Lecture 13 — Belmann-Ford and Dijkstra's Algorithm

The Shortest Path Problem — recap

The problem of finding a shortest path from O to D in a directed graph $G = (\mathcal{N}, \mathcal{A})$ (with arc weights w_{ij} associated with each arc $(i, j) \in \mathcal{A}$) is the problem of finding the directed path $(n_1, \ldots n_l)$ where $n_1 = O$ and $n_l = D$ which minimises the sum $\sum_{i=1}^{l-1} w_{n_i n_{i+1}}$. We will also find it useful to use the convetion that $w_{ij} = \infty$ if $(i, j) \notin \mathcal{A}$. Two methods in common use are Bellman-Ford and Dijkstra's Algorithm.

Dijkstra's Algorithm

Dijkstra's Algorithm was discovered by the pioneering mathematician and programmer E.W.Dijkstra (1930 – 2002). The algorithm finds the shortest path to all nodes from an origin node, on a graph $G = (\mathcal{N}, \mathcal{A})$. It requires that all arc weights are non-negative $\forall (i, j) \in \mathcal{A} : w_{ij} \geq 0$.

Dijkstra's Algorithm involves the labelling of a set of permanent nodes P and node distances D_j to each node $j \in \mathcal{N}$. Assume that we wish to find the shortest paths from node 1 to all nodes. Then we begin with: $P = \{1\}$ and $D_1 = 0$ and $D_j = w_1 j$ where $j \neq 1$. Dijkstra's algorithm then consists of following the procedure:

1. Find the next closest node. Find $i \notin P$ such that:
$$D_i = \min_{j \notin P} D_j$$
2. Update our set of permanently labelled nodes and our nodes distances:
$P := P \cup \{i\}$.
3. If all nodes in $\mathcal{N}$ are also in P then we have finished so stop here.
4. Update the temporary (distance) labels for the new node i. For all $j \notin P$
$$D_j := \min[D_j, w_{ij} + D_i]$$
5. Go to the beginning of the algorithm.

Note that the version of this algorithm is subtly different from that in Bertsekas & Gallager which finds the paths from all nodes to a single destination. (This is not an important detail and it should be obvious how to reverse the algorithm). To understand the algorithm we make the following claims:

Proposition 1. *At the beginning of each iteration of Dijkstra's algorithm then:*

1. *$D_i \leq D_j$ for all $i \in P$ and $j \notin P$.*
2. *D_j is, for all j, the shortest distance from 1 to j using paths with all nodes (except, possibly j) in P.*

If this proposition can be proved then we can see that, when P contains every node in $\mathcal{N}$ then all the D_j are shortest paths by the second part of this proposition. Therefore proving the above proposition is equivalent to proving that Dijkstra's algorithm finds shortest paths.

Proof. The proposition is trivially true at the first step since P consists only of the origin point (node 1) and D_j is 0 for $j = 1$, is $w_{ij} \geq 0$ for nodes reachable directly from node 1 and ∞ otherwise.

The first condition is simply shown to be satisfied since it is preserved by the formula:

$$D_j := \min[D_j, w_{ij} + D_i]$$

which is applied to all $j \notin P$ when node i is added to the set P.

We show the second condition by induction. We have established already that it is true at the very start of the algorithm. Let us assume it is true for the beginning of some iteration of the algorithm and show that it must then be true at the beginning of the next iteration.

Let node i be the node we are adding to our set P and let D_k be the label of each node k at the beginning of the step. The second condition must, therefore, hold for node $j = i$ (the new node we have added) by our induction hypothesis. It must also hold for all nodes $j \in P$ by part one of the proposition which is already proven. It remains to prove that the second condition of the proposition is met for $j \notin P \cup \{i\}$.

Consider a path from 1 to j which is shortest amongst those with all nodes except j in $P \cup \{i\}$ and let D'_j be the corresponding shortest distance. Such a path must contain a path from 1 to some node $r \in P \cup \{i\}$ and an $\text{arc}(r, j)$. We have already established that the length of the path from 1 to r must be D_r and therefore we have:

$$D'_j = \min_{r \in P \cup \{i\}} [D_r + w_{rj}] = \min[\min_{r \in P} [D_r + w_{rj}], D_i + w_{ij}]$$

However, by our hypothesis $D_j = \min_{r \in P}[D_r + w_{rj}]$ therefore, $D'_j = min[D_j, D_i + w_{ij}]$ which is exactly what is set by the fourth step of the algorithm. Thus, after any iteration of the algorithm, the second part of the propostion is true if it was true at the beginning of the iteration. Thus the proof by induction is complete.

□

The observant will have noticed that this provides the shortest distance to each node from the origin. To find the shortest path, simply work backwards from the destination, asking which node in step four the new node was added from.

Bellman-Ford Algorithm

Notation. *D_i^h is the distance of the shortest walk from node 1 to node i of h steps or less.*

Set initially $D_i^0 = \infty$ for $i \neq 1$ and $D_1^0 = 0$.

The Bellman-Ford Algorithm is then simply

$$D_i^{h+1} = \min_j [D_j^h + w_{ji}]$$

The algorithm terminates after h iterations if

$$D_i^h = D_i^{h-1} \qquad \forall i$$

Proposition 2. *The scalars D_i^h generate by the algorithm from the starting values given for D_i^0 are equal to the shortest walk of length $\leq h$ from node 1 to node i.*

Proof. The first iteration will clearly give us $D_i^1 = w_1 i$ for all i apart from $i = 1$ — which is, indeed, correct for the walk lengths of ≤ 1 from 1 to i. Let us suppose that D_i^k is the shortest walk of length $\leq k$ (from 1 to i) then complete the proof by induction by showing that D_i^{k+1} is the shortest walk of length $\leq k + 1$.

One possibility is that the shortest walk of length $\leq (k + 1)$ will be a walk of length k or less — in this case, $D_i^{k+1} = D_i^k$. Otherwise, the walk will be a walk of length $k + 1$ with the final arc being (j, i) added on to some walk of length k to node j. Thus, we can conclude that our shortest walk of length $k + 1$ is given by:

$$\text{Shortest walk to } i \text{ of length} \leq k+1 = \min\left[D_i^h, \min_j D_j^k + w_{ji}\right]$$

However, since a walk of length $\leq k + 1$ must always be shorter or equal to a walk of length $\leq k$ (since the former contains the latter) then this reduces to

$$\text{Shortest walk to } i \text{ of length} \leq k+1 = \min_j D_j^k + w_{ji}$$

which is our original expression for D_j^{k+1} and completes our proof by induction. □

Proposition 3. *The algorithm terminates after a finite number of iterations if, and only if, all cycles not containing node i have non negative length. Furthermore, if the algorithm terminates, it does so after at most $h \leq N$ iterations and, at termination, D_i^h is the shortest path length from 1 to i.*

Proof. If negative length cycles exist then such cycles could be repeated as many times as desired to reduce the shortest walk length and thus the algorithm could never converge. Conversely, if no negative length cycles exist then any walks containing cycles could be made shorter or kept the same length by deleting such a cycle. Thus, our shortest walks contain no cycles. The maximum length of a walk with no cycles is $N - 1$ (since such a walk will have covered every node). Thus, it trivially follows that $D_i^N = D_i^{N-1}$ and the algorithm terminates after, at most, N iterations. □

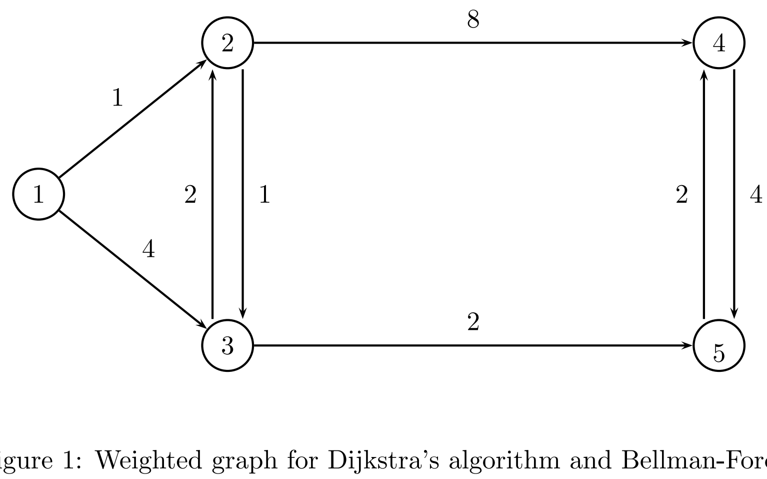

Figure 1: Weighted graph for Dijkstra's algorithm and Bellman-Ford