) it ends at v_n and (b) it contains a negative cost cycle.

   Following the hint, we first look for the length of the shortest bad walk (in terms of number of edges). For this, we can use the algorithm from Problem 2. For a given t, let D_t be the induced subgraph of D on the vertices $\{v_i : B_i(t) < \infty\}$. Now consider running the algorithm on D_t using the distances defined in $\mathbf{B}(t)$. Certainly if there are no bad walks of length t in D, then the algorithm will finish with an in-rooted tree of depth at most t. However, if there is a bad walk of length t, then there will be a point at which the algorithm fails — that is, we will be building a tree T and we will find that there are no vertex pairs (v_i, v_k) where $v_i \in T$ and $v_k \notin T$ for which (1) holds (with equality). Hence

we simply need to find the smallest t for which the algorithm fails, and this can be done very efficiently using a binary search.

Now assume we know that the length of shortest (with respect to number of edges) bad walk in D is t^*. The goal is to find a bad walk of length t^*. To do so we first run the algorithm from Problem 2 on $t = t^* - 1$ (where it succeeds because there are no bad walks this length) and returns an inrooted tree T.

Now consider what happens when we run the algorithm on $t = t^*$. Of course it will fail, because D contains a bad walk of length t^*. Furthermore, it will contain a bad walk which contains each of the v_k that are left over when the algorithm fails (the ones that have not been put in a tree yet and fail (1) with respect to the tree so far). Pick one of those vertices (say v_i) and let w be a walk bad walk containing v_i.

Since w contains a cycle, it must have a repeated vertex. It should be clear that v_i must be that repeated vertex, since otherwise we could remove v_i and have a shorter walk containing a negative cost cycle. Hence w must look like

$$v_i \to \underbrace{v_j \cdots \to v_i \to \cdots \to v_n}_{w'}.$$

where $v_j \neq v_i$ and w' is the walk of length $t^* - 1$ that happens after the first step from v_i. It should be clear that $B_j(t^* - 1) < \infty$ and so B_j appears in the tree T. Furthermore, it should be clear that v_j is a leaf node in T (otherwise w could be made shorter). But there are not that many ways to build a cycle that goes through a leaf node of T — in particular, if we simply test all edges (v_i, v_k) where $v_i \in T$ and v_k is a leaf node in T, then each one will provide a directed cycle of length t^* and we simply need to find one that has negative cost.