

Chapter 8 : Shortest Path Problem - Bellman-Ford Algorithm

Omar Saadi
omar.saadi@um6p.ma

UM6P - College of Computing

1st year of the engineering degree of UM6P-CS
Academic year 2024-2025

Outline

- 1 Introduction
- 2 Bellman-Ford Algorithm

Outline

1 Introduction

2 Bellman-Ford Algorithm

Framework

Definition

A **negative cycle** on a graph (or digraph) is a cycle such that the sum of the weights of its edges is negative (i.e. < 0).

- We have a graph (or digraph) $G = (V, E)$.
- Each edge $(u, v) \in E$ has a weight $w(u, v)$ (that can be negative), but **G has no negative cycles.**
- We have a starting vertex s and a destination vertex d .

We want to solve the shortest path problem :

$$\underset{P \text{ a } s,d\text{-path in } G}{\text{minimize}} \quad w(P),$$

where $w(P)$ is the weight of the path P given as the sum of the weights of the edges forming it.

Outline

1 Introduction

2 Bellman-Ford Algorithm

Bellman-Ford Algorithm

Input : A graph (or digraph) with no negative cycles and a starting vertex s .

Output : Distance $d(s, u)$ from s to each other vertex u and a shortest path tree given by identifying the parent $P(u)$ of each vertex u .

Key idea : $\forall u \in V$, the algorithm computes an estimate $d[u]$ of the distance of u from the source s such that :

- At iteration k , $d[u]$ is the length of a path from s to u . The estimate $d[u]$ is non-increasing and it is updated in a dynamic programming way (a step by step way).
- At the last iteration $n - 1$, $d[u]$ will contain $d(s, u)$.

Bellman-Ford Algorithm

Initialization : Set $d[s] = 0$; $P(s) = \text{NONE}$. For $u \neq s$, set $d[u] = +\infty$ and $P(u) = \text{NONE}$.

Iteration :

for i from 1 to $n - 1$ **do** :

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

$d[v] = \min\{d[v], d[u] + w(u, v)\}$, and **if** $d[v]$ changes **then**
 $P[v] = u$.

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

if $d[v] > d[u] + w(u, v)$ **then**

return "A negative cycle exists"

Return $d[u]$ and $P[u]$ for all $u \in V$.

Remark : The order in which the edges are considered impacts the execution of the algorithm. A possible order of edges is to order the vertices and then take the outgoing edges of each one of them.

Bellman-Ford detects negative cycles

Proposition

If there is a negative cycle reachable from the source s , then at the end we will have for some edge $(u, v) \in E$, $d(v) > d(u) + w(u, v)$.

Proof

Suppose $v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k$ is a negative cycle reachable from s , where $v_0 = v_k$, i.e. $\sum_{i=1}^k w(v_{i-1}, v_i) < 0$.

By absurd, suppose that at the end we have $d(v_i) \leq d(v_{i-1}) + w(v_{i-1}, v_i)$ for all $i = 1, \dots, k$. Then taking the sum, we get

$$\sum_{i=1}^k d(v_i) \leq \sum_{i=1}^k d(v_{i-1}) + \sum_{i=1}^k w(v_{i-1}, v_i).$$

Observing that the first two terms are the same (because $v_0 = v_k$), we deduce that :

$$\sum_{i=1}^k w(v_{i-1}, v_i) \geq 0. \quad (\text{which is absurd})$$

To prove correctness of the distance estimates (at the end), we reformulate the algorithm in a dynamic programming way. This formulation is useful for the proof but not for space (and time) complexity :

Bellman-Ford Algorithm (Variant 1)

Initialization : For all $k \in \{0, \dots, n-1\}$, set $d_k[s] = 0$; $P_k(s) = \text{NONE}$ and for all $u \neq s$, set $d_k[u] = +\infty$ and $P_k(u) = \text{NONE}$.

Iteration :

for k from 1 to $n-1$ **do** :

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

$d_k[v] = \min\{d_{k-1}[v], d_{k-1}[u] + w(u, v), d_k[v]\}$, and **if** $d_k[v]$ changes to $d_{k-1}[v]$ **then** $P_k[v] = P_{k-1}[v]$, and **if** $d_k[v]$ changes to $d_{k-1}[u] + w(u, v)$ **then** $P_k[v] = u$.

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

if $d_{n-1}[v] > d_{n-1}[u] + w(u, v)$ **then**
 return "A negative cycle exists"

Return $d_{n-1}[u]$ and $P_{n-1}[u]$ for all $u \in V$.

*Note that this variant 1 of the algorithm is slower (and takes more storage space). Indeed, to have the same iteration as the original version we need to take $d_k[v] = \min\{d_{k-1}[v], d_{k-1}[u] + w(u, v), d_k[v], d_k[u] + w(u, v)\}$.

\Rightarrow Proving that this variant 1 is correct proves the original version is correct too.

Proposition

If the graph G has no negative cycles, then $d_{n-1}[v] = d(s, v)$ for all vertices v , and backtracking from v to s using the parent list P_{n-1} yields a shortest path from s to v .

Proof

By induction on k , we will prove that $d_k[v]$ is the minimum weight of a path from s to v that uses $\leq k$ edges. This will show that $d_{n-1}[v]$ is the distance from s to v because there is no negative cycles in the graph (a shortest path contains at most $n - 1$ edges).

Base case : If $k = 0$, then $d_k[v] = 0$ for $v = s$, and $+\infty$ otherwise. So the claim is satisfied.

Inductive step : Suppose that for all vertices u , $d_{k-1}[u]$ is the minimum weight of a path from s to u that uses $\leq k - 1$ edges.

If $v \neq s$, let P be a shortest simple path from s to v with $\leq k$ edges, and let u be the node just before v on P . Let Q be the path from s to u . Then Q is a shortest path from s to u that uses at most $k - 1$ edges. By the inductive hypothesis, $w(Q) = d_{k-1}[u]$.

Proof (Cont.)

In iteration k , we update $d_k[v] = \min(d_{k-1}[v], d_{k-1}[u] + w(u, v))$.

We know that $d_k[v] \leq d_{k-1}[u] + w(u, v) = w(Q) + w(u, v) = w(P)$, i.e. $d_k[v] \leq w(P)$.

Furthermore, $d_k[v]$ is the length of a path from s to v with at most k edges, which must be at least as large as $w(P)$.

Therefore, $d_k[v] = w(P)$ is the minimum weight of a path from s to v that uses at most k edges.

Note that the update of the parents $P_k(y)$ insures that the parent list P_k contains the parents in the shortest paths using at most k edges. So that, P_{n-1} contains the parents in the shortest paths from s to v for every $v \in V$.

Complexity

Time complexity : We have the $n - 1$ iterations and in each iteration we go through all the edges in our graph. For each edge $(u, v) \in E$, we do $O(1)$ operations. Therefore the time complexity of the algorithm is :

$$O(|V||E|).$$

Space complexity : The algorithm (in its original version) uses only the two lists d and P . Therefore, the space complexity is $O(|V|)$.

Remark : If at a given iteration k no distance $d[u]$ is changed, then the algorithm can be stopped and the distance and shortest paths are found.

Example

In the following digraph, find the distances from vertex A to all the other vertices and the list of parents providing the shortest paths.

