COMP251: Single source shortest paths

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002)
Which assertions are true?

• A light edge crosses the cut. ✓
• A light edge is unique. ✗
• A MST is unique. ✗
• A graph A that respects the cut has no light edge. ✓
How do we decide if an edge \((i,j)\) belongs to a MST during the execution of the Kruskal's algorithm?

- When this edge connects two sets of vertices that are not connected.  ✓
- When the weight of \((i,j)\) is the lowest among all candidate edges.  ✗
- When the vertices \(i\) and \(j\) have not been used in the solution under construction.  ✗
Let $G=(V,E)$ be a connected undirected weighted graph on which we run the Prim's algorithm to compute a MST. How many iterations the main loop of the algorithm will do?

- $|E|$  
- $|V| - 1$  
- $|E| + |V|$  
- $|V|^2$
Problem

What is the shortest road to go from one city to another?

Example: Which road should I take to go from Montréal to Boston (MA)?

Variants:
- What is the fastest road?
- What is the cheapest road?
Modeling as graphs

Input:
• Directed graph $G = (V,E)$
• Weight function $w: E \rightarrow \mathbb{R}$

Weight of path $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$w(p) = \sum_{k=1}^{n} w(v_{k-1}, v_k)$$

= sum of edges weights on path $p$

Shortest-path weight $u$ to $v$:

$$\delta(u,v) = \begin{cases} 
\min \left\{ w(p) : u \xrightarrow{p} v \right\} & \text{If there exists a path } u \xrightarrow{p} v. \\
\infty & \text{Otherwise.}
\end{cases}$$

Shortest path $u$ to $v$ is any path $p$ such that $w(p) = \delta(u,v)$

Generalization of breadth-first search to weighted graphs
Example

Shortest path from s?
Shortest paths are organized as a tree. Vertices store the length of the shortest path from s.
Shortest paths are not necessarily unique!
Variants

• **Single-source:** Find shortest paths from a given source vertex $s \in V$ to every vertex $v \in V$.

• **Single-destination:** Find shortest paths to a given destination vertex.

• **Single-pair:** Find shortest path from $u$ to $v$.

  *Note: No way to known that is better in worst case than solving the single-source problem!*

• **All-pairs:** Find shortest path from $u$ to $v$ for all $u, v \in V$. 
Negative weight edges can create issues.

**Why?** If we have a negative-weight cycle, we can just keep going around it, and get \( w(s, v) = -\infty \) for all \( v \) on the cycle.

**When?** If they are reachable from the source. Corollary: OK, if the negative-weight cycles is not reachable from the source.

**Who?** Some algorithms work only if there are no negative-weight edges in the graph. We must specify when they are allowed and not.
Cycles

Shortest paths cannot contain cycles:

• Negative-weight: Already ruled out.

• Positive-weight: we can get a shorter path by omitting the cycle.

• Zero-weight: no reason to use them $\Rightarrow$ assume that our solutions will not use them.
Optimal substructure

Lemma
Any subpath of a shortest path is a shortest path.

Proof: (cut and paste)

Suppose this path $p$ is a shortest path from $u$ to $v$.
Then $\delta(u,v) = w(p) = w(p_{ux}) + w(p_{xy}) + w(p_{yv})$. 
**Lemma**
Any subpath of a shortest path is a shortest path.

**Proof:** (cont’d)

Now suppose there exists a shorter path $x \sim y$.

Then $w(p'_{xy}) < w(p_{xy})$.

$$w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv}) < w(p_{ux}) + w(p_{xy}) + w(p_{yv}) = w(p).$$

Contradiction of the hypothesis that $p$ is the shortest path!
Customized breadth-first search

Vertices count the number of edges used to reach them.
Customized breadth-first search
Customized breadth-first search
Customized breadth-first search
Customized breadth-first search

Can we generalize BFS to use edge weights?
Output of single-source shortest-path algorithm

For each vertex \( v \in V \):

- \( d[v] = \delta(s,v) \).
  - Initially, \( d[v] = \infty \).
  - Reduces as algorithms progress, but always maintain \( d[v] \geq \delta(s,v) \).
  - Call \( d[v] \) a **shortest-path estimate**.

- \( \pi[v] = \) predecessor of \( v \) on a shortest path from \( s \).
  - If no predecessor, \( \pi[v] = NIL \).
  - \( \pi \) induces a tree - **shortest-path tree** (see proof in textbook).
Algorithm structure

1. Initialization
2. Scan vertices and relax edges

The algorithms differ in the order and how many times they relax each edge.
Initialization

\[ \text{INIT-SINGLE-SOURCE}(V, s) \]
\[
\text{for each } v \in V \text{ do }
\]
\[
d[v] \leftarrow \infty
\]
\[
\pi[v] \leftarrow \text{NIL}
\]
\[
d[s] \leftarrow 0
\]
RELAX(u,v,w)
if d[v] > d[u] + w(u,v) then
d[v] ← d[u] + w(u,v)
π[v] ← u
Triangle inequality

For all \((u,v) \in E\), we have \(\delta(u,v) \leq \delta(u,x) + \delta(x,v)\).

Proof:
Weight of shortest path \(u \sim v\) is \(<=\) weight of any path \(u \sim v\).
Path \(u \sim x \sim v\) is a path \(u \sim v\), and if we use a shortest path \(u \sim x\) and \(x \sim v\), its weight is \(\delta(u, x) + \delta(x, v)\).
Upper bound property

Always have $\delta(s, v) \leq d[v]$ for all $v$. Once $d[v] = \delta(s, v)$, it never changes.

Proof:
Initially true.
Suppose there exists a vertex such that $d[v] < \delta(s, v)$.
Assume $v$ is first vertex for which this happens.
Let $u$ be the vertex that causes $d[v]$ to change.
Then $d[v] = d[u] + \delta(u, v)$.

\[
d[v] < \delta(s,v) \leq \delta(s, u) + \delta(u, v) \leq d[u] + \delta(u, v) \Rightarrow d[v] < d[u] + \delta(u,v).
\]
(triangle inequality) \hspace{1cm} (v is first violation)

Contradicts $d[v] = d[u] + \delta(u, v)$.
No-path property

If $\delta(s, v) = \infty$, then $d[v] = \infty$ always.

Proof: $d[v] \geq \delta(s,v) = \infty \Rightarrow d[v] = \infty.$
Convergence property

If:
1. $s \sim u \rightarrow v$ is a shortest path,
2. $d[u] = \delta(s, u)$,
3. we call RELAX($u, v, w$),
then $d[v] = \delta(s, v)$ afterward.

Proof:

After relaxation:

$d[v] \leq d[u] + w(u, v)$ \hspace{1cm} (RELAX code)

$= \delta(s, u) + w(u, v)$

$= \delta(s, v)$ \hspace{1cm} (lemma-optimal substructure)

Since $d[v] \geq \delta(s, v)$, must have $d[v] = \delta(s, v)$. 
Path-relaxation property

Let \( p = \langle v_0, v_1, ..., v_k \rangle \) be a shortest path from \( s = v_0 \) to \( v_k \). If we relax, *in order*, \((v_0,v_1), (v_1,v_2), ..., (v_{k-1},v_k)\), even intermixed with other relaxations, then \( d[v_k] = \delta(s,v_k) \).

**Proof:**

Induction to show that \( d[v_i] = \delta(s,v_i) \) after \((v_{i-1},v_i)\) is relaxed.

**Basis:** \( i = 0 \). Initially, \( d[v_0] = 0 = \delta(s,v_0) = \delta(s,s) \).

**Inductive step:** Assume \( d[v_{i-1}] = \delta(s,v_{i-1}) \). Relax \((v_{i-1},v_i)\). By convergence property, \( d[v_i] = \delta(s,v_i) \) afterward and \( d[v_i] \) never changes.
Single-source shortest paths in a DAG

Since a DAG, we are guaranteed no negative-weight cycles.

\[
\text{DAG-SHORTEST-PATHS}(V, E, w, s)
\]

*topologically sort the vertices*

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

for each vertex \( u \) in topological order do

  for each vertex \( v \in \text{Adj}[u] \) do

  RELAX\((u, v, w)\)

---

![Diagram](https://via.placeholder.com/150)
Example
Example

Graph with nodes labeled 0, 2, 6, ∞, and edges labeled with numbers and symbols.
Example
Example
Single-source shortest paths in a DAG

DAG-SHORTEST-PATHS($V, E, w, s$)
topologically sort the vertices
INIT-SINGLE-SOURCE($V, s$)
for each vertex $u$ in topological order do
    for each vertex $v \in \text{Adj}[u]$ do
        RELAX($u, v, w$)

**Time:** $(V + E)$.

**Correctness:**
Because we process vertices in topologically sorted order, edges of any path must be relaxed in order of appearance in the path.
$\Rightarrow$ Edges on any shortest path are relaxed in order.
$\Rightarrow$ By path-relaxation property, correct.
Dijkstra’s algorithm

- No negative-weight edges.
- Weighted version of BFS:
  - Instead of a FIFO queue, uses a **priority queue**.
  - Keys are shortest-path weights \((d[v])\).
- Have two sets of vertices:
  - \(S\) = vertices whose final shortest-path weights are determined,
  - \(Q\) = priority queue = \(V - S\).
- Similar Prim’s algorithm, but computing \(d[v]\), and using shortest-path weights as keys.
- Greedy choice: At each step we choose the light edge.
Dijkstra’s algorithm

\[
\text{DIJKSTRA}(V, E, w, s) \\
\text{INIT-SINGLE-SOURCE}(V, s) \\
S \leftarrow \emptyset \\
Q \leftarrow V \\
\textbf{while } Q \neq \emptyset \textbf{ do} \\
\hspace{1em} u \leftarrow \text{EXTRACT-MIN}(Q) \\
\hspace{1em} S \leftarrow S \cup \{u\} \\
\hspace{1em} \textbf{for each vertex } v \in \text{Adj}[u] \textbf{ do} \\
\hspace{2em} \text{RELAX}(u, v, w)
\]
Example
Example

- Nodes: s, 0, 5, 6, 8, 11
- Edges: s→0 (2), 0→5 (10), 5→6 (2), 6→8 (4), 8→11 (2), 5→11 (6), 0→5 (5), s→5 (2), 5→x (7)
- Query: Q = [x, z]
Example
Example
Example
Correctness

Loop invariant:
At the start of each iteration of the while loop, $d[v] = \delta(s,v)$ for all $v \in S$.

Initialization:
Initially, $S = \emptyset$, so trivially true.

Termination:
At end, $Q=\emptyset \Rightarrow S = V \Rightarrow d[v] = \delta(s,v)$ for all $v \in V$.

Maintenance:
Show that $d[u] = \delta(s,u)$ when $u$ is added to $S$ in each iteration.
Correctness (cont’d)

Show that $d[u] = \delta(s,u)$ when $u$ is added to $S$ in each iteration.

Suppose there exists $u$ such that $d[u] \neq \delta(s,u)$.

Let $u$ be the first vertex for which $d[u] \neq \delta(s, u)$ when $u$ is added to $S$.

- $u \neq s$, since $d[s] = \delta(s,s) = 0$.
- Therefore, $s \in S$, so $S \neq \emptyset$.
- There must be some path $s \rightsquigarrow u$. Otherwise $d[u] = \delta(s,u) = \infty$ by no-path property.
- So, there is a path $s \rightsquigarrow u$. Thus, there is a shortest path $s \rightsquigarrow u$. 
Correctness (cont’d)

Show that \( d[u] = \delta(s,u) \) when \( u \) is added to \( S \) in each iteration.

Just before \( u \) is added to \( S \), path \( p \) connects a vertex in \( S \) (i.e., \( s \)) to a vertex in \( V - S \) (i.e., \( u \)).

Let \( y \) be first vertex along \( p \) that is in \( V - S \), and let \( x \in S \) be \( y \) is predecessor.

Decompose \( p \) into \( s \sim s \rightarrow x \to y \sim u \).
Correctness (cont’d)

**Claim:** $d[y] = \delta(s, y)$ when $u$ is added to $S$.

**Proof:**

$x \in S$ and $u$ is the first vertex such that $d[u] \neq \delta(s, u)$ when $u$ is added to $S \Rightarrow d[x] = \delta(s, x)$ when $x$ is added to $S$.

Relaxed $(x, y)$ at that time, so by the convergence property, $d[y] = \delta(s, y)$.
Correctness (cont’d)

Show that \(d[u] = \delta(s,u)\) when \(u\) is added to \(S\) in each iteration.

Now can get a contradiction to \(d[u] \neq \delta(s, u)\): \(d[u] = \delta(s, u)\)

- \(y\) is on shortest path \(s \sim u\), and all edge weights are nonnegative.
  \(\Rightarrow \delta(s, y) \leq \delta(s, u)\)
  \(\Rightarrow d[y] = \delta(s, y)\)
  \(\leq \delta(s, u)\)
  \(\leq d[u]\) (upper-bound property)

In addition, since \(y\) and \(u\) were in \(Q\) when we chose \(u\):

\(d[u] \leq d[y] \Rightarrow d[u] = d[y].\)

Therefore, \(d[y] = \delta(s, y) = \delta(s, u) = d[u].\)

Contradicts assumption that \(d[u] \neq \delta(s,u).\)
Analysis

Like Prim’s algorithm, it depends on implementation of priority queue.

If binary heap, each operation takes $O(\lg V)$ time
$\Rightarrow O(E \lg V)$.

Note: We can achieve $O(V \lg V + E)$ with Fibonacci heaps.