1 Dijkstra with negative weights

The main reason why Dijkstra’s algorithm works for graphs with nonegative edges is that at any step the distance to the closest unprocessed node to the source $s$ cannot decrease any further. That is why the node $v \in V \setminus S$ with the minimum $d[v]$ is chosen and all outgoing edges $(v,u)$ are relaxed ($V$ is the set of nodes and $S$ is the set of processed nodes). However with negative edges this no longer holds. Below is an example for which Dijkstra’s algorithm does not work.

![Figure 1: An example of a graph where Dijkstra does not work.](image)

Running Dijkstra on the above algorithm starting from vertex $A$, with $d[A] = 0$ and the others infinity, we will relax edges $(A,B), (A,C), (A,D)$ and we will have $d[B] = 1, d[C] = 0, d[D] = 99$. Next up we process node $C$ (the closest unprocessed from $A$), which has no outgoing edges. Next we process node $B$, which has one outgoing edge to $C$ but the path through $B$ is does not improve the path to $C$, so we still have $d[C] = 0, d[B] = 1$. Next we process node $D$ and now we relax edge $(D,B)$ which decreases $d[B]$ to $-201$. Note that $B$ was already processed and we have processed all nodes. However, at the end of Dijkstra’s algorithm we have $d[C] = 0$ which is not optimal.

2 Proof of Bellman Ford Correctness

Bellman Ford solves the problems that Dijkstra’s algorithm has with negative weight edges, by employing a different relaxation ordering on the edges. An pseudocode for Bellman Ford’s algorithm can be found below.

for $v \in V: d[v] \leftarrow \infty$
Lemma 1 Upper Bound Property We always have \( d[v] \geq \delta(s, v) \) and if we ever find \( d[v] = \delta(s, v) \), \( d[v] \) never changes.

Proof. We show that \( d[v] \geq \delta(s, v) \) by induction on the number of relaxation steps \( k \).

Base Case: For \( k = 0 \), we have \( d[s] = 0 \) and \( d[v] = \infty \) for all \( v \in \{V \setminus s\} \). If there is no negative cycle, then \( \delta(s, s) = 0 \); otherwise \( \delta(s, s) = -\infty \) so \( d[s] \geq \delta(s, s) \).

Induction Step: Assume after \( k - 1 \) iterations, \( d[v] \geq \delta(s, v) \) for all \( v \in V \). Now consider the \( k \)th relaxation on edge \((u, v)\). Firstly, for \( w \in \{V \setminus \{v\}\}, d[w] \) does not change so by induction \( d[w] \geq \delta(s, w) \). Now consider we set \( d[v] = d[u] + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v) \) (otherwise \( \delta(s, v) \) would not be the shortest path from \( s \) to \( v \)).

If we ever achieve \( d[v] = \delta(s, v) \), we cannot decrease \( d[v] \) since we have just shown that \( d[v] \geq \delta(s, v) \). Moreover, we cannot increase \( d[v] \) because we only change \( d[v] \) during relaxations and relaxations cannot increase \( d[v] \). Therefore, once we have \( d[v] = \delta(s, v) \) it cannot change.

Lemma 2 Path Relaxation

Assume we have a graph \( G \) with no negative cycles. Let \( p = < v_0, v_1, ..., v_j > \) be a shortest path from \( v_0 \) to \( v_j \). Any sequence of edge relaxations that includes in order the relaxations of \( (v_0, v_1), (v_1, v_2), ..., (v_{j-1}, v_j) \) produces \( d[v_j] = \delta(v_0, v_j) \) after all these relaxations and at all time afterwards. Note that this property holds regardless of what other relaxation calls are made before, during, or after these relaxations.

Proof. We proceed by induction on the \( k \)th vertex in \( p \), showing that after relaxations \((v_0, v_1), ..., (v_{k-1}, v_k)\) have occurred that \( d[v_k] = \delta(v_0, v_k) \).

Base Case: \( k = 0 \). After initialization we have that \( d[v_0] = 0 = \delta(v_0, v_0) \).

Induction Step: Assume that we have relaxed, in order, edges \((v_0, v_1), ..., (v_{k-2}, v_{k-1})\) and \( d[v_{k-1}] = \delta(v_0, v_{k-1}) \). Then eventually we will make a relaxation call to \((v_{k-1}, v_k)\) since we assume this call happens at least once after the call to \((v_{k-2}, v_{k-1})\). By the upper bound property, at the time of this call \( d[v_k] \geq \delta(v_0, v_k) \). In addition, we have that \( \delta(v_0, v_k) = \delta(v_0, v_{k-1}) + w(v_{k-1}, v_k) \) because \( p \) is a shortest path. Therefore \( d[v_k] \geq d[v_{k-1}] + w(v_{k-1}, v_k) \) and after this relaxation call we will have \( d[v_k] = d[v_{k-1}] + w(v_{k-1}, v_k) = \delta(v_0, v_k) \). By the upper bound property this is maintained ever after.

Lemma 3 Bellman Ford Correctness 1: Assume we have a graph \( G \) with no negative cycles reachable by \( s \). Then after running Bellman Ford we have \( d[v] = \delta(s, v) \) for all \( v \in V \) reachable from \( s \).
Proof. If \( v \in V \) is reachable from \( s \) then there must exist an acyclic shortest path \( p = < s, v_1, ..., v_j > \) where \( v_j = v \). Now, since \( p \) is acyclic, \( p \) can contain no more than \( V \) vertices and therefore no more than \( |V| - 1 \) edges. At every step of the Bellman Ford algorithm we relax every edge. Therefore we surely relax \( (s, v_1) \) on the first iteration, \( (v_1, v_2) \) on the second iteration and so on (we also relax \( (v_1, v_2) \) on the first iteration but we cannot guarantee that it is relaxed after the relaxation of \( (s, v_1) \)). By the time we have reached the \( |V| - 1 \) iteration, we must have relaxed every edge in \( p \) in order. Therefore, by the path relaxation property, we have \( d[v] = \delta(s, v) \) after \( |V| - 1 \) all edge relaxation steps.

Corollary 4 Bellman Ford Correctness 2: For vertex \( v \in V \), Bellman Ford terminates with \( d[v] = \infty \) if and only if \( v \) is not reachable from \( s \).

Proof. Let \( d[v] = \infty \) and assume \( v \) is reachable from \( s \). Then from Bellman Ford correctness 1 we have that \( d[v] = \delta(s, v) = \infty \) which is a contradiction.

Assume \( v \) is not reachable from \( s \). By the upper bound property, we have that \( d[v] \geq \delta(s, v) \) at all times. Since \( \delta(s, v) = \infty \) we must have that \( d[v] = \infty \) at all times.

Theorem 5 Correctness of Bellman Ford: If \( G \) contains no negative cycles reachable from \( s \), the algorithm finds no negative cycles and \( d[v] = \delta(s, v) \) for all \( v \in V \). If \( G \) does contain a negative weight cycle, the algorithm will return that there is a negative cycle in the graph.

Proof. If \( G \) contains no negative weight cycles, by the lemmas above, \( d[v] = \delta(s, v) \) for all \( v \in V \) after the termination of the algorithm. Therefore \( d[v] = \delta(s, v) \leq \delta(s, w) + w(u, v) \leq d[u] + w(u, v) \) by the triangle inequality for all edges \( (u, v) \). Thus no edge can still be relaxed.

Now assume \( G \) contains a negative weight cycle that is reachable from \( s \). Let this cycle be \( c = < v_0, v_1, ..., v_j > \) with \( v_0 = v_j \), such that \( \sum_{i=0}^{j} w(v_{i-1}, w_i) < 0 \). We proceed by contradiction. Assume that the last check in Bellman Ford does not return a negative weight cycle. This means that for any edge \( (u, v) \) in the graph \( d[u] \leq d[v] + w(u, v) \). Summing these up for all edges in the cycle we get that \( \sum_{i=0}^{j} d[v_i] \leq \sum_{i=0}^{j-1} d[v_i] + w(v_i, v_{i+1}) \). But \( \sum_{i=0}^{j} d[v_i] = \sum_{i=0}^{j-1} d[v_i] \) since \( v_0 = v_j \), so we get that \( \sum_{i=0}^{j} w(v_i, v_{i+1}) \geq 0 \) which is a contradiction with the fact that the cycle has negative weight. Thus Bellman Ford does return that the graph has a negative weight cycle if such a cycle exists.

3 Running Times of SSP Algorithms

- **DAG**: For directed acyclic graphs we can construct the topological order of the nodes in the graph. Traversing all nodes in the topological order and relaxing all outgoing edges of the current vertex finds all shortest paths in the graph. The running time of this algorithm is \( O(|V| + |E|) \).

- **Non-negative weight edges**: For graphs with non-negative weight edges we can use Dijkstra’s algorithm to find all shortest paths from a source vertex. Running time is \( O(|V| \log(|V|) + |E| \log(|V|)) \), but does not work for.
• **Negative weight edges:** For graphs that might have negative weight edges we use Bellman Ford’s algorithm to find all shortest paths from a source vertex. Bellman Ford relaxes all $O(|E|)$ edges of a graph exactly $O(|V|)$ times, so the running time of the algorithm is $O(|V||E|)$. Bellman Ford can also detect if a graph has a negative weight cycle reachable from the source vertex or not.

4 Application of Bellman Ford: Solving a set of constraints

Assume you have a set of $N$ variables $x_1, x_2, \ldots, x_n$ and $M$ constraints of the form $x_j - x_i \leq b_{ij}$, where $b_{ij}$ can be negative. How can we find an assignment of the variables $x_i$ such that they satisfy the constraints?

We can observe that $x_j - x_i \leq b_{ij} \iff x_j \leq x_i + b_{ij}$. This looks very similar to Bellman Ford ending condition on every edge $d[j] \leq d[i] + w(i, j)$.

We will construct a graph with $N$ nodes and $M$ edges where each node $v_i$ corresponds to the variable $x_i$. For every constraint $x_j - x_i \leq b_{ij}$ we construct an edge from $v_i$ to $v_j$ with weight $w(v_i, v_j) = b_{ij}$. In addition we add a special node $v_0$ and we add an edge from $v_0$ to all other nodes. The weights of these edges are 0 (i.e. $w(v_0, v_i) = 0$). We then run Bellman Ford with the source node $v_0$.

If Bellman Ford ends and finds no negative cycles then $x_i = d[v_i]$ is a solution to the initial constraint problem. Proof: at the end of Bellman Ford for every edge $(v_i, v_j)$ the inequality $d[v_j] \leq d[v_i] + w(v_i, v_j) \iff d[v_j] - d[v_i] \leq w(v_i, v_j) \iff x_j - x_i \leq b_{ij}$ holds (where the last equivalence is by construction of the graph).

If Bellman Ford does find a negative cycle in the graph then there is no solution to the constraint problem. Proof: assume there is a solution which satisfies $x_i - x_j \leq b_{ij}$. Let the negative cycle be $< v_{i_0}, v_{i_1}, \ldots, v_{i_N} >$ with $i_0 = i_N$, where $i_k$ is an index between 1 and $N$. Then $\sum_{i=0}^{j-1} w(v_{i_k}, v_{i_{k+1}}) < 0 \iff \sum_{i=0}^{j-1} b_{i_k,i_{k+1}} < 0$ (by construction). But $\sum_{i=0}^{j-1} b_{i_k,i_{k+1}} \geq \sum_{i=0}^{j-1} x_{i_k} - x_{i_{k+1}} = 0$, where the equality holds because $i_0 = i_N$. Thus we get a contradiction. Thus if Bellman Ford does find a negative cycle in the graph then there is no solution to the constraint problem.

As an example for the problem assume the following constraints:

- $x_1 - x_2 \leq 0$
- $x_1 - x_5 \leq -1$
- $x_2 - x_5 \leq 1$
- $x_3 - x_1 \leq 5$
- $x_4 - x_1 \leq 4$
- $x_4 - x_3 \leq -1$
- $x_5 - x_3 \leq -3$
• $x_5 - x_4 \leq -3$

The corresponding constructed graph can be found below:

![Graph](image)

**Figure 2**: Constructed graph for the constrains.