International Journal For Technological Research In Engineering
Volume 5, Issue 1, September-2017 *ISSN (Online): 2347 - 4718*

SINGLE SOURCE SHORTEST PATH ALGORITHMS - COMPARISON THROUGH IMPLEMENTATION

Sudhanshu Rai[1], Dr. Arun JB[2], Dr Ashish Sharma[3]
[1]Ph.D. Scholar, Jodhpur National University, Jodhpur, Rajasthan, India
[2]Lecturer, Government Polytechnic College, Ajmer, Rajasthan, India
[3]HOD - Computer Science, Jodhpur National University, Jodhpur, Rajasthan

ABSTRACT: Many applications like transportation and communication network use shortest path algorithm to find out the shortest path between two nodes. In the Single source shortest path algorithm, a shortest path is calculating from one node to another node. In this paper, I have compared the results of the shortest path algorithms (Dijkstra, Bellman Ford) on the basis of running time. I am using C# programming language to compare the algorithms. I have also compared the algorithms on the basis of complexity and space. I also tried to give some advantages and disadvantages of both the algorithms. Keywords— Shortest Path, Dijkstra, Bellman Ford, Run-time Analysis

I. INTRODUCTION

Shortest Path Problem is the problem to find out the shortest path between two nodes. There are so many shortest path algorithms depending on the source and destination. a) Single source Shortest Path Algorithm b) Single destination Shortest Path Algorithm c) All pair Shortest path Algorithm In Single source shortest path algorithm, we have to find out the shortest path from a source vertex to another vertex. In single destination shortest path algorithm, we have to find out the shortest path from all vertices to a single destination vertex. In All pair shortest path algorithm, we have to find out the shortest path from all vertices to another vertex. Due to the nature of routing applications, we need flexible and efficient shortest path procedures, both from a processing time point of view and also in terms of the memory requirements. [1] In this paper, I am comparing single source shortest path algorithms (Dijkstra's and Bellman Ford). As mentioned earlier, a graph can be used to represent a map where the cities are represented by vertices and the routes or roads are represented by edges within the graph.[2] In this section, a graph representation of a map is explained further, and brief descriptions and implementations of the shortest path algorithms being studied are presented.

II. WORKING OF DIJKSTRA'S AND BELLMAN FORD ALGORITHM

The working of Dijkstra's algorithm and Bellman-Ford Algorithm is as follows:

DIJKSTRA'S ALGORITHM

The algorithm stores all nodes in a priority queue ordered by distance of the node from the root – in the first iteration of the algorithm, only root has distance set to 0, distance of all other nodes is equal to infinity. Then in each step Dijkstra's Algorithm picks from the queue a node with the highest priority (least distance from the root) a processes it and re-evaluates distances of all unprocessed descendants of the node. This means that the algorithm checks for all descendants that the following condition holds: [3]
Distance + Edge-Weight< Distance

BELLMAN FORD ALGORITHM

The Bellman-Ford algorithm is based on the relaxation operation. The relaxation procedure takes two nodes as arguments and an edge connecting these nodes. If the distance from the source to the first node plus the edge length is less than distance to the second node, than the first node is denoted as the predecessor of the second node and the distance to the second node is recalculated (distance(A)+ edge.length). Otherwise no changes are applied. [4]

COMPARISON ON THE BASIS OF COMPLEXITY AND SPACE

We consider a graph[G] with the vertices or nodes [V] and the edges[E].Now If we find the complexity of Dijkstra Algorithm with the Bellman Ford i.e.

Algorithm	Time Complexity	Space Complexity
Dijkstra Algorithm	O(E+(log V))	O(V)
Bellman-Ford Algorithm	O(EV)	O(V)

ADVANTAGES AND DISADVANTAGES:

DIJKSTRA'S ALGORITHM

The advantages and disadvantages are as follows: 1. It is a Greedy Algorithm. 2. It doesn't work on negative weight. 3. It can work for directed and undirected graph. 4. It requires global information. b) Bellman Ford Algorithm- The advantages and disadvantages are as follows: 1. It is a dynamic Algorithm. 2. It can work on negative weight. 3. It can only work for directed graph. 4. It only requires local information.

BELLMAN FORD ALGORITHM

The advantages and disadvantages are as follows: 1. It is a dynamic Algorithm. 2. It can work on negative weight. 3. It can only work for directed graph. 4. It only requires local information.

COMPARISON USING C# CODE

Now, I will determine the efficiency of shortest path algorithm. I have created a window based application to find out the running time of both the algorithms. I have created a WindowFormsApplication1, in which I have created a Form

and add a list box to display the running time of Dijkstra's and bellman ford algorithm. I have implemented Dijkstra's algorithm and Bellman Ford algorithm using C# code. I have created two functions for Dijkstra's and Bellman Ford algorithms. From the Form_Load () method, both functions are called and display the shortest path for every node from a single source. And I have used stopwatch to calculate the running time of Dijkstra's algorithm and Bellman Ford algorithm in microseconds. I used Random numbers to generate a graph.

To strore a Graph :

```
public struct Edge
{
public int u, v, w;
};
int NODES ;        /* the number of nodes */
int EDGES;         /* the number of edges */
int[]d=new int [10000];     /* d[i] is the minimum distance
from source node s to node i */
double[,] G = new double[1000, 1000];
/* graph to store the graph adjacency matrix */
```

To store the adjacency matrix of graph using Random numbers:

```
Random rn1 = new Random();
        for (m = 0; m length; m++)
        {
                for (n = 0; n < length; n++)
                {
                w[m, n] = rn1.Next(0, 10000);
                G[m, n] = w[m, n];
                }
        }
```

To Store the Edges with their weight:

```
k = 0;
for (i = 0; i < NODES; ++i)
        {
        for (j = 0; j < NODES; ++j)
                {
                if (w[i, j] != 0)
                        {
                        edges[k].u = i;
                        edges[k].v = j;
                        edges[k].w = w[i, j];
                        k++;
                        }
                l++;
                }
        }
        EDGES = k;
```

To Find out the running time using stopwatch:

```
Stopwatch s = new Stopwatch();
s.Start();
BellmanFord(source_vertex); /* Call for Bellman Ford
Algorithm */
s.Stop();
long time = s.ElapsedTicks / Stopwatch.Frequency / (1000L
* 1000L));
listBox1.Items.Add("time taken by Bellman ford is"+ time+"
microseconds");
s.Start();
Dijkstra(source_vertex); /* Call for Dijkstra's Algorithm */
s.Stop();
long time = s.ElapsedTicks / Stopwatch.Frequency / (1000L
* 1000L));
listBox1.Items.Add("time taken by Dijkstra's algorithm is"+
time+" microseconds");
```

TABLE I

First Run		
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1577	741
10	1617	764
50	1853	4655
100	2777	32026
500	23923	4205010
1000	92550	33416106
Second Run		
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1459	657
10	3570	687
50	1918	9631
100	2921	32822
500	23794	4224362
1000	96836	33603691
Third Run		
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1567	667
10	1455	697
50	1758	4557
100	2644	37941
500	25087	4158252
1000	92149	33592017
Fourth Run		
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1460	688
10	1411	678
50	1748	4476
100	2466	32904
500	24285	4196981

1000	92377	34340142
Fifth Run		
N	Dijkstra's Algorithm	Bellman Ford Algorithm
5	1506	670
10	1495	728
50	1659	4486
100	3479	31950
500	24323	4147961
1000	126932	33643137
Average		
5	1513.8	684.6
10	1909.6	710.8
50	1787.2	5561
100	2857.4	33528.6
500	24282.4	4186513.2
1000	100168.8	33719018.6

We can observe from this table that for the small number of vertices (N=5, 10) Bellman Ford is taking less time in comparison with Dijkstra's algorithm. And for the large number of vertices (N=50, 100, 500, 1000) Dijkstra's is taking less time in comparison with Bellman Ford.

III. CONCLUSION

In this study we have studied about two source shortest path algorithms and their comparison. There is advantage and disadvantage in algorithms. To find the running time of each algorithm I used one Program for comparing the running time (in Microseconds). After running the same program on five different runs (for each different value of N=5, 10, 50, 100, 500, 1000), I calculated the average running time for each algorithm and then showed the result with the help of a chart. From the chart I can conclude that for a small number of nodes (N=5, 10) Bellman Ford the most efficient algorithm to find out the shortest path.

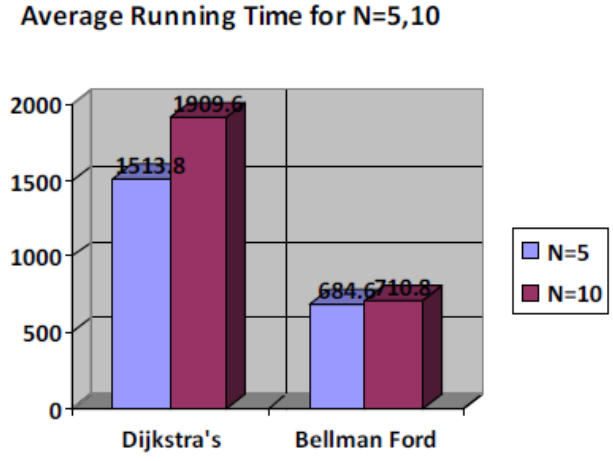

Figure 1

For N = 50, Dijkstra's Algorithm is the efficient algorithm.

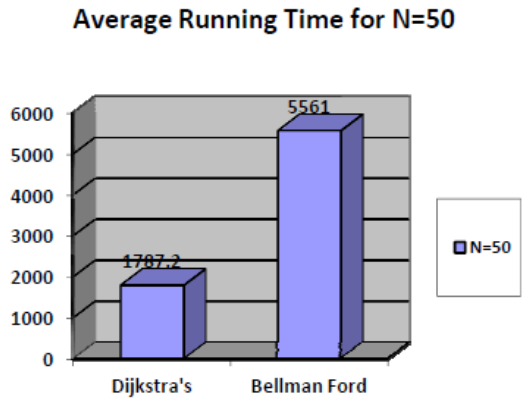

Figure 2

For N=100,again Dijkstra's algorithm, there is a very big difference in running time of Bellman Ford running time and algorithm.

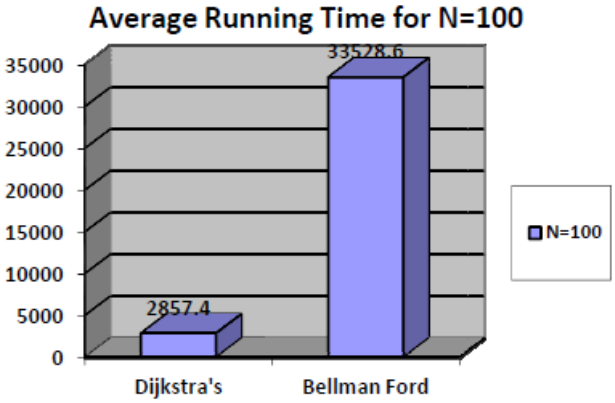

Figure 3

For N = 500, 1000, Dijkstra's Algorithm is the efficient algorithm in comparison to Bellman-Ford Algorithm.

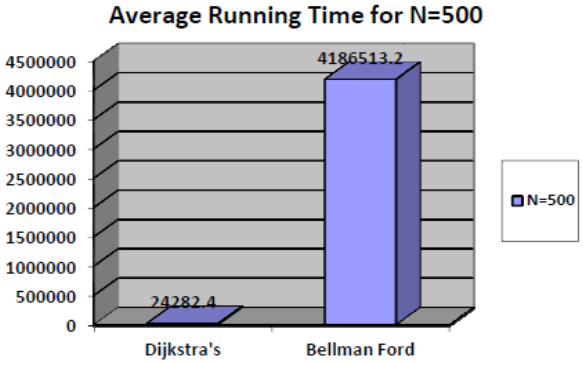

Figure 4

By these all charts, we can conclude that for small number of nodes (N < 50) Bellman Ford perform
better than Dijkstra's algorithm. Dijkstra's algorithm takes twice the running time of Bellman Ford algorithm. But a large number of nodes (N>50) Dijkstra's algorithm becomes more efficient. For N=50, Bellman Ford algorithm is three times to Dijkstra's running time. For N=100, Bellman Ford is 11 times to Dijkstra's algorithm.

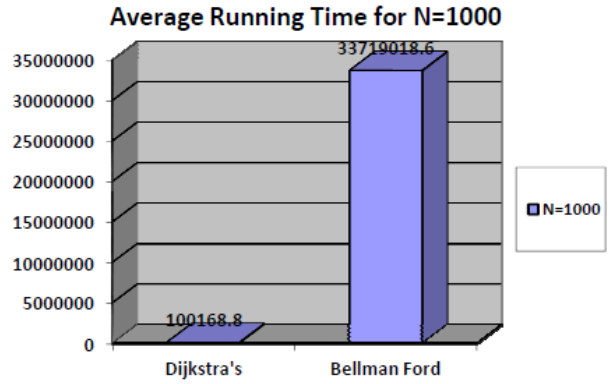

Figure 5

For N=500, 1000, Dijkstra's algorithm outperforms in comparison to Bellman Ford algorithm.

REFERENCES

[1] Faramroze Engineer, Fast Shortest Path Algorithms for Large Networks
[2] Kairanbay Magzhan, Hajar Mat Jani Shortest Path Algorithms
[3] http://en.algoritmy.net/article/45514/Dijkstras Algorithm
[4] http://en.algoritmy.net/article/47389/Bellman Ford Algorithm

