Shortest Paths

Digraph with edge weights (costs, distances)

Shortest path from s to t: path of minimum total wt.

Problems:

- single pair: given s, t, find a shortest path from s to t
- single source: given s, find shortest paths from s to all reachable vertices
- all pairs: find shortest paths between all pairs

Cases:

- acyclic
- no negative wts
- general
- (planar, etc.)
Properties:

If a shortest path from $s$ to $t$ iff there is no negative (total wt.) cycle on a path from $s$ to $t$.

If there is no such cycle, there is a shortest path that is simple (no repeated vertex).

If no neg cycle reachable from $s$, then a shortest path tree: rooted at $s$, contains all vertices reachable from $s$, all tree paths are shortest paths in graph.

New goal: find a negative cycle or construct a shortest path tree.

(single-source problem is central)
Given a spanning tree \( T \) rooted at \( s \),
\[
d(v) = \text{tree wt from } s \text{ to } v, \text{ is } T \text{ a shortest path tree?}
\]

Yes, iff there is no \( (v, w) \) with \( d(v) + c(v, w) < d(w) \)

Edge relaxation algorithm to find a shortest path tree:
\[
d(s) = 0, \ d(v) = \infty \text{ for } v \neq s
\]

while \( \exists \text{ edge } (v, w) \text{ with } d(v) + c(v, w) < d(w) \)
\[
\{ \text{ do } d(w) = d(v) + c(v, w); \ p(w) = v \}
\]

\( d(v) \) is always the wt of some \( s-v \) path

if algorithm stops and \( p \) defines a tree, must be a shortest path tree

stops iff no neg cycle

(alg maintains \( d(w) \geq d(v) + c(v, w) \text{ if } v = p(w) \))
Suppose $T$ not a $sp$ tree. Let $x$ be such that $d(x) > s-x$ distance $L = \{P\}$ be a shortest path from $s$ to $x$, $d'(v) = P -$ distance from $s$ to $(v,w)$ first edge along $P$ such that $d'(w) < d(w)$. Then $d'(v) + c(v,w) = d'(w) + c(v,w) = d'(w) < d(w)$. (This gives the hard direction of $sp$ tree test.)

Suppose edge relaxation algorithm creates a cycle. Then it must be a negative cycle.

\[d'(v) + c(v,w) < d(w) \Rightarrow d'(v) - d(w) + c(v,w) < 0\]

Turn around cycle: \[\sum_{i=1}^{k} d'(v_i) - d(w) + c(v_i,w_i) < 0\]
Labeling and scanning algorithm:

$L = \{ s \}; \quad d(s) = 0; \quad d(v) = \infty \text{ for } v \neq s$

while $L \neq \emptyset$ do

remove $v$ from $L$;

scan($v$); for each $(v,w)$ do

if $d(v) + c(v,w) < d(w)$ then

$d(w) = d(v) + c(v,w); \quad p(w) = v; \quad$ add $w$ to $L$

end

end

(unlabeled)

(labelled)

(scanned)
Algorithm: topological sorting order

\[ D(n) \]

Non-negative weights: shortest-first scanning order

\[ D(n) \text{ original } D(m \log n) \text{ standard heap } D(m \log n + n) \text{ Fibonacci heap} \]

No vertex scanned more than once:

Invariant \( d(x) \leq a(x) = d(x) \)

\[
\begin{array}{c}
\text{smallest } x \\
\end{array}
\]

\[
\begin{array}{c}
x \leftarrow \text{parent} \times \\
\end{array}
\]

\[
\begin{array}{c}
= 0 \\
\end{array}
\]
General case: FIFO scanning order

Maintain $L$ as an (ordinary) queue.

Phase $k$:

phase 0: scan of $s$

phase $k$: scan of vertices added to $L$ during phase $k-1$

After phase $k$, all distances for shortest paths of $k+1$ or fewer edges are correct.

$\Rightarrow$ $k+1$ or fewer phases

$\Rightarrow \mathcal{O}(nm)$ time.
Negative cycle detection:

Method 1: Count phases, stop after first scan of n-th phase. Parent pointers will define a (negative) cycle.

Method (dynamic protection): Maintain a processor list of vertices in tentative shortest path tree. When reaching w using (v,w), explore the subtree rooted at w, disassembling it and looking for v.

Both methods take O(n^2) time total.

(Theoretically) inferior methods:

Method 3: When replacing v using (v,w), follow parent pointers from v looking for w.

Method 4: Maintain tentative shortest path tree as a dynamic tree.
bad (negative cycle)
good
Dijkstra's algorithm:

Heap is maintained: vertices are removed in increasing order by tentative distance.

Can exploit this if edge weights are (small) integers.

Dig: bucket for tentative distance

\# buckets = \max \text{ edge wt. } (c) + 1

\( O(m + cn) \) time

\[ \text{unsorted} \]

\[ \begin{array}{cccccccc}
& & & & & & & \\
\downarrow & & & & & & & \\
\text{smallest} & \rightarrow & \text{non-empty} & \text{bucket} & \\
\end{array} \]

Refinement: Use multiple levels of buckets

\[ \sqrt{c} \text{ buckets} \rightarrow \sqrt{c} \text{ levels} \]

\[ \begin{array}{cccccccc}
& & & & & & & \\
\downarrow & & & & & & & \\
\text{sorted} & \rightarrow & \text{sorted} & \rightarrow & \text{sorted} & \\
\end{array} \]

\( \mathcal{O}(m + \sqrt{cn}) \rightarrow \mathcal{O}(m + n \log \sqrt{c}) \) (binary levels)
a single source

Dijkstra\textsuperscript{o}(n^2 + \log n)

- 30.19n^2 - f(n) = \text{oblivious neg edge cost}\n
\theta(n)

\ell'(v,w) = \ell(v,w) + p(v) - p(w) = 0
All points

Dynamic prog.

\[ P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_n \]

\[ d(x,y) = 0 \]

\[ d(x,y) = \infty \text{ if } y \notin \mathcal{X}_n \]

\[ d(x,y) = d(x,y') + f(y') \text{ if } x = y' \land \forall y' 

for 2

for 1

for 2

\[ d'(y) = d(x,y) + f(y) \]

\[ O(n^3) \]
Heuristic Search: Let $e(v)$ be an estimate of the distance from $v$ to the goal $t$.

Use Dijkstra's algorithm with $d(v) + e(v)$ as the selection criterion.

The method works if

$$e(v) \leq d(v, w) + e(w) \text{ for all } v, w$$

(Estimate $e$ is a consistent lower bound on the actual distance.)

In Euclidean graphs the distance "as the crow flies" works.

Hart, Nilsson, Rafael (1968)
Dijkstra's algorithm

Heuristic search
Bidirectional Search: Search forward from $s$ and backward from $t$ concurrently.

$\Rightarrow$ Getting the stopping rule correct is tricky, especially for bidirectional heuristic search.