Comparison Analysis of Shortest Path Algorithms

Mie Mie Aung
University of Computer studies, Monywa
miemieaung72@gmail.com

Thae Thae Han
University of Computer studies, Meiktila
thae2hann@gmail.com

Thu Zar Aung
University of Computer studies, Meiktila
tza.thuzaraung@gmail.com

Abstract

Today, shortest path problem refers to emerging technology such as map based systems. There are various types of algorithms with shortest paths or routes. The main objective of this paper is to know Dijkstra's Algorithm and Bellman-Ford Algorithm in solving shortest problem of the road. And, in graphical forms, further examples and implementations of algorithms are shown to demonstrate how and of the algorithms operate. Dijkstra's algorithm for non-negative weight graphs and Bellman-Ford algorithm that manages negative weights and it can detect negative cycles, too. The well-know Dijkstra's Algorithm and Bellman-Ford Algorithm ar applied in this paper to find shortest routes to travel fro one town to another. It is therefore essential to choose a efficient route for travelling. Generally speaking, th goal is to get shortest route from one town to anothe The time complexity of Dijkstrs's algorithm is faster than Bellman-Ford algorithm and the travelling paths don't have negative-weight. So, Dijkstrs's algorithm is better to use.

1. Introduction

The shortest path problem is a problem in finding the shortest path of a destination. In general we use graphs to represent the shortest path problem. A graph is an abstract mathematical entity, containing collection of vertices and edges. Edges bind the vertices in pairs. It's possible to walk along the edges of a graph by going from one vertex to another. Depending on whether or not one can walk both sides around the edges, it is determined whether the graph is a directed graph or an undirected. In addition, edge lengths are also referred to as weights and the weights are usually used to measure the shortest path from one point to another.

In the real world the graph theory can be extended to various forms of scenarios. For example we can use a graph to represent a map, where vertices mean cities and edges mean routes that link the cities. If routes are one-way, the graph is directed; otherwise, it is undirected. Various types of algorithms have been solved the shortest path problem. However, this paper addresses some of the most common traditional shortest path algorithms along with one that uses some of the genetic algorithms, and they are as follows:

1. Dijkstra's Algorithm
2. Bellman-Ford Algorithm
3. Floyd-Warshall Algorithm
4. Genetic Algorithm (GA)

People have so many reasons to communicate the town with one another, e.g. to transport goods and local products, to trip for relaxing as excursion and so on. If so, there is a need to think how to reduce cost, time and complex nature of travel. Therefore, it needs to find the shortest paths in reducing cost, time and complexity. The Dijkstra's algorithm and the Bellman Ford algorithm are the efficient algorithms used to find the shortest path between a known vertex to other vertex. So, described the analysis of Dijkstra's algorithm and Bellman Ford algorithm. The paper is intended to provide the following objectives:

-To determine and define shortest path problem principles and the representation of graphs in machine in order to solve the shortest path problem and to recognize the various basic terms of a graph

-To explain the related principles and implementations of the Dijkstra's Algorithm and Bellman-Ford Algorithm.

2. Related Work

There are also several smart shortest path algorithms nowadays which have been used in many previous research papers. The authors in [1], a heuristic GA has been used to solve the Single Source Shortest Path (SSSP) problem. Its main aim was to investigate the SSSP problem within the Internet routing context, the transmitting cost of messages and packages are higher than the search space in another paper[2] for example, used a heuristic approach for calculating the shortest path of traffic networks. They proposed a new restricted dynamic path algorithm, obtained by extending the algorithm of the Dijkstra. In a paper by Li, Qi, and Ruan[3], an effective algorithm called Li-Qi (LQ) has been proposed for the SSSP problem, by finding a simple path of the smallest total weights of a given initial or source vertex of any other vertex within the graph. Based on this newly implemented algorithm; the vertices can be queued many times, and only the source vertex and relaxed vertices are queued[3]. Meghanathan[4] examined the algorithm used by Dijkstra and the Bellman-Ford algorithm to find the shortest path in a graph. He concluded that the time complexity of Dijkstra's algorithm is O ($|E|*log\ |V|$), while Bellman-Ford's time complexity is O ($|V||E|$). Brendan, H. a. [5] Generalizes Dijkstra's Algorithm and Gaussian Elimination for solving MDPs. [12] Fadhil Mukhlif and Abdu Saif, compared the Dijkstra's and Bellman-Ford algorithms is performed as well based on their efficiency on attenuation vs. distance and throughput vs. traffic load.

3. Theoretical Background

A graph is used to describe a map where vertices describe the cities (towns), and edges represent the routes or roads within the network. A graph G is a series of two V&E sets where V is a series of vertices v0,v1, ... vn-1 also known as nodes and E is a collection of edges e1, e2, ... in which an edge is an region connecting two nodes.

This can be represented as – G = (V, E)

V (G) = (v0, v1, ...vn)or set of vertices

E (G) = (e1, e2, ...en)or set of edges

3.1. Overview of Shortest Path Algorithm

The shortest path problem in graph theory is the problem of finding a path in a graph between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. The problem of finding the shortest path between two intersections on a road map can be modeled as a special case of the shortest path problem in graphs, where the vertices correspond to intersections, and the edges correspond to segments of the road, each weighted by segment length. For graphs it is possible to define the shortest path problem, whether undirected, directed or mixed. For undirected graphs it is defined here; for directed graphs the path description includes the connection of consecutive vertices by an appropriate directed edge.

When they both incident to a common side, two vertices are adjacent. A path in an undirected graph is a sequence of vertices $P=(v_1,v_2,...,v_n)$ €V× V×…. V so that v_i is adjacent to v_{i+1} for $1\leq i <n$. Such a path P is considered a path from v_1 to v_n in length n-1. (The v_i are variables; their numbering here refers to their sequence location and does not need to apply to any canonical vertices labeling.) Let $e_{i,j}$ be both v_i and v_j an edge incident. The shortest path from v to v′ is the path $P=(v_1,v_2,...,v_n)$ (where $v_1=v$ and $v_n=v'$), which minimizes the sum of $\sum_{i=1}^{n-1} f(e_{i,i+1})$[9].

Finding the path with the fewest edges is equivalent to the action when each edge in the graph has unit weight or $f: E\rightarrow\{1\}$. The problem sometimes called the shortest path problem for a single pair, to differentiate it from the variants that follow:

- The shortest single-source path problem, where we need to find the shortest paths from the source vertex v to all other vertices in the graph.
- The shortest single-destination path problem, in which we have to find shortest paths from all vertices in the directed graph to a single destination vertex v. This can be reduced to the shortest single-source path problem by reversing the arcs in the directed graph.
- The shortest path problem for all pairs, in which we have to find shortest paths between each pair of vertices v, v' in the graph.

Such generalizations have considerably more effective algorithms on all applicable pairs of vertices than the simplistic method of running a single-pair shortest path algorithm. The shortest problem with the path is to find a path between 2 vertices in a graph to reduce the total edge weights.

3.2. Dijkstra's Algorithm: Description and Configuration

Within a graph, we assign a mark for each vertex which specifies the minimum length of the other vertices v of the graph. We can do it by defining an array[d] on a disk. The algorithm operates sequentially, and decreases the value of the vertices label in each step. If trips are made to all vertices, the algorithm ends. At point s, where the label d[s] is equal to zero; furthermore, labels in other vertices d[v] are equal to infinity means everywhere, meaning that the range from point s to other vertices is unknown. We can simply use a really big number simply to represent infinity by using a computer.

Moreover, we must decide whether or not it has been visited for and vertex v. In order to do so, we declare an array of Boolean form called u[v], where all vertices are initially assigned as unvisited (u[v] = false).

The Dijkstra's algorithm consists of n iterations. When all the vertices are visited, the algorithm ends; otherwise we must select the vertex with the lowest (smallest) value on the mark from the list of unvisited vertices. (We'll pick a starting point of s at the beginning). After that we find the entire neighbor of the vertex. For each unvisited neighbor, we must find a new length equal to the mark's value sum at the initial vertex v (d[v]) and the length of edge l that connects them. If the resulting value is less than the label value the newly acquired value is required to change [4].

Neighbors of d=minimum(d[neighbors],d[v]+l) (1)

We'll allocate the initial vertex as visited after considering all the neighbors where u[v] = true. After performing this step n times, the graph then visit all vertices and the algorithm ends or terminates. The unconnected vertices to the starting point will remain allocated to infinity. To work again the shortest path from the starting point to other vertices, array p[] is defined where we will store the number of vertex p[v], which penultimate vertices in the shortest path for each vertex where $v \neq s$ is used. In other words, a full path from s to v is equal to the assertion below

$P = (s , ..., p[p[p[v]]], p [p [v]], p [v], v)$ (2)

Figure 1 shows an extract from the Dijkstra algorithm, written in Java.

```
for (int i = 0; i < n; i++) {
    // vertex
    int v = -1;
    // finding minimum label among unvisited vertices
    for (int j = 0; j < n; j++) {
        if (u[j] == false && (v == -1 || d[j] < d[v])) {
            v = j;
        }
    }
    // set as visited.
    u[v] = true;

    for (int j = 0; j < n; j++) {
        // if there is exists path between v and j vertices
        if (a[v][j] > 0) {
            if (d[v] + a[v][j] < d[j]) {
                d[j] = d[v] + a[v][j];
            }
        }
    }
}
```

Figure 1. The use of Dijstra's Algorithm

3.2.1. Dijkstra's Algorithm

Solution to graph theory's shortest single-source path problem

- Directed and undirected graphs
- All edges must have non-negative weights
- Graph must be connected

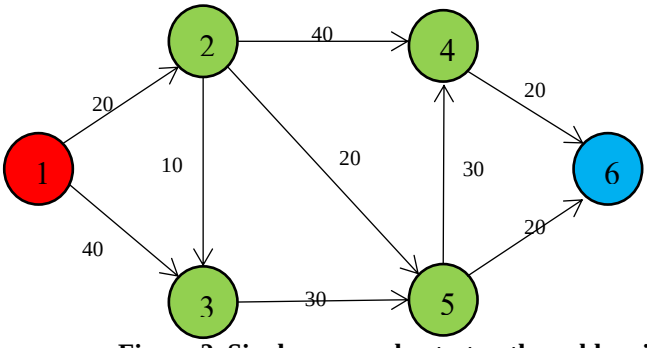

Figure 2. Single source shortest path problem in graph theory

Relaxation of Dijkstra's algorithm is
If (d [u] + C (v, u) < d [v])
d [v] = d [u] + c (u, v)

3.2.2. Output of Dijkstra's Algorithm

- Initial algorithm outputs value of the shortest path not the path itself.
- The route can be obtained with minor modification.

Value: σ (1, 6) = 60
Path: {1, 2, 5, 6}

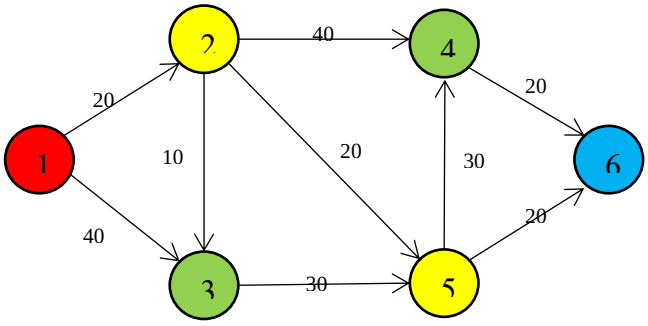

Figure 3. Output values of shortest path problem in graph theory

3.2.3. Advantages of Dijkstra's Algorithm

- To our purposes, a possible benefit of Dijkstra's algorithm is that the algorithm also doesn't need to explore all edges. When edges are fairly costly to calculate, otherwise Dijkstra's algorithm may be quicker.
- An algorithm uses a process definite. It is not based on any programming language, so, even without programming knowledge, it is easy for everyone to understand. Each step in an algorithm has its own logical sequence, so that debugging is simple.
- Google Maps, regional charts and mobile networks are included.

3.2.4. Disadvantages of Dijkstra's Algorithm

- Blind searching results in a loss of time during processing.
- Incapable of treating negative edges.
- It leads to acyclic graphs and the shortest path can't very often be found.

3.3. Bellman-Ford Algorithm: Description and Configuration

The Bellman-Ford algorithm recognizes or recognizes the edges with negative weights, as compared to the Dijkstra's algorithm. A graph therefore containing loops of negative weights that generate multiple paths from the starting point to the end destination, where each cycle will decrease the length of the shortest path. Expected to give this actual fact if our graph does not contain cycles of negative weight. The d[] array will hold the minimum length from beginning point s to other vertices.

The algorithm consists of many steps in which, in each step, all edges values must be decreased by replacing d[b] with the statement d[a]+c; a and b are graph vertices, and c is the corresponding edge that links them. By calculating the length of all the shortest paths in a graph, n – 1 phases are needed, but the value of the array elements will remain by setting infinity for those unattainable graph vertices [7]. [3] The authors represented the algorithm is based on the ideas of the queue and the relaxation. This is an extract from Bellman-Ford's Java Algorithm. Figure2 displays an extract of the Bellman-Ford algorithm written in Java.

```
// the distance from start point to inself
d[0] = 0;

for (int i = 0; i < n - 1; i++) {
    for (int j = 0; j < m; j++) {
        // if less that infinity (30000)
        if (d[a[j][0] - 1] < 30000) {

            d[a[j][1] - 1] =
            Math.min(d[a[j][1] - 1], d[a[j][0] - 1] + a[j][2]);
        }
    }
}
```

Figure 4. The use of Bellman-Ford Algorithm

3.3.1. Bellman-Ford Algorithm

The Bellman Ford algorithm will be used for finding the shortest paths from the source vertex to all other vertices in a weighted graph. This relies on the following concept: The shortest path at most contains n−1 edges, since there is no loop on the shortest path. Why shouldn't a loop have the shortest path, then? No need to shift a vertex again, as the shortest path to all other vertices could be found for any vertices without the need for a second visit.

Algorithm Steps:

- From 0 the outer loop goes through: n−1.
- Loop over all edges, test whether the next node distance > current node distance + edge weight, then change the next node distance to "actual node distance + edge weight".

This algorithm depends on relaxation theory, where the shortest distance for all vertices is slowly replaced by more complex values before eventually finding the optimum solution. For first all vertices have "Infinity" distance, but only the source vertex distance = 0, then change all vertices related to the new distances (source vertex distance + edge weights), instead apply the same idea of different distances to new vertices, and so on.

Works with negative weights

• Notice a negative loop if there is one

• If there is no negative loop, find the shortest simple path if graph G = (V, E) includes a cycle of negative weight, then there could be no shortest paths.

Relaxation of Bellman-Ford Algorithm is

If (d [u] + C (v, u) < d [v])

d [v] = d [u] + c (u, v)

The key advantage of the Bellman-Ford algorithm is that it works well on a graph with negative edge weights but the Dijkstra's algorithm may or may not yield the correct result in this case. Yet the Bellman-Ford algorithm is not working if the graph has a negative weight loop and is slower than Dijkstra's algorithm.

3.4. Differences between Dijkstra's Algorithm and Bellman-Ford Algorithm

Table 1. Differences between two Algorithms

Bellman-Ford Algorithm	Dijkstra's Algorithm
Works when there is negative weight edge, it also detects the negative weight cycle.	Doesn't work when there is negative weight edge.
The result contains the vertices which contains the information about the other vertices they are connected to.	The result contains the vertices containing whole information about the network, not only the vertices they are connected to.
It can easily be implemented in a distributed way.	It cannot be implemented easily in a distributed way.
It is relatively less time consuming.	It is more time consuming than Bellman Ford's algorithm.
Dynamic Programming approach is taken to implement the algorithm.	Greedy approach is taken to implement the algorithm.

4. Flowchart Diagram

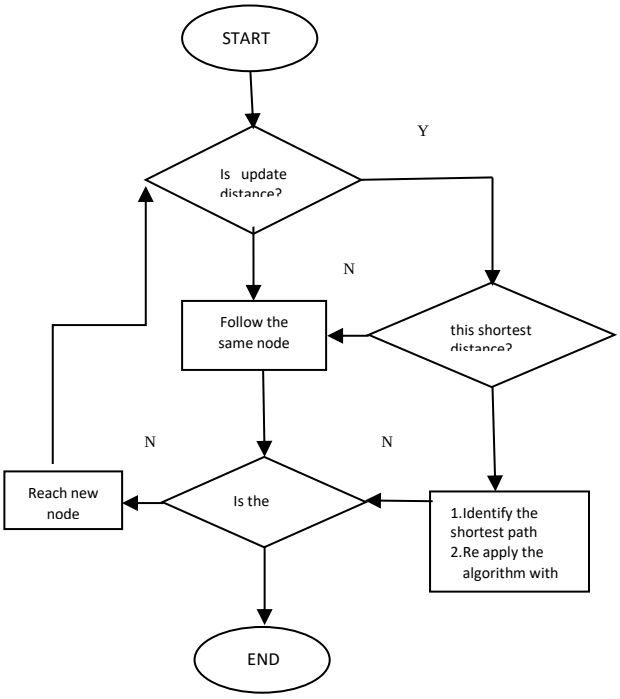

Figure 5. Flowchart diagram for shortest path algorithm

5. Analysis of Data

This section explains the approaches for integrating data and theoretical approach used in this article. The locations used for this paper work were obtained from Yangon to Chaungthar, and the distance between each location in the Kilometer (km) diagram was also measured using Google maps position.

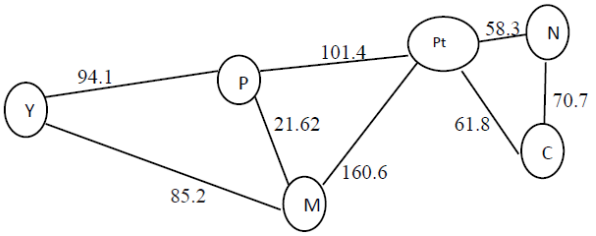

Figure 6. An example of the single source shortest path (SSSP) problem

Table 2. Names of location and Nodes identification

City	Nodes
Yangon	Y
Pantanaw	P
Maubin	M
Pathein	Pt
Ngwesaung	N
Chaungthar	C

Table 3. The selected nodes for shortest path

Vertices	Y	P	M	Pt	N	C
Y	0_Y	94.1_Y	85.2_Y	∞	∞	∞
P	0_Y	94.1_Y	85.2_Y	195.5p	∞	∞
M	0_Y	94.1_Y	85.2_Y	195.5p	∞	∞
Pt	0_Y	94.1_Y	85.2_Y	195.5p	253.8 pt	315.6 pt
N	0_Y	94.1_Y	85.2_Y	195.5p	253.8 pt	315.6 pt

5.1. The Time Complexity

Table4 displays the time-complexity in each algorithm; where n is the total number of vertices, and m is the total number of edges.

• The Bellman-Ford algorithm's time complexity is Θ(|V||E|) where |V| is the number of vertices and |E| is number of edges. If the graph is complete, value of |E| becomes Θ(|V|2).

• Time complexity of Dijkstra's Algorithm is O(V^2) but it drops down to O(V + E log V) with a min-priority queue.

Table 4. The time complexity

Shortest-path Algorithm	Time complexity
Dijkstra	n^2+m
Bellman-Ford	n^3

The time complexity of Dijkstrs's algorithm is faster than Bellman-Ford algorithm.

5.2. Discussion

The calculated time complexity of Dijkstra's algorithms and Bellman-Ford algorithms shows that these algorithms are suitable to solve the shortest path problem. Single-source Shortest Path Algorithm is used to obtain the best performance, with actual distances and time between any two cities.

6. Conclusion

This system, demonstrated by calculating the shortest paths from Yangon to Chaungthar in Myanmar began using real weighted values. The experimental results show that the journey will incorporate the use of Dijkstra's algorithm to get the shortest possible way from Yangon to another city of choice. The Bellman-Ford algorithm is ignored because the negative-weight values are not suited for as the travelling paths. And, Dijkstra's algorithm is better, because the time complexity of Dijkstra's algorithm is faster than Bellman-Ford algorithm.

References

[1] B.S. Hasan, M.A. Khamees, and A.S.H. Mahmoud, ―A Heuristic Genetic Algorithm for the Single Source Shortest Path Problem, ‖ Proc. of International Conference on Computer Systems and Applications, pp. 187-194, 2007. (A Review)

[2] C. Xi, F. Qi, and L. Wei, ―A New Shortest Path Algorithm based on Heuristic Strategy,‖ Proc. of the 6th World Congress on Intelligent Control and Automation, Vol. 1, pp. 2531–2536, 2006. (A Review)

[3] T. Li, L. Qi, and D. Ruan, ―An Efficient Algorithm for the Single-Source Shortest Path Problem in Graph Theory, Proc. of 3rd International Conference on Intelligent System and Knowledge Engineering, Vol. 1, pp. 152-157, 2008. (A Review)

[4] Meghanathan, D. N. (n.d.). Review of Graph Theory Algorithms . MS: Department of Computer Science , Jackson State University.

[5] Brendan, H. a. (2005). Generalizing Dijkstra's Algorithm and Gaussian Elimination for solving MDPs. International Conference on Automated Planning and Scheduling/Artificial Intelligence Planning System, (pp. 151 - 160).

[6] Dijkstra's Algorithm, Available at http:// informatics. mccme.ru/moodle/mod/statements/ view.php?id=193#1. 2012.

[7] Bellman-Ford Algorithm, Available at http:// informatics .mccme.ru/moodle/mod/statements/vie w.php?id=260#1. 2012.

[8] M. Muthulakshm, M.M. Shanmugapriya, ―Shortest Path Algorithm and its implementation‖, International Journal of Mathematics Trends and Technology (IJMTT) – Vol. 36 No. 2- August 2016.

[9] Dijkstra'sAlgorothmWikipedia website (2019). [Online]. Available: https://en.mwikipedia.org/ wiki/ Dijkstar% 27s_algorithm

[10] Myanmar Distance Calculator website (2018). [Online]. Available: https://distance calculator. globefeed.com/Myanmar_Distance_Calculator.asp

[11] Zafar Ali, ―Comparison of Dijkstra's Algorithm with other proposed algorithms, International Academic Journal of Science and Engineering. Vol. 3, No. 7, 2016, pp. 53-66.

[12] Fadhil Mukhlif and Abdu Saif, __Comparative Study On Bellman-Ford And Dijkstra Algorithms, Proc. Of International Conference on Communication, Electrical and Computer Networks (ICCECN 2020) Kuala Lumpur, Malaysia, 1 – 2 February 2020.