ABSTRACT

In most of the graph theory problem, the shortest path problem or SPP is the problem of determining the distance between two nodes (or vertices) in a weighted graph in order that the sum of the weights of its constituent edges is minimized. In a connected graph there exists at least one path between every pair of vertices. In a weighted graph the path between a pair of vertices for which the sum of the weights of the constituent edges is minimum is called the shortest path between them. Here in this paper we developed a new algorithm to determine the shortest path of a weighted graph which is an improvement over the Dijkstra’s algorithm so as to obtain all the shortest paths of a given weighted graph. Let us first recollect Dijkstra’s algorithm so as to easily comprehend our result.

Keywords: All-pairs Shortest Path, Adjacency Matrix, Binary heap, Dijkstra’s Algorithm, Single-Destination Path, Time Complexity

1. INTRODUCTION

Very popular, extensively studied, and widely applied Dijkstra algorithm [1, 2] for finding shortest paths of weighted graphs got published in 1959. Dijkstra algorithm gives a shortest path from a given vertex of a weighted graph to each of its remaining vertices. However, Dijkstra algorithm has a shortcoming that it gives exactly one shortest path of a given weighted graph. As a practical case the problem of finding the shortest path between two intersections on a road map may be modelled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections and the edges correspond to road segments, each weighted by the length of the segment. The single-source shortest path problem is one in which it is required to find shortest paths from a source vertex to all other vertices in the graph. The single-destination shortest path problem is one in which it is required to find shortest paths from all vertices in the directed graph to a single destination vertex. The all-pairs shortest path problem is one in which it is required to find shortest paths between every pair of vertices in the graph.

2. DIJKSTRA ALGORITHM [2]

Let $G = (V, E, W)$ is a non-negative weight network. Let $V = (v_1, v_2, ..., v_n)$. Let $w_{ij}$ be the weight of the edge $(v_i, v_j)$. The length of a shortest path in $G(v_i, v_j) \in E$ satisfies the following equation:

$$u_1 = 0, \quad u_j = \min_k \left( u_k + w_{kj} \right) \text{ for } j = 2, 3, ..., n$$  \hspace{1cm} (3.1)

If in $G$ from the vertex $v_1$ to the rest of the vertices, the shortest path length sorting by size:

$$u_{i_1} \leq u_{i_2} \leq ... \leq u_{i_n}$$

Here, $i_1 = 1, u_{i_1} = 0$, Then from (3.1) have:

$$u_j = \min_{k \neq j} \left\{ u_k + w_{kj} \right\}$$

$$(j = 2, 3, ..., n)$$
\[ = \min\{\min_{k<j}\{u_{ik} + w_{kij}\}, \min_{k>j}\{u_{ik} + w_{kij}\}\} \]

When \( k > j, i \neq j, u_{ik} \geq u_{ij} \), and \( w_{kij} \geq 0 \), thus

\[ u_{ij} \leq u_{ik} + w_{kij} \]

That is \( u_{ij} \leq \min_{k>j}\{u_{ik} + w_{kij}\} \)

Therefore \( u_{ij} = \min_{k>j}\{u_{ik} + w_{kij}\} \) \( (j = 1, 2, 3, \ldots, n) \)

\( u_{ij} \), one of the solution \( (u_{i1}, u_{i2}, \ldots, u_{in}) \) is the shortest path length of \( G(u_i, u_j) \).

2.1 Calculation Steps and the Problems of Dijkstra Algorithm

Weighted graph can be expressed as adjacency matrix \( \text{cost}[i][j] \), which states: if between \( v_i \) and \( v_j \) have no direct path, the \( \text{cost}[i][j] = \infty \); if between \( v_i \) and \( v_j \) have direct path, the \( \text{cost}[i][j] = w_{ij} \); if \( i = j \), then the \( \text{cost}[i][j] = 0 \). The set of \( S \) storage initial source of shortest path, during the process of calculating, the vertex have determined the shortest path added to the \( S \). Final \( \text{dist}[i] \) storage the shortest path length, the source point to the vertex, specific steps are as follows:

1. Initialize \( S \) and \( \text{dist} \).

\[ S = \{v_0\}, \text{dist}[i] = \text{cost}[0][i], i = 0, 1, \ldots, n - 1. \]

2. Select \( v_j \), so \( \text{dist}[j] = \min\{\text{dist}[i]|v_j \in (v - S)\} \);

\[ (3) \quad s = s \cup v_j \]

3. Modify the length of the shortest path from \( v_0 \) to \( v_k \in (V - S) \).

\[ \text{If dist}[j] + \text{cost}[j][k] < \text{dist}[k], \]

\[ \text{So dist}[k] = \text{dist}[j] + \text{cost}[j][k]. \]

4. Repeat steps (2),(3), until obtain the shortest path that from the source point \( v_0 \) to the rest of vertices \( v_j \).

However, Dijkstra algorithm can only find one shortest path from \( v_0 \) to \( v_j \), and cannot find all the shortest paths. In this chapter we develop a new algorithm by slightly modifying the Dijkstra algorithm, i.e. using Dijkstra algorithm in a more intuitive way to find all the shortest paths from a given vertex to all other vertices of a given undirected weighted graph.

3. RESULT

The shortest path and minimum spanning tree problems are two of the classic problems in combinatorial optimization. Using Dijkstra algorithm for searching shortest path problem is an important content of the application of GIS (Geographic Information System), but it fails to find all the shortest paths. Here we modify Dijkstra algorithm, add some data structure and propose an algorithm that finds all the shortest paths of one vertex to others in a given undirected weighted graph. The effectiveness of the algorithm is explained through a numerical example.

3.1 Improved Algorithm

1. The introduction of Dijkstra algorithm, that is, step (1) to step (4) in Algorithm 3.1.1.

2. According to the cost and the distance, creating a correction matrix \( \text{corr} \).

Method:

\[ \text{If } 0 < \text{cost}[i][j] < \infty, \]

then \( \text{corr}[i][j] = \text{cost}[i][j] + \text{dist}[i] \);

otherwise, \( \text{corr}[i][j] = \text{cost}[i][j] \) ;

3. Create a successor node set \( \text{succ}(v_i) \).

Method: By \( \text{succ}(i) = \{j|\text{corr}[i][j] = \text{dist}[j] \text{ and } i \neq j\} \).
find the set that the successor node for each vertex;

(4) According to the corr and succ output the shortest path from the source to all other vertices.

(5) The above algorithm finds the shortest path from one vertex to all other vertices of a weighted graph. The following example illustrates implementation of Algorithm 3.1.

3.1.1 Example:
Figure 1 shows a directed graph $G_1$, to find the shortest path from vertex $v_0$ to all other vertices the specific steps are as follows.

**Figure 1: Weighted directed graph $G_1$.**

(1) Adjacency matrix of the graph $G_1$ is given:

\[
\begin{bmatrix}
0 & 6 & \infty & \infty & 33 \\
\infty & 0 & 28 & 11 & \infty \\
\infty & \infty & 0 & \infty & 13 \\
23 & \infty & 7 & 0 & 15 \\
8 & \infty & \infty & \infty & 0 \\
\end{bmatrix}
\]

Initialize $S$ and $\text{dist}[i]$ ($0 \leq i \leq 4$): $S = \{v_0\}$, $\text{dist}[i] = \{0, 6, \infty, \infty, 33\}$.

(2) based on $\text{dist}[i] = \min\{\text{dist}[i] \mid v_i \in V - S\}$, to find $\text{dist}[1] = 6$; $S = \{v_0, v_1\}$;

(3) modify the shortest path length, starting from $v_0$ to any node on the $V - S$

$v_k$ (then $2 \leq k \leq 4$), $\text{dist}[i] = \{0, 6, 34, 17, 33\}$;

(4) $\text{dist}[3] = \min\{\text{dist}[i] \mid v_i \in V - S\} = 17$; $S = \{v_0, v_1, v_3\}$; $\text{dist}[i] = \{0, 6, 24, 17, 32\}$.

(5) $\text{dist}[2] = \min\{\text{dist}[i] \mid v_i \in V - S\} = 24$; $S = \{v_0, v_1, v_2, v_3\}$; $\text{dist}[i] = \{0, 6, 24, 17, 32\}$.

Although at this point there is a vertex not incorporated in set $S$, but it is the shortest distance have been determined, so the whole operation is over.

(6) Created corr matrix through the cost and dist:

\[
\begin{bmatrix}
0 & 6 & \infty & \infty & 33 \\
\infty & 0 & 34 & 17 & \infty \\
\infty & \infty & 0 & \infty & 37 \\
40 & \infty & 24 & 0 & 32 \\
40 & \infty & \infty & \infty & 0 \\
\end{bmatrix}
\]

According to corr, dist and $\text{succ}(i) = \{j \mid \text{corr}[i][j] = \text{dist}[j] \text{ and } i \neq j\}$, find the collection from the successor node for each vertex, here: $\text{succ}(v_0) = \{v_1\}$, $\text{succ}(v_1) = \{v_3\}$, $\text{succ}(v_2) = \{\text{null}\}$, $\text{succ}(v_3) = \{v_2, v_4\}$, $\text{succ}(v_4) = \{\text{null}\}$. Then all
the shortest paths originating from \( v_0 \) are \( v_0v_1; v_0v_1v_3v_2; v_0v_1v_3; v_0v_1v_3v_4 \).

### 3.1.2 Time Complexity Analysis:

**Case-1:** This case is valid if the following conditions are satisfied:

- The given graph \( G \) is represented by an adjacency matrix.
- Priority queue \( Q \) is represented as an unordered list

Here,
- \( A[i,j] \) stores the information about edge \((i,j)\).
- Time taken for selecting \( i \) with the smallest distance is \( O(V) \)
- For each neighbour of \( i \), time taken for updating \( \text{dist}[j] \) is \( O(1) \) and there will be maximum \( V \) neighbours
- Time taken for each iteration of the loop is \( O(V) \) and one vertex is deleted from \( Q \)

Thus, total time complexity becomes \( O(V^2) \)

**Case-2:** This case is valid if the following conditions are satisfied:

- The given graph \( G \) is represented as an adjacency list
- Priority Queue \( Q \) is represented as a binary heap

Here,
- With adjacency list representation, all vertices of the graph can be traversed Using BFS is \( O(V + E) \) times
- In Min-heap, operation like extract minimum and decrease key value takes \( O(\log V) \) times.

So, overall time complexity becomes

\[
O(E + V) \times O(\log V) = O(E \log V)
\]

### 3.1.3 Improved Algorithm: Correctness by induction

We prove the correctness by the method of induction. In the following \( G \) is the input graph, \( S \) is the source vertex \( L(u,v) \) is the length of an edge from the vertex \( u \) to \( v \) and \( V \) is the set of vertices. Let \( d(v) \) be the label found by the algorithm and let \( \delta(v) \) be the shortest path distance from \( S \) to \( v \). We want to show that \( d(v) = \delta(v) \) for every vertex \( v \) at the end of the algorithm. Showing that the algorithm correctly computes the distances. We prove this by induction on the length of shortest path that is \( |P| \) through the following lemma.

### 3.1.3.1 Lemma

For each \( x \in P, d(x) = \delta(x) \)

**Proof by induction:**

**Base case (|\( P \)| = 1):** Since \( P \) only grows in size, the only time \( |P| = 1 \) is when \( P = \{ S \} \) and \( d(S) = 0 = \delta(S) \) which is correct.

**Induction hypothesis:**

Let \( u \) be the last vertex added to \( P \). Let \( P' = P \cup \{ u \} \) our induction hypothesis is for \( x \in P', d(x) = \delta(x) \)

By the induction hypothesis for every vertex \( P' \) that is not \( u \), we have the correct distance label. We need only show that \( d(u) = \delta(u) \) to complete the proof.

Assume to the contrary that the shortest path from \( S \) to \( u \) is \( R \) and has length \( L(R) < d(u) \).

\( R \) starts in \( P' \) and at some leave \( P' \) (to get to \( u \) which is not in \( P' \) let \( xy \) be the first edge along \( R \) that leaves \( P' \) Let \( R_y \) be the \( S \) to \( x \) sub path of \( R \)

Now \( L(R_y) + L(xy) \leq L(R) \)

Since \( d(x) \) is the length of the shortest \( S \) to \( x \) path by the induction hypothesis, we have

\[
d(x) \leq L(R_y), \text{ giving us}
\]

\[
d(x) + L(xy) \leq L(R_y)
\]

Since \( y \) is adjacent to \( x \), \( d(y) \) must be have been updated by the algorithm, So
Finally, since \( u \) was picked by the algorithm \( u \) must have the smallest distance label

\[
d(u) \leq d(y)
\]

Combining these inequalities in reverse order gives us the contradiction that \( d(x) < d(x) \)

Therefore, no such shortest path \( R \) must exist and so \( d(u) = \delta(u) \).

4. CONCLUSION AND FUTURE SCOPE

The time complexity for each of the Dijkstra’s, Floyd Warshall and Bellman-Ford algorithms show that these algorithms are acceptable in terms of their overall performance in solving the shortest path problem. All of those algorithms produce just one solution. However, the main advantage of our modified algorithms to find out all possible path from from source to destination vertex and it may produce a number of different optimal solutions since the result can differ every time the it is executed. In the future our proposed frame work will be extended and improved in finding the shortest path or distance between two places in a map that represents any types of networks. In addition, other AI techniques like symbolic logic and neural networks also can be implemented in improving existing shortest path algorithms so as to form them more intelligent and more efficient.

5. REFERENCES