Johnson's all-pairs shortest paths algorithm

Vassos Hadzilacos

Throughout this document $G = (V, E)$ is a directed graph, $n = |V|$, $m = |E|$, and $\mathbf{wt} : E \to \mathbb{Z}$ is an edge weight function. Edge weights can be positive, negative, or zero.

Given an algorithm A that solves the single-source shortest paths problem we can solve the all-pairs shortest paths problem by running A n times, once with each node as the source. If A runs in $\Theta\big(f(m, n)\big)$ time, this takes $\Theta\big(nf(m, n)\big)$ time. In particular, if A is Dijkstra's algorithm, it takes $\Theta(nm \log n)$ time. (For simplicity, we assume here that $n = O(m)$, which is typically the case.) This is worse that the $\Theta(n^3)$ running time of the Floyd-Warshall algorithm if G is a dense graph, i.e., m is roughly n^2. But if G is a sparse graph, say $m = O(n)$ or even $O(n \log n)$, $\Theta(nm \log n)$ is faster than $\Theta(n^3)$.

Unfortunately, as we have seen, Dijkstra's algorithm does not work if edges can have negative weights. So the question arises: Can we re-weigh the edges in a way that

(a) makes all edge weights non-negative, and

(b) preserves shortest paths: a $u \to v$ path p is shortest under the original weight function $\mathbf{wt}$ if and only if p is shortest under the new weight function $\mathbf{wt'}$.

We have seen that a naive way to re-weigh the edges so as to satisfy (a), namely by adding the same amount to the weight of every edge so as to make them all non-negative, does not satisfy (b): it punishes disproportionately paths with many edges.

Johnson's algorithm involves a clever way of re-weighing the edges that achieves both (a) and (b), ***when that is possible***: If G has no negative-weight cycles under $\mathbf{wt}$, we can find (relatively quickly, using the Bellman-Ford algorithm) new weights for the edges that satisfy both (a) and (b). We can then run Dijkstra n times with these new weights, and obtain an all-pairs shortest paths algorithm that runs faster than the Floyd-Warshall algorithm, if the graph is sparse.

Let us first focus on goal (b). Suppose we assign weight x_u to each node u of G; for now, this is an arbitrary integer — it could be positive, negative, or zero. Having assigned these weights to the nodes, we can now define the new weight function $\mathbf{wt'}$ on the edges:

$$\mathbf{wt'}(u, v) = \mathbf{wt}(u, v) + x_u - x_v, \quad \text{for each } (u, v) \in E. \tag{1}$$

So, the weight of edge (u, v) increases (by the amount $x_u - x_v$) if $x_u > x_v$; it decreases if $x_u < x_v$, and it remains unchanged if $x_u = x_v$.

Now, consider any path $p = u_1, u_2, \ldots, u_k$. We have

$$\begin{aligned}
\mathbf{wt'}(p) = \quad & \mathbf{wt}(u_1, u_2) + x_{u_1} - \cancel{x_{u_2}} \\
& + \mathbf{wt}(u_2, u_3) + \cancel{x_{u_2}} - \cancel{x_{u_3}} \\
& + \mathbf{wt}(u_3, u_4) + \cancel{x_{u_3}} - \cancel{x_{u_4}} \\
& \ \ \vdots \\
& + \mathbf{wt}(u_{k-2}, u_{k-1}) + \cancel{x_{u_{k-2}}} - \cancel{x_{u_{k-1}}} \\
& + \mathbf{wt}(u_{k-1}, u_k) + \cancel{x_{u_{k-1}}} - x_{u_k}
\end{aligned}$$

Therefore, $\mathbf{wt'}(p) = \mathbf{wt}(p) + x_{u_1} - x_{u_k}$. In other words, if we fix two nodes u and v in the graph, the weight of ***every*** path from u to v under the new weight function $\mathbf{wt'}$ changes by the same amount relative to its weight under the old weight function, namely the difference $x_u - x_v$. In particular, a shortest path from u to v under the old weight function $\mathbf{wt}$ remains a shortest path from u to v under the new weight function $\mathbf{wt}'$. So, this way of re-weighing the edges achieves property (b): it preserves shortest paths. It remains to determine whether there are values we can choose for the node weights x_u that will also make the new edge weights non-negative, thereby also achieving property (a).

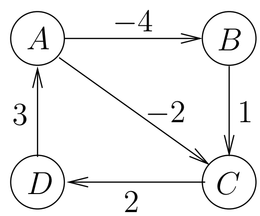
Let us view the x_u's as unknown variables. The question is whether there are values we can assign to these variables that satisfy, for every edge (u, v) of G

$$\underbrace{\mathbf{wt}(u,v) + x_u - x_v}_{\mathbf{wt}'(u,v)} \geq 0.$$

Or, rearranging the inequalities, the question is whether there are values for the variables x_u, $u \in V$, such that

$$x_v - x_u \leq \mathbf{wt}(u,v), \quad \text{for every } (u,v) \in E \tag{2}$$

Example: Consider the following graph:



This gives rise to the following inequalities, one for each edge of the graph:

$$\begin{aligned}
x_B - x_A &\leq -4 \\
x_C - x_A &\leq -2 \\
x_C - x_B &\leq 1 \\
x_D - x_C &\leq 2 \\
x_A - x_D &\leq 3
\end{aligned}$$

It turns out that these can be satisfied, for example by setting $x_A = 0$, $x_B = -4$, $x_C = -3$, $x_D = -1$. (Verify that this assignment satisfies all of the above inequalities; we will see shortly how these values were obtained.)

Claim: *The inequalities (2) are satisfiable if and only if G has no negative-weight cycle (under the edge weight function $\mathbf{wt}$).*

PROOF: [ONLY IF] Suppose there are values $\hat{x}_u$ for the variables x_u, $u \in V$, that satisfy (2), and let $C = u_1, u_2, \ldots, u_k, u_1$ be any cycle of G. We want to prove that $\mathbf{wt}(C) \geq 0$. Since the inequalities (2) are satisfied, we have:

$$\begin{aligned}
\hat{x}_{u_2} - \hat{x}_{u_1} &\leq \mathbf{wt}(u_1, u_2) \\
\hat{x}_{u_3} - \hat{x}_{u_2} &\leq \mathbf{wt}(u_2, u_3) \\
\hat{x}_{u_4} - \hat{x}_{u_3} &\leq \mathbf{wt}(u_3, u_4) \\
&\vdots \\
\hat{x}_{u_k} - \hat{x}_{u_{k-1}} &\leq \mathbf{wt}(u_{k-1}, u_k) \\
\hat{x}_{u_1} - \hat{x}_{u_k} &\leq \mathbf{wt}(u_k, u_1)
\end{aligned}$$

If we add these inequalities, all the terms on the left-hand side cancel out and the terms on the right-hand side add up to the weight of the cycle C. So, $\mathbf{wt}(C) \geq 0$, as wanted.

[IF] Suppose G has no negative-weight cycle. Therefore, shortest paths between any two nodes of G are well-defined. We want to show that there are values $\hat{x}_u$ for the variables x_u, $u \in V$, that satisfy all the inequalities (2). Rewrite (2) as

$$x_v \leq x_u + \mathbf{wt}(u,v), \quad \text{for every } (u,v) \in E. \tag{3}$$

Let s be a new "dummy" node that is not in V. We define the graph $G^s = (V^s, E^s)$ as follows: $V^s = V \cup \{s\}$ and $E^s = E \cup \{(s,u) : \ u \in V\}$ — that is, we add to G a new node s and edges from s to every other node. We also define an edge weight function $\mathbf{wt}^s$ as follows: $\mathbf{wt}^s(u,v) = \mathbf{wt}(u,v)$ for every edge $(u,v) \in E$ and $\mathbf{wt}^s(s,u) = 0$ for every $u \in V$. That is, the weights of G's edges do not change, and the weights of the new edges (from s to all nodes) are 0.

Since s has no incoming edges, G and G^s have the same cycles, and the weight of each cycle is the same in both graphs. In particular, G^s has no negative weight cycle — since, by assumption, G does not. Therefore, the weight of a shortest $s \to u$ path is well-defined in G^s.

If, for every node v, we interpret the variable x_v as the weight of a shortest $s \to v$ path in G^s, the inequalities (3) hold: They state that, for any node v and any predecessor u of v, the weight of a shortest $s \to v$ path is no greater than the weight of a shortest $s \to u$ path plus the weight of the edge (u,v); this statement is certainly true, by a straightforward cut-and-paste argument. Therefore, we can satisfy (2) by assigning to each x_v the weight of a shortest $s \to v$ path in G^s. □

The proof of the "if" direction of the above claim suggests the following effective procedure to ***find*** values for the variables x_u that satisfy the inequalities (2), which we need in order to re-weigh the edges so as to satisfy requirements (a) and (b): Construct the graph G^s from G, run the Bellman-Ford algorithm on G^s using s as the source node and the edge weight function $\mathbf{wt}^s$. If the algorithm reports that a negative-weight cycle is reachable from s, then G has a negative weight cycle and shortest paths on G are not well-defined. Otherwise, we can use the weight of a shortest $s \to u$ path computed by the Bellman-Ford algorithm as the weight x_u of node u, and then use these node weights to re-weigh the edges of G. (In the example on page 2 the weights of the nodes are shortest paths from node A. In this example, we don't need to invent a dummy source s since there already exists a node, namely A, so that there is a path from it to every node.) Since all edges now have non-negative weights, and the new weights preserve shortest paths, we can use Dijkstra's algorithm to compute shortest paths from each node u. This is Johnson's algorithm. It is described in pseudocode below.

```
JOHNSON(G = (V, E), wt)
V^s := V ∪ {s}                                   ▶ construct G^s and wt^s from G and wt
E^s := E ∪ {(s,u) : u ∈ V}
for each u ∈ V do wt^s(s,u) = 0
for each (u,v) ∈ E do wt^s(u,v) = wt(u,v)
L := BELLMAN-FORD(G^s, s, wt^s)    ▶ compute node weights as weights of shortest paths from s in G^s
if L = ⊥ then return ⊥                            ▶ shortest paths not well-defined
else
    for each u ∈ V do x_u := L[u]                 ▶ assign node weigts
    for each (u,v) ∈ E do                         ▶ reweigh the edges
        wt'(u,v) = wt(u,v) + x_u − x_v
    for each u ∈ V do                             ▶ run Dijkstra from each node using new weights
        D'[u] := DIJKSTRA(G, u, wt')
    for each u ∈ V do    ▶ adjust the weights of shortest paths found under wt' to the original weights wt
        for each v ∈ V do
            D[u,v] := D'[u,v] + x_v − x_u
return D[−,−]
```

In the pseudocode we assume that $\textsc{Bellman-Ford}(G, s, \mathbf{wt})$ returns the special value $\perp$ if G has a negative weight cycle reachable from s; otherwise it returns an array L indexed by the nodes, with $L[u]$ containing the weight of a shortest $s \to u$ path in G. We also assume that $\textsc{Dijkstra}(G, s, \mathbf{wt})$ returns an array D indexed by the nodes of G, so that $D[v]$ is the weight of a shortest $s \to v$ path. So in line 12, the assignment $D'[u] := \textsc{Dijkstra}(G, u, \mathbf{wt}')$ sets $D'[u]$ to an array indexed by the nodes of G so that $D'[u, v]$ is the weight of a shortest $u \to v$ path (under weight function $\mathbf{wt}'$).

Running time: We assume that the graph $G = (V, E)$ is given in adjacency list form, which is well-suited for sparse graphs. The weight $\mathbf{wt}(u, v)$ of edge (u, v) is stored together with node v in the adjacency list of node u. As usual let $n = |V|$ and $m = |E|$.

The construction of G^s from G (lines 1-4) can be done in $\Theta(m+n)$ time. The Bellman-Ford algorithm in line 5 takes $\Theta(mn)$ time. The computation of the new edge weights in lines 9-10 takes $\Theta(m)$ time. The n executions of Dijkstra's algorithm (lines 11-12) take $\Theta(nm \log n)$ time. Finally, the computation of the weight of shortest paths between every pair of nodes under the original weight function $\mathbf{wt}$ (lines 14-15) takes $\Theta(n^2)$ time. So, the overall running time of the algorithm is

$$\Theta(m + n) + \Theta(mn) + \Theta(nm \log n) + \Theta(n^2) = \Theta(nm \log n).$$

The running time is dominated by the n executions of Dijkstra's algorithm in lines 11-12.