 doi:10.20944/preprints202008.0627.v1

Review

A Discussion on the Evolution of the Pathfinding Algorithms

Tong Pan [1], and Shuk Ching PUN-Cheng [2]

[1] The Hong Kong Polytechnic University; betty-tong.pan@connect.polyu.hk
[2] The Hong Kong Polytechnic University; lilian.pun@polyu.edu.hk

Abstract: In daily travel and activities, pathfinding is a significant process. They are often used in transportation routes calculation. They have now evolved to be able to solve most situations of the pathfinding and its related problems. This review describes previous and recent studies on the pathfinding algorithms. It reviews the development of pathfinding algorithms in a classification base on their usage. The aim is to summarize the application of the pathfinding algorithms for the readers interested in the subject that can be used as a supplement.

Keywords: pathfinding; algorithms; multi-criteria; multi-modal; multi-network; transportation

1. Introduction

Scientists and mathematizes have long studied the problem of pathfinding. A general cognition of the problem is to find the optimal path from origin to destination. As a classical graph theory algorithm problem, this problem was first solved by the Dijkstra's algorithm [13]. The initial goal of pathfinding problem is simply to find the best route from one place to another. However, this success is only the node of the development for solving the main category of optimal path problems. With the rapid development of local hardware facilities, electronic products, and computer science, the pathfinding's needs are also evolving. Different scenarios are considered in later studies. These include having multiple origins or destinations, obstacles that occur between the origin and destination, or the requirement of non-linear non-additive least fare and so on, resulting in the enhancement and evolution of various algorithmic solutions. These have been applied differently in passenger and goods commuting, and in logistics management with varying transportation modes.

As a universally accessible topic, the pathfinding study still has research spaces. To give an overview of the pathfinding algorithms development, this paper is backtrack to the history of pathfinding algorithms starting with the Dijkstra's algorithm invention. Several main existing pathfinding algorithms is analyzed, including the A star (A*) algorithm [22], Yen's algorithm [44], Double-sweep algorithm [38-39], Martins' delete path algorithm [12], Bellman Floyd algorithm [8,17,34,40], Shortest Path Faster Algorithm (SPFA) [14], Floyd-Warshall Algorithm [16,36,43], and Johnson's algorithm [25]. Lastly, an analysis is given to the increasing demand for pathfinding solutions, which arises from the increasing complexity of today's transport network, thus proposing possible ways forward.

2. A Review of Evolution of Pathfinding Algorithms

Evans and Minieka [15] summarized most path algorithms to fall into three categories: the single-source shortest paths algorithms, the all-pairs shortest paths algorithms, and the *k*-shortest paths algorithms. In detail, the single-source shortest paths algorithms are calculating the shortest paths from a specified source or vertex to a destination vertex or all other vertices. The all-pairs shortest paths algorithms are computing the shortest paths between every pair of vertices. Moreover, the *k*-shortest paths algorithms are reckoning not only the best or optimal path but also the second,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

third, and until the k_{th} best paths, which k represent. Furthermore, under every category, the algorithms can also be grouped as two labeling related settings [9-10] – the label setting method and are the label correcting method. Mainstream research is generally classified according to the solution types. Therefore, in this paper, a classification category base on the needs of the problem-solving may help guidance

To represent the pathfinding problem, graphs that contain sets of vertices and edges are commonly used. One edge usually connects a pair of vertices in the graph. Each vertex represents one location, and the edges are the possible path connecting every vertex. For example, if representing a country's map into a graph, the location point of the cities is described as vertices, the routes like road, railway, and flight lines between the cities are represented as edges. These edges usually align the same as the features on the ground but can also use as virtual edges. Each edge contains a weight, such as distance or travel time between the pair of connected vertices. The weights are added up gradually, and the least total weight is derived by checking and comparing the different possible edge combinations for a route reaching the destination.

For the explanation of different shortest path algorithms, Figure 1 is referred first to illustrate the concept of a planar graph network G. The sets of vertices (V) and edges (E) in the graph have their quantities v and e respectively. The edge between two vertices (i and j) is expressed as $E[i][j]$. The specific vertices of source and destination are denoted as $V[0]$ and $V[n]$, where n equal to v-1. Besides, a weight value (W) is attached to every edge, and the specific weight is $W[i][j]$. With this, pathfinding algorithms of different categories are explained.

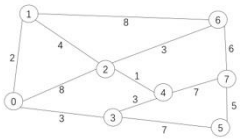
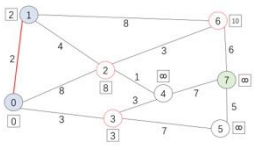


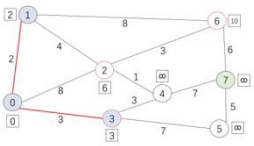
Figure 1. A sample planar graph network of G.

2.1. One origin to one destination problem with the single criterion

The problem of finding the best path or route between two places with a single criterion of the shortest distance, least time, or cost has been widely studied over the years. Many optimal path algorithms and their optimizations are proposed to compute the best or optimal option in any cost or weight, among which Dijkstra's Algorithm is the most orthodox. The main idea is to adopt the greedy algorithm strategy for the optimal choice and traverse the adjacent vertex with the lowest accumulated weight and have not been visited starting from the source vertex. As the example shows in Figure 2, it starting with searching every vertex connecting to the source vertex $V[0]$ and calculate the tentative distance for the unlabeled vertices (Highlight in red circle) (Figure 2a). label the vertex that has the lowest weight $W[0][i]$ and $V[1]$ and $E[0][1]$ are labeled as it has the minimum distance among unlabeled vertices (Figure 2a). Then the vertices connecting to the labeled vertex are further compared and labeled the one that has the lowest accumulated weight ($W[0][i] + W[i][j]$). Re-calculate and update the tentative distances for all the unlabeled vertices connect to previously labeled vertex, label the vertex that has the minimum distance among unlabeled vertices (Figures 2b-e). Then iterate the process of compare, labeling, and update till destination vertex is labeled. Accordingly, the recorded lowest accumulated weight shows the shortest route (Figure 2f).



(**a**)



(**b**)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

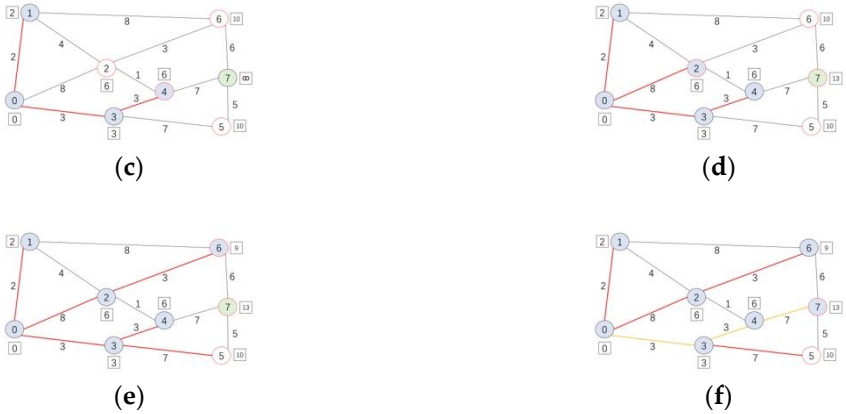


Figure 2. An example of Dijkstra's algorithm.

Because the algorithm is making a locally optimal choice every time it labels a vertex for a global optimum; and stops operation when reaching the destination. Therefore, the possible negative weighted edge may have been ignored, which means there is another possible optimal path that exists and invalidates the optimal result. Consequently, Dijkstra's algorithm is not suitable for the graph that contains a negative weight edge. Correspondingly, it is unable to handle the negative weight cycle in the graph.

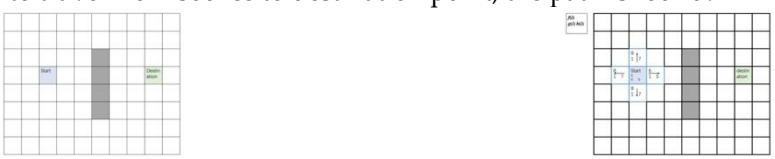
2.2. One origin to on destination problem with avoiding obstacles need

Besides the single criterion, in walking, driving, or road planning, avoiding obstacles' requirements are needed. An example of avoiding obstacles' problem is when visiting a museum hall that some exhibits are set up in the center of the room, and the exhibits in the center block the goal. The exhibits in the middle can be seen as obstacles, and finding the path to the goal turns into avoiding obstacles problem. A star (A*) algorithm was then developed for defining the obstacle. Base on the idea of the Dijkstra's algorithm, Hart et al. developed the A* algorithm to improve the capability of searching efficiency and accuracy with an evaluation function:

$$f(i) = g(i) + h(i), \tag{1}$$

where $g(i)$ as the weight from $V[0]$ to any other vertex $V[i]$, $h(i)$ as the estimated weight of the optimal path between $V[i]$ and $V[n]$ using Manhattan distance, and $f(i)$ as the weight between $V[0]$ and $V[n]$. This is a pre-process to evaluate the importance of every vertex other than the source vertex for vertices prioritization. When a higher search efficiency is needed, $g(i)$ can be ignored, but the completeness of the search will be affected. An optimal result is achieved when $h(i)$ is smaller than the real weight from $V[i]$ to $V[n]$. However, as $h(i)$ gets smaller, efficiency will be lowered with more vertices being calculated. Maximum computation occurs when $h(i)$ reaches 0, which will be the same as Dijkstra's algorithm. Similarly, because the A* algorithm is also unable to cater for a negative weight edge.

However, the A* algorithm is usually applied to the case using the grid in the raster format to represent edge or vertex. The searching area is divided into grids (Figure 3) or other polygonal shapes and store as a 2-d array. To depict the reality, every grid may represent the different sizes of reality area, and the smaller the grid size can represent the closer to reality. Furthermore, in order to better eliminate the gaps between grid boxes of different shapes, it is common to set a node as the center of the grid box at any position within the grid box or on edge. For ease of explanation, it is supposed to be the grid center. Each grid in the array is flagged either "available" or "unavailable." By calculating which grids need to travel from source to destination point, the path is found.



Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

Figure 3. An example of the A* algorithm. Blank grid as "available" grid, and gray grid as "unavailable" grid. (**a**) The source and destination points are highlighted as blue and green.

Figure 3 shows an example of the A* algorithm in a grid area of 10 by 8 and suppose the moving weight to the surrounding grids is 1. Accordingly, $f(i)$, $g(i)$, and $h(i)$ of the starting node are 0. An open list then creates to store the grids around the starting point and collects them during the continuous search of a new starting point (highlight in blue) (Figures 3a-b). By choosing the grid with the lowest f value and move the grid value to the close list. The chosen grid then becomes the new starting point (highlight in red). A close list then creates to store the grids that have been the starting point in the process of constantly searching for a new starting point, avoiding encountering duplicate points in the pathfinding process (Figure 3c). Iterate the process of the search until the destination is stored in the close list.

2.3. One origin to one destination problem with more than one options

In the real-world case, when the optimal path is no longer the best path due to some unpredictable conditions, the second or the third best paths are needed. The k-shortest paths problem has been mentioned and studied by the researchers since the 1950s. As a morph of the optimal path problem, the goal of the k-shortest paths algorithms is to seek multiple alternative optimization paths to satisfy different demands. There are two categories: the loopless and the general k-shortest paths algorithm. The former has limitations on the negative weight cycle and high complexity graph, while the latter has no restriction for the input graph. All can be applied to both single-source and all-pairs approaches.

2.3.1. Loopless algorithms

Yen's algorithm is one of the main algorithms dealing with loopless and simple graph input. It constructs all the shortest paths from a fixed vertex to all other vertices in the non-negative graph. This algorithm is characterized by fewer additions and comparisons than other k-shortest algorithms, but the time complexity is relatively high. It shares the same heuristic function as the typical A* algorithm described above, but the meaning of the state and the way to extend the state are different.

Referring to the example in Figure 4, the deviation method is adopted to calculate the second, third, and until k-1 shortest path. Suppose asking the shortest path for $k = 3$ for the graph, which needs to find the first, second, and third best path. First, the best or shortest path is computed with the resultant path stored in list A. In this case, Dijkstra's algorithm is used, and the path $P(1)$ is store in list A (Figure 5 Step1). The nodes in $P(1)$ is the priority queue in later calculations. Based on the deviation from the first path, the second shortest path is found (Figure 5, Step 2). When calculating the rest of the path, all other vertices except for the end node are set as deviation nodes, followed by finding the shortest path from each deviation node to the terminating node. Each edge includes in the first shortest path is set piecewise to ∞, and the graph using Dijkstra's algorithm is recalculated for k-1 times (Tables 1 and 2). Whenever recalculation finishes, the edge that is set as ∞ will be restored, and the resulting shortest path will be stored in a temporary list B (Figure 5). The final step is to find the shortest path among the paths stored in list B and move it to list A as the second shortest path (Figure 5, Step 2). Other candidate paths remain and continue to be candidates for the third, fourth, and k^{th} shortest path. The process is iterated until all the edges have been traversed (Figure 5, Step 3).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

Figure 4. An example of Yen's algorithm.

Table 1. Finding the second path using Yen's algorithm.

Deviation nodes	∞ edge sand nodes	Distance to *V*[7]	Result path (path weight)
V[0]	*E*[0][3], *null*	*W*[0][1][6][7] = 16	*E*[0][1][6][7] (16)
V[3]	*E*[3][4], *V*[0]	*W*[3][5][7] = 12	*E*[0][3][5][7] (15)
V[4]	*E*[4][7], *V*[0],V[3]	*W*[4][7] = 7	~~*E*[0][3][4][7] (13)~~

Table 2. Finding the third path using Yen's algorithm.

Deviation nodes	∞ edge sand nodes	Distance to *V*[7]	Result path (path weight)
V[0]	*E*[0][3], *null*	*W*[0][1][6][7] = 16	~~*E*[0][1][6][7] (16)~~
V[3]	*E*[3][5], *E*[3][4], *V*[0]	*null*	*null*
V[5]	*E*[5][7], *V*[0],*V*[3]	*W*[5][7] = 7	~~*E*[0][3][5][7] (15)~~

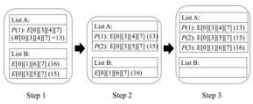


Figure 5. List A and List B's progress in Yen's algorithm. The resulting paths in Table 1 and 2 stores to List B (Steps 1 and 2). The shortest path in List B move to List A (Steps 2 and 3).

2.3.2. General algorithms

Other methods that solve the *k* shortest path problem are based on the optimality principle, such as the labeling algorithm and delete path algorithm.

The double-sweep algorithm was one of the early label correcting algorithms proposed by Shier. The algorithm' basic idea is to perform multiple iterations of the forward and backward sweeps until the labeling weight value under every vertex does not change anymore. In essence, the forward sweep process is to search the previous vertex *i* that connect to the vertex *j* ($i < j$) in ascending order of the index. The searching is to determine if the k^{th} shortest path from *V*[0] to *V*[*j*] is going through the *V*[*i*] or not. If so, the new estimated weight is saved for the next iteration. The backward sweep process is similar except that the post vertex *i* that connects to the vertex *j* ($i > j$) is searched in descending order of the index.

In 1984, the first delete path algorithm was proposed by De Queirós Vieira Martins. Creating a new graph that obviating the previous shortest path outcomes is one of the iconic features of the delete path algorithm. The steps of the delete path algorithm are:

1. delete the shortest path from the current operating graph;
2. determine the shortest path for the latest graph;
3. iterate step 1. and 2. until the k^{th} shortest path is found.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

For example, if the first shortest path is found in graph G, then the second shortest path will be found in a new graph G' that excluding the vertices of the first shortest path.

In summary, algorithms dealing with one origin to one destination mostly consider the optimal choices locally. The drawbacks are some algorithms are picking the optimal solutions locally too soon, so the possible negative weight that can reduce the total weight is ignored.

2.4. One origin to multiple destinations problem with the single criterion

When there is a need to plan how to reach multiple destinations more efficiently in terms of distance, time, or costs, such as tourists' route planning and fleet management, it calls for solutions that can calculate optimal path sequence from one place to all other locations. The Bellman-Floyd algorithm and shortest path faster algorithm (SPFA) developed by Duan is capable of solving the one origin to many destinations problem by traversing all other vertices from an origin.

Negative weight value may apply. For example, in the case of public transport fare calculation, the negative weight edge represents the road section that offers a discount. However, the negative weight edges may not fit the original pathfinding algorithms that deal with positive edges. Therefore, to provide the correct calculation of the optimal path in a graph that has a negative weight, the scholars program a reweight process of balancing the negative weight back to the positive weight may also solve the problem. Nevertheless, under the circumstances that the negative edge weight is acting as the penalties or a function of the graph, the negative weight calculation ability is needed.

The Bellman-Ford algorithm found improvements on the Dijkstra's algorithms so that an optimum global result of whether the weights are positive or negative can be achieved. It applies label correcting setting, checking, and updating a better path progressively for every path between the source vertex and all other vertices n-1 times (Figure 6a). Since the algorithm does not ignore any edge for calculation, negative weight edge, if any, in the graph, can be appropriately detected and considered. Base on the negative weight cycle, the total weight can be unlimitedly lowered when updating the labels. However, owing to the requirement of a complete calculation of all paths between the source vertex and all other vertices, the Bellman-Ford algorithm's time complexity is attaining ev, resulting in a high time complexity value among other popular shortest path algorithms.

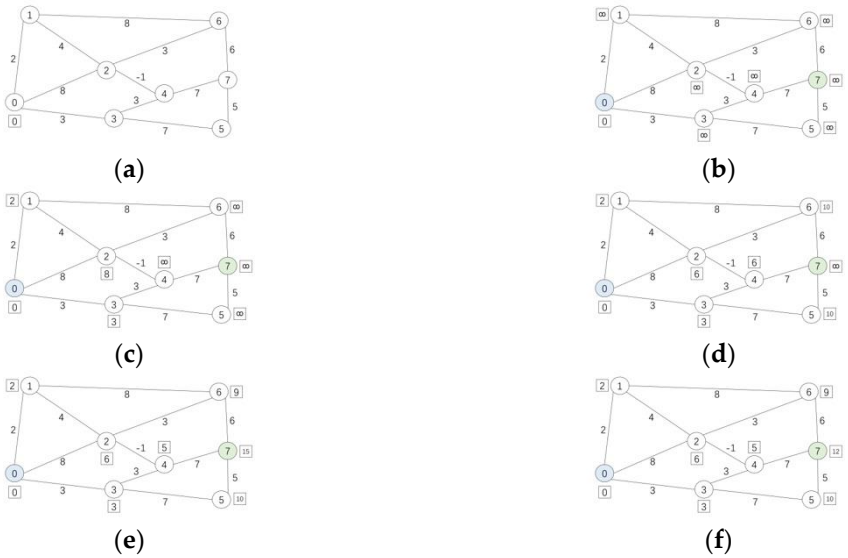


Figure 6. An example of the Bellman-Ford algorithm.

An example of implementing the Bellman-Ford algorithm is shown in Figure 6. The stating graph of Figure 6a, sets the starting point as 0, and all other vertices are infinity. First, establish the edge object information, start from the source point, from near to far (Figure 6b; Table 3). Then do the first relaxation operation and count the value of a node that can be reached by one edge, two edges, and three edges (Figures 6c-e; Tables 4-6) — followed by the Second relaxation operation (Figure 6f;

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

Table 6). The third relaxation operation is done afterward, the result is not updated again, thus end traversal ahead of time to optimize efficiency.

Table 3. Bellman-Ford algorithm edge object information.

Index	Weight [*i*][*j*]	Weight value
1	*W*[0][1]	2
2	*W*[0][2]	8
3	*W*[0][3]	3
4	*W*[1][2]	4
5	*W*[1][6]	8
6	*W*[2][4]	-1
7	*W*[2][6]	3
8	*W*[3][4]	3
9	*W*[3][5]	7
10	*W*[4][7]	7
11	*W*[5][7]	5
12	*W*[6][7]	6

Table 4. The cost of a node that can be reached by one edge in the first relaxation of the Bellman-Ford algorithm.

Parent node	Node	Cost
V[0]	*V*[0]	0
V[0]	*V*[1]	2
V[0]	*V*[2]	8
V[0]	*V*[3]	3
V[]	*V*[4]	∞
V[]	*V*[5]	∞
V[]	*V*[6]	∞
V[]	*V*[7]	∞

Table 5. The cost of a node can be reached by two edges in the first relaxation of the Bellman-Ford algorithm.

Parent node	Node	Cost
V[0]	*V*[0]	0
V[0]	*V*[1]	2
V[1]	*V*[2]	6
V[0]	*V*[3]	3
V[3]	*V*[4]	6
V[3]	*V*[5]	10
V[1]	*V*[6]	10
V[]	*V*[7]	∞

Table 6. The cost of a node can be reached by three edges in the first relaxation of the Bellman-Ford algorithm.

Parent node	Node	Cost
V[0]	*V*[0]	0
V[0]	*V*[1]	2
V[1]	*V*[2]	6
V[0]	*V*[3]	3
V[2]	*V*[4]	5

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

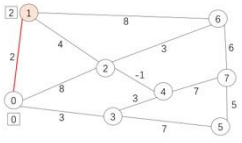
V[3]	*V*[5]	10
V[2]	*V*[6]	9
V[5]	*V*[7]	15

Table 7. The cost of a node can be reached by three edges in the second relaxation of the Bellman-Ford algorithm.

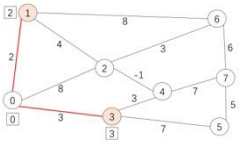
Parent node	Node	Cost
V[0]	*V*[0]	0
V[0]	*V*[1]	2
V[1]	*V*[2]	6
V[0]	*V*[3]	3
V[2]	*V*[4]	5
V[3]	*V*[5]	10
V[2]	*V*[6]	9
V[4]	*V*[7]	12

The shortest path faster algorithm (SPFA) is another significant algorithm solving the 1:*n* vertices problem. This algorithm is regarded as an enhancement of the Bellman-Ford algorithm. In Figure 7, the SPFA labels all vertices a weight accumulates from *V*[0] to itself, and updates the labels every time of weight reduction. Similar to the Bellman-Ford algorithm, SPFA uses every vertex for the next alternative and connected vertex. The difference is that SPFA maintains a queue of alternative vertices, so it does not necessarily try all vertices. A queue table is created that only allows delete operation for the top records and insert operation for the bottom ones. Furthermore, unlike the Dijkstra's algorithm and A* algorithm, SPFA inherits the time complexity and the ability to consider negative weight edges from the Bellman-Ford algorithm.

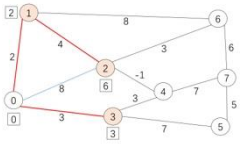
The example of the SPFA is shown in Figure 7. First, create a list to store the shortest path for every vertice. The first shortest route that *V*[0] directly reach is *V*[1], then path *E*[0][1] and *W*[0][1] = 2 will be store in the list for *V*[1] (Figure 7a). Next, since *V*[1] is reached, the vertices connected to it can be considered. The second shortest route from *V*[0] is *V*[3], then path *E*[0][3] and *W*[0][3] = 3 will be store in the list for *V*[3] (Figure 7b).



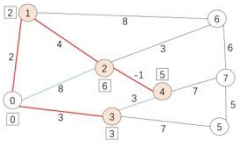
(**a**)



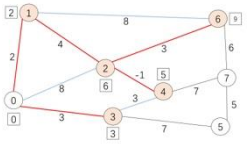
(**b**)



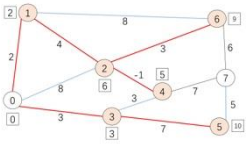
(**c**)



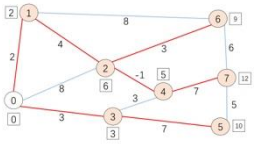
(**d**)



(**e**)



(**f**)



(**g**)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

Figure 7. An example of SPFA.

Now considers the vertices connect to *V*[0], *V*[1], and *V*[3], the next shortest route from *V*[0] is *V*[2], then path *E*[0][1][2] and *W*[0][1][2] = 6 will be store in the list for *V*[2]. There are two ways to get to *V*[2] from *V*[0], and only the shortest path will be stored. Thus the edge E[0][2] will be ignored for later calculation (Figure 7c).

Next, are the vertices connect to *V*[1], *V*[2], and *V*[3], the next shortest route from *V*[0] is *V*[4], then path *E*[0][1][2][4] and *W*[0][1][2][4] = 5 will be store in the list for *V*[4]. There are two ways to get to *V*[4] from *V*[0], and only the shortest path will be stored. Thus the edge *E*[3][4] will be ignored for later calculation (Figure 7d).

Then check the vertices connect to *V*[1], *V*[2], *V*[3], and *V*[4] the next shortest route from *V*[0] is V[6], then path *E*[0][1][2][6] and *W*[0][1][2][6] = 9 will be store in the list for *V*[4]. There are two way to get to *V*[6] from *V*[0], and only the shortest path will be stored. Thus the edge *E*[1][6] will be ignored for later calculation (Figure 7e).

Continually examine the vertices connect to *V*[3], *V*[4], and *V*[6] the next shortest route from *V*[0] is *V*[5], then path E[0][3][5] and *W*[0][3][5] = 10 will be store in the list for i[5]. There are two ways to get to *V*[5] from *V*[0], and only the shortest path will be stored. Thus the edge *E*[5][7] will be ignored for later calculation (Figure 7f).

Lastly view on the vertices connect to *V*[4], *V*[5], and *V*[6] the next shortest route from *V*[0] is V[7], then path *E*[0][1][2][4][7] and *W*[0][1][2][4][7] = 12 will be store in the list for *V*[7]. There are two ways to get to *V*[7] from *V*[0], and only the shortest path will be stored. Thus the edge *E*[6][7] will be ignored for later calculation. Finally, the process terminates when all the vertices are calculated (Figure 7g).

2.5. Multiple origins and multiple destinations problem with the single criterion

For city planning, especially in evaluating the accessibility of different places or detecting whether certain areas are over-served or under-served by transportation infrastructure, there is a need to compute and evaluate paths for multiple origins to multiple destinations. Among all the all-pairs shortest path algorithms, two mainstreams provide a complete set of the pathfinding result between any two vertices in the graph – the Floyd-Warshall algorithm developed by Floyd, Roy, and Warshall, and the Johnson's algorithm.

The Floyd-Warshall algorithm (Figure 8) is to check every pair of vertices *V*[*i*] and *V*[*j*] to see if a path that passes through *V*[*k*] is having a smaller weight than the known path, i.e. *W*[*i*][*j*] > *W*[*i*][*k*] + *W*[*k*][*j*]. If so, the weight of *W*[*i*][*j*] and their intermediate vertex *V*[*k*] are updated until *W*[*i*][*j*] is equal to the minimum value of *W*[*i*][*k*] + *W*[*k*][*j*]. This algorithm can also cater to negative weight edge because it considers all the possible vertices and edges, but the negative weight cycle is maladjustment for this algorithm. Compare with the Dijkstra's and Bellman-Ford algorithms, it has higher efficiency and lower coding difficulty. However, the relatively high time complexity makes it unsuitable for large datasets.

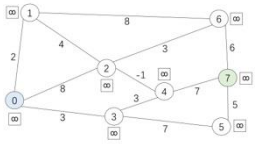


Figure 8. An example of the Floyd-Warshall algorithm, starting graph.

The Floyd-Warshall algorithm is using the matrix to solve the shortest path problem. Give matrixes, where matrix *A* in (2) is the adjacency matrix, and matrix *Path* in (3) records the point that the shortest path between two points *i*, *j* must pass.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

$$A(-1)=\begin{matrix}0&2&8&3&\infty&\infty&\infty&\infty\\2&0&4&\infty&\infty&\infty&8&\infty\\8&4&0&\infty&-1&\infty&3&\infty\\3&\infty&\infty&0&3&7&\infty&\infty\\\infty&\infty&-1&3&0&\infty&\infty&7'\\\infty&\infty&\infty&7&\infty&0&\infty&5\\\infty&8&3&\infty&\infty&\infty&0&6\\\infty&\infty&\infty&\infty&7&5&6&0\end{matrix} \quad (2)$$

$$Path(-1)=\begin{matrix}-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1'\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\end{matrix} \quad (3)$$

First, cross out row 0, column 0, and diagonal line to calculate $A(0)$:

$$A(0)=\begin{matrix}0&2&8&3&\infty&\infty&\infty&\infty\\2&0&4&\infty(5)&\infty&\infty&8&\infty\\8&4&0&\infty(11)&-1&\infty&3&\infty\\3&\infty(5)&\infty(11)&0&3&7&\infty&\infty\\\infty&\infty&-1&3&0&\infty&\infty&7'\\\infty&\infty&\infty&7&\infty&0&\infty&5\\\infty&8&3&\infty&\infty&\infty&0&6\\\infty&\infty&\infty&\infty&7&5&6&0\end{matrix} \quad (4)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(0)=\begin{matrix}-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&0&-1&-1&-1&-1\\-1&-1&-1&0&-1&-1&-1&-1\\-1&0&0&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1'\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\end{matrix} \quad (5)$$

Second, cross out row 1, column 1, and diagonal line to calculate $A(1)$:

$$(1)=\begin{matrix}0&2&8(6)&3&\infty&\infty&\infty(10)&\infty\\2&0&4&5&\infty&\infty&8&\infty\\8(6)&4&0&11(9)&-1&\infty&3&\infty\\3&5&11(9)&0&3&7&\infty(13)&\infty\\\infty&\infty&-1&3&0&\infty&\infty&7'\\\infty&\infty&\infty&7&\infty&0&\infty&5\\\infty(10)&8&3&\infty(13)&\infty&\infty&0&6\\\infty&\infty&\infty&\infty&7&5&6&0\end{matrix} \quad (6)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(1)=\begin{matrix}-1&-1&1&-1&-1&-1&1&-1\\-1&-1&-1&0&-1&-1&-1&-1\\1&-1&-1&1&-1&-1&-1&-1\\-1&0&1&-1&-1&-1&1&-1\\-1&-1&-1&-1&-1&-1&-1&-1'\\-1&-1&-1&-1&-1&-1&-1&-1\\1&-1&-1&1&-1&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\end{matrix} \quad (7)$$

Third, cross out row 2, column 2, and diagonal line to calculate A(2):

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

$$A(2)=\begin{matrix}0&2&6&3&\infty(5)&\infty&10(9)&\infty\\2&0&4&5&\infty(3)&\infty&8(7)&\infty\\6&4&0&9&-1&\infty&3&\infty\\3&5&9&0&3&7&13(12)&\infty\\\infty(5)&\infty(3)&-1&3&0&\infty&\infty(2)&7'\\\infty&\infty&\infty&7&\infty&0&\infty&5\\10(9)&8(7)&3&13(12)&\infty(2)&\infty&0&6\\\infty&\infty&\infty&\infty&7&5&6&0\end{matrix} \quad (8)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(2)=\begin{matrix}-1&-1&1&-1&2&-1&2&-1\\-1&-1&-1&0&2&-1&2&-1\\1&-1&-1&1&-1&-1&-1&-1\\-1&0&1&-1&-1&-1&2&-1\\2&2&-1&-1&-1&-1&2&-1'\\-1&-1&-1&-1&-1&-1&-1&-1\\2&2&-1&2&2&-1&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\end{matrix} \quad (9)$$

Fourth, cross out row 3, column 3, and diagonal line to calculate $A(3)$:

$$A(3)=\begin{matrix}0&2&6&3&5&\infty(10)&9&\infty\\2&0&4&5&3&\infty(12)&7&\infty\\6&4&0&9&-1&\infty(16)&3&\infty\\3&5&9&0&3&7&12&\infty\\5&3&-1&3&0&\infty(10)&2&7'\\\infty(10)&\infty(12)&\infty(16)&7&\infty(10)&0&\infty(19)&5\\9&7&3&12&2&\infty(19)&0&6\\\infty&\infty&\infty&\infty&7&5&6&0\end{matrix} \quad (10)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(3)=\begin{matrix}-1&-1&1&-1&2&3&2&-1\\-1&-1&-1&0&2&3&2&-1\\1&-1&-1&1&-1&3&-1&-1\\-1&0&1&-1&-1&-1&2&-1\\2&2&-1&-1&-1&3&2&-1'\\3&3&3&-1&3&-1&3&-1\\2&2&-1&2&2&3&-1&-1\\-1&-1&-1&-1&-1&-1&-1&-1\end{matrix} \quad (11)$$

Fifth, cross out row 4, column 4, and diagonal line to calculate $A(4)$:

$$A(4)=\begin{matrix}0&2&6(4)&3&5&10&9(7)&\infty(12)\\2&0&4(2)&5&3&12&7(5)&\infty(10)\\6(4)&4(2)&0&9(2)&-1&16(9)&3(1)&\infty(6)\\3&5&9(2)&0&3&7&12(5)&\infty(10)\\5&3&-1&3&0&10&2&7'\\10&12&16(9)&7&10&0&19(12)&5\\9(2)&7(5)&3(1)&12(5)&2&19(12)&0&6\\\infty(12)&\infty(10)&\infty(6)&\infty(10)&7&5&6&0\end{matrix} \quad (12)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(4)=\begin{matrix}-1&-1&4&-1&2&3&4&4\\-1&-1&4&0&2&3&4&4\\4&4&-1&4&-1&4&4&4\\-1&0&4&-1&-1&-1&4&4\\2&2&-1&-1&-1&3&2&-1'\\3&3&4&-1&3&-1&4&-1\\4&4&4&4&2&4&-1&-1\\4&4&4&4&-1&-1&-1&-1\end{matrix} \quad (13)$$

Sixth, cross out row 5, column 5, and diagonal line to calculate $A(5)$:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

$$A(5) = \begin{matrix} 0 & 2 & 4 & 3 & 5 & 10 & 7 & 12 \\ 2 & 0 & 2 & 5 & 3 & 12 & 5 & 10 \\ 4 & 2 & 0 & 2 & -1 & 9 & 1 & 6 \\ 3 & 5 & 2 & 0 & 3 & 7 & 5 & 10 \\ 5 & 3 & -1 & 3 & 0 & 10 & 2 & 7' \\ 10 & 12 & 9 & 7 & 10 & 0 & 12 & 5 \\ 2 & 5 & 1 & 5 & 2 & 12 & 0 & 6 \\ 12(7) & 10 & 6 & 10 & 7 & 5 & 6 & 0 \end{matrix} \quad (14)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(5) = \begin{matrix} -1 & -1 & 4 & -1 & 2 & 3 & 4 & 4 \\ -1 & -1 & 4 & 0 & 2 & 3 & 4 & 4 \\ 4 & 4 & -1 & 4 & -1 & 4 & 4 & 4 \\ -1 & 0 & 4 & -1 & -1 & -1 & 4 & 4 \\ 2 & 2 & -1 & -1 & -1 & 3 & 2 & -1' \\ 3 & 3 & 4 & -1 & 3 & -1 & 4 & -1 \\ 4 & 4 & 4 & 4 & 2 & 4 & -1 & -1 \\ 5 & 4 & 4 & 4 & -1 & -1 & -1 & -1 \end{matrix} \quad (15)$$

Seventh, ross out row 6, column 6, and diagonal line to calculate $A(6)$:

$$A(6) = \begin{matrix} 0 & 2 & 4 & 3 & 5 & 10 & 7 & 12 \\ 2 & 0 & 2 & 5 & 3 & 12 & 5 & 10 \\ 4(3) & 2 & 0 & 2 & -1 & 9 & 1 & 6 \\ 3 & 5 & 2 & 0 & 3 & 7 & 5 & 10 \\ 5(4) & 3 & -1 & 3 & 0 & 10 & 2 & 7' \\ 10 & 12 & 9 & 7 & 10 & 0 & 12 & 5 \\ 2 & 5 & 1 & 5 & 2 & 12 & 0 & 6 \\ 7 & 10 & 6 & 10 & 7 & 5 & 6 & 0 \end{matrix} \quad (16)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(6) = \begin{matrix} -1 & -1 & 4 & -1 & 2 & 3 & 4 & 4 \\ -1 & -1 & 4 & 0 & 2 & 3 & 4 & 4 \\ 6 & 4 & -1 & 4 & -1 & 4 & 4 & 4 \\ -1 & 0 & 4 & -1 & -1 & -1 & 4 & 4 \\ 6 & 2 & -1 & -1 & -1 & 3 & 2 & -1' \\ 3 & 3 & 4 & -1 & 3 & -1 & 4 & -1 \\ 4 & 4 & 4 & 4 & 2 & 4 & -1 & -1 \\ 5 & 4 & 4 & 4 & -1 & -1 & -1 & -1 \end{matrix} \quad (17)$$

At last, cross out row 7, column 7, and diagonal line to calculate A(7):

$$A(7) = \begin{matrix} 0 & 2 & 4 & 3 & 5 & 10 & 7 & 12 \\ 2 & 0 & 2 & 5 & 3 & 12 & 5 & 10 \\ 3 & 2 & 0 & 2 & -1 & 9 & 1 & 6 \\ 3 & 5 & 2 & 0 & 3 & 7 & 5 & 10 \\ 4 & 3 & -1 & 3 & 0 & 10 & 2 & 7' \\ 10 & 12 & 9 & 7 & 10 & 0 & 12(11) & 5 \\ 2 & 5 & 1 & 5 & 2 & 12(11) & 0 & 6 \\ 7 & 10 & 6 & 10 & 7 & 5 & 6 & 0 \end{matrix} \quad (18)$$

The two-order determinant for the elements that are not in the above 3 lines are:

$$Path(7) = \begin{matrix} -1 & -1 & 4 & -1 & 2 & 3 & 4 & 4 \\ -1 & -1 & 4 & 0 & 2 & 3 & 4 & 4 \\ 6 & 4 & -1 & 4 & -1 & 4 & 4 & 4 \\ -1 & 0 & 4 & -1 & -1 & -1 & 4 & 4 \\ 6 & 2 & -1 & -1 & -1 & 3 & 2 & -1' \\ 3 & 3 & 4 & -1 & 3 & -1 & 7 & -1 \\ 4 & 4 & 4 & 4 & 2 & 7 & -1 & -1 \\ 5 & 4 & 4 & 4 & -1 & -1 & -1 & -1 \end{matrix} \quad (19)$$

Matrix $A(7)$ in (18) is the result of the shortest distance to all the vertices, and matrix $Path(7)$ in (19) records the resulting path.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

On the other hand, Johnson's algorithm is adopting a completely different approach (Figure 9). It first re-weights all the edges to balance the negative weight edges for the later stages of the algorithm. Second, an extra fictitious vertex $V[s]$ is added into the graph, which is assumed to be connected to all other vertices. Third, the Bellman-Ford algorithm is applied for computing $W[s][i]$ with $V[s]$ as the source vertex. $W[s][i]$ is the resultant shortest weight between $V[s]$ and all other vertices. All the original edges are then re-weighted, where new weight $w[i][j] = W[i][j] + (W[s][i] - W[s][j])$. Lastly, with the deletion of vertex $V[s]$, Dijkstra's algorithm is applied to every pair of vertices.

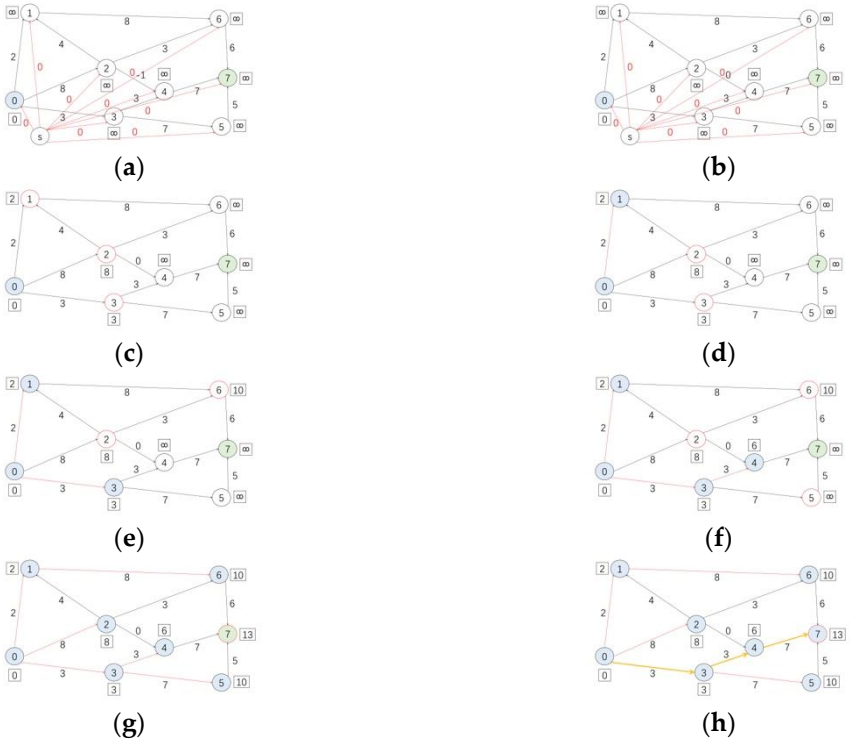


Figure 9. An example of Johnson's algorithm.

In detail, first, add the extra vertex $V[s]$ to the graph, and make the weight to all other vertices are 0 (highlight in red) (Figure 9a). Run the bellman-ford method for the vertex $V[s]$. Establish the edge object information, start from the source point, from near to far. Thus, the shortest distance for $V[s]$ to all other vertices list in Table 8. Then after the re-weighting process using the equation of

$$w[i][j] = W[i][j] + (W[s][i] - W[s][j]), \quad (20)$$

the weight of each edge shows in Table 9. After removing $V[s]$, the graph is no longer have negative value, so the Dijkstra's algorithm can be applied.

In detail, first, add the extra vertex $V[s]$ to the graph, and make the weight to all other vertices are 0 (highlight in red) (Figure 9a). Run the bellman-ford method for the vertex $V[s]$. Establish the edge object information, start from the source point, from near to far. Thus, the shortest distance for $V[s]$ to all other vertices list in Table 8. Then after the re-weighting process using the equation of (20), the weight of each edge shows in Table 9. After removing $V[s]$, the graph is no longer have negative value, so the Dijkstra's algorithm can be applied.

Table 8. The shortest distance for $V[s]$ to all other vertices of Johnson's algorithm.

Weight [s][i]	Value
$W[s][0]$	0
$W[s][1]$	0
$W[s][2]$	0
$W[s][3]$	0
$W[s][4]$	-1

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

$W[s][5]$	0
$W[s][6]$	0
$W[s][7]$	0

Table 9. Re-weighting process of Johnson' algorithm.

$w[i][j]$	$W[i][j] + (W[s][i] - W[s][j])$	Weight
$w[0][1]$	2 + (0 - 0)	2
$w[0][2]$	8 + (0 - 0)	8
$w[0][3]$	3 + (0 - 0)	3
$w[1][6]$	8 + (0 - 0)	8
$w[2][1]$	4 + (0 - 0)	4
$w[2][4]$	-1 + (0 – (-1))	0
$w[2][6]$	3 + (0 - 0)	3
$w[3][4]$	3 + (0 - 0)	3
$w[3][5]$	7 + (0 - 0)	7
$w[4][7]$	7 + (0 - 0)	7
$w[5][7]$	5 + (0 - 0)	5
$w[6][7]$	6 + (0 - 0)	6

Firstly, calculate the tentative distance for the unlabeled vertices connect to the source node ($V[0]$) (Highlight in red) (Figure 9c). Secondly, re-calculate and update the tentative distances for all the unlabeled vertices connect to $V[1]$. $V[3]$ has a minimum distance among unlabeled vertices. Hence $V[3]$ and $E[0][3]$ are labeled (Figure 9d). Thirdly, re-calculate, and update the tentative distances for all the unlabeled vertices connect to $V[3]$. $V[4]$ has the minimum distance among unlabeled vertices, hence $V[4]$ and $E[3][4]$ are labeled (Figure 9e). Fourthly, re-calculate and update the tentative distances for all the unlabeled vertices connect to $V[2]$. $V[5]$ and $V[6]$ have the minimum distance among unlabeled vertices; hence $V[5]$, $V[6]$, $E[3][5]$, and $E[1][6]$ are labeled (Figure 9f). Finally, re-calculate, and update the tentative distances for all the unlabeled vertices connect to $V[5]$ and $V[6]$. The destination vertex $V[7]$ is reached, and $E[4][7]$ are labeled for minimum tentative distance (Figure 9g). Figure 9h highlights the resulting path in yellow.

Johnson's algorithm solves the problem of negative weight edge in Dijkstra's algorithm but is still unable to detect the negative weight cycle's existence because of the balancing re-weight step. The improvement allows the all-pairs shortest path problem to be solved by a single source shortest path solution so that the properties of obtaining the optimal solution using the Dijkstra's algorithm can be maintained.

After the introductions of the mainstream pathfinding algorithms, Table 10 summarizes their properties of efficiency, complexity, and accuracy [26,41,45], and allows them to be compared more intuitively. The time complexity depicts the cost of time for an algorithm to run in computer science, usually express as notation O, and often consider the worst-case as the expression standard. On the other hand, the space complexity is considering the worst-case computer memory space occupied condition during algorithm program execution, also using the notation O to express.

Table 10. Shortest path algorithms. Where d is the depth of the solution (the shortest path), and b is the average number of successors per state (branching factor).

Classification	Algorithms	Time complexity	Space complexity	Negative weight edge solution	Negative cycle detection	Label-setting (relaxation)

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

One origin to one destination problem	**Dijkstra's Algorithm**	$O(v^2)$	$O(e)$	No	No	Yes
	A* Algorithm	$O(b^d)$	$O(b^d)$	No	No	Yes
One origin to multiple destinations problem	**Shortest Path Faster Algorithm (SPFA)**	$O(ev)$	$O(e)$	Yes	Yes	Yes
	Bellman-Ford algorithm	$O(ev)$	$O(e)$	Yes	Yes	Yes (label correcting)
Multiple origins and multiple destinations problem	**Floyd-Warshall Algorithm**	$O(v^3)$	$O(v^2)$	Yes	No	No
	Johnson's algorithm	$O(ev + v^2 \log v)$	$O(e + v)$	Yes	No	No
One origin to one destination problem with more than one	**Loopless (without negative weight cycle)**	**Yen's algorithm**	$O(kv(e + v \log v))$	$O(v^2)$	No	No	No

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 August 2020 doi:10.20944/preprints202008.0627.v1

options	**Double-sweep algorithm**	$O(v^3k^3)$		Yes	Yes	No
General	**Martins' delete path algorithm**	$O(e + k\,v\log v)$	$O(kv + m)$	Yes	No	No

Despite the simplicity in the calculation, the time complexity of Dijkstra's Algorithm is high. Correspondingly the algorithms that employ the method of Dijkstra's Algorithm, such as A* algorithm, are having a relatively high. Besides, the space complexity of Dijkstra's algorithm lacks competitiveness among other shortest path algorithms (Table 10). Compare to others, the computation difficulty of the algorithms that solve the one origin to one destination problem is comparatively low, and the efficiency is relatively higher.

As the SPFA is born out of the Bellman-Ford Algorithm, the time complexity of them is the same. In contrast with the Floyd-Warshall Algorithm, Johnson's algorithm has a higher efficiency in terms of time complexity.

2.6. Multi-criteria path computation

The general cognition of the multi-criteria pathfinding is integrating different factors into one cost for each edge and applying the simple optimal path algorithm. When having two or more criteria in the optimal path problem such as distance and cost in a public transport network, the conventional way is to attach edges in the graph with separate weights. It is because ordinary shortest path algorithms for the single objective can only reckon one criterion at a time. The most common multi-criteria pathfinding algorithm is to express each objective or criterion with a weight or percentage according to the user preference, such as 75% for time, and 25% for fare in deriving an optimal public transport route. There is no guarantee to derive the best results for multiple criteria or factors without a standard weight unit.

Given such limitations of traditional pathfinding algorithms, Hansen [21] brought up the study of multi-criteria decision making did not study in depth. The initial multi-criteria shortest path algorithm raised by João Carlos Namorado and Ernesto Queirós Vieira [24] is only feasible for treating a few objectives. This algorithm applies the *k*-shortest path algorithm to both criteria under the same network. It stores the results into a Pareto optimal set, including a set of the shortest paths with an ideal state of the objective. Both are then tested to improve the objective goal without sacrificing the objective of any criterion by using Pareto improvement theory. Due to the *k*-shortest path algorithm, it has no limitation of negative values and loop. The efficiency depends on the type of *k*-shortest algorithm used and the size of the input data.

Based on Hansen's multi-criteria decision-making theory, the algorithm from Martins [30] processes any number of objectives by establishing general conditions to ensure a Pareto optimal path correlates with the temporary label. Later studies of Tung Tung and Lin Chew [42] proposed a method to determine the complete set of Pareto optimal path estimation using multiple labeling schemes. These algorithms have the advantages of high computational efficiency. Depending on the input size and accuracy, the finite number of estimated Pareto optimal paths derives from a polynomial function.

For multi-criteria shortest path algorithms solving multiple objectives problems, the Pareto optimal theory results in an approximate set of optimal paths only. It treats all criteria in the same

 doi:10.20944/preprints202008.0627.v1

order of importance, which may not be ideal in a real-world implementation. Thus, the multi-criteria shortest path algorithms can only consider the diverse criteria in a single network or the unite criteria across multiple networks.

The pathfinding problem combines multi-modal networks criterion, aiming to improve the traffic efficiency, fairness, and costs of the overall transportation network. Considering the user's preference for the transportation modal and valued criteria, the multi-modal pathfinding method was mainly used as the city transportation arrangement and planning [27,32]. The studies of the multi-modal pathfinding often focus on the connectivity checking and costs reducing [31]. This algorithm solves the problem base on multi-modal networks and multi-criteria. However, the recent researches of the multi-modal pathfinding are focusing on the global calculation for the area. Even though applying this method to a single pair of origin and destination, only the low-cost criteria results are available.

3. Limitations and way forward

There are still a lot of challenges for pathfinding algorithmic developments to cater for real-world requirements including multi-criteria or multi-objective, multi-modal involving very different networks [5,7,35], multi-destinations, time-dependent planning [11,29,37], traffic speed, real-time planning and so on [2,28].

In a real-world implementation, the complexity of economic, social, and natural environments requires search results of more than one influencing factor, apart from a mere distance. The functions dealing with single-objective issues, which the shortest path algorithms explained in previous sections, are not satisfying the requirement. Multi-criteria pathfinding thus evolves to be more accessible and practical. In general, most of the shortest path algorithms can only deal with one single network with a single criterion, which far from reality. Even the multi-criteria solutions are just combined and treated as a single criterion in each network.

There is so far no solution to compute an optimal route integrating two different networks, each with a set of varying criteria. For instance, a walking network needs to consider the level of difficulty expressed in gradient, the greenness of the path, and the extent of covered paths for rain-avoidance, whereas a vehicular network is more focused on time, distance and fare. Currently, many commercial and governmental pathfinding applications such as Google, Baidu, Bing tend to provide a solution based on a single network only. Users have to enter a single criterion of time or fare separately each time for an origin-destination pair and have to choose the network type of walking, driving or public transport separately. Even when walking is including in the result of a public transport route, it just depends on the end stops derived from the public transport network, from or to which with the origin and destination to compute an optimal walking route from the walking network separately.

Other studies of multi-criteria and/or multi-network are focused on solving the decision conflict of the resultant routes [20], improving calculation complexity [23,30, 33,46] and figuring time-dependent solutions [1,3,4,6,18,28]. These studies only assume the best solutions for most users being the fastest or cheapest way to the destination.

With improved means of transportation and more options for transportation modes, there is an increasing demand for getting more information to satisfy the varying needs of today's commuters. For example, one may want to find a healthy walking route for 2 kilometers from an origin, followed by a least fare bus route to the destination. The current solution of separating this route request into two networks, each with predefined origin and destination may not be ideal. To have a total solution involving two or more mutually exclusive networks, so the algorithm needs to consider all networks' interdependence before an optimum changing point can be derived.

4. Conclusions

This paper reviews the development of pathfinding algorithms in a classification base on their usage. Includes the shortest path algorithms that solve one origin to one destination problem with the single criterion, one origin to one destination problem with avoiding obstacles need, one origin to one destination problem with more than one options, one origin to multiple destinations problem

with the single criterion, multiple origins and multiple destinations problem with the single criterion, and multi-criteria path computation.

At the same time, this paper critically analyzes those algorithms in multiple dimensions that comprise the computation time and space efficiency and advantages and disadvantages in implementation. Those single network pathfinding methods embody the upgrading request from ever-changing technology and market demand for more convenience solutions. However, the studies of pathfinding methods tend to optimize calculating efficiency, complexity, and accuracy. Most of the optimal path algorithms mentioned above are dedicated to optimizing those calculation related optimizations. Other pathfinding studies aim to fit the pathfinding algorithms to different network environments, such as mobile routes, DTN route, and indoor environment; or to settle the multi-agent dynamic network, which involves with time layer and simultaneously multiple paths planning.

Although the pathfinding algorithms have been developed and used for years that might be powerful for the original purposes, these algorithms also have their disadvantages for other extended applications. As the market demand, optimal pathfinding for a single network can not satisfy the needs of multiple-purpose trips, especially the trip that requests transportation mode change. Combining the requirement of the multi-criteria environment, the multi-network and multi-criteria pathfinding problem drew the attention for study [19]. Current research of the multi-criteria and multi-network pathfinding did not regard the aspect of solving the pathfinding of diverse unit criteria in distinct networks. Therefore, the solution to the multi-network various criteria pathfinding is worth further investigation and also is a way forward.

Author Contributions: Conceptualization, Tong Pan and Shuk Ching Pun-Cheng; Writing-Original Draft Preparation, Tong Pan; Writing-Review & Editing, Tong Pan and Shuk Ching Pun-Cheng; Supervision, Shuk Ching Pun-Cheng; Project Administration, Shuk Ching Pun-Cheng. All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Abbaspour, R.; Samadzadegan, F. A solution for time-dependent multimodal shortest path problem. *Journal of Applied Sciences*. **2009**, *9*, 3804-3812.
2. Abbaspour, R. A.; Samadzadegan, F. An evolutionary solution for multimodal shortest path problem in metropolises. *Comput. Sci. Inf. Syst*. **2010**, *7*, 789-811.
3. Abdelgawad, H.; Abdulhai, B.; Wahba, M. Multiobjective optimization for multimodal evacuation. *Transportation Research Record*. **2010**, *2196*, 21-33.
4. Aifadopoulou, G.; Ziliaskopoulos, A.; Chrisohoou, E. Multiobjective optimum path algorithm for passenger pretrip planning in multimodal transportation networks. *Transportation Research Record*. **2007**, *2032*, 26-34.
5. Alajlan, M.; Koubâa, A.; Châari, I.; Bennaceur, H.; Ammar, A. Global path planning for mobile robots in large-scale grid environments using genetic algorithms, Paper presented at the 2013 International Conference on Individual and Collective Behaviors in Robotics - Proceedings of ICBR 2013; 2013.
6. Androutsopoulos, K. N.; Zografos, K. G. Solving the multi-criteria time-dependent routing and scheduling problem in a multimodal fixed scheduled network. *European Journal of Operational Research*. **2009**, *192*, 18-28.
7. Balasubramanian, A.; Levine, B. N.; Venkataramani, A. DTN routing as a resource allocation problem. *Computer Communication Review*. **2007**, *37*, 373-384.
8. Bellman, R. On A Routing Problem. *Quarterly of Applied Mathematics*. **1958**, *16*, 87-90.
9. Bertsekas, D. P. *Linear network optimization : algorithms and codes*; Cambridge, Mass.: MIT Press; 1991.
10. Bertsekas, D. P. *Network optimization : continuous and discrete models*; Belmont, Mass.: Athena Scientific; 1998.
11. Boccaletti, S.; Bianconi, G.; Criado, R.; del Genio, C. I.; Gómez-Gardeñes, J.; Romance, M.; . . . Zanin, M. The structure and dynamics of multilayer networks. *Physics Reports*. **2014**, *544*, 1-122.
12. De Queirós Vieira Martins, E. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*. **1984**, *18*, 123-130.
13. Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*. **1959**, *1*, 269-271.

14. Duan, F. A Faster Algorithm for Shortest-Path--SPFA. *Journal of Southwest Jiaotong University*. **1994**, *29*, 207-212.
15. Evans, J. R.; Minieka, E. *Optimization algorithms for networks and graphs*, 2nd ed.; Hong Kong: M. Dekker., New York, 1992.
16. Floyd, R. W. Algorithm 97: shortest path. *Communications of the ACM*. **1962**, *5*, 345.
17. Ford, L. R. Network flow theory. Santa Monica, Calif, 1956.
18. Gräbener, T.; Berro, A.; Duthen, Y. Time dependent multiobjective best path for multimodal urban routing. *Electronic Notes in Discrete Mathematics*. **2010**, *36*, 487-494.
19. Hai-cong, Y.; Feng, L. A multi-modal multi-criteria route planning method based on genetic algorithm. *Acta Geodaetica et Cartographica Sinica*. **2014**, *43*, 89-96.
20. Haicong, Y.; Feng, L. A Multi-criteria Route Planning Approach Considering Walking Guidance. *Journal of Image and Graphics*. **2010**, *4*.
21. Hansen, P. Bicriterion path problems. In *Multiple Criteria Decision Making: Theory and Application*. In: Springer-Verlag, Berlin, Germany; 1980.
22. Hart, P. E.; Nilsson, N. J.; Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*. **1968**, *4*, 100-107.
23. Huang, B.; Fery, P.; Xue, L.; Wang, Y. Seeking the Pareto front for multiobjective spatial optimization problems. *International Journal of Geographical Information Science*. **2008**, *22*, 507-526.
24. João Carlos Namorado, C.; Ernesto Queirós Vieira, M. A bicriterion shortest path algorithm. European *Journal of Operational Research*. **1982**, *11*, 399-404.
25. Johnson, D. B. Efficient Algorithms for Shortest Paths in Sparse Networks. *J. ACM*. **1977**, *24*, 1-13.
26. Lamarche, F.; Donikian, S. Crowd of virtual humans: A new approach for real time navigation in complex and structured environments. *Computer Graphics Forum*. **2004**, *23*, 509-518.
27. Litman, T. *Introduction to multi-modal transportation planning*. Canada: Victoria Transport Policy Institute; 2017.
28. Lozano, A.; Storchi, G. Shortest viable path algorithm in multimodal networks. *Transportation Research Part A: Policy and Practice*. **2001**, *35*, 225-241.
29. Luan, X.; Miao, J.; Meng, L.; Corman, F.; Lodewijks, G. Integrated optimization on train scheduling and preventive maintenance time slots planning. *Transportation Research Part C: Emerging Technologies*. **2017**, *80*, 329-359.
30. Martins, E. Q. V. On a multicriteria shortest path problem. *European Journal of Operational Research*. **1984**, *16*, 236-245.
31. Mishra, Sabyasachee, Welch, Timothy F; Jha, Manoj K. Performance indicators for public transit connectivity in multi-modal transportation networks. *Transportation Research Part A*. **2012**, *46*, 1066-1085.
32. Modesti, P.; Sciomachen, A. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *European Journal of Operational Research*. **1998**, *111*, 495-508.
33. Mooney, P.; Winstanley, A. An evolutionary algorithm for multicriteria path optimization problems. *International Journal of Geographical Information Science*. **2006**, *20*, 401-423.
34. Moore, E. F. 1959. The Shortest Path Through a Maze: Bell Telephone System. Cambridge: Harvard University Press, Proceedings of an International Symposium on the Theory of Switching, Cambridge, Massachusetts, 2–5 April; 1957.
35. Nguyen, Q. H.; Vu, H.; Tran, T. H.; Nguyen, Q. H. Developing a way-finding system on mobile robot assisting visually impaired people in an indoor environment. *Multimedia Tools and Applications*. **2017**, *76*, 2645-2669.
36. Roy, B. Transitivité et connexité. *Comptes Rendus Hebdomadaires Des Seances De L Academie Des Sciences*. **1959**, *249*, 216-218.
37. Sharon, G.; Stern, R.; Felner, A.; Sturtevant, N. R. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*. **2015**, *219*, 40-66.
38. Shier, D. Computational experience with an algorithm for finding the k shortest paths in a network. *Journal of Research of the National Bureau of Standards, Section B: Mathematical Sciences*. **1974**, *78B*, 139-116.
39. Shier, D. R. Iterative methods for determining the k shortest paths in a network. *Networks*. **1976**, *6*, 205-229.
40. Shimbel, A. Structure in communication nets. *Proceedings of the Symposium on Information Networks*. **1954**, 119-203.

41. Tharwat, A.; Elhoseny, M.; Hassanien, A. E.; Gabel, T.; Kumar, A. Intelligent Bézier curve-based path planning model using Chaotic Particle Swarm Optimization algorithm. *Cluster Computing*. **2018**, 1-22.
42. Tung Tung, C.; Lin Chew, K. A multicriteria Pareto-optimal path algorithm. *European Journal of Operational Research*. **1992**, *62*, 203-209.
43. Warshall, S. A Theorem on Boolean Matrices. *J. ACM*. **1962**, *9*, 11-12.
44. Yen, J. Y. Finding the K Shortest Loopless Paths in a Network. *Management Science*. **1971**, *17*, 712-716.
45. Zimmerman, P. M. Single-ended transition state finding with the growing string method. *Journal of Computational Chemistry*. **2015**, *36*, 601-611.
46. Zitzler, E.; Deb, K.; Thiele, L. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*. **2000**, *8*, 173-195.

Bibliography

1. Eppstein, D. Finding the k shortest paths. *SIAM Journal on computing*.**1998**, *28*, 652-673.
2. Fox, B. Kth shortest paths and applications to the probabilistic networks. *ORSA/TIMS Joint National Mtg*. **1975**, *23*, B263.
3. XU Tao, DING Xiao-lu, & LI Jian-fu. Review on K Shortest Path Algorithms. *Computer Engineering and Design*. **2013**, *34*, 3900-3906