COMPARATIVE DIFFERENCE BETWEEN DIJKSTRA & BELLMAN-FORD ALGORITHM

Mir Mursalin Ankur[1] *& Md. Resbi Anik*[1]

[1]Dept. of Computer Science & Engineering, RUET

ABSTRACT

Both Dijkstra and Bellman-Ford algorithm are used to find the shortest path between two or more nodes. They have different characteristics. Both algorithms have advantages and limitations upon each other. They are generally used in different sectors like as networking and communication, transportation, system analysis, business sector etc. Here we will discuss about performances of both algorithms like as time complexity, space complexity and negative cycle checking. To compare the algorithms we will use C++ programming language.

1. INTRODUCTION

In mathematics graph theory is the study of graphs, which are mathematical structures used to model pairwise relations between objects. A graph in this context is made up of vertices, nodes, or points which are connected by edges, arcs, or lines. In computer science, a graph is an abstract data type that is meant to implement the undirected graph and directed graph concepts from mathematics. These pairs are known as edges, arcs, or lines for an undirected graph and as arrows, directed edges, directed arcs, or directed lines for a directed graph. [1]

In general sense we can divide graph into two parts. One of them is weighted graph and another is un-weighted graph. In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. [2]

Shortest path problem means the problem of finding the shortest path in graph from one vertex to another. "Shortest" may be least number of edges, least total weight etc. There are different kinds of shortest path algorithms:

1. Single source shortest path Algorithm
2. Single destination shortest path Algorithm
3. All pair shortest path Algorithm

In single source shortest path algorithm we will consider both directed and undirected graphs. In graph, all edges must have non-negative weights and graph must be connected. In this paper , we are comparing time, space, cycle checking and other criteria of single source Dijkstra and Bellman-Ford algorithm. For comparing complexity we are taking Big O notation as standard. Here cycle checking means capability of negative cycle detecting between both algorithms.

The term Negative cycle means a cycle with weights that sum to a negative number in a graph. [3]

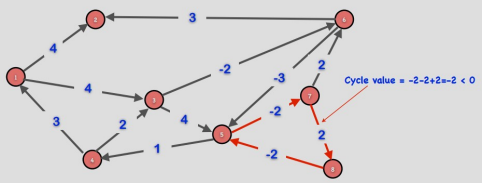

Fig. 1. Sample Graph with negative cycle

Why the comparison is needed between Dijkstra and Bellman-Ford Algorithm:

Bellman-Ford algorithm is a single source shortest path algorithm, which allows for negative edge weight and can detect negative cycles in a graph. Dijkstra algorithm is also another single-source shortest path algorithm. However, the weight of all the edges must be non-negative. A comparative discussion between both algorithm will lead us to finding better algorithm for real life problem.

2. GENERAL DISCUSSION ON DIJKSTRA AND BELLMAN-FORD ALGORITHM

Primary deliberation of both algorithms are given beneath :-

2.1. Dijkstra Algorithm

Dijkstra algorithm is the method of choice for finding shortest paths in an edge-weighted or vertex-weighted graph. Given a particular start vertex s, it finds the shortest path from s to every other vertex in the graph.

Psudocode: Shortest Path - Dijkstra(G, s) [4]

Algorithm 1 Dijkstra Algorithm

```
1:  procedure DIJKSTRA(G, s)
2:
3:      for all v ∈ G do
4:          dis[v] := ∞
5:          previous[v] := undefined
6:      end for
7:      dis[s] := 0
8:      Q := the set of all nodes in Graph
9:      while Q != empty do
10:         u := node in Q with smallest dist[]
11:         remove u from Q
12:     end while
13:     for all neighbor v of u: do
14:         alt := dist[u] + dist_between(u, v)
15:         if alt < dist[v] then
16:             dist[v] := alt
17:             previous[v] := u
18:         end if
19:     end for
20:     return previous[]
21: end procedure
22:
23: ▷ G = Graph,s = source,v = vertex,dis = distance
```

Dijkstra Implementation Technique:

Detailed steps used in Dijkstra's algorithm to find the shortest path from a single source vertex to all other vertices in the given graph.

1. Create a set sptSet (s=shortest p=path t=tree set) that keeps track of vertices included in shortest path tree, i.e., whose minimum distance from source is calculated and finalized. Initially, this set is empty.

2. Assign a distance value to all vertices in the input graph. Initialize all distance values as INFINITE. Assign distance value as 0 for the source vertex so that it is picked first.

3. While sptSet doesn't include all vertices

   (a) Pick a vertex u which is not there in sptSet and has minimum distance value.

   (b) Include u to sptSet.

   (c) Update distance value of all adjacent vertices of u. To update the distance values, iterate through all adjacent vertices. For every adjacent vertex v, if sum of distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

Example of Dijkstra Algorithm:

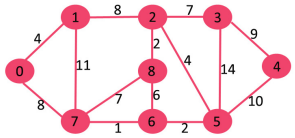

Fig. 2. A Weighted Graph [5]

The set sptSet is initially empty and distances assigned to source vertex are 0 and else other vertices are ∞.
Now there should pick the vertex having minimum distance value. The vertex 0 is picked, include it in sptSet. SptSet becomes 0 and only one element is inserted into set.
Now update distance values of its adjacent vertices.0's adjacent vertices are 1 and 7. The distance values of 1 is updated as 4 and 7 is updated as 8. [5]
The subgraph in Fig.3 shows vertices and their distance

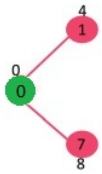

Fig. 3. Sub Graph of Fig.2 [5]

values as indicated at the top of the vertices,. The vertices included in SPT are shown in another color. Here it is green.

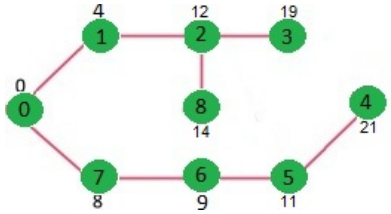

Fig. 4. Shortest Path according to Dijkstra Algorithm [5]

Now it should pick the vertex having minimum distance value and not already included in the set of sptSet. This process repeats this steps until sptSet doesn't include all vertices of given graph which shortest path is needed. The result is showed in fig 4.

2.2. Bellman-Ford Algorithm

Bellman–Ford is also a shortest path algorithm. In this algorithm, an approximation to the correct distance is gradually replaced by more accurate values until eventually reaching the optimum solution.

Pseudocode: Shortest Path - BELLMAN-FORD(G,w,s) [6]

Algorithm 2 Bellman-Ford Algorithm

```
1:  procedure BELLMAN-FORD(G, w, s)
2:
3:      for i = 1 to |G.V| − 1 do
4:          for all edge (u, v) ∈ G.E do RELAX(u,v,w)
5:          end for
6:      end for
7:      for all edge (u, v) ∈ G.E do
8:          if v.d > u.d + w(u, v) then
9:              return false
10:         else
11:             return false
12:         end if
13:     end for
14: end procedure
15:
16:  ▷ G = Graph, w = weight, s = source, V = Vertex
17:                ▷ E = Edge, d = distance, (u, v) = node
```

Bellman-Ford Implementation Technique:

1. This step initializes distances from source to all vertices as infinite and distance to source itself as 0. Create an array dist[] of size $|V|$ with all values as infinite except dist[src] where src is source vertex.

2. This step calculates shortest distances. Do following $|V| - 1$ times where $|V|$ is the number of vertices in given graph. Do following for each edge u-v.
   If $dist[v] > dist[u] + weight$ of edge uv, then update dist[v].
   $dist[v] = dist[u] + weight$ of edge u-v.

3. This step reports if there is a negative weight cycle in graph. Do following for each edge u-v. If $dist[v] > dist[u] + weight$ of edge uv, then "Graph contains negative weight cycle".

   The idea of step 3 is, step 2 guarantees shortest distances if graph doesn't contain negative weight cycle. If we iterate through all edges one more time and get a shorter path for any vertex, then there is a negative weight cycle.

Example of Bellman-Ford Algorithm:

Let the given source vertex be 0. Initialize all distances as infinite, except the distance to source itself. Total number of vertices in the graph is 5, so all edges must be processed 4 times.

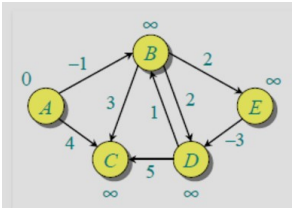

Fig. 5. Negative Cycle Checking With Bellman-Ford Algorithm [7]

Let all edges are processed in following order: (B,E), (D,B), (B,D), (A,B), (A,C), (D,C), (B,C), (E,D). We get following distances when all edges are processed first time. The first row in shows initial distances.
The first iteration guarantees to give all shortest paths which are at most 1 edge long. We get following distances when all edges are processed second time.

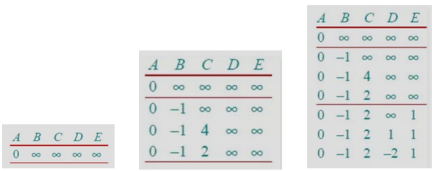

Fig. 6. Sequential iteration of Fig. 5 implementing Bellman-Ford Algorithm [7]

The second iteration guarantees to give all shortest paths which are at most 2 edges long. The distances are minimized after the second iteration, so third and fourth iterations don't update the distances. The last row in Fig.6 shows final values.

3. COMPARISON

		Negative edges	Negative cycle	Time complexity	Space complexity
Dijkstra	Single source	NO	NO	O(Vlog V+E)	O(V+E)
Bellman-Ford	Single source	YES	YES	O(V.E)	O(V)

Fig. 7. Comparative Difference Table [8]

Complexity :

Computational complexity is the study of the inherent limits of efficient computation measured in terms of time, space, and other resources such as randomness. Complexity between Dijkstra and Bellman-Ford algorithm on different aspect is shown in Fig. 7.

No of Nodes	Dijkstra's (sec)	Bellman ford's (sec)
1000	0.02354	0.02355
5000	0.5337	0.5539
10000	2.1728	2.3027
15000	5.04	5.71
20000	9.1839	9.5033
30000	20.73	23.2003
50000	55.3941	75.4112

Fig. 8. Chart of experimental values

Experimental value :

We have calculated some experimental values by using C++ programming language. A PC with intel core i3 CPU of 3^{rd} generation and 4GB DDR3 RAM configuration show experimental value in Fig. 8. and Fig. 9.

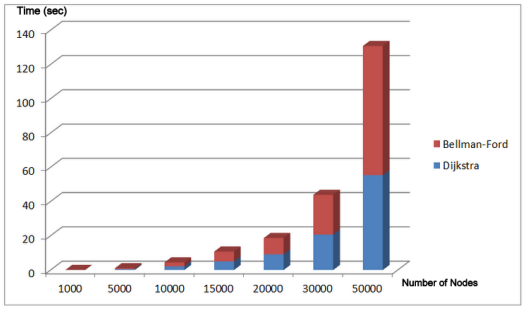

Fig. 9. Histogram according to Fig. 8 showing comparative time complexity

4. CONCLUSION:

After studying these two algorithms we conclude that Dijkstra algorithm is faster than Bellman-Ford algorithm. Though at the perspective of negative cycle checking Bellman-Ford is better. But time complexity of Bellman-Ford algorithm is larger than Dijkstra algorithm for finding shortest path. Which is a important factor in real life graph based problem like GIS (Geographic Information System), networking, pattern finding etc. So, Dijkstra algorithm is widely used in real time application. For better and faster performance in larger system Dijkstra is automatic choice between Dijkstra and Bellman-Ford algorithm.

5. REFERENCES

[1] https://en.wikipedia.org

[2] Introduction to Algorithms by Thomas H. Corman

[3] http://algo.epfl.ch

[4] http://www.gitta.info

[5] http://www.geeksforgeeks.org

[6] The Algorithm Design Manual by Steve S. Skiena

[7] http://www.shafaetsplanet.com

[8] http://www.stackoverflow.com