1 Shortest paths (continued)

1.1 Bellman-Ford algorithm for single source shortest paths

This algorithm can be used with networks that contain negative weighted edges. Therefore it detects the existence of negative cycles. In words, the algorithm consists of \( n \) pulses. (pulse = update of labels along all \( m \) arcs). (Henceforth we will assume that we want to find the shortest paths from node 1 to all other nodes.)

**Input:** A (general) graph \( G = (V, A) \), and two nodes \( s \) and \( t \).

**Output:** The shortest path from \( s \) to \( t \).

**Pseudocode**

Initialize:

\[ d(1) := 0, d(i) := \infty, i = 2, \ldots, n \]

For \( i = 1, \ldots, n \) do

Pulse:

\[ \forall (i, j) \in A, d(j) := \min\{d(j), d(i) + C_{ij}\} \]

To prove the algorithm’s correctness we will use the following lemma.

**Lemma 1.** If there are no negative cost cycles in the network \( G = (N, A) \), then there exists a shortest path from \( s \) to any node \( i \) which uses at most \( n - 1 \) arcs.

**Proof.** Suppose that \( G \) contains no negative cycles. Observe that at most \( n - 1 \) arcs are required to construct a path from \( s \) to any node \( i \). Now, consider a path, \( P \), from \( s \) to \( i \) which traverses a cycle.

\[ P = s \to i_1 \to i_2 \to \ldots \to (i_j \to i_k \to \ldots \to i_j) \to i_\ell \to \ldots \to i. \]

Since \( G \) has no negative length cycles, the length of \( P \) is no less than the length of \( \bar{P} \) where

\[ \bar{P} = s \to i_1 \to i_2 \to \ldots \to i_j \to i_k \to \ldots \to i_\ell \to \ldots \to i. \]

Thus, we can remove all cycles from \( P \) and still retain a shortest path from \( s \) to \( i \). Since the final path is acyclic, it must have no more than \( n - 1 \) arcs.

**Theorem 2** (Invariant of the algorithm). After pulse \( k \), all shortest paths from \( s \) of length \( k \) (in terms of number of arcs) or less have been identified.

**Proof.** Notation: \( d_k(j) \) is the label \( d(j) \) at the end of pulse \( k \).

By induction. For \( k = 1 \), obvious. Assume true for \( k \), and prove for \( k + 1 \).

Let \((s, i_1, i_2, ..., i_\ell, j)\) be a shortest path to \( j \) using \( k + 1 \) or fewer arcs.

It follows that \((s, i_1, i_2, ..., i_\ell)\) is a shortest path from \( s \) to \( i_\ell \) using \( k \) or fewer arcs.

After pulse \( k + 1 \), \( d_{k+1}(j) \leq d_k(i_\ell) + c_{i_\ell,j} \).

But \( d_k(i_\ell) + c_{i_\ell,j} \) is the length of a shortest path from \( s \) to \( j \) using \( k + 1 \) or fewer arcs, so we know that \( d_{k+1}(j) = d_k(i_\ell) + c_{i_\ell,j} \).

Thus, the \( d() \) labels are correct after pulse \( k + 1 \).
Note that the shortest paths identified in the theorem are not necessarily simple. They may use negative cost cycles.

**Theorem 3.** If there exists a negative cost cycle, then there exists a node \( j \) such that \( d_n(j) < d_{n-1}(j) \) where \( d_k(i) \) is the label at iteration \( k \).

**Proof.** Suppose not.

Let \((i_1 - i_2 - \ldots - i_k - i_1)\) be a negative cost cycle.

We have \( d_n(i_j) = d_{n-1}(i_j) \), since the labels never increase.

From the optimality conditions, \( d_n(i_{r+1}) \leq d_{n-1}(i_r) + C_{r,r+1} \) where \( r = 1, \ldots, k-1 \), and \( d_n(i_1) \leq d_{n-1}(i_k) + C_{i_k,i_1} \).

These inequalities imply \( \sum_{j=1}^{k} d_n(i_j) \leq \sum_{j=1}^{k} d_{n-1}(i_j) + \sum_{j=1}^{k} C_{i_{j-1},i_j} \).

But if \( d_n(i_j) = d_{n-1}(i_j) \), the above inequality implies \( 0 \leq \sum_{j=1}^{k} C_{i_{j-1},i_j} \).

\( \sum_{j=1}^{k} C_{i_{j-1},i_j} \) is the length of the cycle, so we get an inequality which says that 0 is less than a negative number which is a contradiction. \( \square \)

**Corollary 4.** The Bellman-Ford algorithm identifies a negative cost cycle if one exists.

By running \( n \) pulses of the Bellman-Ford algorithm, we either detect a negative cost cycle (if one exists) or find a shortest path from \( s \) to all other nodes \( i \). Therefore, finding a negative cost cycle can be done in polynomial time. Interestingly, finding the most negative cost cycle is NP-hard (reduction from Hamiltonian Cycle problem).

**Remark** Minimizing mean cost cycle is also polynomially solvable.

**Complexity Analysis**

Complexity \( = O(mn) \) since each pulse is \( O(m) \), and we have \( n \) pulses.

Note that it suffices to apply the pulse operations only for arcs \((i, j)\) where the label of \( i \) changed in the previous pulse. Thus, if no label changed in the previous iteration, then we are done.

### 1.2 Floyd-Warshall Algorithm for the all pairs shortest paths problem

The Floyd-Warshall algorithm is designed to solve all pairs shortest paths problems for graphs with negative cost edges. As all algorithms for shortest paths on general graphs, this algorithm will detect negative cost cycles.

In words, the algorithm maintains a matrix \( (d_{ij}) \) such that at iteration \( k \), \( d_{ij} \) is the shortest path from \( i \) to \( j \) using nodes \( 1, 2, \ldots, k \) as intermediate nodes. After the algorithm terminates, assuming that no negative cost cycle is present, the shortest path from nodes \( i \) to \( j \) is \( d_{ij} \).

**Pseudocode**

**Input:** An \( n \times n \) matrix \([c_{ij}]\)

**Output:** An \( n \times n \) matrix \([d_{ij}]\) is the shortest distance from \( i \) to \( j \) under \([c_{ij}]\)

An \( n \times n \) matrix \([e_{ij}]\) is a node in the path from \( i \) to \( j \).

**begin**

for all \( i \neq j \) do \( d_{ij} := c_{ij} \);
for \( i = 1, \ldots, n \) do \( d_{ii} := \infty \);
for \( j = 1, \ldots, n \) do

for \( i = 1, \ldots, n, i \neq j, \) do

for \( k = 1, \ldots, n, k \neq j, \) do

end
\[
d_{ik} := \min \{d_{ik}, d_{ij} + d_{jk}\}
\]
\[
e_{ik} := \begin{cases} 
  j & \text{if } d_{ik} > d_{ij} + d_{jk} \\
  e_{ik} & \text{otherwise}
\end{cases}
\]

The main operation in the algorithm is: \textit{Pulse} \( j \): \( d_{ik} = \min(d_{ik}, d_{ij} + d_{jk}) \). This operation is sometimes referred to as a \textit{triangle operation} (see Figure 1).

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{triangle_operation.png}
\caption{Triangle operation: Is the (dotted) path using node \( j \) as intermediate node shorter than the (solid) path without using node \( j \)?}
\end{figure}

\textbf{Invariant property of Floyd-Warshall Algorithm:}
After iteration \( k \), \( d_{ij} \) is the shortest path distance from \( i \) to \( j \) involving a subset of nodes in \( \{1, 2, \ldots, k\} \) as intermediate nodes.

\textbf{Complexity Analysis}
At each iteration we consider another node as intermediate node \( \rightarrow O(n) \) iterations. In each iteration we compare \( n^2 \) triangles for \( n^2 \) pairs \( \rightarrow O(n^2) \). Therefore complexity of the Floyd-Warshall algorithm = \( O(n^3) \).

See the handout titled \textit{All pairs shortest paths}.

At the initial stage, \( d_{ij} = c_{ij} \) if there exist an arc between node \( i \) and \( j \)
\[
d_{ij} = \infty \quad \text{otherwise}
\]
\[
e_{ij} = 0
\]
First iteration :
\[
d_{24} \leftarrow \min(d_{24}, d_{21} + d_{14}) = 3
\]
\[
e_{24} = 1
\]
Update distance label
Second iteration :
\[
d_{41} \leftarrow \min(d_{41}, d_{42} + d_{21}) = -2
\]
\[
d_{43} \leftarrow \min(d_{43}, d_{42} + d_{23}) = -3
\]
\[
d_{44} \leftarrow \min(d_{44}, d_{42} + d_{24}) = -1
\]
\[
e_{41} = e_{43} = e_{44} = 2
\]
No other label changed

Note that we found a negative cost cycle since the diagonal element of matrix \( D, d_{44} = -1 \leq 0 \). Negative \( d_{ii} \) means there exists a negative cost cycle since it simply says that there exists a path of negative length from node \( i \) to itself.