IE170: Algorithms in Systems Engineering: Lecture 21

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

March 21, 2007

Taking Stock

Last Time

- Shortest Paths
- TSP in Lab

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 $w : \mathbb{E} \rightarrow \mathbb{R}^{|E|}$.
- We are interested in finding the shortest-path weights from u to v, which we will denote $\delta(u, v)$.
- $\delta(u, v) = \infty$ if there is no path from u to v in G
- (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 and Relaxing

Init-Single-Source(V, s)

```
for each v in V
do d[v] ← ∞
     π[v] ← NIL
d[s] ← 0
```

Relax(u, v, w)

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

Lemmas

Lemma

Any subpath of a shortest path is a shortest path

Lemma

Shortest paths can't contain cycles

Path Relaxation Property

Let $P = \{v_0, v_1, \dots v_k\}$ 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 $d[v_k] = \delta(s, v_k)$

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
```


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
  - Can You Do It!?!?!

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

$\text{Dijkstra}(V, E, w, s)$

$\text{Init-Single-Source}(V, s)$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$
do $u \leftarrow \text{Extract-Min}(Q)$
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
    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

Dijkstra Analysis

- Analysis: Like Prim's Algorithm, depends on the time it takes to perform priority queue operations.
- Suppose we use a binary heap.
  - EXTRACT-MIN: Called $O(|V|)$ times
  - RELAX: Called $O(|E|)$ times
  - How long does each of these operations take?
- Dijkstra's Algorithm Runs in $O(E \lg V)$, with a binary heap implementation.
- Better Heap implementations get it down to $O(V \lg V + E)$.
- Our "List/Container " implementation took $O(V^2)$

Dijkstra Correctness.

Loop Invariant

- At the start of each iteration of the **while** loop $\delta(s, v) = d[v] \; \forall v \in S$
- Initially: $S = \emptyset$, so this is trivially true
- At end: $S = V$, so we have the shortest path weights
- Maintenance: Must show that $d[u] = \delta(s, u)$ when u is added to S

We'll Give Proof (If Time)