All-Pairs Shortest Paths

Jared Saia
University of New Mexico

Outline

- All-pairs shortest paths by repeated SSSP
- Choosing subproblems for dynamic programming
- The Floyd–Warshall algorithm

All-Pairs Shortest Paths

- Given a directed graph with n vertices and m edges, compute $dist(u, v)$ for every ordered pair of vertices.
- $dist(u, v)$ is the minimum weight of a path from u to v.
- Edge weights may be negative; assume there are no negative-weight cycles.
- We will compute distances. Path reconstruction is an extension: $pred(u, v)$ records the vertex just before v on a chosen shortest path.

Distance Conventions

- $dist(v, v) = 0$: the empty path has weight zero.
- If no path leads from u to v, then $dist(u, v) = \infty$.
- If a shortest path consists of the edge (u, v), then $dist(u, v) = w(u, v)$.
- For formulas, let $w(u, v) = \infty$ when no edge (u, v) exists. The minimum of an empty set is ∞.
- Set $pred(u, v) =$ None when $u = v$ or v is unreachable from u.

The Distance Matrix

- Store the n^2 distances in an $n \times n$ matrix.
- Row u gives distances from u; column v gives distances to v.
- A road-map distance matrix works the same way: look up the Albuquerque row and the Ruidoso column.
- In a directed graph, $dist(u, v)$ need not equal $dist(v, u)$.

Lots of Single Sources

- Run an SSSP algorithm n times, once from each source s.
- Put the resulting distances in row $dist(s, *)$.
- Use Dijkstra for nonnegative weights, or Bellman–Ford when negative edges are allowed.
- Call this algorithm ObviousAPSP.

ObviousAPSP

ObviousAPSP(G)

  for each vertex $s \in V$ **do**

    $dist(s, *) \leftarrow \mathsf{SSSP}(G, s)$

  return $dist$

Repeated SSSP: Runtime

- Bellman–Ford from each source takes

$$O\Big(n(n+nm)\Big) = O(n^2 + n^2 m).$$

  For $m \geq 1$, this is $O(n^2 m)$; for dense graphs, $O(n^4)$.
- For nonnegative weights, Dijkstra with a Fibonacci heap takes

$$\boxed{O(nm + n^2 \log n)}.$$

  For dense graphs, this is $O(n^3)$.

Our Goal

- Compute all-pairs distances in $O(n^3)$ time, allowing negative edges but no negative-weight cycles.
- Floyd–Warshall achieves this using dynamic programming.
- Johnson's algorithm is another approach, faster on sparse graphs; we will not cover it here.

Dynamic Programming

- Recall: recursion + memoization avoids solving the same subproblem repeatedly.
- We first need a recurrence in terms of smaller subproblems.
- Shortest-path distances satisfy these equations:

$$dist(u,v) = \begin{cases} 0 & \text{if } u = v, \\ \min\limits_{x\,:\,x \to v \in E} \Big(dist(u,x) + w(x,v) \Big) & \text{otherwise.} \end{cases}$$

The Circular Dependency

- Try each predecessor x: find a shortest path from u to x, then add the last edge $x \to v$.
- These equations do not give a well-founded recursive algorithm.
- A call for $dist(u, v)$ can depend on $dist(u, x)$, which can in turn depend on $dist(u, v)$.
- Memoization alone does not break this cycle: neither call has finished computing its answer.

The Solution

- Add a parameter that decreases with each recursive call.
- Define $dist(u, v, k)$ to be the minimum weight of a path from u to v using *at most* k edges.
- Without negative-weight cycles, a shortest path can be chosen simple, so it uses at most $n - 1$ edges.
- The answer is $dist(u, v, n - 1)$.

Edge-Hop Recurrence

Base cases: $dist(u, u, k) = 0$ and $dist(u, v, 0) = \infty$ for $u \neq v$.

For $u \neq v$ and $k \geq 1$,

$$dist(u, v, k) = \min_{x\,:x \to v \in E} \Big(dist(u, x, k-1) + w(x, v) \Big).$$

Remove the last edge (x, v): the remaining path uses at most $k - 1$ edges.

Edge-Hop DP: Runtime

- Fill the states in increasing order of k.
- There are n^2 vertex pairs and n choices of k: $\Theta(n^3)$ states.
- Each non-base state checks all n possible predecessors x: $\Theta(n)$ work per state.
- Total time:

$$\Theta(n^3) \cdot \Theta(n) = \boxed{\Theta(n^4)}.$$

A Better Parameter

- The edge-hop DP takes $\Theta(n^4)$ time, matching the dense-graph bound for repeated Bellman–Ford.
- To reach $\Theta(n^3)$ time, we will use a different third parameter in $dist$.
- Instead of bounding the number of edges, restrict which vertices may appear inside the path.

Floyd–Warshall

- Number the vertices from 1 to n.
- Define $dist(u, v, r)$ to be the minimum weight of a path from u to v whose intermediate vertices are numbered at most r.
- Intermediate vertices exclude the two endpoints.
- For $r = 0$, only the empty path or a single edge is allowed.
- The unrestricted answer is $dist(u, v, n)$.

The Floyd–Warshall Recurrence

Initialize

$$dist(u,v,0) = \begin{cases} 0 & \text{if } u = v, \\ w(u,v) & \text{if } u \neq v \text{ and } (u,v) \in E, \\ \infty & \text{otherwise.} \end{cases}$$

For $r \geq 1$, either avoid r or use it as an intermediate vertex:

$$dist(u,v,r) = \min \left\{ \begin{array}{l} dist(u,v,r-1), \\ dist(u,r,r-1) + dist(r,v,r-1) \end{array} \right\}.$$

Floyd–Warshall: Correctness

Theorem: The recurrence computes the stated distances.

Proof. Induct on r, the largest allowed intermediate label.

Base case ($r = 0$): Only an empty path or a single edge is allowed, exactly as in the initialization.

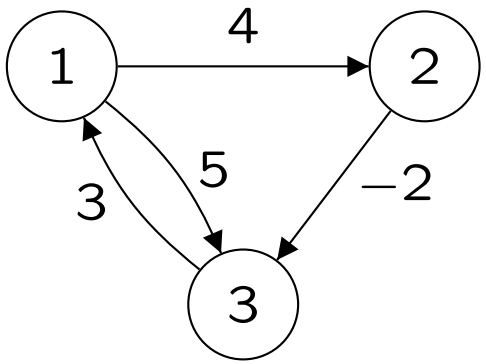
Inductive step: Assume all entries for $r - 1$ are correct (IH). Consider a shortest path from u to v whose intermediate vertices have labels at most r.

If the path avoids r as an intermediate vertex, then by IH the first argument of the minimum equals the shortest-path length.

If the path traverses r, split it at r. Both subpaths have intermediate labels at most $r - 1$. By IH, their $dist$ values equal their shortest-path lengths under that restriction. Thus the second argument equals the shortest-path length.

This establishes the inductive step. □

Floyd–Warshall: Example



Negative edge; no negative-weight cycle.

Each matrix gives $dist(u, v, r)$, with rows u and columns v ordered $1, 2, 3$. **Blue** entries improve in that round.

$r = 0$

$$\begin{bmatrix} 0 & 4 & 5 \\ \infty & 0 & -2 \\ 3 & \infty & 0 \end{bmatrix}$$

$r = 1$

$$\begin{bmatrix} 0 & 4 & 5 \\ \infty & 0 & -2 \\ 3 & \mathbf{7} & 0 \end{bmatrix}$$

$r = 2$

$$\begin{bmatrix} 0 & 4 & \mathbf{2} \\ \infty & 0 & -2 \\ 3 & 7 & 0 \end{bmatrix}$$

$r = 3$

$$\begin{bmatrix} 0 & 4 & 2 \\ \mathbf{1} & 0 & -2 \\ 3 & 7 & 0 \end{bmatrix}$$

The Floyd–Warshall Algorithm

FloydWarshall(G)

for each $u, v \in V$ **do**

$dist(u, v, 0) \leftarrow \infty$

for each $u \in V$ **do**

$dist(u, u, 0) \leftarrow 0$

for each $(u, v) \in E$ **do**

$dist(u, v, 0) \leftarrow \min\{dist(u, v, 0), w(u, v)\}$

for $r \leftarrow 1$ **to** n **do**

for each $u, v \in V$ **do**

$dist(u, v, r) \leftarrow \min\{dist(u, v, r-1),$
$dist(u, r, r-1) + dist(r, v, r-1)\}$

return $dist(*, *, n)$

Floyd–Warshall: Analysis

- There are n^2 vertex pairs and $n+1$ layers: $\Theta(n^3)$ states.
- Each non-base state takes $\Theta(1)$ work: one addition and one minimum.
- Total time is $\boxed{\Theta(n^3)}$.
- Storing all layers of $dist(u, v, r)$ takes $\Theta(n^3)$ space.

Take Away

- Floyd–Warshall computes all-pairs distances in $\Theta(n^3)$ time, allowing negative edges but no negative-weight cycles.
- Adding a parameter can make recursive dependencies decrease toward a base case.
- The parameter matters: bounding edges gives $\Theta(n^4)$ time; bounding intermediate labels gives $\Theta(n^3)$.