4.4 Shortest Paths

- edge-weighted DAGs

Q. Suppose that an edge-weighted digraph has no directed cycles, so it's a DAG (directed acyclic graph). Is it easier to find shortest paths than in a general digraph?

A. Yes!

Acyclic shortest paths algorithm

- Consider vertices in topological order.
- Relax all edges pointing from that vertex.

Shortest paths in edge-weighted DAGs

Proposition: Topological sort algorithm computes SPT in any edge-weighted DAG in time proportional to $E + V$.

Compare to $E \log V$ for Dijkstra's algorithm. --
Where $E > V$ (most cases), $O(E \log V) > O(E + V) = O(E)$.
So this is faster. It takes advantage of the acyclic property.

Shortest paths in edge-weighted DAGs

```java
public class AcyclicSP {
    private DirectedEdge[] edgeTo;
    private double[] distTo;

    public AcyclicSP(EdgeWeightedDigraph G, int s) {
        edgeTo = new DirectedEdge[G.V()];
        distTo = new double[G.V()];

        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;

        distTo[s] = 0.0;
        Topological topological = new Topological(G);

        for (int v : topological.order())
            for (DirectedEdge e : G.adj(v))
                relax(e);
    }
}
```
Content-aware resizing

Seam carving. [Avidan and Shamir] Resize an image without distortion for display on cell phones and web browsers.

Critical path method

**CPM.** To solve a parallel job-scheduling problem, create edge-weighted DAG:
- Source and sink vertices.
  - Two vertices (begin and end) for each job.
- Three edges for each job:
  - begin to end (weighted by duration)
  - source to begin (0 weight)
  - end to sink (0 weight)
- One edge for each precedence constraint (0 weight).

Critical path method

**CPM.** Use longest path from the source to schedule each job.

Longest paths in edge-weighted DAGs: application

**Parallel job scheduling.** Given a set of jobs with durations and precedence constraints, schedule the jobs (by finding a start time for each) so as to achieve the minimum completion time, while respecting the constraints.

In the wild. Photoshop CS 5, Imagemagick, GIMP, ...
Shortest paths with negative weights: failed attempts

Dijkstra. Doesn’t work with negative edge weights.

Dijkstra selects vertex 0 immediately after 0. But shortest path from 0 to 6 is 0→1→2→3→6.

Re-weighting. Add a constant to every edge weight doesn’t work.

Adding 8 to each edge weight changes the shortest path from 0→1→2→3 to 0→3.

Conclusion. Need a different algorithm.

Negative cycles

Def. A negative cycle is a directed cycle whose sum of edge weights is negative.

Algorithm:

Bellman-Ford algorithm

Bellman-Ford algorithm

- Initial distTo[s] = 0 and distTo[v] = for all other vertices.
- Repeat V times:
  - Relax each edge.

Proposition. A SPT exists if no negative cycles.

Shortest paths summary

Nonnegative weights.
- Arises in many applications.
- Dijkstra’s algorithm is nearly linear-time.

Acyclic edge-weighted digraphs.
- Arise in some applications.
- Topological sort algorithm is linear time.
- Edge weights can be negative.

Negative weights and negative cycles.
- Arise in some applications.
- If no negative cycles, can find shortest paths via Dijkstra.
- If negative cycles, can find one via Bellman-Ford.

Remark 1. Directed cycles make the problem harder.
Remark 2. Negative weights make the problem harder.
Remark 3. Negative cycles makes the problem intractable.

Single source shortest-paths implementation: cost summary

<table>
<thead>
<tr>
<th>algorithm</th>
<th>initialization</th>
<th>typical case</th>
<th>worst case</th>
<th>extra space</th>
</tr>
</thead>
<tbody>
<tr>
<td>topological sort</td>
<td>no directed edges</td>
<td>$E = V$</td>
<td>$E = V$</td>
<td>$E$</td>
</tr>
<tr>
<td>Dijkstra (binary heap)</td>
<td>no negative weights</td>
<td>$E \log V$</td>
<td>$E \log V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Bellman-Ford</td>
<td>no negative weight</td>
<td>$E V$</td>
<td>$E V$</td>
<td>$V$</td>
</tr>
<tr>
<td>Bellman-Ford (queue-based)</td>
<td>no negative weight</td>
<td>$E V$</td>
<td>$E V$</td>
<td>$V$</td>
</tr>
</tbody>
</table>

Remark 1. Directed cycles make the problem harder.
Remark 2. Negative weights make the problem harder.
Remark 3. Negative cycles makes the problem intractable.

Bellman-Ford algorithm

Bellman-Ford algorithm

- Initial distTo[s] = 0 and distTo[v] = for all other vertices.
- Repeat V times:
  - Relax each edge.

for (int i = 0; i < G.V(); i++)
for (int v = 0; v < G.V(); v++)
for (DirectedEdge e : G.adj(v))
relax(e);

Proposition. A SPT exists if no negative cycles.