Data Structures and Algorithms
CMPSC 465

Lecture 38

- Bellman-Ford Shortest Paths Algorithm
- Detecting Negative Cost Cycles

Adam Smith
This graph causes Dijkstra’s algorithm to fail
(incorrectly estimate the distance and shortest path to A)
Shortest Paths: Negative Cost Cycles

Negative cost cycle.

![Graph](image)

Observation. If some path from \( s \) to \( t \) contains a negative cost cycle, there does not exist a shortest \( s-t \) path; otherwise, there exists one that is simple.

![Graph](image)

\( c(W) < 0 \)
Shortest Paths: Dynamic Programming

Def. \( \text{OPT}(i, v) = \text{length of shortest } v-t \text{ path } P \text{ using at most } i \text{ edges} \)  
\( (\infty \text{ if no such path exists}) \)

- **Case 1:** \( P \) uses at most \( i-1 \) edges.  
  - \( \text{OPT}(i, v) = \text{OPT}(i-1, v) \)

- **Case 2:** \( P \) uses exactly \( i \) edges.
  - if \((v, w)\) is first edge, then \( \text{OPT} \) uses \((v, w)\), and then selects best \( w-t \) path using at most \( i-1 \) edges

\[
\text{OPT}(i, v) = \begin{cases} 
0 \text{ or } \infty & \text{if } i = 0 \\
\min \left\{ \text{OPT}(i-1, v), \min_{(v, w) \in E} \left\{ \text{OPT}(i-1, w) + c_{vw} \right\} \right\} & \text{otherwise}
\end{cases}
\]

Remark. By previous observation, if no negative cycles, then \( \text{OPT}(n-1, v) = \text{length of shortest } v-t \text{ path} \).
Shortest Paths: Implementation

Shortest-Path(G, t) {
    foreach node v ∈ V
        M[0, v] ← ∞
    M[0, t] ← 0

    for i = 1 to n-1
        foreach node v ∈ V
            M[i, v] ← M[i-1, v]
        foreach edge (v, w) ∈ E
            M[i, v] ← min { M[i, v], M[i-1, w] + c_{vw} }
}

Analysis. Θ(mn) time, Θ(n^2) space.

Finding the shortest paths. Maintain a "successor" for each table entry.
Example of Bellman-Ford

The demonstration is for a slightly different version of the algorithm (see CLRS) that computes distances from the source node rather than distances to the destination node.
Example of Bellman-Ford

Initialization.
Example of Bellman-Ford

Order of edge relaxation.

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Example of Bellman-Ford
Example of Bellman-Ford
Example of Bellman-Ford
Example of Bellman-Ford
Example of Bellman-Ford

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Example of Bellman-Ford
Example of Bellman-Ford

\begin{figure}
\centering
\includegraphics[width=\textwidth]{bellman_ford_example}
\end{figure}
Example of Bellman-Ford
Example of Bellman-Ford

End of pass 1.
Example of Bellman-Ford

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Example of Bellman-Ford
Example of Bellman-Ford
Example of Bellman-Ford

A

B

C

D

E

0

-1

4

5

3

1

2

4

7

3

1

6

2

8

-3

1

2

-3

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Example of Bellman-Ford
Example of Bellman-Ford

Graph with nodes A, B, C, D, E and edges with weights as shown in the diagram.
Example of Bellman-Ford
Example of Bellman-Ford

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova, K. Wayne
Example of Bellman-Ford

End of pass 2 (and 3 and 4).
Shortest Paths: Improvements

Maintain only one array $M[v] = \text{length of shortest } v\text{-}t \text{ path found so far.}$ No need to check edges of the form $(v, w)$ unless $M[w]$ changed in previous iteration.

Theorem. Throughout the algorithm, $M[v]$ is length of some $v\text{-}t \text{ path, and after } i \text{ rounds of updates, the value } M[v] \text{ is no larger than the length of shortest } v\text{-}t \text{ path using } \leq i \text{ edges.}$

Overall impact.
- Memory: $O(m + n)$.
- Running time: $O(mn)$ worst case, but substantially faster in practice.
Bellman-Ford-Shortest-Path(G, s, t) {
    foreach node v ∈ V {
        M[v] ← ∞
        successor[v] ← φ
    }
    M[t] = 0
    for i = 1 to n-1 {
        foreach node w ∈ V {
            if (M[w] has been updated in previous iteration) {
                foreach node v such that (v, w) ∈ E {
                    if (M[v] > M[w] + c_{vw}) {
                        M[v] ← M[w] + c_{vw}
                        successor[v] ← w
                    }
                }
            }
            If no M[w] value changed in iteration i, stop.
        }
    }
}
Distance Vector Protocol

Communication network.
- Nodes $\approx$ routers.
- Edges $\approx$ direct communication links.
- Cost of edge $\approx$ delay on link. \(\leftarrow\) naturally nonnegative, but Bellman-Ford used anyway!

Dijkstra's algorithm. Requires global information of network.

Bellman-Ford. Uses only local knowledge of neighboring nodes.

Synchronization. We don't expect routers to run in lockstep. The order in which each `foreach` loop executes is not important. Moreover, algorithm still converges even if updates are asynchronous.
Distance vector protocol.

- Each router maintains a vector of shortest path lengths to every other node (distances) and the first hop on each path (directions).
- Algorithm: each router performs $n$ separate computations, one for each potential destination node.
- "Routing by rumor."

Ex. RIP, Xerox XNS RIP, Novell's IPX RIP, Cisco's IGRP, DEC's DNA Phase IV, AppleTalk's RTMP.

Caveat. Edge costs may change during algorithm (or fail completely).
Path Vector Protocols

Link state routing.
- Each router also stores the entire path.
- Based on Dijkstra's algorithm.
- Avoids "counting-to-infinity" problem and related difficulties.
- Requires significantly more storage.

Ex. Border Gateway Protocol (BGP), Open Shortest Path First (OSPF).
Detecting Negative Cycles: Application

Currency conversion. Given $n$ currencies and exchange rates between pairs of currencies, is there an arbitrage opportunity?

Remark. Fastest algorithm very valuable!
Detecting Negative Cycles

Bellman-Ford is guaranteed to work if there are no negative cost cycles.

How can we tell if neg. cost cycles exist?
- We could pick a destination vertex $t$ and check if the cost estimates in Bellman-Ford converge or not.
- What is wrong with this?
  - (What if a cycle isn’t on any path to $t$?)
Detecting Negative Cycles

Theorem. Can find negative cost cycle in $O(mn)$ time.

- Add new node $t$ and connect all nodes to $t$ with 0-cost edge.
- Check if $OPT(n, v) = OPT(n-1, v)$ for all nodes $v$.
  - if yes, then no negative cycles
  - if no, then extract cycle from shortest path from $v$ to $t$
Lemma. If \( OPT(n,v) = OPT(n-1,v) \) for all \( v \), then no negative cycles are connected to \( t \).

**Pf.** If \( OPT(n,v) = OPT(n-1,v) \) for all \( v \), then distance estimates won’t change again even with many executions of the for loop. In particular, \( OPT(2n,v) = OPT(n-1,v) \). So there are no negative cost cycles.

Lemma. If \( OPT(n,v) < OPT(n-1,v) \) for some node \( v \), then (any) shortest path from \( v \) to \( t \) contains a cycle \( W \). Moreover \( W \) has negative cost.

**Pf.** (by contradiction)
- Since \( OPT(n,v) < OPT(n-1,v) \), we know \( P \) has exactly \( n \) edges.
- By pigeonhole principle, \( P \) must contain a directed cycle \( W \).
- Deleting \( W \) yields a \( v\rightarrow t \) path with < \( n \) edges \( \Rightarrow \) \( W \) has negative cost.
Detecting Negative Cycles: Summary

Bellman-Ford. $O(mn)$ time, $O(m + n)$ space.
- Run Bellman-Ford for $n$ iterations (instead of $n-1$).
- Upon termination, Bellman-Ford successor variables trace a negative cycle if one exists.
- See p. 304 for improved version and early termination rule.