Shortest Paths

- Dijkstra’s algorithm
- Bellman-Ford algorithm

Shortest Path Problem

- Shortest path network.
  - Directed graph.
  - Source s, destination t.
  - cost(v-w) = cost of using edge from v to w.

Shortest path problem: find shortest directed path from s to t.
- Cost of path = sum of arc costs in path.

Cost of path s-2-3-5-t = 9 + 23 + 2 + 16 = 48.

Brief History

- Ford (1956). RAND, economics of transportation.
Applications

More applications.
- Robot navigation.
- Typesetting in TeX.
- Urban traffic planning.
- Tramp steamer problem.
- Optimal pipelining of VLSI chip.
- Telemarketer operator scheduling.
- Subroutine in higher level algorithms.
- Routing of telecommunications messages.
- Approximating piecewise linear functions.
- Exploiting arbitrage opportunities in currency exchange.
- Open Shortest Path First (OSPF) routing protocol for IP.
- Optimal truck routing through given traffic congestion pattern.


Graphs

<table>
<thead>
<tr>
<th>Graph</th>
<th>Vertices</th>
<th>Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>communication</td>
<td>telephones, computers</td>
<td>fiber optic cables</td>
</tr>
<tr>
<td>circuits</td>
<td>gates, registers, processors</td>
<td>wires</td>
</tr>
<tr>
<td>mechanical</td>
<td>joints</td>
<td>rods, beams, springs</td>
</tr>
<tr>
<td>hydraulic</td>
<td>reservoirs, pumping stations</td>
<td>pipelines</td>
</tr>
<tr>
<td>financial</td>
<td>stocks, currency</td>
<td>transactions</td>
</tr>
<tr>
<td>transportation</td>
<td>street intersections, airports</td>
<td>highways, airway routes</td>
</tr>
<tr>
<td>scheduling</td>
<td>tasks</td>
<td>precedence constraints</td>
</tr>
<tr>
<td>software systems</td>
<td>functions</td>
<td>function calls</td>
</tr>
<tr>
<td>internet</td>
<td>web pages</td>
<td>hyperlinks</td>
</tr>
<tr>
<td>games</td>
<td>board positions</td>
<td>legal moves</td>
</tr>
<tr>
<td>social relationship</td>
<td>people, actors</td>
<td>friendships, movie casts</td>
</tr>
<tr>
<td>neural networks</td>
<td>neurons</td>
<td>synapses</td>
</tr>
<tr>
<td>protein networks</td>
<td>proteins</td>
<td>protein-protein interactions</td>
</tr>
<tr>
<td>chemical compounds</td>
<td>molecules</td>
<td>bonds</td>
</tr>
</tbody>
</table>

Shortest Path

Some versions of the problem that we consider.
- Undirected.
- Directed.
- Single source.
- All-pairs.
- Arc costs are $\geq 0$.
- Points in plane with Euclidean distances.

Valid weights. For all $v$, $w_t[v]$ is length of some path from $s$ to $v$.

Edge relaxation.
- Consider edge $v-w$ with $G.cost(v, w)$.
- If path from $s$ to $v$ plus edge $v-w$ is better than current path to $w$, then update.

```
if (w_t[w] > w_t[v] + G.cost(v, w)) {
    w_t[w] = w_t[v] + G.cost(v, w);
    pred[w] = v;
}
```

Shortest Path: Edge Relaxation
The question of whether computers can think is like the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.

---

**Dijkstra's Algorithm: Implementation**

- Initialize $w_t[v] = \infty$ and $w_t[s] = 0$.
- Insert all vertices $v$ onto $PQ$ with priorities $w_t[v]$.
- Repeatedly delete node $v$ from $PQ$ that has min $w_t[v]$.
  - add $v$ to $S$
  - for each $v-w$, relax $v-w$

```java
while (!pq.isEmpty()) {
    int v = pq.delMin();
    IntIterator i = G.neighbors(v);
    while (i.hasNext()) {
        int w = i.next();
        if ($w_t[w] > w_t[v] + G.cost(v, w)$) {
            $w_t[w] = w_t[v] + G.cost(v, w);$;
            pq.decrease(w, $w_t[w]$);
            pred[w] = v;
        }
    }
}
```

---

**Dijkstra's Algorithm: Proof of Correctness**

**Invariant.** For each vertex $v$, $w_t[v]$ is length of shortest $s-v$ path whose internal vertices are in $S$; for each vertex $v$ in $S$, $w_t[v] = w^*_t[v]$.

**Proof:** by induction on $|S|$.

**Base case:** $|S| = 0$ is trivial.

**Induction step:**
- Let $v$ be next vertex added to $S$ by Dijkstra's algorithm.
- Let $P$ be a shortest $s-v$ path, and let $x-y$ be first edge leaving $S$.

1. We show $w_t[v] = w^*_t[v]$.

$$w_t[v] \geq w^*_t[v] \geq w^*_t[y] = w_t[y] \geq w_t[v]$$
Dijkstra’s Algorithm: Implementation Cost Summary

<table>
<thead>
<tr>
<th>Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap $^+$</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td>$V$</td>
<td>$V$</td>
<td>$\log V$</td>
<td>$d \log V$</td>
<td>$1$</td>
</tr>
<tr>
<td>delete-min</td>
<td>$V$</td>
<td>$V$</td>
<td>$\log V$</td>
<td>$d \log V$</td>
<td>$\log V$</td>
</tr>
<tr>
<td>decrease-key</td>
<td>$E$</td>
<td>$1$</td>
<td>$\log V$</td>
<td>$\log V$</td>
<td>$1$</td>
</tr>
<tr>
<td>is-empty</td>
<td>$V$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
</tr>
<tr>
<td>total</td>
<td>$V^2$</td>
<td>$E \log V$</td>
<td>$E \log E/V$</td>
<td>$E + V \log V$</td>
<td></td>
</tr>
</tbody>
</table>

$^+$ Individual ops are amortized bounds

Observation: algorithm is almost identical to Prim’s MST algorithm!
Priority first search: variations on a theme.

Shortest Path in Euclidean Graphs

Euclidean graph (map).
- Vertices are points in the plane.
- Edges weights are Euclidean distances.

Sublinear algorithm.
- Assume graph is already in memory.
- Start Dijkstra at $s$.
- Stop as soon as you reach $t$.

Exploit geometry. ($A^*$ algorithm)
- For edge $v-w$, use weight $d(v, w) + d(w, t) - d(v, t)$.
- Dijkstra’s proof of correctness still applies.
- In practice only $O(V^{1/2})$ vertices examined.

Shortest Path Application: Currency Conversion

Given currencies and exchange rates, what is best way to convert one ounce of gold to US dollars?
- 1 oz. gold $\Rightarrow$ £208.10 $\Rightarrow$ 208.10 (1.5714) $\Rightarrow$ $327.00$.
- 1 oz. gold $\Rightarrow$ 455.2 Francs $\Rightarrow$ 304.39 Euros $\Rightarrow$ $327.28$.

<table>
<thead>
<tr>
<th>Currency</th>
<th>£</th>
<th>Euro</th>
<th>¥</th>
<th>Franc</th>
<th>$</th>
<th>Gold</th>
</tr>
</thead>
<tbody>
<tr>
<td>UK Pound</td>
<td>1.0000</td>
<td>0.6853</td>
<td>0.005290</td>
<td>0.4569</td>
<td>0.6368</td>
<td>208.100</td>
</tr>
<tr>
<td>Euro</td>
<td>1.4599</td>
<td>1.0000</td>
<td>0.007721</td>
<td>0.6677</td>
<td>0.9303</td>
<td>304.028</td>
</tr>
<tr>
<td>Japanese Yen</td>
<td>189.050</td>
<td>129.520</td>
<td>1.0000</td>
<td>85.4694</td>
<td>120.400</td>
<td>393.467</td>
</tr>
<tr>
<td>Swiss Franc</td>
<td>2.1904</td>
<td>1.4978</td>
<td>0.011574</td>
<td>1.0000</td>
<td>1.3929</td>
<td>455.200</td>
</tr>
<tr>
<td>US Dollar</td>
<td>1.5714</td>
<td>1.0752</td>
<td>0.008309</td>
<td>0.7182</td>
<td>1.0000</td>
<td>327.25</td>
</tr>
<tr>
<td>Gold (oz.)</td>
<td>0.004816</td>
<td>0.003295</td>
<td>0.0000255</td>
<td>0.002201</td>
<td>0.003065</td>
<td>1.0000</td>
</tr>
</tbody>
</table>

Shortest Path Application: Currency Conversion

Graph formulation.
- Create a vertex for each currency.
- Create a directed edge for each possible transaction, with weight equal to the exchange rate.
- Find path that maximizes product of weights.
Shortest Path Application: Currency Conversion

Reduction to shortest path problem.

- Let $γ_{vw}$ be exchange rate from currency $v$ to $w$.
- Let $c_{vw} = -\lg γ_{vw}$.
- Shortest path with costs $c$ corresponds to best exchange sequence.

Dijkstra's Algorithm With Negative Costs

Dijkstra's algorithm fails if there are negative weights.
- Ex: Selects vertex $v$ immediately after $s$.
- But shortest path from $s$ to $v$ is $s-x-y-v$.

Challenge: shortest path algorithm that works with negative costs.

Dynamic Programming

Dynamic programming.
- Initialize $w_t[v] = w$, $w_t[s] = 0$.
- Repeat $v$ times: relax each edge $v-w$.

Bellman-Ford-Moore Algorithm

Practical improvement.
- If $w_t[v]$ doesn't change during phase $i$, don't relax any edges of the form $v-w$ in phase $i+1$.
- Programming solution: maintain queue of nodes that have changed.

Running time. Still $\Omega(EV)$ in worst case, but linear in practice!
Negative Cycles

Negative cycle. Directed cycle whose sum of edge costs is negative.

\[-6 + 7 - 4 = -5\]

Caveat. Bellman-Ford terminates and finds shortest (simple) path after at most \(V\) phases if and only if no negative cycles.

Observation. If negative cycle on path from \(s\) to \(t\), then shortest path can be made arbitrarily negative by spinning around cycle.

\[
\begin{aligned}
\text{negative cost cycle} & \\
-6 + 7 - 4 &= -5
\end{aligned}
\]

Shortest Path Application: Arbitrage

Arbitrage.

- Is there an arbitrage opportunity in currency graph?
- Ex: \(\$1 \Rightarrow 1.3941\) Francs \(\Rightarrow 0.9308\) Euros \(\Rightarrow \$1.00084\).
- Is there a negative cost cycle?
- Fastest algorithm very valuable!

![Currency graph diagram]

Bellman-Ford-Moore Algorithm

Finding the shortest path itself.

- Trace back \(\text{pred}[v]\) as in Dijkstra's algorithm.

Finding a negative cycle.

- If any node \(v\) is enqueued \(V\) times, there must be a negative cycle.
- Fact: can trace back \(\text{pred}[v]\) to find cycle.

![Bellman-Ford diagram]

Single Source Shortest Paths Implementation: Cost Summary

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Worst Case</th>
<th>Best Case</th>
<th>Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dijkstra (classic) †</td>
<td>(V^2)</td>
<td>(V^2)</td>
<td>linear</td>
</tr>
<tr>
<td>Dijkstra (heap) †</td>
<td>(E \log V)</td>
<td>(E \log V)</td>
<td>linear</td>
</tr>
<tr>
<td>Dynamic Programming ‡</td>
<td>(E)</td>
<td>(E)</td>
<td>linear</td>
</tr>
<tr>
<td>Bellman-Ford ‡</td>
<td>(E)</td>
<td>(E)</td>
<td>linear</td>
</tr>
</tbody>
</table>

† nonnegative costs
‡ no negative cycles or negative cycle detection

Remark 1: negative weights makes the problem harder.
Remark 2: negative cycles makes the problem intractable.