An Analysis of Bellman-Ford and Dijkstra's Algorithm

Melita D'souza
Department of Computer Science,
Indiana University Bloomington,
Bloomington, IN.
dsouzam@indiana.edu

Dwayne Dexter D'souza
Department of Computer Science,
Indiana University Bloomington,
Bloomington, IN.
dsouzad@indiana.edu

Abstract - This article represents the study of two different shortest path algorithms-Dijkstra's algorithm and Bellman-Ford algorithm. The aim of this study is to compare the two algorithms based on their run time. The graph sizes and their average degree will be varied to decide which one of the two algorithms is optimal. We generate random graphs using Erdos-Renyi model.

1. INTRODUCTION

Shortest path problem is a way to find the shortest distance between vertices in a graph from the source node to the destination node. The most popularly used shortest path algorithms are Dijkstra's and Bellman-Ford algorithm. Both these algorithms have the same end result i.e. both give the shortest path from one node to all the other nodes. Erdos-Renyi model is used to generate random graphs. These graphs have nodes and edges; each edge has a probability p of existing.
Work distribution:
Melita Dsouza-Bellman-Ford(code and analysis)
Dwayne Dsouza-Dijkstra's algorithm(code and analysis)
Erdos-Renyi-Collaboration

1.1 Bellman-Ford algorithm

The Bellman–Ford algorithm computes shortest paths from a single source vertex to all of the other vertices in a weighted digraph[1]. This algorithm is compatible with negative edge weights. The fundamental idea behind Bellman-Ford is that there can be at most |V| - 1 edges in one of our paths from the starting node to any other node in the graph, where |V| is the number of vertices in the graph. Therefore, if we iterate |V| - 1 times, we are guaranteed to find the shortest path from source to destination. Bellman-Ford updates along all edges for every iteration i.e. it examines each edge if it lessens the shortest path distance.

Procedure [2]

Bellman-Ford(G,l,s)

Input: Directed graph G=(V,E);
edge lengths $\{l_e : e \in E\}$ with no negative cycles;
vertex s ∈ V

Output: dist(u) = the distance from s to
// $\forall u$ reachable from s; where u∈V

for all $u \in V$:
    dist(u)=∞
    prev(u)=nil

dist(s)=0

repeat |V| - 1 times:
    for all e ∈ E:
        update(e)

update($(u, v) \in E$)
$\text{dist}(v) = \min\{\text{dist}(v), \text{dist}(u) + l\}$

1.2 Dijkstra's algorithm

Dijkstra's algorithm is an algorithm for finding shortest paths in a graph with edges that have non-negative edge weights. This algorithm follows a greedy approach which is why it fails with negative edge weights. Dijkstra's algorithm works faster than Bellman-Ford algorithm.

Procedure

Dijkstra(G,l,s)

Input: Directed/ undirected graph G=(V,E);
positive edge lengths $\{l_e : e \in E\}$;
vertex s ∈ V

Output: dist(u) = the distance from s to
// $\forall u$ reachable from s; where u∈V

for all $u \in V$:
dist(u)=∞
prev(u)=nil

dist(s)=0
//Create a set for unvisited nodes:
Z: set of unvisited nodes
while Z != []:
remove minimum valued node from the set Z and assign it to u
for $\forall\ (u, v) \in E$:
if dist(v)> dist(u)+$l(u,v)$:
dist(v)> dist(u)+$l(u,v)$
prev(v)=u
decreasekey(Z,v) //making nodes far away move closer to the top of the queue

2. ALGORITHMS [3]

2.1 Bellman-Ford algorithm

```
BELLMAN-FORD(G,l,s)
1.  INITIALIZE-SINGLE-SOURCE(G,s)
2.  for i= 1 to |G. V| -1
3.      for each edge (u,v) ∈ G.E
4.          RELAX(u,v,l)
5.  for each edge(u,v) ∈ G.E
6.      if v.d > u.d +l(u,v)
7.          return FALSE
8.  return TRUE
```

2.2 Dijkstra's algorithm

```
DIJKSTRA(G,l,s)
1.INITIALIZE-SINGLE-SOURCE(G,s)
2.  S = ∅
3.  Q = G.V
4. while Q ≠ ∅
5.     u=EXTRACT-MIN(Q)
6.     S= S ∪ {u}
7.     for each vertex v ∈ G.Adj[u]
8.         RELAX(u,v,l)
```

INITIALIZE-SINGLE-SOURCE(G,s):

1. for each vertex $v \in G.V$
2. $v.d=\infty$
3. $v.\pi$ = NIL
4. $s.d$ =0

RELAX(u,v,l):

1. if $v.d>u.d+l(u,v)$for each vertex $v \in G.V$
2. v.d=$u.d + l(u,v)$
3. v.π = u

3. TIME COMPLEXITY

3.1 Bellman-Ford algorithm

Line 1 of algorithm: The initial for loop runs through each vertex once. Hence this loop runs in *O(V)* time.

Lines 2-4: This consists of 2 loops; one executing *V* times and the other *E* times. So, the time complexity will be *O(VE).*

Lines 5-7: This loop runs through every edge, therefore running in *O(E)* time.

Thus, the total time complexity is *O(V.E)*

3.2 Dijkstra's algorithm

Line 1 of algorithm: The initial for loop runs through each vertex once. Hence this loop runs in *O(V)* time.

EXTRACT_MIN(Q) function runs in *O(logV)* time.

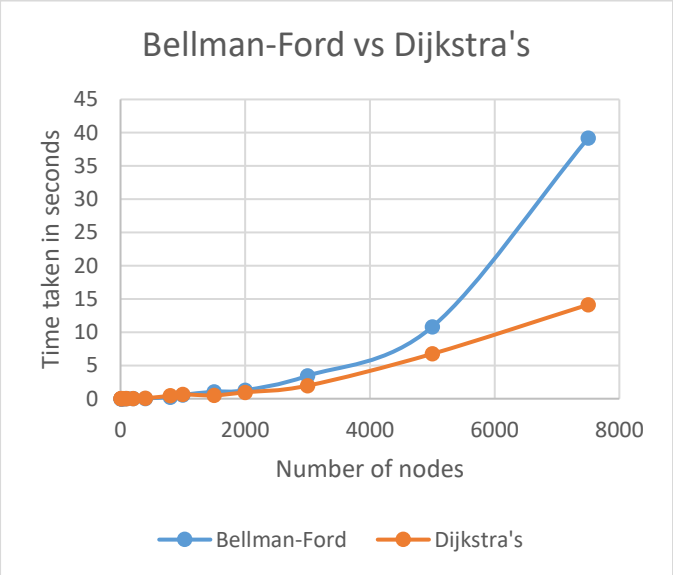
Line 7: This loop runs through every vertex, thus it runs in *O(E)* time.

The algorithm loops through the edges of each node. Thus, the total time complexity of this algorithm is *O(E+VlogV).*

4. EXPERIMENTATION RESULTS

Comparing the running time complexity of Bellman-Ford and Dijkstra's algorithm by varying the number of nodes in the graph using Erdos-Renyi model.

Number of nodes	Time taken in seconds		Probability
	Bellman Ford	Dijkstra's	
5	0.001235	0.02576	0.4
10	0.001335	0.027194	0.4
20	0.001617	0.028241	0.4
40	0.002634	0.027268	0.4
80	0.00493	0.029866	0.4
100	0.005823	0.032361	0.4
200	0.015194	0.039393	0.4
400	0.049738	0.069224	0.4
800	0.237473	0.441966	0.4
1000	0.576739	0.635708	0.4
1500	1.055265	0.50863	0.4
2000	1.284116	0.963116	0.4
3000	3.447053	1.957851	0.4
5000	10.80324	6.743982	0.4
7500	39.18482	14.133911	0.4



From the above graph, we can see that Dijkstra's algorithm works faster than Bellman-Ford approximately after 1300 nodes. Bellman-Ford is comparatively slower than Dijkstra's for higher number of nodes.

EXAMPLE:

The example below demonstrates the working of Dijkstra's and Bellman-Ford algorithm. We use the graph generated by Erdos-Renyi model as input. This graph is given in the form of an adjacency matrix as input to both the algorithms.

Erdos-Renyi (graph):

G=erdosRenyi(5,0.4,1)

G =

```
   0   10   10    0    0
   2    2    4    0    6
   6    9    8    8    9
   0    0    8    7    8
   0    3    6    6    0
```

This graph is now given as input to the two algorithms.

O/P of Bellman-Ford Algorithm:

```
Shortest path values are from Node 1 ( Origin)
Vertex(Destination) = [Total path value, Predecessor Node]
Vertex( 1 ) = [ 0 , 0 ]
 Vertex( 2 ) = [ 1 , 10 ]
 Vertex( 3 ) = [ 1 , 10 ]
 Vertex( 4 ) = [ 3 , 18 ]
 Vertex( 5 ) = [ 2 , 16 ]
Elapsed time is 0.001816 seconds.
```

O/P of Dijkstra's Algorithm:

```
Enter  source: 1
Enter destination: 5

e =

    16

Elapsed time is 0.024561 seconds.
```

5. TOOLS

We used MATLAB Version 7.6.0.324 R2008a to code shortest path algorithms- Bellman-Ford and Dijkstra's algorithm. We generated random graphs using Erdos-Renyi model which was coded in MATLAB as well. [4]

The code was run on a Windows 10 64-bit system @2.4GHz.

6.CONCLUSION

The analysis of the two shortest path algorithms shows that Bellman-Ford algorithm runs with a time complexity of *O(V.E)* whereas Dijkstra's algorithm runs the same problem with a time complexity of *O(E+VlogV)*. Thus, Dijkstra's algorithm has a much lower running time as compared to Bellman-Ford; which is why we choose Dijkstra's algorithm over Bellman-Ford for graphs with positive edges. Bellman-Ford algorithm should only be used with graphs having negative edge weights.

Both the algorithms fail if there is a negative cycle involved in the graph.

7. REFERENCES

[1] https://en.wikipedia.org/wiki/Bellman%E2%80%93Ford_algorithm

[2] https://www.youtube.com/watch?v=9PHkk0UavIM.

[3] Cormen, T.H. ;Leiserson, C.E.;Rivest R.L.;Stein C. Introduction to Algorithms, MIT Press & McGraw-Hill.

[4] Code: https://drive.google.com/drive/folders/0B5qF1hWVIfkQNkFSRHRiWkFIUkU?usp=sharing