Computational Problem: Finding Shortest Paths

Debswapna Bhattacharya, PhD
Assistant Professor
Computer Science and Software Engineering
Auburn University

Shortest Paths Problems

Given a weighted directed graph g=(V,E), with each edge having real-valued weights, the weight of a path from node u to node v is the sum of the weights of its edges. The shortest path from u to v is the path with the minimum path weight (if no such path exists, its weight is

Single Source Shortest Paths Problem

Given a single start node s, find shortest paths from s to all other nodes.

Single Source Shortest Paths

- The breadth-first search algorithm finds the shortest paths (i.e. paths with the least number of edges) from the start node to all other nodes when edges have no weights.
- This is the same as finding the shortest paths if all edge weights are 1.
- Thinking Assignment: Will the BFS algorithm find the shortest paths if all edges had the same constant weight c?

Single Source Shortest Paths

An algorithm for finding SSSPs can also solve the following problems:

1. Single Destination Shortest Paths (how?)
2. Single Pair Shortest Path (how?)
3. All Pairs Shortest Paths (how?)

Optimal Substructure Property

Lemma 24.1

If $p=<v_0,\ldots,v_k>$ is a shortest path from v_0 to v_k in a weighted directed graph $G=(V,E)$, then, for any i and j such that $0 \leq i \leq j \leq k$, let p_{ij} be the subpath of p from v_i to v_j. Then p_{ij} is the shortest path from v_i to v_j.

Proof: By contradiction.

Negative weight edges

- Edges with negative weights pose no problem.
- Shortest path weight may be negative in this case.
- But if there is a negative weight cycle that is reachable from the start node s, it is a problem! Why?
- In this case the shortest path is not defined and shortest path weight is $-\infty$

Positive or zero weight cycles

- Thus, a shortest path cannot contain a negative weight cycle.
- Can a shortest path contain any positive or zero weight cycles. Why or why not?

Shortest Paths

- Thus we can conclude that <u>shortest paths must be simple paths </u>(a simple path is one that contains no cycles).
- Any simple (acyclic) path in a graph G with n nodes and m edges can only contain at most n nodes and n-1 edges.
- Therefore, we can also conclude that <u>shortest paths can have at most n nodes and at most n-1 edges</u>.

Shortest Path Algorithms

- Use the attribute predecessor or previous (π) attached to nodes.
- Use the attribute distance (d) attached to nodes.

Relaxation

INITIALIZE-SINGLE-SOURCE (G,s) Complexity $\Theta(n)$

```
for each node v in G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

v.d = upper bound on the weight of a shortest path from s to v

v.π = previous node on the shortest path from s to v

G is the adjacency list representation; Each node in the adjacency list of a vertex contains the edge weight.

Relaxing an edge (u,v): testing if the currently known shortest path from s to v can be improved by going through the currently known shortest path from s to u and then from u to v along the edge (u,v). This is the only way in which a current estimate of a shortest path can change.

RELAX (u,v,w) Complexity $\Theta(1)$

```
if v.d>u.d+w(u,v)
    v.d=u.d+w(u,v)
    v.π= u
```

Bellman-Ford Algorithm

Returns a Boolean indicating whether there is a negative weight cycle reachable from source node s. If not, it determines shortest paths from s to all other vertices.

```
BELLMAN-FORD(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
for i=1 to n-1
        for each edge (u,v) in G.E
                RELAX (u,v,w)
for each edge (u,v) in G.E
        if v.d>u.d+w(u,v)
                return false
return true
```

Complexity: $O(nm)$ where $n=|V|$ and $m=|E|$ [or $O(VE)$]

SSSP Algorithm for DAGs

```
DAG-SHORTEST-PATHS(G,w,s)
topologically sort vertices of G
INITIALIZE-SINGLE-SOURCE(G,s)
for each vertex u taken in topological order
        for each vertex v in G.Adj[u]
            RELAX(u,v,w)
```

By relaxing the edges of a weighted directed acyclic graph according to the topological order of its nodes, shortest paths can be computed much faster, in $\Theta(m+n)$ time.

Note that s can be any vertex, not necessarily the first in the topological order

Thinking Assignments

- Redo the example worked out on the white board, but with a different start node and verify that all shortest paths from this start node are correctly found.

- Why does this strategy of considering vertices in the topological order work? How can we get away with not relaxing each edge n-1 times as in Bellman-Ford? Hint: if there is a path from s to v, s will appear before v in the topological order

Dijkstra's Algorithm

Edge weights must not be negative. Maintains a set S of vertices whose shortest paths from s have already been determined. Repeatedly selects a vertex u from V-S with a minimum shortest-path weight estimate u.d (use a min priority queue based on the d attribute), and relaxes all edges leaving u.

```
DIJKSTRA(G,w,s)
INITIALIZE-SINGLE-SOURCE(G,s)
S=empty set
Build min priority queue Q with nodes in G.V based on d
while Q≠empty
        u=EXTRACT-MIN(Q)
        S=S U {u}
        for each vertex v in G.AdjacencyList[u]
            RELAX(u,v,w) //substitute step 2 of RELAX with a DECREASE-KEY operation
```

Thinking Assignments

- Why/how does Dijkstra's algorithm fail if there are negative cost edges? Create a graph with a negative weight edge and run the algorithm on that graph to see why.
- Why does the algorithm work correctly if all edge costs are positive? I.e., why is it guaranteed that when a node is removed from the priority Q by the algorithm, the shortest path to it has already been found?

Shortest Path Algorithms

All algorithms go through the initialization step and then relax graph edges repeatedly. They differ in the number of times and order in which edges are relaxed.

1. Bellman-Ford
   - each edge is relaxed exactly $|V|-1=n-1$ times
   - works on graphs with negative weight edges
   - is able to detect negative weight cycles
   - complexity $O(mn)$
2. Shortest Paths in Directed Acyclic Graphs
   - each edge is relaxed exactly once
   - works only on acyclic graphs
   - linear algorithm: $\Theta(m+n)$
   - works on graphs with negative weight edges
3. Dijkstra's Algorithm
   - each edge is relaxed exactly once
   - edge weights must be nonnegative
   - complexity $O(n^2)$

Reading Assignment

- Chapter 24
- Read sections 24.1 – 24.3
- Omit all theorems, corollaries, lemmas and proofs (except those mentioned in these slides)
- Omit "Properties of shortest paths and relaxation" (p. 649-650)
- Omit the complexity analysis of Dijkstra's algorithm (p. 661-662)
- Read everything else
- Understand the PERT chart application discussed on p. 657 to find the longest path – not discussed in class

Thinking Assignments

Try problems

24.1-1 24.1-4

24.2-1, 24.2-2, 24.2-4

24.3-1, 24.3-2, 24.3-3