Shortest Paths with Dynamic Programming

Bellman-Ford Algorithm

Paul Beame
Shortest Path Problem

- Dijkstra’s Single Source Shortest Paths Algorithm
  - $O(m \log n)$ time, positive cost edges

- Bellman-Ford Algorithm
  - $O(mn)$ time for graphs with negative cost edges
Shortest paths with negative cost edges

- Dijsktra’s algorithm failed with negative-cost edges
  - What can we do in this case?
  - Negative-cost cycles could result in shortest paths with length $-\infty$
    - but these would be infinitely long...

- What if we just wanted shortest paths of exactly $i$ edges?
Shortest paths with negative cost edges (Bellman-Ford)

- We want to grow paths from $s$ to $t$ based on the # of edges in the path.
- Let $Cost(s, w, i) =$ cost of minimum-length path from $s$ to $w$ using exactly $i$ edges.
  - $Cost(s, w, 0) = \begin{cases} 0 & \text{if } w = s \\ \infty & \text{otherwise} \end{cases}$
  - $Cost(s, w, i) = \min_{(v, w) \in E}(Cost(s, v, i-1) + c_{vw})$
Bellman-Ford

- Observe that the recursion for \( \text{Cost}(s,w,i) \) doesn’t change \( s \)
  - Only store an entry for each \( w \) and \( i \)
    - \( \text{OPT}_i(w) \)

- \( \text{OPT}_0(w) = \begin{cases} 0 & \text{if } w = s \\ \infty & \text{otherwise} \end{cases} \)

- \( \text{OPT}_i(w) = \min_{(v,w) \in E} (\text{OPT}_{i-1}(v) + c_{vw}) \)
Shortest paths with negative cost edges (Bellman-Ford)

- Suppose no negative-cost cycles in $G$
  - Shortest path from $s$ to $t$ has at most $n-1$ edges
    - If not, there would be a repeated vertex which would create a cycle that could be removed since cycle can’t have –ve cost
Algorithm, Version 1

foreach w

\[ M[0, w] = \infty; \]

\[ M[0, s] = 0; \]

for i = 1 to n-1

foreach w

\[ M[i, w] = \min_v (M[i-1, v] + \text{cost}[v, w]); \]

What if we want to allow \textit{up to i} edges rather than require exactly \textit{i} edges?
Algorithm, Version 2

foreach \( w \)

\[
M[0, w] = \text{infinity};
\]

\[
M[0, s] = 0;
\]

for \( i = 1 \) to \( n-1 \)

foreach \( w \)

\[
M[i, w] = \min(M[i-1, w], \min_v(M[i-1,v] + \text{cost}[v,w]))
\]

Now \( M[i,w] \leq M[i-1,w] \leq \ldots \leq M[0,w] \).

If all we only care about is finding short paths we can use the shortest length we have found and forget # of hops.
Algorithm, Version 3

foreach w
    M[w] = infinity;
M[s] = 0;
for i = 1 to n-1
    foreach w
        M[w] = min(M[w], min_v(M[v] + cost[v,w]))
Key lemma – at the end of iteration i, for all w, $M[w] \leq M[i, w]$;

Reconstructing the path:
- Set $P[w] = v$, whenever $M[w]$ is updated from vertex $v$
Bellman-Ford
Bellman-Ford
Bellman-Ford
Bellman-Ford
Bellman-Ford
Bellman-Ford
Bellman-Ford
Other details

- Can run algorithm and stop early if $M$ doesn’t change in an iteration
  - Even better, one can update only neighbors $x$ of vertices $w$ whose $M$ value changed in an iteration
If the pointer graph has a cycle, then the graph has a negative cost cycle

- If \( P[w] = v \) then \( M[w] \geq M[v] + \text{cost}(v,w) \)
  - Equal when \( w \) is updated
  - \( M[v] \) could later be reduced after update

- Let \( v_1, v_2, \ldots v_k \) be a cycle in the pointer graph with \( (v_k, v_1) \) the last edge added
  - Just before the update
    - \( M[v_j] \geq M[v_{j+1}] + \text{cost}(v_{j+1}, v_j) \) for \( j < k \)
    - \( M[v_k] > M[v_1] + \text{cost}(v_1, v_k) \)
  - Adding everything up
    - \( 0 > \text{cost}(v_1,v_2) + \text{cost}(v_2,v_3) + \ldots + \text{cost}(v_k, v_1) \)
Finding negative cost cycles

What if you want to find negative cost cycles?
Foreign Exchange Arbitrage

<table>
<thead>
<tr>
<th></th>
<th>USD</th>
<th>EUR</th>
<th>CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td>USD</td>
<td>------</td>
<td>0.8</td>
<td>1.2</td>
</tr>
<tr>
<td>EUR</td>
<td>1.2</td>
<td>------</td>
<td>1.6</td>
</tr>
<tr>
<td>CAD</td>
<td>0.8</td>
<td>0.6</td>
<td>------</td>
</tr>
</tbody>
</table>

Diagram:
- USD to CAD: 1.2
- EUR to CAD: 1.6
- USD to EUR: 0.8
- CAD to EUR: 0.6
- CAD to USD: 0.8
Bellman-Ford with a DAG

Edges only go from lower to higher-numbered vertices
• Update distances in order of topological sort
• Only one pass through vertices required
• $O(n+m)$ time