IE170: Algorithms in Systems Engineering: Lecture 20

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

March 19, 2007

Taking Stock

Last Time

- Minimum Spanning Trees
- Strongly Connected Components

This Time

- Shortest Paths

Shortest Paths—Definitions

- For the next few lectures, we will have a directed graph $G = (V, E)$, and a weight function $w : \mathbb{E} \rightarrow \mathbb{R}^{|E|}$.
- The weight of a path $P = \{v_0, v_1, \ldots v_k\}$ is simply the weight of the edges taken on the sequence of nodes:

$$w(P) = \sum_{i=1}^{k} w_{v_{i-1},v_i}.$$

- We are interested in finding the shortest-path weights from u to v, which we will denote $\delta(u, v)$.
- We use the convention that $\delta(u, v) = \infty$ if there is no path from u to v in G

Example

- The example (hopefully) makes it clear that shortest paths are organized as a tree
- Many algorithms work like a generalization of BFS to weighted graphs.

Shortest Path Variants

- Single-Source: Find the shortest path from $s \in V$ to every vertex $v \in V$
- Single-Destination: Find the shortest path from every vertex $v \in V$ to a given destination vertex $t \in V$
- Single-Pair: Find the shortest path from given $s \in V$ to given $t \in V$. There is now way known that is better (in the worst case) that solving the single-source version.
- All-Pairs: Find the shortest path from every $u \in V$ to every vertex $v \in V$

Negative Weight Edges

- In Minimum Spanning Tree, negative weight edges posed no significant challenge to the algorithms. However, for shortest path, this is not the case
- If we have a negative weight cycle, we can just keep going around it, and $\delta(s, v) = -\infty$ for all v on the cycle.
- Some algorithms work only if there are no negative weight-edges in the graph

Just Like DP

Lemma

Any subpath of a shortest path is a shortest path

- Proof. (Same as DP)

Lemma

Shortest paths can't contain cycles

- (Single Source) shortest-path algorithms produce a label: $d[v] = \delta(s, v)$.
- Initially $d[v] = \infty$, reduces as the algorithm goes, so always $d[v] \geq \delta(s, v)$
- Also produce labels $\pi[v]$, predecessor of v on a shortest path from s.

Initializing

$\text{INIT-SINGLE-SOURCE}(V, s)$

for each v **in** V
do $d[v] \leftarrow \infty$
$\quad \pi[v] \leftarrow \text{NIL}$
$d[s] \leftarrow 0$

Relax!

- The algorithms work by improving (lowering) the shortest path estimate $d[v]$
- This operation is called relaxing an edge (u, v)
- Can we improve the shortest-path estimate for v by going through u and taking (u, v)?

$\textsc{Relax}(u, v, w)$

```
if d[v] > d[u] + w_uv
   then d[v] ← d[u] + w_uv
        π[v] ← u
```


How To Do It!

- All algorithms call $\textsc{Init-Single-Source}$ and then $\textsc{Relax}$, they differ in the order and number of times relax is called for an edge.

More Lemmas, (Lemml?)

Shortest Path Weights Obey Triangle Inequality

$$\delta(s, v) \leq \delta(s, u) + w_{uv} \quad \forall (u, v) \in E.$$

Relax Only Lowers Path Length Estimates

$$d[v] \geq \delta(s, v) \ \forall v \in V$$

Lemma, Lemma, Lemma

Path Relaxation Property

Let $P = \{v_0, v_1, \ldots v_k\}$ be a shortest path from $s = v_0$ to v_k. If the edges (v_0, v_1), (v_1, v_2), (v_{k-1}, v_k) are relaxed in that order, (there can be other relaxations in-between), then $d[v_k] = \delta(s, v_k)$

- Proof. Induction. (True for $i = 0$, since $d[s] = 0$). Assume $d[v_{i-1}] = \delta(s, v_{i-1})$, by calling $\textsc{Relax}(v_{i-1}, v_i)$, then $d[v_i] = \delta(s, v_i)$ must be a shortest path to v_i, and the label can never change.

Bellman-Ford Algorithm

- Works with Negative-Weight Edges
- Returns true is there are no negative-weight cycles reachable from s, false otherwise

```
BELLMAN-FORD(V, E, w, s)
INIT-SINGLE-SOURCE(V, s)
for i ← 1 to |V| − 1
do for each (u, v) in E
    do RELAX(u, v, w)
for each (u, v) in E
do if d[v] > d[u] + w_uv
      then return False
    return True
```


Analysis

- Here I'll show Example and Code
- Analysis
  - $\Theta(|V||E|)$
- Correctness?
  - Let v be reachable from s, and let $P = \{v_0, v_1, \ldots v_k\}$ be a shortest path to v. Each iteration of the **for** loop relaxes all edges. The first iteration relaxes (v_0, v_1), the next (v_1, v_2), the kth iteration relaxes (v_{k-1}, v_k), by the path relaxation Lemma, $d[v] = \delta(s, v)$,

Single Source Shortest Path on a DAG

```
DAG-SHORTEST-PATHS(V, E, s, w)
INIT-SINGLE-SOURCE(V, s)
topologically sort the vertices (HOW)
for each u in topologically sorted V
do for each v ∈ Adj[u]
    do RELAX(u, v, w)
```


SSSP-DAG Analysis and Correctness

- Correctness
  - Since vertices are processed in topolgically sorted order, edges of any path are relaxed in order of appearance on the path
  - Thus, edges on any shortest path are relaxed in order
  - Thus, by the path-relaxation lemma, the algorithm is correct
- Analysis

Dijkstra's Algorithm

- Works only if the graph as no negative-weight edges
- This is essentially a weighted-version of BFS
  - Instead of a FIFO Queue (like you used for BFS in the lab), use a priority queue
  - Keys (in PQ) are the shortest-path weight estimates $(d[v])$
- In Disjkstra's Algorithm, we have two sets of vertices
  - S: Vertices whose final shortest path weights are determines
  - Q: Priority queue: $V \setminus S$

Dijkstra's Algorithm

$\text{DIJKSTRA}(V, E, w, s)$
$\quad \text{INIT-SINGLE-SOURCE}(V, s)$
$\quad S \leftarrow \emptyset$
$\quad Q \leftarrow V$
$\quad$ **while** $Q \neq \emptyset$
$\quad$ **do** $u \leftarrow \text{EXTRACT-MIN}(Q)$
$\qquad S \leftarrow S \cup \{u\}$
$\qquad$ **for each** $v \in Adj[u]$
$\qquad$ **do** $\text{RELAX}(u, v, w)$

Dijkstra's Algorithm

- Note: Looks a lot like Prim's algorithm, but computing $d[v]$, and using the shortest path weights as keys
- Dijkstra's Algorithm is greedy, since it always chooses the "lightest" vertex in $V \setminus S$ to add to S
- Analysis: Like Prim's Algorithm, depends on the time it takes to perform priority queue operations.
- Suppose we use a binary heap.
  - How many times is whole loop called: $O(|E|)$
  - Inside loop, takes: $(O(\lg V))$ to EXTRACT-MIN.
- Dijkstra's Algorithm Runs in $O(E \lg V)$, with a binary heap implementation.
- Better Heap implementations get it down to $O(V \lg V + E)$.

Next Time

- Today in Lab: A "very" related problem... Traveling Salesman.
- Next Time: More Shortest Paths