Three popular shortest path algorithms and their applications in real life

Thi Hong, Dang [Student id: 103140022]

Master student, Swinburne University of Technology, Melbourne, Victoria, Australia

Abstract. Finding the shortest path between two graphical vertices is one of the most popular problem in any type of graph: undirected, directed, or mixed graph[1]. There are many solutions for this problem and the algorithms solving this problem is widely used to many areas including road network, network routing, artificial intelligence, game design, and so on. This paper introduces three basic algorithms and compares them theoretically by analysing the time and space complexity, experimentally by a small python application. Then, the paper compares their performances and summaries the best application range. At the end it introduces some application of shortest path algorithms in real life problem.

1. Introduction

A path is considered shortest path when the sum of the weights of its constituent edges is minimised and finding this path is the original of the shortest path algorithm. The mentioned weight can be any kind of measurement. For example, in the graph represents road network, weight is the distance between two points, but in communication networks, it is reliability measure or time connection between 2 nodes. According the characteristics of the graph, the shortest path problem can be divided into three types: **the single-source shortest path problem (SSSPP)**, in which we have to find shortest paths from a source vertex *v* to all other vertices in the graph; **the single-destination shortest path problem(SDSPP)**, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex *v*; and, **the all-pairs shortest path problem (APSPP)**, in which we have to find shortest paths between every pair of vertices *v*, *v′* in the graph. Since second problem(**SDSPP)** can be reduced to the single-source shortest path problem by reversing the arcs in the directed graph, this paper only introduces the first and the third shortest path problems[1].

There are three popular algorithms that are the basic solutions for the single-source shortest path problem and the all-pairs shortest path problem**.** The classical single source shortest path algorithm Dijkstra's algorithm, which was conceived by computer scientist Edsger W. Dijkstra in 1956, is in the widespread used in road networks. Later, Richard Bellman and Lester Ford published the Bellman-Ford algorithm in 1958 and 1956, which has a significant difference with Dijkstra's algorithm, it can be used on graphs with negative edge weights, as long as the graph contains no negative cycle reachable from the source vertex [3]. In 1962, the Floyd-Warshall algorithm could find shortest paths in a weighted graph with positive or negative edge weight, which is a typical multi-source shortest path.The rest of this paper is organised as following: The first portion will introduce three algorithms which are mentioned above. Then the second section will compare the performances and the present the most suitable field of three algorithms, like in the time and space complexity. Next, there is an application to represents the differences of three algorithms. The last part of the paper is a summary.

2. The three algorithms

2.1 The Dijkstra algorithm

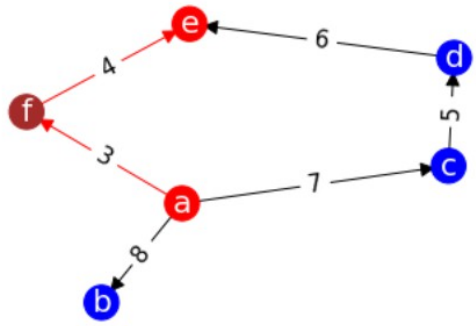
The Dijkstra algorithm is used to find a single node as the source node and calculate the shortest paths from the source to all other nodes in the graph, increasing node by node to get a shortest path tree. Here is the algorithm [3]:

Step 1: Mark all nodes unvisited and store them.

Step 2: Set the distance to zero for our initial node and to infinity for other nodes.
Step 3: Select the unvisited node with the smallest distance, it's current node now.
Step 4: Find unvisited neighbours for the current node and calculate their distances through the current node. Compare the newly calculated distance to the assigned, save the smaller one, update predecessor mapping. *For example, if the node A has a distance of 6, and the A-B edge has length 2, then the distance to B through A will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8.*
Step 5: Mark the current node as visited and remove it from the unvisited set.
Step 6: Using the derived distances and predecessor tables to get shortest path

Below are how Dijkstra works in different scenarios
(*Node(Red: start/end point, Brown: on the path, Blue: normal node), Edge(Red: on the path, black: normal)*)

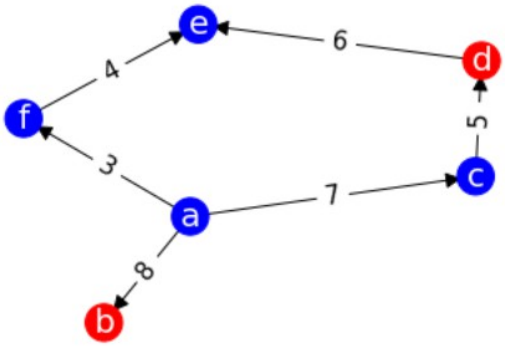
Case 1: There is a shortest path



Distances					
a	b	c	d	e	f
8	0	15	20	15	11

Predecessor					
a	b	c	d	e	f
b	None	a	c	f	a

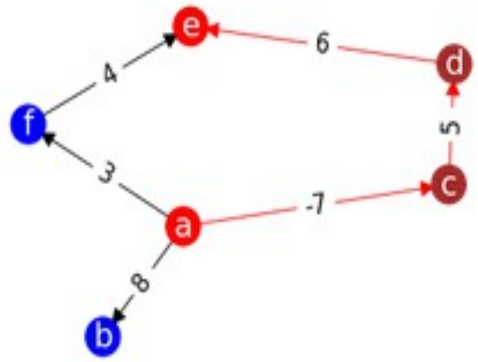
Case 2: There is a no path



Distances					
a	b	c	d	e	f
inf	inf	inf	inf	inf	inf

Predecessor					
a	b	c	d	e	f
None	None	None	None	None	None

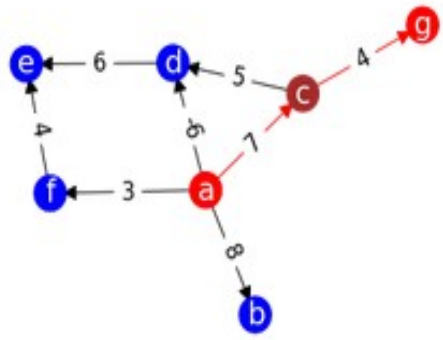
Case 3: There is negative value



Distances					
a	b	c	d	e	f
0	8	-7	-2	4	3

Predecessor					
a	b	c	d	e	f
None	a	a	c	d	a

Case 4: There is negative cycle



Distances						
a	b	c	d	e	f	g
0	8	7	-6	0	3	11

Predecessor						
a	b	c	d	e	f	g
None	a	a	c	d	a	c

Basically, in the heart of Dijkstra algorithm, distances and predecessor tables must be calculated to find the shortest path. In the first case, it is very easy to find the shortest path from b to d by looking at the distances and predecessor tables. Looking into details, distances table shows that the shortest distance from b to d is 20(weight) and predecessor table shows that: to get to d, it has to go c; to get to c, it has to go to a; to get to a, it has to go to b.

Dijkstra can handle well even though there is negative weight in the graph except if there is a negative cycle. This is Dijkstra's hugest drawback. Looks closely to the fourth case. It is clearly to see that the correct path is a ->d, d->c, c->g. In this scenario, both distances and predecessor tables were corrupted by the negative value of edge connecting a and d (-6).

That is why we need an algorithm that can work well with negative weight and it still can give us the shortest path. In the next section, Bellman–Ford algorithm will be introduced as a way to deal with this troublesome.

2.2 The Bellman–Ford algorithm

Just like Dijkstra's algorithm, Bellman-Ford algorithm also is used to find a single node as the source node and calculate the shortest paths from the source to all other nodes in the graph but it can handle negative weight. Here is the algorithm [4]:

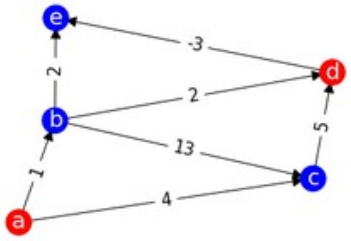
Step 1: Mark all nodes unvisited and store them.
Step 2: Set the distance to zero for our initial node and to infinity for other nodes.
Step 3: Find all neighbours for the current node and calculate their distances through the current node. Compare the newly calculated distance to the assigned, save the smaller one, update predecessor mapping
Step 4: Check for negative weight cycles. If there is, stop here.
Step 5: Using the derived distances and predecessor tables to get shortest path

Here two example of Bellman–Ford with and without negative cycle
(*Node(Red: start/end point, Brown: on the path, Blue: normal node), Edge(Red: on the path, black:*

normal))

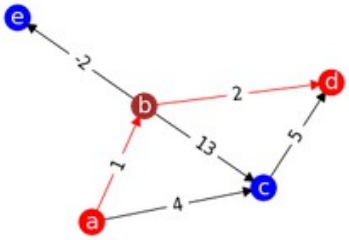
- There is a negative cycle

```
distances {'a': 0, 'c': 4, 'b': 1, 'e': 3, 'd': 3}
previous_vertices {'a': None, 'c': 'a', 'b': 'a', 'e': 'b', 'd': 'b'}
Graph contains negative weight cycle at d e
shortest path:  []
```



- There is no negative cycle

```
distances {'a': 0, 'c': 4, 'b': 1, 'e': -1, 'd': 3}
previous_vertices {'a': None, 'c': 'a', 'b': 'a', 'e': 'b', 'd': 'b'}
shortest path:  [('a', 'b'), ('b', 'd')]
```



So what Bellman-ford has done better than Dijkstra is it can detect negative cycle, however it does not mean it can return correct shortest path in that particular case. Still, not wrong is better than wrong. The third and last algorithm, Floyd-Warshall, that is introduced in this paper, can give the shortest path even there is a negative cycle.

2.3 The Floyd–Warshall algorithm

While two above algorithms Dijkstra and Bellman-ford only calculate shortest distance of two give points , the Floyd–Warshall algorithm compares and find all possible paths through the graph between each pair of vertices, namely, it calculates the shortest path between all nodes.

The basic idea of Floyd algorithm has four steps [5]:

Step1: Find two vertices from the network, put each vertex in the network into these two points as an intermediate.

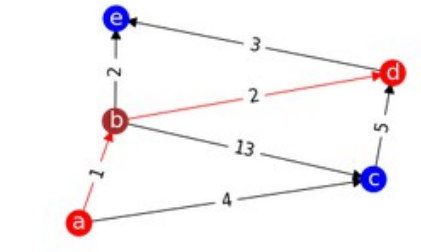
Step2: Compare the original distance with the new distance between these two points, treat the smaller distance as the new shortest distance.

Step3: Sequentially construct n matrix S(1), S(2), □ , S(n) by looping iteration, the each element in the last matrix S(n) represents the shortest distance between the two points. Figure 3 and Table 2 show the detailed process.

Step4: Get the minimum distance of one point to the other points by summing the elements in each lines of S(n), one or more best locations can be found by comparing these summations.

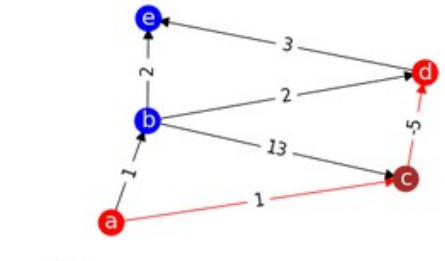
Here two example of Floyd–Warshall with and without negative cycle
(*Node(Red: start/end point, Brown: on the path, Blue: normal node), Edge(Red: on the path, black: normal)*)

```
     a     b    c    d    e
a  inf   inf  inf  inf  inf
b  1.0   inf  inf  inf  inf
c  4.0  13.0  inf  inf  inf
d  3.0   2.0  5.0  inf  inf
e  3.0   2.0  8.0  3.0  inf
      a     b     c     d     e
a  None  None  None  None  None
b     b  None  None  None  None
c     c     c  None  None  None
d     b     d     d  None  None
e     b     e     d     e  None
shortest path:  [('a', 'b'), ('b', 'd')]
```



In case there is a cycle negative

```
      a     b    c    d    e
a  inf   inf  inf  inf  inf
b  1.0   inf  inf  inf  inf
c  1.0  13.0  inf  inf  inf
d -4.0   2.0 -5.0  inf  inf
e -1.0   2.0 -2.0  3.0  inf
      a     b     c     d     e
a  None  None  None  None  None
b     b  None  None  None  None
c     c     c  None  None  None
d     c     d     d  None  None
e     c     e     d     e  None
shortest path:  [('a', 'c'), ('c', 'd')]
```



Above example gives an impression that Floyd–Warshall can solve the shortest path problem better than the other mentioned algorithms. However, we need to check the efficiency comparison to get the final conclusion.

3. Efficiency comparison

3.1. Big O notation

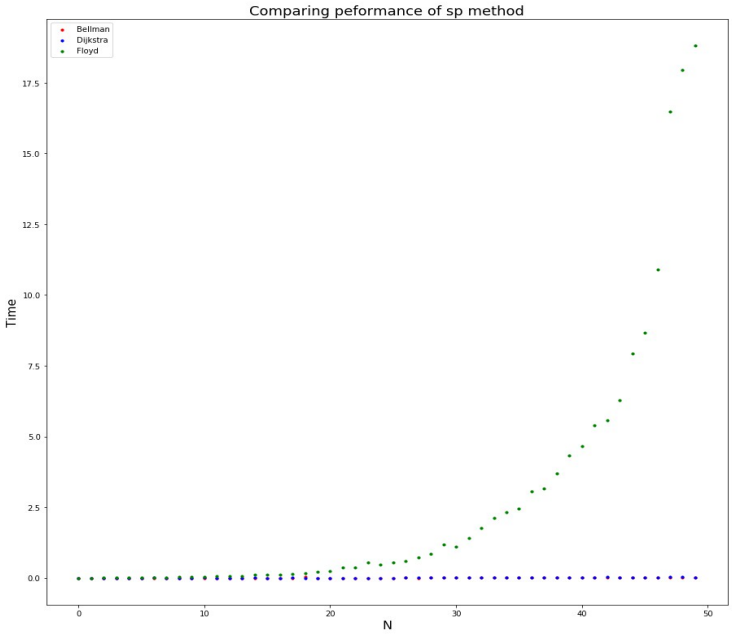
Where *V* is the number of vertices and *E* is the number of edges in the graph, the complexity of three algorithms are:

Name	Time complexity
Dijkstra	O(E + V) (binary heap)[1] O(E*V)(no heap, no priority queue)
Bellman-Ford	O(E*V)
Floyd–Warshall	O(V^3)

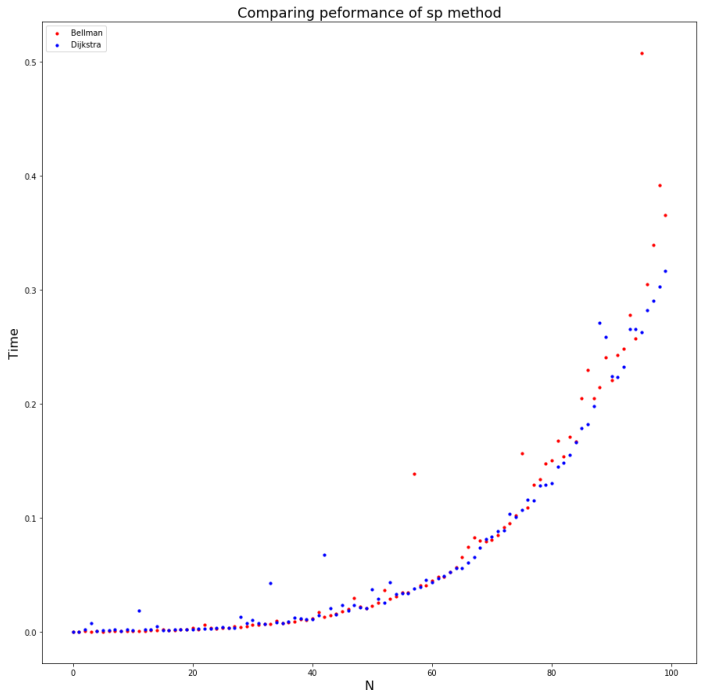
The Big O notation says that Dijkstra's algorithm is the fastest one and Floyd-Warshall is the slowest one. In the next section, the python application will prove this.

3.2. Experiment

Running three algorithms with differences number of vertices (from 1- 50) shows that Floyd-Warshall started to be slow exponentially when the number of vertices is growing.



After removing Floyd-Warshall from the picture, only comparing Dijkstra and Bellman-ford, visually, it gives that Bellman-ford is slightly slower than Dijkstra's (the red dots lie above blue dots in most cases after n >40).



These above graphs reassure the big O notation hypothesis. The analysis shows that there is always a trade off when picking an appropriated algorithm. Even though, Dijkstra is the fastest, if you run Dijkstra's algorithm n times, on n different vertices, you will have a theoretical time complexity of O(n* n2)=O(n3). In other words, if you use Dijkstra's algorithm to find a path from every vertex to every other vertex you will have the same efficiency and result as using Floyd's algorithm. Also, Floyd's algorithm is much easier to implement. Lets dive into some real life problem to see when and how to use which.

4. Applications of three algorithm in real life

There are many application of shortest path algorithms in daily life[5]. The most popular one could be network routing. To route between network nodes, network has to determines the outgoing route of the received packet. Figuratively, network is graph of billion of routers and all routers are connected by communication lines, in graph language, they are edges. In order to select the routing among a pair of routers, there is the need of finding the shortest path in the diagram. Even though, Dijkstra is the fastest algorithm for calculating the shortest path of a router to other routers, but if the problem states that there is a need of finding shortest path from random node network to other node, Dijkstra is slow since it always has to calculate its distance and predecessor tables. On the other hand, after creating distance and predecessor matrix, Floyd-Warshall has a better performance. That is why in real-life, Floyd-Warshall is picked to solve network routing problem.

Dijkstra still can be shine in another computer networking problem. Modern computer networks usually use dynamic routing algorithms, namely link state routing algorithm and distance vector routing algorithm. Link state routing protocol collects all kinds of information of the whole network, which constitute a topological database of routers. Open Shortest Path First (OSPF) is a typical protocol which is an internal gateway protocol which used to make routing decision within a single autonomous system. Besides, it is a specific implementation of Dijkstra algorithm. It mainly uses the algorithm to generates a tree without loops. Then starting from a router and passing the information to all the routers in the tree. Thus, all the routers is shared by all the state of the link. Each router is calculated in the local routing and avoids updating the routing table blindly.

The distance vector routing algorithm is that each router maintains a table. The table has the best path and route for each destination through exchanging the information with neighbouring routers to update the table information. RIP protocol is a dynamic routing protocol and uses the Bellman-Ford algorithm. The process of routing announcement is the process of the Bellman-Ford algorithm's implementation. The routers collect all different paths to the destination and save the number of sites about information of each destination path. Any other information will be discarded, except the best route to the destination. The algorithm is distributed execution. All the routers are in the execution of the algorithm and the results are calculated together by all machines. In OSPF, the algorithm only executes on one machine which is not distributed.

Collaboration between algorithms can also be an option. For example, using both the Dijkstra algorithm and the Floyd-warshall algorithm in an application that shows the availability of ambulance and accurate information about victims and road conditions in an accident to help the first aid process for victims or patients. This application applies Dijkstra's algorithm to determine the fastest travel time to the nearest hospital. The Floyd-warshall algorithm is implemented to determine the closest distance to the hospital. Data on some nearby hospitals will be collected by the system using Dijkstra's algorithm and then the system will calculate the fastest distance based on the last traffic condition using the Floyd-warshall algorithm to determine the best route to the nearest hospital recommended by the system. This application is built with the aim of providing support for the first handling process to the victim or the emergency patient by giving the ambulance calling report and determining the best route to the nearest hospital.

5. Conclusion

The shortest path issue is still one of the hottest topics in the research field. This paper introduces the basic principle of three shortest path algorithms and compares them by analyzing the time and space complexity. Besides, it summaries the application range of different algorithms.

- The Dijkstra algorithm is the fastest algorithm comparison with others, however, it fails when graph carries negative weight.
- The Bellman-Ford algorithm can be a supplementary for Dijktra when there is a negative loop. However, it only does negative loop detection.
- The Floyd-Warshall the slowest algorithm among three but it can find shortest path from every node in the graph.

In many practical applications, these three algorithms are applied combinationally to maximise efficiency.

References

[1] Shortest path problem - https://en.wikipedia.org/wiki/Shortest_path_problem
[2] Li, T., Qi, L., & Ruan, D. (2008, November). An efficient algorithm for the single-source shortest path problem in graph theory. In Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on (Vol. 1, pp. 152-157). IEEE.

[3] https://dev.to/mxl/dijkstras-algorithm-in-python-algorithms-for-beginners-dkc

[4] https://www.programiz.com/dsa/bellman-ford-algorithm
[5] The Comparison of Three Algorithms in Shortest Path Issue, Xiao Zhu Wang 2018 *J. Phys.: Conf. Ser.* **1087** 022011