The all-pairs shortest path problem

Combinatorial optimization

Giovanni Righini

University of Milan

UNIVERSITÀ DEGLI STUDI DI MILANO

All-pairs shortest paths

By a repeated execution (n times) of the Bellman-Ford algorithm, it is possible to compute shortest paths from any node to any other node in a weighted digraph $D = (\mathcal{N}, \mathcal{A})$. However the complexity is $O(n^2 m)$.

The same result can be obtained more efficiently with an algorithm due to Kleene (1956), Roy (1959), McNaughton e Yamada (1960), Warshall (1962), Floyd (1962), known as Floyd-Warshall algorithm.

The Floyd-Warshall algorithm is a dynamic programming algorithm.

Floyd-Warshall algorithm (1962)

Consider an arbitrary ordering of the nodes $v_1, v_2, \ldots, v_n$.

For each pair of nodes $s \in \mathcal{N}$ and $t \in \mathcal{N}$ and for each $k = 0, 1, \ldots, n$ we define $d_k(s, t)$, as the cost of the optimal path from s to t using only intermediate nodes in $\{v_1, \ldots, v_k\}$.

Initially, with $k = 0$, we have $d_0(s, t) = c_{st}$ for each arc $(s, t) \in \mathcal{A}$ and $d_0(s, t) = \infty$ for each pair $(s, t) \notin \mathcal{A}$.

The following recursive property holds:

$$d_k(s,t) = \min\{d_{k-1}(s,t), d_{k-1}(s,v_k) + d_{k-1}(v_k,t)\} \quad \forall k = 1, 2, \ldots, n.$$

A matrix π of optimal predecessors is also computed and it is used to reconstruct the shortest paths, recursively: we update $\pi[s, t] := k$ whenever $d_{k-1}(s, v_k) + d_{k-1}(v_k, t) < d_{k-1}(s, t)$.

Floyd-Warshall algorithm (1962)

Algorithm 1 Floyd-Warshall algorithm

for $u = 1, \ldots, n$ **do**
  for $v = 1, \ldots, n$ **do**
    if $u = v$ **then**
      $d[0, u, v] \leftarrow 0$
    else
      $d[0, u, v] \leftarrow c_{uv}$
    $\pi[u, v] \leftarrow 0$
for $k = 1, \ldots, n$ **do**
  for $u = 1, \ldots, n$ **do**
    for $v = 1, \ldots, n$ **do**
      if $d[k-1, u, k] + d[k-1, k, v] < d[k-1, u, v]$ **then**
        $d[k, u, v] \leftarrow d[k-1, u, k] + d[k-1, k, v]$
        $\pi[u, v] \leftarrow k$
      else
        $d[k, u, v] \leftarrow d[k-1, u, v]$

The computational complexity is $O(n^3)$.

Negative circuits

If the digraph contains negative cost circuits, then the Floyd-Warshall algorithm detects at least one of them and stops.

A negative cost circuit corresponds to a negative entry on the main diagonal (at any iteration).

Therefore the Floyd-Warshall algorithm can be used as a pre-processing sub-routine, to check whether a given digraph contains negative cost circuits or not.

Johnson algorithm

We now consider the case in which

- the digraph is strongly connected;
- there are no negative cost circuits;
- arc costs can be negative.

We can run:

- Bellman-Ford n times, once from each node: $O(n^2 m)$.
- Floyd-Warshall: $O(n^3)$.
- Dijkstra n times, once from each node, if all arc costs are non-negative: $O(nm + n^2 \log n)$.

Johnson algorithm (1977) allows for $O(nm + n^2 \log n)$ complexity even when arc costs can be negative.

Johnson algorithm

Johnson algorithm runs in three steps:

- run Bellman-Ford from a node s to all the other nodes;
- define modified arc costs such that:
  - the new costs are non-negative;
  - the rank of paths does not change (shortest paths remain shortest paths);
  - negative cost circuits are not introduced;
  - the new cost is computed in $O(m)$ (i.e. $O(1)$ for each arc);
- run Dijkstra from the other $n - 1$ nodes.

Johnson algorithm

Consider a potential function $p : \mathcal{N} \mapsto \Re$ and a new cost function

$$\overline{c}_{ij} = c_{ij} - p_i + p_j \ \ \forall (i,j) \in \mathcal{A}.$$

Effects on paths:

$$\begin{aligned}\overline{c}(P(1,k)) &= \overline{c}_{12} + \overline{c}_{23} + \ldots + \overline{c}_{k-1,k} = \\ &= c_{12} - p_1 + p_2 + c_{23} - p_2 + p_3 + \ldots + c_{k-1,k} - p_{k-1} + p_k = \\ &= c(P(1,k)) - p_1 + p_k.\end{aligned}$$

For each pair of nodes $(1,k)$ all path costs are modified by the same amount $p_k - p_1$: in particular, the shortest paths between the two nodes remain the same.

Effects on circuits:

$$\begin{aligned}\overline{c}(C) &= \overline{c}_{12} + \overline{c}_{23} + \ldots + \overline{c}_{k1} = \\ &= c_{12} - p_1 + p_2 + c_{23} - p_2 + p_3 + \ldots + c_{k1} - p_k + p_1 = \\ &= c(C).\end{aligned}$$

For each circuit C, the cost does not change: in particular, no negative cost circuits are introduced.

Johnson algorithm

The potential function we use is

$$p_i = -dist(s, i) \;\; \forall i \in \mathcal{N},$$

where $dist(s, i)$ is the shortest path cost from s to i, computed with Bellman-Ford algorithm.

With this choice, the modified arc costs are the *reduced costs*.

$$\overline{c}_{ij} = c_{ij} - p_i + p_j = c_{ij} + dist(s, i) - dist(s, j).$$

These reduced costs are all non-negative. The optimality conditions for shortest path (feasibility conditions for the dual problem) are:

$$dist(s, j) - dist(s, i) \leq c_{ij} \;\; \forall (i, j) \in \mathcal{A}$$

from which

$$\overline{c}_{ij} \geq 0 \;\; \forall (i, j) \in \mathcal{A}.$$