A Survey of Trilogy Shortest Path Algorithms

Shaista Sarwar, Laiba Shaheen

Abstract—Shortest path problem is one of the most classical problem in graph theory, aiming to discover the shortest path between two nodes in a graph. In this problem, we have to find the minimum-cost tracks or shortest paths between the starting node and final destination in a given graph.This-work gives a brief introduction of the most famous algorithms of the shortest path problem i.e. Dijkstra's, Bellman-Ford, and Floyd Warshall algorithm. A comparative analysis of these algorithms is performed based on their advantages, disadvantages, and efficiency, and application areas.

Index Terms—Dijkstra; Shortest path; Floyd; Bellman

1 Introduction

GRAPH is made out of nodes and these nodes are connected by edges and each of these edges has a number next to them and that number is a weight, the bigger the weight is the more expensive it is to get from one node to the other node that the edge connects to[1]. In graph theory, there is a problem to discover the least expensive path between nodes which is called the shortest path problem. In this problem, we have tofind the minimum-cost tracks or shortest paths between the starting node and final destination in a given graph[2]. Several algorithms exist for the shortest path problem but here we discuss only the most popular and competitive algorithms of the shortest path problem, and which are as follow:

1. Dijkstra's Algorithm
2. Bellman-Ford Algorithm
3. Floyd Warshall Algorithm

In this survey, our research contribution can be summarized as follows:

- Critical review on each algoritm by sharing its procedure, main features , applications and limitations.
- Comparisons of the existing shortest path algorithms for better understanding.
- And describing which algorithm work best in which scenario.
- Running the algorithms on different graphs and view results.

The rest of the paper is divided into three main sections. Section 2 discusses the basic concept related to the three classical algorithm. Section 3 presents the Comparative analysis of these three algorithms. In the end, section 4 we discuss the conclusion.

2 Three Classical Algorithms

2.1 Dijkstra's Algorithm

Edsger Dijkstra was a Dutch computer scientist who presents Dijkstraan algorithm in 1959, in which he finds the minimumroute from the source node to all other nodes in a given graph[3]. He develops Dijkstra's algorithm to address one of the optimization problems i.e. single-source shortest path problem. Dijkstra algorithm work on the directed weighted graph in which the values of the edges are always positive[4]. The conception of the Dijkstra algorithm: let G be a directed, weighted graph in which there is the number of nodes $v \in G.V$ and these nodes are connected by edges $(u, v) \in G.E$ and each of these edges have a number next to them and that number is a weight, the bigger the weight is
more expensive it is to get from one node to the other node that the edge connects to.
In the Dijkstra algorithm, we manage two sets of vertices i.e. Q and P.P contains all visited nodes. Atthestart, Q contains all vertices and P is empty.The set Q contains all non-visited vertices/nodes and the set In Dijkstra's algorithm, we maintain two attributes for each vertex i.e. v.d and $v.\pi$. v.d shows the shortest path estimate from the source s to vertex v and $v.\pi$ gives the predecessor of v. At each step, the smallest entry from set Q is removed and added to our visited node-set P. Whenever the node is removed from Q and added in P, then we compute $\delta(u, v)$ that node with their adjacent nodes which are called as neighbors of that selected node. For all vertices $\delta(u, v)$ are computed and algorithm stop when allnodes in the graph are visited[5,6].

2.1.1 Algorithm Procedure

Following is a detailed note of each step of Dijkstra'salgorithm.

Inputs:

1. Graph G with vertices V and edges E.
2. Weight of each edge w (u,v).
3. A source node $s \in G.V$

Output: the shortest path weight solution set P from source s node to all other nodes.

Algorithm:

Step 1: In step 1, we initialize all vertices in a given graph by setting $v.d = \infty$ and $v.\pi$ = NIL. For sources s we set s.d= 0.

Step 2: we initialize two sets i.e. P and Q for visited and non-visited vertices. At the start, P is empty and inserts all G.V vertices in Q.

Step 3: now we iterate the following points until Q becomes empty.

a. First, we extract a vertex v from Q whose v.d is minimum among all of them. And add this vertex in set P.

b. Let's name the extracted vertex as curr_vertex (current vertex). Now check the neighbors of current

- *Shaista Sarwar is with Dept of Computer Science & Information Technology ,University of Sargodha, Sargodha.*
  E-mail: shaistasarwar193@gmail.com.
- *Laiba Shaheen is with Dept of Computer Science & Information Technology ,University of Sargodha, Sargodha.*
  E-mail: laibashaheen0@gmail.com

vertex (neighbors also called adjacent nodes of current vertex) and name it as neighbor_vertex and for each neighbor vertex check if neighbor_vertex.d>curr_vertex.d + w (curr_vertex, neighbor_vertex) then set the neighbor_vertex.d = curr_vertex.d + w (curr_vertex, neighbor_vertex) and neighbor_vertex.π = curr_vertex.
Compute shorted path estimate and predecessor for each neighbor of the current vertex.

c. Repeat a and b until all vertices are visited and when Q = Φ the algorithm stops.

The algorithm return solution set P which provides us the shortest path weights from source to every single vertex in a given graph.

2.1.2 Detailed Example

Here is a simple example that shows the working of Dijkstra's algorithm in detail.
Input: a directed graph with non-negative weights and a source vertex is given.

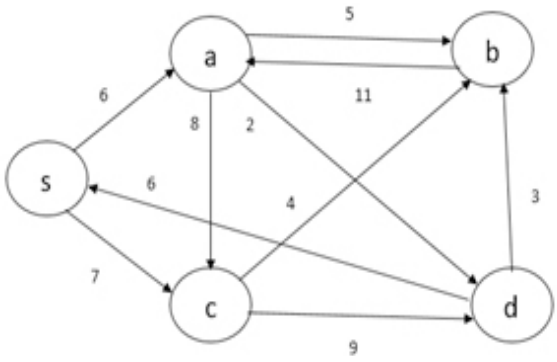

Fig. 1. An Example of Dijkstra Algorithm

Find out: we have tocompute the shortest path weight set P.
Solution: following table shows the working of the Dijkstra algorithm.

TABLE 1
ITERATION OF DIJKSTRA ALGORITHM

Iterations	Visited Nodes	s.d / s.π	a.d/ a.π	b.d/ b.π	c.d/ d.π	d.d/ d.π
Initialization		0/ -	∞/ -	∞/ -	∞/ -	∞/ -
1	{s}	0/ -	6/s	∞/ -	7/s	∞/ -
2	{s,a}	0/ -	6/s	11/a	7/s	8/a
3	{s,a,c}	0/ -	6/s	11/a	7/s	8/a
4	{s,a,c,d}	0/ -	6/s	11/a	7/s	8/a
5	{s,a,c,d,b}	0/ -	6/s	11/a	7/s	8/a

2.2 Bellman-Ford Algorithm

To get a short distance from a starting node to all destinations in the graph that has negative weights we are using a bellmen-ford algorithm. We use the Bellmen ford algorithm to compute the -ve cycle exists in the graph or not and if a graph having a -ve-weight cycle then it will make various from the initial point to the endpoint, in which every cycle will decrease the rate of shortest distance. Because of that factor, let us assume that our graph has no -ve weight cycles the array dist[] will maintain a minimum length starting the first position to the other nodes. This algorithm contains many sections, in which every step wants to decrease the cost from the entire edges by substituting dist[*n*] the statement dist[*m*] + *e*; *m* and *n* are the graph nodes, and e will be the related edge that connects both nodes. Graph required an n - 1 sectionto compute the value of all the shortest distances, but for those inaccessible graph values, the amount to the array will be assigned to infinity[7].
A boolean value is returned by the algorithm that specifying that either there is a -ve weight cycle present that is retrieved by-source or not. If the graph has no cycle then the shortest distance is returned by the algorithm, but if the graph contains a -ve cycle then the algorithm does not return the shortest path. The solution is their Bellman-Ford algorithm can perceive -ve cycles and define that they exist. These algorithms return esteem that the negative cycle is available or not and return the shortest-path. This algorithm finds the shortest path in a bottom-up manner. The algorithm returns true if the graph does not consist of any negative weight cycles accessible to the source.
The elementary composition of bellman for dislike Dijkstra's algorithm, althoughin its place of greedy choosing the minimum-weight vertex not still handled to relaxing, it merely relaxes entire edges and performs this |N| - 1 time, where |N| represents the number of nodes in the graph[8]. Repetition allows small gaps to spread precisely across the entire graph, because, in the lack of -ve cycles, a very short path visits every node at a time.

2.2.1. Algorithm Procedure

Following is a detailed note on each step of the bellman-ford algorithm.
Inputs:

1. Graph G with vertices V and edges E.
2. Weight of each edge w (u,v).
3. A source node s ∈ G.V

Output: the shortest path weight solution set P from source s to all other nodes.

Algorithm:
Step 1: In this step, we initialize all vertices in a given graph by setting v.d= ∞ and v.π = NIL. For sources s we set s.d= 0.
Step 2: now we start iterating the vertices and inside this iteration also iterate each edge by their weight by checking that either there is less cost to reach from one node to another if we find the minimum cost then we update the node with this cost.
Step 3: Now after updating nodes with their minimum cost we again make an iteration to check either there is a -ve weight cycle exist in the graph or not if we find -ve cycle the algorithm returns False otherwise it returns True.

2.2.2 Detailed Example

Input: a graph with non-negative weights and a source vertex is

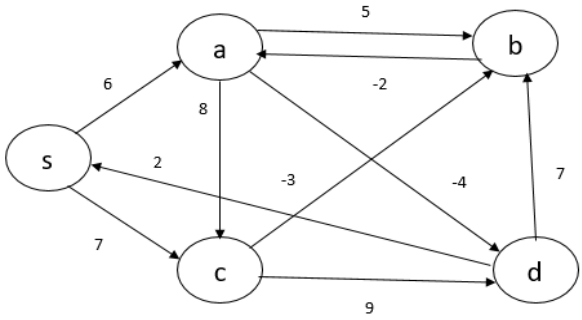

Fig. 2. An Example of Bellman Ford Algorithm

Findout: we have to find out the shortest path from the
Find out: we have to compute the shortest path weight set P.
Solution: following table shows the working of the bellman ford algorithm.

TABLE 2
ITERATION OF BELLMEN-FORD ALGORITHMS

Iterations	s.d/ s.π	a.d/ a.π	b.d/ b.π	c.d/ d.π	d.d/ d.π
Initialization	0/ -	∞/ -	∞/ -	∞/ -	∞/ -
1	0/ -	6/s	4/c	7/ s	2/a
2	0/ -	2/b	4/c	7/s	2/a
3	0/ -	2/b	4/c	7/s	-2/a
4	0/ -	2/b	4/c	7/s	-2/a

2.3 Floyd Warshall Algorithm

Floyd-Warshall algorithm is used to determine the shortest distance among the entire sets of nodes on agraph having +ve or -ve weights on edges. A matrix of square length is given as its input to the algorithm. The matrix specifies the distance of each vertex without any intermediate nodes called the distance or length matrix. This matrix containing the length in the matrix if an edge exists among vertex *a* and *b*.At matrix diagonal it contains zeros and If it has no edge among the ends *a* and *b*, there is a place (*a, b*) containing infinity value. That matrix is recalculated in each iteration of the algorithm[9].

For this, this keeps a record of the shortest distance among any two nodes, by using a subsection of a whole set of nodes as middle steps beside the path. A matrix, formed for the initial iteration of the process, consists of paths between all nodes that use a single (previously defined) intermediate node. It contains distances applying two pre-defined intermediary nodes. After All, the matrix employs "N" intermediary nodes[10]. This method can be explained by applying the following recursive formula:

D_{ij}^{n}= *minimum* (D_{ij}^{n-1}, $D_{ik}^{n-1} + D_{kj}^{n-1}$) [11]

The algorithm runs and determining the shortest distance (a, b, c) for each pair of (a, b) for, c = 1 and then c = 2, etc. This procedure runs continuously till c = n, after which determines the shortest distance for each (a, b) pairs by applying any of the middle nodes. To retrieve short paths between all nodes, you must make another matrix during the operation of the matrix, this matrix is utilized to store the shortest distance.

If we have graph G, in which each node is labeled from 1 to n. The shortest distances from vertex *a* to vertex *b* are represented by Notation d_{ab}^{c}, which is as well goes over the *c* vertex. So, if there is an edge among vertex *a* and *b* that will be equivalent to d_{ab}^{0}, else it can be valued as infinity[11].

Though for some values of d_{ab}^{c} here we have two options: (a) If the shortest distance from *a* to *b* does not exceed node*c* the d_{ab}^{c} value will be equivalent to d_{ab}^{c1}. (b) If the shortest distance as of vertex *a* to *b* reaches from vertex so, firstly it reaches from vertex *a* to *c*, and then from vertex *c* to *b*. so for that the value of d_{ab}^{c} will be equivalent to $d_{ac}^{c-1} + d_{cb}^{c-1}$.To find the shortest distance we have to calculate the least cost between them this is represented by the following two statements:

D_{ab}^{0}= the distance of edge among the nodes a and b

$d_{ab}^{c} = \min (d_{ab}^{c-1}, d_{ac}^{c-1} + d_{cb}^{c-1})$

2.3.1. Algorithm Procedure

Following is a detailed note of each step of the Floyd-warshall algorithm.

Inputs:

2. Graph G with vertices V and edges E.
3. Weight of each edge w (u, v).
4. A source node s ∈ G.V

Output: the shortest path weight solution set P from source s node to all other nodes.

Algorithm:

Step 1: First we have to make two square matrices A and B. A matrix store the distance and B store path, now, make iteration to calculate the distance we have to modify the matrix from the starting node to the intermediary point, then determine the shortest distance among every two node so, obviously,

Step 2: After that, we have to modify the matrix by using the second node as the intermediary point. Given that the shortest distance from node *a* to node *b* through the first node is availableand also stores the path in the B matrix.

Step 3: Repeat step 2 till Nth nodes, and then we determine the shortest distance among each vertex.

2.3.2 Detailed Example

Here is a simple example that shows the working of the Floyd Warshall algorithm in detail.

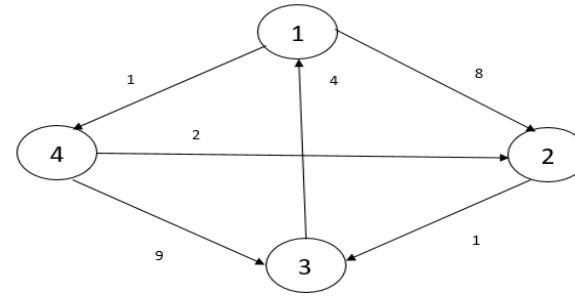

Fig. 3. A Example of Floyd-Warshall Algorithm

Step1: initialization: (k=0)

D

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	∞	0	∞
4	∞	2	9	0

Π

	1	2	3	4
1	/	1	/	1
2	/	/	2	/
3	3	/	/	/
4	/	4	4	/

Step2: iteration 1 (k=1) shorter paths from 3 → 2 and 3 → 4 are found through vertex 1

D

	1	2	3	4
1	0	8	∞	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	9	0

Π

	1	2	3	4
1	/	1	/	1
2	/	/	2	/
3	3	1	/	1
4	/	4	4	/

Step3: iteration 2 (k=2) shorter paths from 1 → 3 and 4 → 3 are found through vertex 2

D

	1	2	3	4
1	0	8	9	1
2	∞	0	1	∞
3	4	12	0	5
4	∞	2	3	0

Π

	1	2	3	4
1	/	1	2	1
2	/	/	2	/
3	3	1	/	1
4	/	4	2	/

Step4: iteration 3 (k=3) shorter paths from 2 → 1, 2 → 4, and 4 → 1 are found through vertex 3

D

	1	2	3	4
1	0	8	9	1
2	5	0	1	6
3	4	12	0	5
4	7	2	3	0

Π

	1	2	3	4
1	/	1	2	1
2	3	/	2	1
3	3	1	/	1
4	3	4	2	/

Step5: iteration 4 (k=4) shorter paths from 1 → 2 and 3 → 2 are found through vertex 4

D

	1	2	3	4
1	0	3	4	1
2	5	0	1	6
3	4	7	0	5
4	7	2	3	0

Π

	1	2	3	4
1	/	4	2	1
2	3	/	2	1
3	3	4	/	1
4	3	4	2	/

3 COMPARATIVE ANALYSIS OF THESE THREE ALGORITHMS

3.1 Dijkstra's Algorithm

3.1.1 Advantages:

Dijkstra's algorithm is used to compute the result in the single-pair, single-source/destination, and shortest path problem. Dijkstra's algorithm does not need that the distance_matrix represents a dense matrix, which makes the algorithm work better on memory for sparse graphs. With a large number of vertex, Dijkstra's algorithm is superior and more effective[12]. Moreover, Dijkstra's algorithm would create shorter time in minor and large graphs. Dijkstra's algorithm has lower time complexity as compared to other shortest path problem algorithms and is introduced with good extensibility.

3.1.2 Disadvantages:

Dijkstra's is the more efficient algorithms of single-source shortest path problem but the major problem with this algorithm is that it can't handle the graphs having negative weight edges[13].

3.1.3 Applications:

Dijkstra's algorithm is the most efficient and much faster single-source shortest path problem algorithm and is commonly used in real-time applications. Now a days, Dijkstra's algorithm is most frequently applied in networking areas, in GPS systems, and in 3D wireless sensors.

3.2 Bellman-Ford Algorithm

3.2.1 Advantages:

Bellman-Ford algorithm getsaresult in the SSP if the edges having -ve weights and thatcan sensing a negative edges cycle in the graph. The quality of the algorithm performs best while the graph contains fewer amount of nodes, although Dijkstra's algorithm can perform effectively where the graph is has a larger number of nodes[13].

3.2.2 Disadvantages:

Based on the complexity of time, we conclude the Bellmenford algorithm getsa longer time instead of Dijkstra's algorithm. For single-source shortest path problem, Bellman-Ford is appliedbarely when the weights are negative on edges otherwise Dijkstra's algorithm is the best option.

3.2.3 Applications:

Bellman-Ford algorithm is used in routing to find optimal routes in a network [14].

3.3 Floyd WarshallAlgorithm

3.3.1 Advantages:

The Floyd Warshall algorithm assigns the shortest path among each set of nodes by a graph examing algorithm. This algorithm is quicker wheres the graph is densely connected but not in the case of sparse graphs. In the Floyd-Warshall algorithm, there is an improved memory performance than a small matrix execution for a densely connected matrix calculation usually has a high performance of floating points operation in memory. This algorithm can solve the -ve weight problemhavingabroader range than Dijkstra's algorithm. It's significantly desirable along with these three algorithms. It is mostrelated to the shortest path problem among entire point pairs.
longer time instead of Dijkstra's algorithm. For single source shortest path problem, Bellman-Ford is applied barely when the weights are negative on edges otherwise Dijkstra's algorithm is the best option[15].

3.3.2 Disadvantages:

Though, that has a greater complexity of time than Dijkstra's and Bellman-Ford's Algorithm.

3.3.3 Applications:

This algorithm is used to handle several problems like computation of fast path finder networks, finding the path with maximum flow between two vertices, and shortest path in directed graphs.

TABLE 3
COMPARISONS OF SHORTEST PATH ALGORITHMS

Factors	Dijkstra's	Bellman-Ford	Floyd Warshall
Space complexity	O(V)	O(V)	$O(V^2)$
Time complexity	$O(V^2)$	O(VE)	$O(V^3)$
Use condition	Dense graph	Sparse graph thoroughly related to the side	Dense graph closely related to the vertexes
Negative weight edges	No, can't deal with negative edge values	Yes, can deal with negative edge values	Yes, can deal with negative weight edges

4 CONCLUSION

This paper performed a comparative examination in terms of minimum path optimization between three algorithms. The three comparable algorithms are used to determine which is best to discover the minimum distance between the two vertices. Our research shows that the Dijkstra algorithm is the best option to choose when the graph has positive weights and it solves the same problem in less time as compared to Bell-man ford. Floyd Warshall is suitable in the situation where the dense graph is given and we have to computeall shortest pair's pathand it is more widely used in real-time applications. It essential to recalled that if the input graph has a negative cycle then no shortest path exists.

5 REFERENCES

[1] Li, Tianrui, Luole Qi, and Da Ruan. "An efficient algorithm for the Single-Source Shortest Path Problem in graph theory." *2008 3rd International Conference on Intelligent System and Knowledge Engineering*. Vol. 1. IEEE, 2008.

[2] Sadavare, A. B., and R. V. Kulkarni. "A review of application of graph theory for network." *International Journal of Computer Science and Information Technologies* 3.6 (2012): 5296-5300.

[3] Manjaiah, D. H. "A Study on Contrast and Comparison between Bellman-Ford algorithm and Dijkstra's algorithm."

[4] Johnson, Donald B. "A note on Dijkstra's shortest path algorithm." *Journal of the ACM (JACM)* 20.3 (1973): 385-388.

[5] Zhao, Lingling, and Juan Zhao. "Comparison study of three shortest path algorithm." *2017 International Conference on Computer Technology, Electronics and Communication (ICCTEC)*. IEEE, 2017.

[6] Noto, Masato, and Hiroaki Sato. "A method for the shortest path search by extended Dijkstra algorithm." *Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics.'cybernetics evolving to systems, humans, organizations, and their complex interactions'(cat. no. 0*. Vol. 3. IEEE, 2000.

[7] Goldberg, Andrew, and Tomasz Radzik. *A heuristic improvement of the Bellman-Ford algorithm*. STANFORD UNIV CA DEPT OF COMPUTER SCIENCE, 1993.

[8] Manan, Abdul, and Syed Imran Ali Lakyari. "Single source shortest path algorithm Dijkstra and Bellman-Ford Algorithms: A Comparative study." *International Journal of Computer Science and Emerging Technologies* 3.2 (2019): 25-28.

[9] Burfield, Chandler. "Floyd-warshall algorithm." *Massachusetts Institute of Technology* (2013).

[10] Hougardy, Stefan. "The Floyd–Warshall algorithm on graphs with negative cycles." *Information Processing Letters* 110.8-9 (2010): 279-281.

[11] Magzhan, Kairanbay, and Hajar Mat Jani. "A review and evaluations of shortest path algorithms." *International journal of scientific & technology research* 2.6 (2013): 99-104.

[12] AbuSalim, Samah WG, et al. "Comparative Analysis between Dijkstra and Bellman-Ford Algorithms in Shortest Path Optimization." *IOP ConferenceSeries: Materials Science and Engineering*. Vol. 917. No. 1. IOP Publishing, 2020.

[13] Xiao, Ji-Xian, and Fang-Ling Lu. "An improvement of the shortest path algorithm based on Dijkstra algorithm." *2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE)*. Vol. 2. IEEE, 2011.

[14] Wang, Xiao Zhu. "The Comparison of Three Algorithms in Shortest Path Issue." *Journal of Physics: Conference Series*. Vol. 1087. No. 2. IOP Publishing, 2018.

[15] Mukhlif, Fadhil, and Abdu Saif. "_Comparative Study On Bellman-Ford AndDijkstra Algorithms." *Proc. Of International Conference on Communication, Electrical and Computer Networks (ICCECN 2020) Kuala Lumpur, Malaysia*. 2020.