Advanced Algorithms Analysis and Design

By

Nazir Ahmad Zafar

Lecture No 33

Single-Source Shortest Paths

Today Covered

- Road map problem
- Linking road map problem with graph theory
- Shortest paths
- Cycles and their role in finding shortest paths
- The Bellman-Ford Algorithm
  - Initialization of graphs
  - Relaxation property
  - Algorithm design and analysis
  - Proof of correctness
- Applications
- Conclusion

Road Map Problem

- We are given a road map on which the distance between each pair of adjacent cities is marked, and our goal is to determine the shortest route from one city to another.
- The number of possible routes can be huge.
- How do we choose which one routes is shortest?
- This problem can be modelled as a graph
- And then we can find the shortest path from one city to another using graph algorithms.
- How to solve this problem efficiently?

Linking with Graphs

Linking Road Map with Graph Theory

Road map problem

- This problem can be modeled as a graph problem
- Road map is a weighted graph:

    set of *vertices* = set of cities

    set of *edges* = road segments between cities

    edge weight = length between two cities

  – Goal: find a shortest path between two vertices i.e. between two cities

Defining Shortest Path

Shortest Path?

- In a **shortest path problem**, a weighted, directed graph $G = (V, E)$ is given with weight function $w : E \rightarrow R$ mapping edges to real-valued weights.
- The **weight** of path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituents edges

$$w(p) = \sum_{i=1}^{k} \mathrm{w}(\mathrm{v}_{i-1}, \mathrm{v}_i)$$

$$= w(v_0, v_1) + w(v_1, v_2) + \ldots + w(v_{k-1}, v_k)$$

- A **shortest path** from vertex u to v is any path p with weight $w(p) = \delta(u, v)$

Path variants

Path Variants

- The **shortest path weight** from vertex u to v by

$$\delta(u, v) = \begin{cases} \min\{w(p) : u \xrightarrow{p} v\} & \text{if there is a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- Weight of edges can represent any metric which accumulates linearly along a path
  - Distance,
  - time,
  - cost,
  - penalty,
  - loss etc.

Problem variants

Variants of Shortest Path

- **Single-source shortest path**
  - G = (V, E) ⇒ find a shortest path from a given source vertex s to each vertex v ∈ V
- **Single-destination shortest path**
  - Find a shortest path to a given destination vertex **t** from each vertex v
  - Reverse the direction of each edge ⇒ single-source
- **Single-pair shortest path**
  - Find a shortest path from u to v for given vertices u and v
  - Solve the single-source problem
- **All-pairs shortest-paths**
  - Find shortest path for every pair of vertices u and v of G

Lemma optimality

Lemma : subpath of a shortest path, a shortest path

Statement

Given a weighted, directed graph $G = (V, E)$ with weight function $w : E \rightarrow \mathbf{R}$, let $p = \langle v_1, v_2, \ldots, v_k \rangle$ be a shortest path from vertex v_1 to vertex v_k, for any i, j such that $1 \le i \le j \le k$, let $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ be subpath of p from v_i to vertex v_j. Then, p_{ij} is a shortest path from v_i to v_j.

Proof

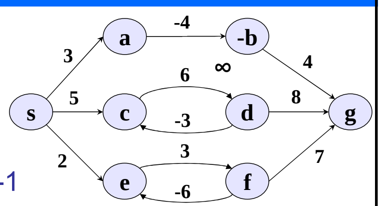
- We prove this lemma by contradiction
- If we decompose path p into $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{p_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$, then we have that $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$

Lemma (Contd..)

- Now, assume that there is a path p'_{ij} from v_i to v_j with weight $w(p'_{ij}) < w(p_{ij})$
- That is there is a subpath $p'_{i,j}$ from v_i to vertex v_j *which is shortest than* $p_{i,j}$
- Then, $v_1 \overset{p_{1i}}{\leadsto} v_i \overset{P'_{ij}}{\leadsto} v_j \overset{p_{jk}}{\leadsto} v_k$ is a path from vertices v_1 to v_k whose weight $w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$ is less than $w(p)$.
- It contradicts the assumption that p is a shortest path from v_1 to v_k.
- Hence subpath of a given shortest path is also a shortest path.

Why cycle not?

Why Positive Cycle Not?

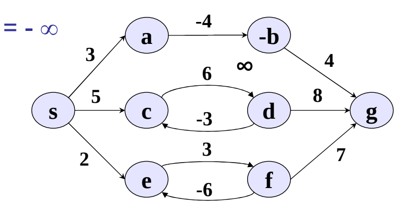


- s → a: only one path
  δ(s, a) = w(s, a) = 3
- s → b: only one path
  δ(s, b) =
  w(s, a) + w(a, b) = -1
- s → c: infinitely many paths
  ⟨s, c⟩, ⟨s, c, d, c⟩, ⟨s, c, d, c, d, c⟩
- cycle has positive weight (6 - 3 = 3)
  ⟨s, c⟩ shortest path with weight δ(s, c) = w(s, c) = 5,
- Positive cycle increases length of paths

Why cycle not?

Why Negative Cycle Not?

- s → e: infinitely many paths:
  - ⟨s, e⟩, ⟨s, e, f, e⟩, ⟨s, e, f, e, f, e⟩ etc.
  - cycle ⟨e, f, e⟩ has negative weight: 3 + (- 6) = -3
  - paths from s to e with arbitrarily large negative weights
  - δ(s, e) =- ∞ ⇒ no shortest path exists between s and e
  - Similarly:
    δ(s, f) = - ∞, δ(s, g) = - ∞



Removing cycle

Removing cycles from shortest paths

- If $p = \langle v_0, v_1, ..., v_k \rangle$ is a path and $c = \langle v_i, v_{i+1}, ..., v_j \rangle$ is a positive weight cycle on this path then the path $p' = \langle v_0, v_1, ..., v_i, v_{j+1}, v_{j+2}, ..., v_k \rangle$ has weight

$$w(p') = w(p) - w(c) < w(p),$$

and so p cannot be a shortest path from v_0 to v_k

- As long as a shortest path has 0-weight cycles, we can repeatedly remove these cycles from path until a cycle-free shortest path is obtained.

When no path

When is no shortest path?

- There may be edges with negative weight.
- A cycle $p = v_0, v_1, \ldots, v_k, v_0$ is a **negative cycle** such that $w(p) < 0$
- If a graph $G = (V, E)$ contains no negative weight cycle reachable from the source s, then for all $v \in V$, shortest path $\delta(s, v)$ remains well defined.
- If there is a negative weight cycles reachable from s, then shortest path weight are not well defined.
- If there is a path from u to v that contains a negative cycle, then shortest path is defined as

$$\delta(u, v) = -\infty$$

Cycles in shortest path

Summary of cycles in SPP

- Can shortest paths contain cycles?
- Negative-weight cycles: No!
- Positive-weight cycles: No!
  - By removing the cycle we can get a shorter path
- Zero-weight cycles
  - No reason to use them
  - Can remove them to obtain a path with similar weight

Note

- We will assume that when we are finding shortest paths, the paths will have no cycles

Predecessor graph

Representing Shortest Paths

- For a graph $G=(V, E)$, a **predecessor** $\pi[v]$ is maintained for each vertex $v \in V$
  - Either vertex or NIL
- We are interested in **predecessor subgraph** $G_\pi=(V_\pi, E_\pi)$ induced by π values, such that

$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$$
$$E_\pi = \{(\pi[v], v) \in E : v \in V_\pi - \{s\}\}$$

Predecessor graph

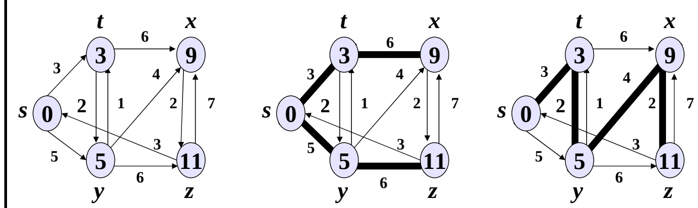
Representing Shortest Paths

- Let $G = (V, E)$ be a weighted, directed graph with weight function $w : E \rightarrow R$ and assume that G contains no negative weight cycles reachable from the source vertex $s \in V$, so that shortest paths are well defined.
- A shortest path tree rooted at s is a directed subgraph $G' = (V', E')$, where $V' \subseteq V$ and $E' \subseteq E$
- Shortest path are not necessarily unique and neither are shortest path trees.

Example, s. p. not unique

Shortest path not unique

- Shortest path are neither necessarily
  - unique and
  - nor shortest path trees



Initialization, relaxation

Initialization and Relaxation

Contd..

Initialization

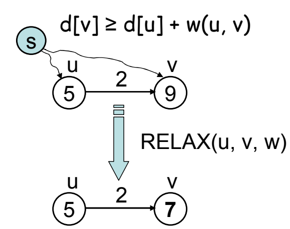
- All the shortest-paths algorithms start with initialization of vertices.

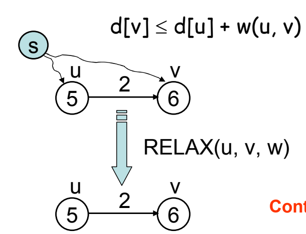
Relaxation

- For each vertex $v \in V$, an attribute $d[v]$ is defined and called a **shortest path estimate,** maintained
  - which is in fact, an upper bound on the weight of a shortest path from source s to v
- Process of **relaxing** an edge (u, v) consists of testing whether we can improve shortest path to v found so far, through u, if so update $d[v]$ and $\pi[v]$.

Relaxation

- **Relaxing** edge (u, v), testing whether we can improve shortest path to v found so far through u
  - If d[v] > d[u] + w(u, v)
  - we can improve the shortest path to v
  - $\Rightarrow$ update d[v] and π[v]





Contd..

Initialization and Relaxation

INITIALIZE-SINGLE-SOURCE (G, s)

for each vertex $v \in V[G]$
  do $d[v] \leftarrow \infty$
    $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$

Running time = Θ (V)

RELAX (u, v, w)

if $d[v] > d[u] + w(u, v)$
  then $d[v] \leftarrow d[u] + w(u, v)$
$\pi[v] \leftarrow u$

B. F algo.

Note:
All the single-source shortest-paths algorithms, start by calling INIT-SINGLE-SOURCE then relax edges. The algorithms differ in the order and how many times they relax each edge

The Bellman-Ford Algorithm

Input:

- Weighted, directed graph G, edges may be negative with weight function w : E$\rightarrow$ R,

Output

- it returns boolean value indicating whether or not there is a negative-weight cycle reachable from source.
- If there is such a cycle, it indicates no solution exists
- Else it produces shortest paths and their weights.

Note:

- It uses relaxation progressively decreasing estimate d[v] on weight of a shortest path from source s to each vertex v $\in$ V until it achieves actual SP weight δ(s, v).

Contd..

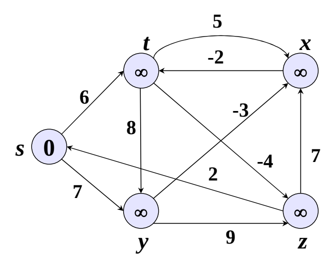
The Bellman-Ford Algorithm

BELLMAN-FORD (G, w, s)
INITIALIZE-SINGLE-SOURCE (G, s) — Θ (V)
for $i \leftarrow 1$ to $|V[G]| - 1$
  do for each edge $(u, v) \in E[G]$
    do RELAX (u, v, w)
} Θ (E)
for each edge $(u, v) \in E[G]$
  do if $d[v] > d[u] + w(u, v)$
    then return FALSE
} $O(E)$
return TRUE

Total Running Time = O(E)

Contd..

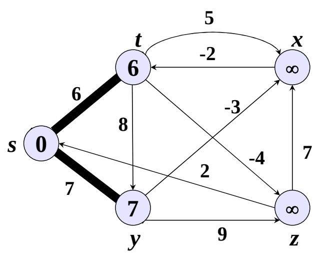
The Bellman-Ford Algorithm



For each vertex $v \in V(G)$
  $d[v] \leftarrow \infty$
  $\pi[v] \leftarrow$ NIL

Considering s as root node
  $d[s] \leftarrow 0$

The Bellman-Ford Algorithm



Considering edge (s, t)

$d[t] > d[s] + w(s, t)$ $(\infty > 0 + 6)$

$d[t] \leftarrow d[s] + w(s, t)$

$d[t] \leftarrow 0 + 6 = 6$

$\pi[t] \leftarrow s$

Considering edge (s, y)

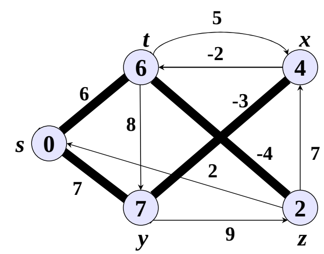
$d[y] > d[s] + w(s, y)$ $(\infty > 0 + 7)$

$d[y] \leftarrow d[s] + w(s, t)$

$d[y] \leftarrow 0 + 7 = 7$

$\pi[y] \leftarrow s$

The Bellman-Ford Algorithm



Considering edge (t, y)

$d[y] > d[t] + w(t, y)$

But $(7 < 6 + 8)$

Considering edge (t, z)

$d[z] > d[t] + w(t, z)$ $(\infty > 6 + (-4))$

$d[z] \leftarrow d[t] + w(t, z)$

$d[z] \leftarrow 6 + (-4) = 2$

$\pi[z] \leftarrow t$

Considering edge (y, x)

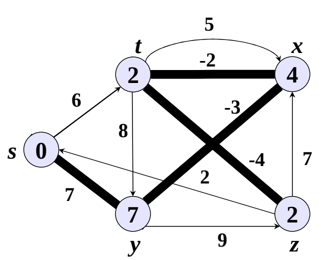
$d[x] > d[y] + w(y, x)$ $(\infty > 7 + (-3))$

$d[x] \leftarrow d[y] + w(y, x)$

$d[x] \leftarrow 7 + (-3) = 4$

$\pi[x] \leftarrow y$

The Bellman-Ford Algorithm



Considering edge (x, t)

$d[t] > d[x] + w(x, t)$ $(6 > 4 + (-2))$

$d[t] \leftarrow d[x] + w(x, t)$

$d[t] \leftarrow 4 + (-2) = 2$

$\pi[t] \leftarrow x$

Considering edge (y, z)

$d[z] > d[y] + w(y, z)$

but $(2 < 7 + 9)$

Considering edge (z, x)

$d[x] > d[z] + w(z, x)$

but $(4 < 2 + 7)$

Considering (z, s)

$d[s] > d[z] + w(z, s)$

but $(0 < 2 + 2)$

lemma

Lemma

Statement

Let $G = (V, E)$ be a weighted, directed graph with source s and weight function $w : E \rightarrow \mathbf{R}$, and assume that G contains no negative-weight cycles that are reachable from s. Then, after the $|V| - 1$ iterations of the ***for*** loop of lines 2-4 of BELLMAN-FORD, we have $d[v] = \delta(s, v)$ for all vertices v that are reachable from s.

Proof

- We prove the lemma by appealing to the path-relaxation property.

Lemma (Contd..)

- Consider any vertex v that is reachable from s, and let $p = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = s$ and $v_k = v$, be any acyclic shortest path from s to v.
- Path p has at most $|V| - 1$ edges, and so $k \leq |V| - 1$.
- Each of the $|V| - 1$ iterations of the **for** loop of lines 2-4 relaxes all E edges.
- Among the edges relaxed in the ith iteration, for $i = 1, 2, \ldots, k$, is (v_{i-1}, v_i).
- By the path-relaxation property, therefore,
  $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$.

Correctness

Theorem : Correctness of Bellman-Ford algorithm

Let BELLMAN-FORD be run on weighted, directed graph $G = (V, E)$, with source vertex s, and weight function $w : E \rightarrow R$.

- If G contains no negative-weight cycles that are reachable from s, then
  - $d[v] = \delta(s, v)$ for all vertices $v \in V$, and
  - the algorithm returns TRUE
  - the predecessor subgraph $G\pi$ is shortest-paths tree rooted at s.
- If G does contain a negative weight cycle reachable from s, then the algorithm returns FALSE.

Proof

Case 1

Suppose graph G contains no negative-weight cycles that are reachable from the source s.

- We first prove the claim that at termination, $d[v] = \delta(s, v)$ for all vertices $v \in V$.
  - If v is reachable from s, **Lemma above** proves it.
  - If v is not reachable from s, then the claim follows from the no-path property. Thus, the claim is proven.
- The predecessor subgraph property, along with the claim, implies that G_π is a shortest-paths tree.

Contd..

- Now we use the claim to show that BELLMAN-FORD returns TRUE.
  - At termination, for all edges (u, v)
  - $d[v] = \delta(s, v) \leq \delta(s, u) + w(u, v) = d[u] + w(u, v)$,
  - It therefore returns TRUE

Case 2,

- Suppose that graph G contains a negative-weight cycle that is reachable from the source s
- Let this cycle be $c = \langle v_0, v_1, \ldots, v_k \rangle$, where $v_0 = v_k$,

Then, $$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \qquad \text{(A)}$$

Contd..

- Assume for the purpose of contradiction that the Bellman-Ford algorithm returns TRUE.
- Thus, $d[v_i] \le d[v_{i-1}] + w(v_{i-1}, v_i)$ for $i = 1, 2, \ldots, k$.
- Summing the inequalities around cycle c gives us

$$\sum_{i=1}^{k} d[v_i] \le \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i))$$
$$= \sum_{i=1}^{k} (d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Contd..

- Since $v_0 = v_k$, each vertex in c appears exactly once in each of the summations and, and so

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

- Of course $d[v_i]$ is finite for $i = 1, 2, \ldots, k$. Thus,

$$0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Which contradicts inequality (A). And hence it proves the theorem

Applications

Different applications of shortest path

- Transportation problems
  - finding the cheapest way to travel between two locations
- Motion planning
  - what is the most natural way for a cartoon character to move about a simulated environment
- Communications problems
  - how look will it take for a message to get between two places which two locations are furthest apart i.e.
  - what is the diameter of network