Finding Shortest Paths

Shortest Path Problem

Shortest Path Problem

We are given a graph $G = (V, E)$ and an edge weight function $\omega\colon E \longrightarrow \mathbb{R}$.

Length of a Path

The *length* or *weight* $\omega(P)$ of a path $P = \{v_1, v_2, \ldots, v_l\}$ with at least two vertices is

$$\omega(P) = \sum_{i=1}^{l-1} \omega(v_i v_{i+1})$$

If $|P| = 1$, $\omega(P) = 0$.

Shortest Path Problem

Shortest Path

For two vertices u and v, the *shortest path* from u to v is the path P for which $\omega(P)$ is minimal. The *distance* $d(u, v)$ from u to v is the length of a shortest path from u to v.

Shortest Path Problem

Variants

- *Single Pair Shortest Path* (SPSP)
  Find a shortest path from a vertex u to some vertex v.
- *Single Source Shortest Path* (SSSP)
  Find shortest paths from a source vertex v to all other vertices in the graph.
- *All Pairs Shortest Path* (APSP)
  Find shortest paths fall vertex pairs u and v.

There is no algorithm for SPSP which is better in general than an algorithm for SSSP.

Shortest Path Properties

Theorem — Optimal Substructure Property

Each subpath of a shortest path is a shortest path.

Theorem — Triangle Inequality

For all vertices u, v, and w,

$$d(u, v) \leq d(u, w) + d(w, v).$$

Negative Weight Edges and Cycles

Negative Weight Edges

- Natural in some application
- Makes finding a shortest path harder

Theorem

If there is a path from u to v containing a vertex w and w is in a cycle C with $\omega(C) < 0$, then there is no shortest path from u to v.

Avoiding Cycles

- Only permit simple paths, i. e., no vertex twice
- Follows if graph has no negative cycles
- With negative cycles, shortest simple path problem equal to longest simple path problem
- Problem: loss of optimal substructure property

General Approach

General Approach

Store for each vertex v

- $\mathrm{dist}_s(v)$, length of currently best known path P from start vertex s to v
- $\mathrm{par}_s(v)$, parent of v in P

Relaxation

- Updates best known distance.

Procedure *Relax(u, v)*
  If $\mathrm{dist}_s(v) > \mathrm{dist}_s(u) + \omega(uv)$ **Then**
    Set $\mathrm{par}_s(v) := u$ and $\mathrm{dist}_s(v) := \mathrm{dist}_s(u) + \omega(uv)$.

General Approach

Initialization

- Set $\mathrm{par}_s(v) := \texttt{null}$ and $\mathrm{dist}_s(v) := \infty$ for each vertex v.
- Set $\mathrm{dist}_s(s) := 0$ for start vertex s.

Iteration

- Pick vertex pair u, v.
- Call Relax(u, v)
- Repeat

Open Questions

- How do we pick u and v?
- When do we stop the iteration?

Single Source Shortest Path

Shortest Path for DAGs

Directed Acyclic Graphs

- No (negative) cycles
- Topological order

Algorithm Idea

- Find a topological order $\langle v_1, v_2, \ldots, v_n \rangle$.
- For $i := 1$ to n, relax all outgoing edges of v_i.

Properties

- Invariant: For all v_j with $j \leq i$, $\mathrm{dist}(v_j)$ is optimal.
- Runtime: linear
- Works with negative edges, i. e., can be used to compute longest path.

Observation

- A shortest path has at most $n-1$ edges.
- If we know all shortest path with k edges, we can compute all shortest paths with $k+1$ edges by relaxing all edges once.

```
For Each v ∈ V
    Set dist(v) := ∞ and par(v) = null.
Set dist(s) := 0.
For i := 1 To |V| − 1
    For Each (u, v) ∈ E
        Relax(u, v)
```

Bellman-Ford

Properties

- Runtime: $\mathcal{O}(nm)$
- Works with negative weight edges
- Can detect negative cycles

Detecting negative cycles

- Negative cycle $\longrightarrow$ There is always an edge (u, v) for which Relax(u, v) updates $\mathrm{dist}(v)$.
- If Relax(u, v) still updates $\mathrm{dist}(v)$ for $i \geq n$, then (u, v) is part of a negative cycle.

Dijkstra’s Algorithm

Idea

- Let S be set of vertices where shortest path is known.
- Relax all outgoing edges (u, v), i. e., $u \in S$ and $v \notin S$.
- If $\mathrm{dist}(v)$ is minimal for all vertices not in S, then $\mathrm{dist}(v)$ is optimal.
- Add v to S and repeat.

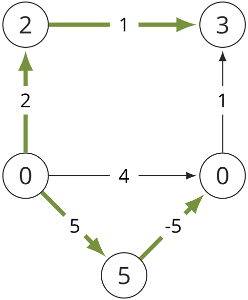
```
Initialize(G, s)
Create a priority Q and add all vertices in V.
While Q is not empty
    Remove v with minimal dist(v) from Q.
    For Each (v, w) ∈ E
        Relax(v,w)
```

Dijkstra's Algorithm

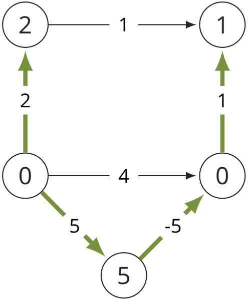
Properties

- Runtime: $\mathcal{O}(m \log n)$ with binary heaps and $\mathcal{O}(n \log n + m)$ with Fibonacci-Heaps
- Invariant: For all vertices in S, $\mathrm{dist}(s)$ is optimal.
- Requirement: No negative edges. The algorithm assumes that distances are always increasing.

What happens if there are negative edges?



Dijkstra



Dijkstra

All Pairs Shortest Path

Floyd-Warshall

Idea

- Assume that we know, for all i and j, the shortest path from v_i to v_j using only the (additional) vertices $\langle v_1, v_2, \ldots, v_{k-1}\rangle$. Let $d_{ij}^{(k-1)}$ be this distance.
- Then, we can add v_k in the next iteration and get

$$d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$$

- If $k = n$, then $d_{ij}^{(k)} = d(v_i, v_j)$ for all i and j.
- Initial values

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ \omega(v_i v_j) & \text{if } v_i v_j \in E \\ \infty & \text{else} \end{cases}$$

Floyd-Warshall

For Each pair i, j with $1 \leq i, j \leq |V|$
    Set $d_{ij}^{(0)} := 0$ if $i = j$, $\omega(v_i v_j)$ if $v_i v_j \in E$, and ∞ otherwise.
For $k := 1$ **To** $|V|$
    For Each pair i, j with $1 \leq i, j \leq |V|$
        Set $d_{ij}^{(k)} = \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\}$.

To represent d_{ij}, use two $n \times n$ arrays.

Detecting Negative Cycles

- Check if, for some i and some k, $d_{ii}^{(k)} < 0$.

Runtime: $\mathcal{O}(n^3)$

Dijkstra vs. Floyd-Warshall

Runtime for APSP

- Dijkstra: $\mathcal{O}(n^2 \log n + nm)$
- Floyd-Warshall: $\mathcal{O}(n^3)$

Observation

- Since $m \leq n^2$, Dijkstra would be better, especially for sparse graphs.
- Problem: negative weight edges.

Question

- Is there a way to avoid these negative edges?

Johnson's Algorithm

Algorithm

- Add a new vertex q and add, for each $v \in V$, the directed edge qv with weight 0.
- Run Bellman-Ford with start vertex q. Let $h(v)$ be the length of a shortest path from q to v.
- For each edge uv, set $\tilde{\omega}(uv) := \omega(uv) + h(u) - h(v)$.
- Remove q and run Dijkstra's algorithm on each vertex using $\tilde{\omega}$ as edge weights.

Properties

- Runtime $\mathcal{O}(n^2 \log n + nm)$
- Works with negative weight edges and can detect negative cycles.

A* and Branch and Bound

Single Pair Shortest Path

Single Pair Shortest Path

- Weighted graph
- Find shortest path from s to t.

Dijkstra

- Explores all in distance $d(s, t)$ before terminating.
  (Can be improved to $d(s, t)/2$ with bidirectional search)
- Next vertex is selected by distance from s.

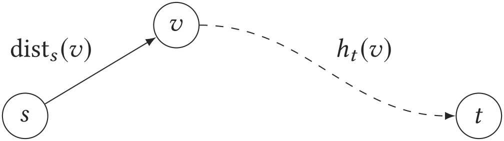
Problem

- Some vertices go in the wrong direction.

A*

Idea

- For a vertex v, make an estimation $h_t(v)$ of $d(v, t)$
- Important: $h_t(v) \leq d(v, t)$



Algorithm

- Basically Dijkstra
- For next iteration, pick vertex v for which $\mathrm{dist}_s(v) + h_t(v)$ is minimal.

Generalised A*

Idea

- Take decision tree.
- Find a shortest path from root to leaf.
- Important: Do not construct whole tree. Only construct explored parts.

Branch and Bound

- Start at root.
- Branch: Determine the children of a node.
- Bound: Compute for every node a lower bound for the cost of the solutions in this subtree.
- Select next node where estimated lower bound is minimal.

Note

- Finding the optimal lower bound (i. e., $h_t(v) = d(v, t)$) is as hard as solving the original problem.