

Bellman-Ford Algorithm

Computes single source shortest paths even when some edges have negative weight.

The algorithm detects if there are negative cycles reachable from s .

If there are no such negative cycles, it returns the shortest paths.

The algorithm has two parts:

Part 1: Computing Shortest Paths Tree:

$|V| - 1$ iterations, iteration i computes the shortest path from s using paths of up to i edges.

Part 2: Checking for Negative Cycles.

Bellman-Ford (G, w, s)

1. For all $v \in V$ do
 - (a) $d[v] \leftarrow \infty$;
 - (b) $\pi[v] \leftarrow NIL$;
2. $d[s] = 0$;
3. For $i \leftarrow 1$ to $|V| - 1$ do
 - (a) For all $(u, v) \in E$ do
 - i. If $d[v] > d[u] + w(u, v)$ then
 - A. $d[v] \leftarrow d[u] + w(u, v)$;
 - B. $\pi[v] \leftarrow u$;
4. For all $(u, v) \in E$ do
 - (a) If $d[v] > d[u] + w(u, v)$ then return FALSE;
5. return TRUE

Run Time

Theorem 1. *The run time of the algorithm is $O(V \times E)$.*

Proof.

The initialization (1) takes $O(V)$.

The path creation (3) takes $O(V \times E)$.

The negative cycle detection (4) takes $O(E)$. \square

Correctness

Theorem 2. *Assuming that G contains no negative cycles reachable from s , the algorithm computes the shortest paths for all vertices of G .*

Proof. Fix a vertex $u \in V$, we prove that the algorithm computes a shortest path from s to u .

Let $P = v_0, v_1, \dots, v_k$, where $v_0 = s$ and $v_k = u$ be a shortest path from s to u .

Since there are no negative cycles P is a simple path, $k \leq |V| - 1$.

We prove by induction on i that after the i -th iteration of the (3) loop, the algorithm computed the shortest path for v_i .

The hypothesis holds for $v_0 = s$.

Assume that it holds for $j \leq i - 1$. After the i -th iteration

$$d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$$

which is the shortest path from s to v_i , since P is a shortest path from s to v_i , and the RHS is the distance between s to v_i on that path.

□

Theorem 3. *The algorithm returns TRUE if there are no negative cycles reachable from s , otherwise it returns FALSE.*

Proof. Assume that there are no negative cycles reachable from s , then by the previous theorem, the algorithm returns a shortest path tree, and $d[v]$ is the weight of the shortest path to s .

Thus, all inequalities in 4.1 don't hold.

Assume that there is a negative weight cycle v_0, \dots, v_k reachable from s ($v_0 = v_k$).

Since the path is reachable from s the values $d[v_i]$ are defined.

$$\sum_{i=1}^k d[v_{i-1}] = \sum_{i=1}^k d[v_i] \text{ and}$$

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0.$$

Thus,

$$\sum_{i=1}^k d[v_i] > \sum_{i=1}^k d[v_{i-1}] + \sum_{i=1}^k w(v_{i-1}, v_i)$$

So there must be an i such that

$$d[v_i] > d[v_{i-1}] + w(v_{i-1}, v_i)$$

and the algorithm returns FALSE. \square

All-Pairs Shortest Paths

Single-Source Shortest Paths:

Compute shortest paths from a given source to all vertices in the graph.

All-Pairs Shortest Paths:

Given a graph $G = (V, E)$, $|V| = n$, and a weight function w on the edges, compute the shortest paths between all pairs of vertices.

The problem is not well defined in the presence of a **negative weight cycle** in the graph.

All-Pairs Shortest Path Algorithms

We can solve an all-pairs shortest-paths problem by running a single source shortest-paths algorithm n times, once for each vertex as a source. Then the running time is:

- $O(n^2 \lg n + n|E|)$ if we use the Dijkstra algorithm (assuming no negative edge weights);
- $O(n^2|E|)$ if we use Bellman-Ford algorithm - i.e., $O(n^4)$ if the graph is dense.

Direct approach: First we will give an $O(n^4)$ algorithm, then improve it to $O(n^3 \log n)$. Later we will give a even better algorithm running in $O(n^3)$ time.

All Pairs Shortest Paths

Input: an adjacency matrix W where $w_{i,j}$ is the weight of the edge (i, j) :

$$w_{i,j} = \begin{cases} 0 & \text{if } i = j \\ \text{weight of } (i, j) & \text{if } i \neq j \text{ and } (i, j) \in E \\ \infty & \text{if } (i, j) \notin E \end{cases}$$

Output:

A matrix D where $d_{i,j}$ is the shortest path from i to j .

The Idea

Define $d_{i,j}^{(m)}$ to be the shortest path between i and j using paths of up to m edges. When $m = 0$, we have

$$d_{ij}^{(0)} = \begin{cases} 0 & : i = j \\ \infty & : i \neq j \end{cases}$$

Recursively we define,

$$d_{i,j}^{(m)} = \min_{1 \leq k \leq n} [d_{i,k}^{(m-1)} + w_{k,j}].$$

If there are no negative weight cycles then no shortest path has more than $n - 1$ edges.

Computing the Matrices

Define a sequence of matrices $D^{(1)}, D^{(2)}, \dots, D^{(n-1)}$, where for $m = 1, 2, \dots, n-1$, we have $D^{(m)} = \left(d_{ij}^{(m)} \right)$.

Note that $D^{(1)} = W$.

The key procedure (called EXTEND-SHORTEST-PATHS) is to compute the matrix $D^{(m)}$ given $D^{(m-1)}$ and W : extending the shortest paths computed so far by one more edge.

EXTEND-SHORTEST-PATHS($D^{(m-1)}, W$)

for $i = 1$ to n

for $j = 1$ to n

$$d_{ij}^{(m)} = \infty$$

for $k = 1$ to n

$$d_{ij}^{(m)} = \min(d_{ij}^{(m)}, d_{ik}^{(m-1)} + w_{kj})$$

Shortest Paths and Matrix Multiplication

Let $C = A \cdot B$ be the product of two $n \times n$ matrices A and B . For $i, j = 1, 2, \dots, n$, we compute

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

Note that by substituting,

$$\begin{aligned}d^{(m-1)} &\rightarrow a \\w &\rightarrow b \\d^{(m)} &\rightarrow c \\min &\rightarrow + \\+ &\rightarrow \cdot\end{aligned}$$

in

$$d_{i,j}^{(m)} = \min_{1 \leq k \leq n} [d_{i,k}^{(m-1)} + w_{k,j}]$$

we get matrix multiplication.

Slow All-Pairs Algorithm

Theorem 4. For $n \times n$ matrices D and W , the procedure *EXTEND-SHORTEST-PATHS*($D^{(m-1)}, W$) takes $\Theta(n^3)$ steps.

Theorem 5. There is an algorithm that computes the correct shortest paths and terminates in $\Theta(n^4)$ steps.

Repeated Squaring

Let $D' = D \cdot W$, where the operation \cdot is the output of the procedure EXTEND-SHORTEST-PATHS(D, W).

Like the “product” operation our “ \cdot ” operation is **associative**, i.e.

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C$$

Assume that $n - 1 = 2^k$. A faster method for computing D^{n-1} is:

For $t = 1$ to k do

$$D^{2^t} = D^{2^{t-1}} \cdot D^{2^{t-1}}$$

Fast All-Pairs Shortest Paths Algorithm

We need to compute $D^{(m)}$ for some $m \geq n - 1$.

Let $m = 2^{\lceil \log_2(n-1) \rceil}$

Theorem 6. *There is an algorithm that computes the correct shortest-path distances in $\Theta(n^3 \log n)$ steps.*

The Floyd-Warshall algorithm

For a path $P = v_1, v_2, \dots, v_{k-1}, v_k$, the edges v_2, \dots, v_{k-1} are intermediate edges.

Let $V = \{1, \dots, n\}$.

In iteration k , the algorithm computes all pairs shortest paths with intermediate vertices in $\{1, \dots, k\}$.

In iteration k , the algorithm computes all pairs shortest paths with intermediate vertices in $\{1, \dots, k\}$.

Let $d_{i,j}^{(k)}$ = the distance of a shortest path from i to j using only vertices of $\{1, \dots, k\}$.

Lemma 1.

$$d_{i,j}^{(k)} = \begin{cases} w_{i,j} & \text{if } k = 0 \\ \text{MIN} [d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}] & \text{if } k \geq 1 \end{cases}$$

Proof. Let P be a shortest path from i to j using vertices in $\{1, \dots, k\}$.

If P does not use k then $d_{i,j}^{(k)} = d_{i,j}^{(k-1)}$.

Otherwise P consists of a path P_1 from i to k , followed by a path P_2 from k to j .

P_1 is a shortest path from i to k in $\{1, \dots, k-1\}$ and P_2 is a shortest path from k to j in $\{1, \dots, k-1\}$. \square

Theorem 7. *The run-time of the Floyd-Warshall algorithm is $O(n^3)$ steps.*

Transitive Closure

Given a directed graph G , the **transitive closure** of G is a directed graph $G^* = (V, E^*)$, where

$$E^* = \{(i, j) \mid \text{there is a path from } i \text{ to } j \text{ in } G\}.$$

Given G , we can compute G^* by computing all pairs shortest paths with all edges having weight 1.

More efficiently:

Let

$$t_{i,j}^{(0)} = \begin{cases} 0 & \text{if } i \neq j \text{ and } (i, j) \notin E \\ 1 & \text{if } i = j \text{ or } (i, j) \in E \end{cases}$$

For $k \geq 1$

$$t_{i,j}^{(k)} = t_{i,j}^{(k-1)} \vee (t_{i,k}^{(k-1)} \wedge t_{k,j}^{(k-1)}).$$