Shortest Paths

Single-Source Shortest Paths
Dijkstra’s Algorithm
Bellman-Ford Algorithm
Difference Constraints
All-Pairs Shortest Paths
Floyd-Warshall Algorithm
Johnson’s Algorithm
Shortest-Paths

Shortest path problems on weighted graphs (directed or undirected) have two main types:

*Single-Source Shortest-Path:* find the shortest paths from source vertex \( s \) to all other vertices.

*All-Pairs Shortest-Path:* find the shortest paths between all pairs of vertices.

Some notation:
- \( w(u, v) \) = weight of edge \((u, v)\)
- \( w(p) \) = sum of weights on path \( p \)
- Let \( \delta(u, v) \) = shortest distance from \( u \) to \( v \).
- Let \( s^\text{opt} \to v \) stand for shortest path from \( s \) to \( v \).

Example Weighted Graphs

Example Bad Graph

Graphs with negative-weight cycles can’t be solved.

Key Properties

*Triangle Inequality:* If \((u, v)\) is an edge, then \( \delta(s, v) \leq \delta(s, u) + w(u, v) \). If \( p \) is a path from \( u \) to \( v \), then \( \delta(s, v) \leq \delta(s, u) + w(p) \).

Proof: \( \delta(s, v) > \delta(s, u) + w(u, v) \) contradicts \( \delta \)'s definition. So does \( \delta(s, v) > \delta(s, u) + w(p) \).

*Optimal Subpath Property:* If \( u \) is on \( s^\text{opt} \to v \), then \( \delta(s, v) = \delta(s, u) + \delta(u, v) \).

Proof: Let \( p \) be \( s^\text{opt} \to v \), and let \( p_1 \) and \( p_2 \) be the subpaths \( s \to u \) and \( u \to v \).

By definition, \( w(p) = w(p_1) + w(p_2) \geq \delta(s, u) + \delta(u, v) \). A lower \( \delta(s, u) \) or \( \delta(s, v) \) would contradict \( p \)'s optimality.
### Key Properties Continued

**Convergence Property:** If \( s \leadsto u \to v \) is a shortest path, then 
\[
\delta(s, v) = \delta(s, u) + w(u, v).
\]
**Proof:** Follows from the optimal subpath property.

Let \( v.d \geq \delta(s, v) \) for all \( v \) with \( s.d = 0 = \delta(s, s) \).

**Path-Relaxation Property:** If \( v.d > \delta(s, v) \), then some edge \((x, y)\) satisfies
\[
v.d = \delta(s, x) \quad \text{and} \quad y.d > x.d + w(x, y).
\]
**Proof:** Some edge \((x, y)\) on \( s \leadsto v \) must be the first edge with \( x.d = \delta(s, x) \) and \( y.d > x.d + w(x, y) = \delta(s, y) \).

### Basic Procedures

These properties justify the following procedures for single-source shortest-paths.

- \( v.d = \text{distance from } s \text{ to } v \).
- \( v.\pi = \text{previous vertex on } s \leadsto v \text{ path.} \)

**Initialize-Single-Source**

\[
\text{for each vertex } v \text{ in } G
\]
\[
v.d \leftarrow \infty
\]
\[
v.\pi \leftarrow \text{NIL}
\]
\[
s.d \leftarrow 0
\]

Initial \( \infty \) ensures that \( d.v \geq \delta(s, v) \) for all \( v \).

\( s.d = 0 \) ensures that \( s.d = \delta(s, s) \).

### Basic Procedures Continued

**RELAX**

\[
\text{if } v.d > u.d + w(u, v)
\]
\[
v.d \leftarrow u.d + w(u, v)
\]
\[
v.\pi \leftarrow u
\]

If \( u.d \geq \delta(s, u) \) and \( v.d \geq \delta(s, v) \) before \( \text{RELAX}(u, v, w) \) then they remain true afterwards.

Also, if \( v.d > \delta(s, v) \), then the Path Relaxation Property implies that some call to \( \text{RELAX} \) will improve \( d \) somewhere.

### Dijkstra's Algorithm

Dijkstra's Algorithm assumes all weights are nonnegative.

**DIJKSTRA**

\[
\text{INITIALIZE-SINGLE-SOURCE}(G, s)
\]
\[
S \leftarrow \emptyset
\]
\[
Q \leftarrow G.V \text{ using } d \text{ field for priority queue}
\]
\[
\text{while } Q \neq \emptyset
\]
\[
u \leftarrow \text{EXTRACT-MIN}(Q)
\]
\[
S \leftarrow S \cup \{u\}
\]
\[
\text{for each } v \in G.\text{Adj}[u]
\]
\[
\text{RELAX}(u, v, w)
\]
**Dijkstra Analysis**

Proof of Correctness:

Basis: $s$ is assigned to $u$, $s.d = 0 = \delta(s, s)$
Assume: For all $x \in S$, $x.d = \delta(s, x)$
Show: $u.d = \delta(s, u)$ when $u$ is extracted.
Induction: Proof by contradiction.
Suppose $\delta(s, u) < u.d$.
Some edge $(x, y)$ in $s \Rightarrow u$ goes from $S$ to $Q$.
$y.d = \delta(s, y)$ because $(x, y)$ has been relaxed,
but $u.d \leq y.d = \delta(s, y) \leq \delta(s, u) < u.d$
is a contradiction.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Difference Constraints**

A difference constraint is $x_j - x_i \leq b_k$, where $x_i$ and $x_j$ are variables and $b_k$ is a constant.

Difference constraints can represent time relationships between events.
If $x_2$ must occur within 2 hours after $x_1$, then $x_2 - x_1 \leq 2$.
If $x_2$ must occur at least 2 hours after $x_1$, then $x_2 - x_1 \geq 2$, which is equivalent to $x_1 - x_2 \leq -2$. 

**Bellman-Ford Algorithm**

This works with negative weights. FALSE is returned if a negative cycle is detected.

**Difference Constraints**

A difference constraint is $x_j - x_i \leq b_k$, where $x_i$ and $x_j$ are variables and $b_k$ is a constant.

Difference constraints can represent time relationships between events.
If $x_2$ must occur within 2 hours after $x_1$, then $x_2 - x_1 \leq 2$.
If $x_2$ must occur at least 2 hours after $x_1$, then $x_2 - x_1 \geq 2$, which is equivalent to $x_1 - x_2 \leq -2$. 

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$. 

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.

**Bellman-Ford Algorithm**

Proof of Correctness:

All finite shortest paths have $\leq V - 1$ edges.
The $i$th iteration relaxes $i$th edge in the optimal path.
If shortest path is finite, then after $V - 1$ iterations, $v.d = \delta(s, v)$.
If there is an infinite shortest path, then $v.d > u.d + w(u, v)$ for some edge $(u, v)$.
Difference Constraints Continued

The triangle inequality for shortest paths is a difference constraint.
\[ \delta(s, v) \leq \delta(s, u) + w(u, v) \] is equivalent to
\[ \delta(s, v) - \delta(s, u) \leq w(u, v). \]

A set of difference constraints \( x_j - x_i \leq b_k \) can be reduced to a weighted graph by \( w(v_i, v_j) = b_k \) and \( w(s, v_j) = w(s, v_i) = 0 \).

There is a solution for single-source shortest paths if and only if there is a solution to the difference constraints.

All-Pairs Shortest Paths

The all-pairs shortest-paths problem is finding the shortest path for each pair of vertices. Algorithms for this problem can be adapted for transitive closure.

Let \( \delta(i, j, m) \) be the shortest length from \( i \) to \( j \) using \( \leq m \) edges.

\[
\begin{align*}
\delta(i, j, 1) &= 0 & \text{if } i = j \\
&= w(i, j) & \text{if } (i, j) \text{ is an edge} \\
&= \infty & \text{otherwise}
\end{align*}
\]

\[
\delta(i, j, 2m) = \min_{k=1}^{n} (\delta(i, k, m) + \delta(k, j, m))
\]

All-Pairs Recursion

This recursive definition is correct.

Basis: Equation for \( \delta(i, j, 1) \) is correct.

Assume: \( \delta(i, j, m) \) is correct.

Show: Equation for \( \delta(i, j, 2m) \) is correct.

Induction: Optimal path of \( \leq 2m \) edges consists of two optimal paths of \( \leq m \) edges.

This leads to a \( O(V^3 \lg V) \) algorithm.
**All-Pairs Algorithm**

**ALL-PAIRS**($G, w$)

\[ n \leftarrow |G.V| \]
\[ D \leftarrow \text{an } n \times n \text{ matrix initialized to } \delta(i, j, 1) \]
\[ m \leftarrow 1 \]

while \( m < n - 1 \)

\[ \text{EXTEND}(D, n) \]
\[ m \leftarrow 2m \]

return \( D \)

**EXTEND**($D, n$)

\[ \text{for } i \leftarrow 1 \text{ to } n \]
\[ \text{for } j \leftarrow 1 \text{ to } n \]
\[ \text{for } k \leftarrow 1 \text{ to } n \]
\[ D[i, j] \leftarrow \min(D[i, j], D[i, k] + D[k, j]) \]

**Floyd-Warshall Algorithm**

**FLOYD-WARSHALL**($G, w$)

\[ n \leftarrow |G.V| \]
\[ D \leftarrow \text{an } n \times n \text{ matrix initialized to } \delta(i, j, 1) \]
\[ \text{for } k \leftarrow 1 \text{ to } n \]
\[ \text{for } i \leftarrow 1 \text{ to } n \]
\[ \text{for } j \leftarrow 1 \text{ to } n \]
\[ D[i, j] \leftarrow \min(D[i, j], D[i, k] + D[k, j]) \]

return \( D \)

**Floyd-Warshall** is \( O(V^3) \).

For transitive closure, use a 0/1 matrix and:
\[ D[i, j] \leftarrow D[i, j] \lor (D[i, k] \land D[k, j]) \]

**Floyd-Warshall Analysis**

Let \( \lambda(i, j, k) \) be the shortest distance from \( i \) to \( j \) with all intermediate vertices \( \leq k \).

Basis: \( D[i, j] \leq \lambda(i, j, 0) \) due to initialization.

Assume: \( D[i, j] \leq \lambda(i, j, k-1) \) before \( k \)th iteration.

Show: \( D[i, j] \leq \lambda(i, j, k) \) after \( k \)th iteration.

Induction: Either \( \lambda(i, j, k) = \lambda(i, j, k-1) \) or
\[ \lambda(i, j, k) = \lambda(i, k, k-1) + \lambda(k, j, k-1) \]
Floyd-Warshall Example

\[
\begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & \infty & -5 & 0 & \infty \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\begin{pmatrix}
0 & 3 & 8 & \infty & -4 \\
\infty & 0 & \infty & 1 & 7 \\
\infty & 4 & 0 & \infty & \infty \\
2 & 5 & -5 & 0 & -2 \\
\infty & \infty & \infty & 6 & 0
\end{pmatrix}
\]

Johnson’s Algorithm

\text{JOHNSON}(G, w, s)
\begin{equation}
(G', w') \leftarrow (G, w) \text{ with new vertex } s, \ w'(s, v) = 0
\end{equation}
\begin{algorithmic}
\If{Bellman-Ford}(G', w', s) = \text{FALSE}
\State \textbf{error} negative-weight cycle
\EndIf
\For{(u, v) \in G}
\State \(\hat{w}(u, v) \leftarrow w(u, v) + \delta'(s, u) - \delta'(s, v)\)
\EndFor
\For{each vertex \(u \in G\)}
\State \(\text{DIJKSTRA}(G, \hat{w}, u)\)
\EndFor
\For{each vertex \(v \in G\)}
\State \(D[u, v] \leftarrow \hat{\delta}(u, v) - \hat{\delta}'(s, u) + \delta'(s, v)\)
\EndFor
\Return \(D\)
\end{algorithmic}

Johnson’s Algorithm for Sparse Graphs

Idea of Johnson’s Algorithm:
Run Dijkstra’s algorithm \(n\) times.
Obtain \(O(VE\log V)\) time using binary heaps.
Better than \(O(V^3)\) if \(E\) is \(o(V^2)\).
Problem: Negative weights.
Solution: Clever way to transform the graph so that all weights are positive.

Johnson’s Algorithm Analysis

Triangle inequality, \(\delta'(s, v) \leq \delta'(s, u) + w(u, v)\), implies
\(0 \leq \delta'(s, u) + w(u, v) - \delta'(s, v) = \hat{\delta}(u, v)\).
Thus, \(\hat{\delta}(u, v) \geq 0\) for all edges \((u, v)\).

For any path \(p\) from \(u\) to \(v\), \(\hat{\delta}(p) = w(p) + \delta'(s, u) - \delta'(s, v)\) because of the telescoping sum.

\text{JOHNSON} is \(O(VE\log V)\) if binary heaps are used for \text{DIJKSTRA}.
Johnson Example

CS 5633 Analysis of Algorithms
Chapter 24 and 25: Slide – 26