Homework: The Shortest Path Problem

Solutions

1. **(a)** The Generic Shortest Path Algorithm (GSPA) maintains a vector $d = (d_1, d_2, \ldots, d_N)$ of node labels and a set $\mathcal{V}$ of nodes not processed yet. Assuming node 1 is the source, the algorithm initializes as: $\mathcal{V} = \{1\}$, $d = (0, \infty, \ldots, \infty)$. At each iteration k, as long as $\mathcal{V}$ remains nonempty, select *any* node $i \in \mathcal{V}$. For every out-neighbor j of node i (i.e., (i, j) is an outgoing arc from i), if

$$d_j > d_i + a_{ij},$$

then set $d_j \leftarrow d_i + a_{ij}$ and add j to $\mathcal{V}$, if not already there. The algorithm terminates if no path from i contains a cycle with negative cost. After the algorithm has terminated, d_i will be the shortest distance from node 1 to node i.

(b) The following tables illustrate how the algorithm evolves. The red color indicates the chosen node and the blue its out-neighbors.

k	d	$\mathcal{V}$	$\mathcal{V} \to i$
1	$(0, \infty, \infty, \infty, \infty)$	$\{1\}$	1
2	$(0, 2, 2, \infty, \infty)$	$\{2, 3\}$	2
3	$(0, 2, 2, 6, 5)$	$\{3, 4, 5\}$	3
4	$(0, 2, 2, 6, 3)$	$\{4, 5\}$	4
5	$(0, 2, 2, 6, 3)$	$\{5\}$	5
6	$(0, 2, 2, 6, 3)$	$\emptyset$	

(c) The main difference between label setting and label correcting methods is how they select the node to be removed from the candidate node list, $\mathcal{V}$. In label *setting* methods the node to be removed from the candidate list is selected to have the minimum label, whereas in label *correcting* methods, nodes to be removed need not have such a property. Hence, in label setting methods nodes enter the candidate list at most once, and when they are removed, their labels remain fixed. On the other hand, in label correcting methods, a node may enter the candidate list more than once and they are more general since they can handle negative arc lengths.

2. **(a)** The Dijkstra's Algorithm is a type of label setting algorithm. The GSPA is modified in the sense that nodes from the candidate list, $\mathcal{V}$, are removed in the ascending order of their label values, i.e., the node i having the minimum label value in $\mathcal{V}$ is removed first:

$$i = \arg\min_{i \in \mathcal{V}} d_i.$$

(b) Since nodes enter the candidate list, $\mathcal{V}$, at most once and their label values remain fixed once they are removed from the list, the method is a label setting method.

(c) The following table shows Dijkstra's algorithm at each iteration k:

k	d	$\mathcal{V}$	$\mathcal{V} \to i$
1	$(0,\infty,\infty,\infty,\infty,\infty)$	$\{1\}$	1
2	$(0,2,8,\infty,\infty,\infty)$	$\{2,3\}$	2
3	$(0,2,7,5,\infty,\infty)$	$\{3,4\}$	4
4	$(0,2,6,5,12,11)$	$\{3,5,6\}$	3
5	$(0,2,6,5,6,11)$	$\{5,6\}$	5
6	$(0,2,6,5,6,8)$	$\{6\}$	6
7	$(0,2,6,5,6,8)$	$\emptyset$	

3. **(a)** The Bellman-Ford algorithm is a type of label correcting algorithm, where nodes from the candidate list, $\mathcal{V}$, are removed in the chronological order of their entrance, i.e., the node which enters the queue first is removed first (first-in-first-out queuing).

(b) Since nodes may enter the candidate list, $\mathcal{V}$, more than once, their label values may be altered when they re-enter the list, i.e., their final values are not fixed after their first removal from the candidate list.

(c) The following tables illustrate how the Bellman-Ford algorithm evolves. The red color indicates the chosen node and the blue its out-neighbors that are updated.

k	d	$\mathcal{V}$	$\mathcal{V} \to i$
1	$(0,\infty,\infty,\infty,\infty)$	$\{1\}$	1
2	$(0,4,2,\infty,\infty)$	$\{2,3\}$	2
3	$(0,4,2,7,\infty)$	$\{3,4\}$	3
4	$(0,3,2,7,3)$	$\{4,2,5\}$	4
5	$(0,3,2,7,3)$	$\{2,5\}$	2
6	$(0,3,2,6,3)$	$\{5,4\}$	5
7	$(0,3,2,6,3)$	$\{4\}$	4
8	$(0,3,2,6,3)$	$\emptyset$	

4. **(a)** The graph has negative arcs but none of them can form a negative cycle. Therefore, the algorithm will converge. The following table shows the Bellman-Ford algorithm at each iteration k:

k	d	$\mathcal{V}$	$\mathcal{V} \to i$
1	$(0,\infty,\infty,\infty,\infty,\infty,\infty)$	$\{1\}$	1
2	$(0,3,10,\infty,\infty,\infty,\infty)$	$\{2,3\}$	2
3	$(0,3,7,10,4,\infty,\infty)$	$\{3,4,5\}$	3
4	$(0,3,7,5,4,14,\infty)$	$\{4,5,6\}$	4
5	$(0,3,7,5,4,-1,\infty)$	$\{5,6\}$	5
6	$(0,3,7,5,4,-1,19)$	$\{6,7\}$	6
7	$(0,3,7,5,4,-1,5)$	$\{7\}$	7
8	$(0,3,7,5,4,-1,5)$	$\emptyset$	

The Bellman-Ford algorithm uses:

$$v_i(t+1) = \min_{k|(i,k)\in\mathcal{A}} \{c_{ik} + v_k(t)\}, \quad i = 1,\ldots,n-1.$$

A table of v_i therefore looks like:

t/i	1	2	3	4	5	6
1					15	6
2		16	13	0	15	6
3	19	7	-2	0	15	6
4	8	2	-2	0	15	6
5	5	2	-2	0	15	6

The shortest path from 1 to 7 has cost 5. To find the path we should backtrack from 7. This gives the shortest path $1 \to 2 \to 3 \to 4 \to 6 \to 7$.

(b) We reformulate the problem with

$$\overline{c}_{ij} = c_{ij} + v_i - v_j,$$

which ensures that all costs are nonnegative (see pp. 64 in Network Optimization book). The length of any path from a node i to a node j will be increased by $v_j - v_i$, where the shortest path will be unaffected!

