Floyd-Warshall algorithm

Liyang ZHANG

July 30, 2020

Introduction

Here I introduce an algorithm to find the path with the smallest edge between every pair of vertices at once in an oriented edge-weighted finite graph with negative weight edges permitted, but without any negative weight circle.

Dijkstra algorithm cannot deal with graphs with negative weighted edges, since it relies on a fact that if all weights are non-negative, adding an edge can never make a path shorter. [1] It fails to find the cheaper path when a largely negative edge is hidden after a largely positive edge, [1] for example the path from point 2 to 3 in Figure 1 (a).

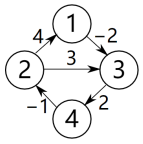
Now let me introduce Floyd-Warshall Algorithm, which is applicable in oriented edge-weighted finite graphs with negative weight edges permitted, but without any negative weight circle, with the graph of Figure 1 (a) as example.

My main references are [2], [3], and [4].

Algorithm

- Input: an oriented edge-weighted finite graph G, vertices labelled by consecutive integers from 1 to $n = \text{order}(G)$
- Output: a matrix $M \in \mathbb{R}^{n\times n}$ with M_{ij} showing the least weight among paths from vertex i to vertex j, and a set of shortest routes from each vertex to each vertex.

1.
   - Write down a matrix $M \in \mathbb{R}^{n\times n}$, with 0 on the diagonal, and the weight of edge from the i^{th} vertex to the j^{th} one on M_{ij}, and ∞ on M_{ij} if there is no edge from the i^{th} vertex to the j^{th} one. [*Example in Figure 1 (b)*]
   - Save down each edge as a route. [*Example in Figure 3 ($k = 0$)*]
2.
   - For each vertex k from 1 to n, For each ordered pair of vertices (i, j) with i and j from 1 to n different from k, If $M_{ij} > M_{ik} + M_{kj}$,
   - Then let $M_{ij} = M_{ik} + M_{kj}$, [*Example in Figure 2*]
   - Delete the old route from i to j,
   - And save down the new route as the route from i to k that we already have connected to the route from k to j that we already have. [*Example in Figure 3 ($k = 1, 2, 3, 4$)*]
     Notice that in the case of our example, as shown in Figure 3, at $k = 1$, we have replaced the route $2 \to 3$ by $2 \to 1 \to 3$, and $2 \to 1 \to 3$ is now the route from 2 to 3 that we already have; then, at $k = 2$, we find that it is shorter to go from 4 to 3 via 2. Here, we need to add the route $4 \to 2 \to 1 \to 3$ instead of $4 \to 2 \to 3$, as stated in the algorithm. Indeed, $4 \to 2 \to 1 \to 3$ is shorter than $4 \to 2 \to 3$, and the algorithm gives the correct thing.
3.
   - If for any i from 1 to n, $M_{ii} = 0$, then return M;
   - Else if $M_{ii} < 0$, then the input graph has at least one negative circle starting and ending at the vertex i. *Since the diagonal elements are initially 0 and only decreases in the following steps, $M_{ii} > 0$ is impossible.*



(a)[2]

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 3 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

(b)

Figure 1: (a) an oriented graph with negative edges; (b) the matrix M^0 listing the weight of its all edges.

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ \infty & -1 & \infty & 0 \end{pmatrix}$$

$k = 1$

$$\begin{pmatrix} 0 & \infty & -2 & \infty \\ 4 & 0 & 2 & \infty \\ \infty & \infty & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$k = 2$

$$\begin{pmatrix} 0 & \infty & -2 & 0 \\ 4 & 0 & 2 & 4 \\ \infty & \infty & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$k = 3$

$$\begin{pmatrix} 0 & -1 & -2 & 0 \\ 4 & 0 & 2 & 4 \\ 5 & 1 & 0 & 2 \\ 3 & -1 & 1 & 0 \end{pmatrix}$$

$k = 4$

Figure 2: Floyd–Warshall Algorithm on the graph of Figure 1 (a).
After examining all points, at $k = 4$, we have the final result.

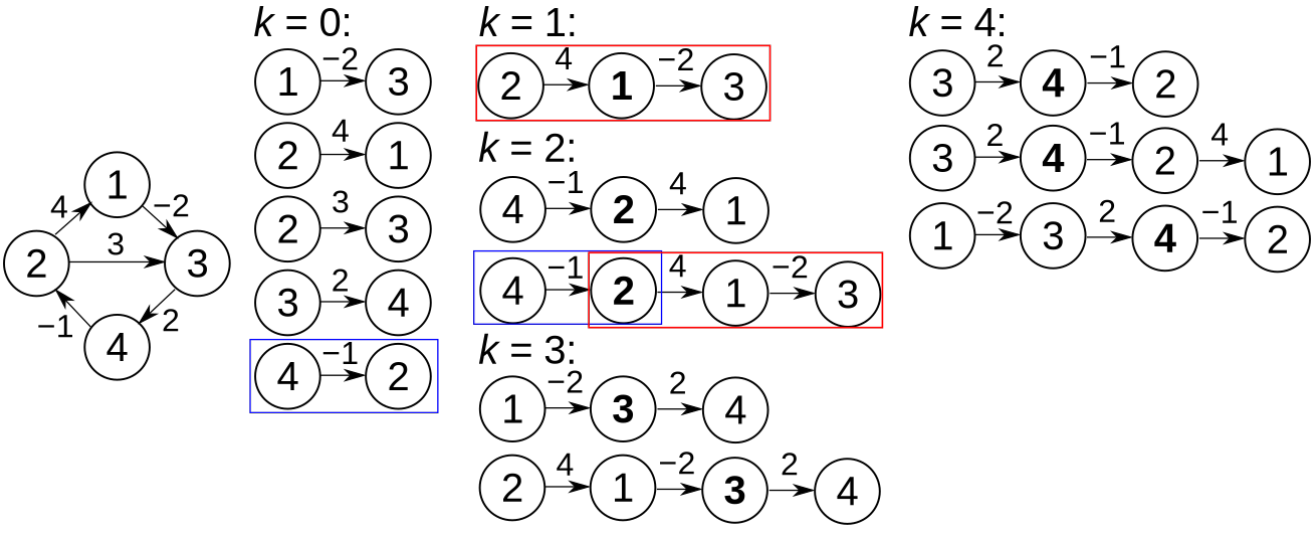


Figure 3: the temporary cheapest paths between each pair of vertices after each step. [2]
The final cheapest paths are in the last step $k = 4$.

Justification

The second step means that if the cheapest path from i to j not passing $k, k+1, ..., n$ is more expensive than the cheapest one passing k but not passing $k+1, ..., n$, then we replace it. After the loop from 1 to n, all possible midway points are examined once. In a figure without negative loop, paths passing a point more than once cannot be cheaper than its concatenated one, so it is enough to examine all points once and only once.

References

[1] www.quora.com/Why-doesnt-Dijkstra-work-with-negative-weight-graphs

[2] en.wikipedia.org/wiki/Floyd-Warshall_algorithm

[3] ithelp.ithome.com.tw/articles/10209186 (in Chinese)

[4] youtube.com/watch?v=4OQeCuLYj-4 (video of 4.5 minutes)