Study and Analysis of Shortest Path Algorithms

Rabia Arshad
Department of Electrical Engineering
The University of Lahore
Lahore, Pakistan
rabia.arshad@ee.uol.edu.pk

Danista Khan
Department of Electrical Engineering
The University of Lahore
Lahore, Pakistan
danista.khan@ee.uol.edu.pk

Muhamamd Arslan Shahid
Department of Electrical Engineering
The University of Lahore
Lahore, Pakistan
arslan.shahid@ee.uol.edu.pk

Syed Hammad Hussain Shah
Department of Computer Science and I. T
The University of Lahore
Gujrat, Pakistan
hammad.hussain@cs.uol.edu.pk

***Abstract*—Shortest path problem is a fundamental technique in computer networking for route discovery. Utilizing the shortest path algorithms overall costs of setting the network is reduced. Many new technologies are implemented using the shortest path problem e.g. the road map system. Shortest path problem is an optimization technique [1] used in many applications. This paper's objective is to study the shortest path algorithms i.e. Dijkstra's Algorithm, A* Search, Floyd-Warshall Algorithm, Johnson's Algorithm and Bellman-Ford Algorithm. We will study and analyze the behavior of different shortest path algorithms in this paper. First we will study each algorithm to analyze its performance and then we will compare algorithms on the basis of their time complexity.**

Keywords-Shortest path problem, Weight Values, Dijkstra's, Bellman Ford, A* Search.

I. INTRODUCTION

Shortest path problem [2, 11] refers to find a path from source point to end point that is the shortest path among all other paths. Generally, shortest path problem is represented using graph theory. A graph consists of sets of vertices and set of edges. Edges are connections between pairs of vertices. Moreover some of the edges in a graph are associated with some value that is called weights. Weights values are used to find the shortest path from one vertex to another vertex. Graph theory [3] is widely used in real life applications e.g. in representation of a map. Many algorithms have been proposed to solve the shortest path problem. However, we discussed only a few of important and widely used algorithms for shortest path problem in this paper. Algorithms discussed in this paper are: Dijkstra's Algorithm, Floyd-Warshall Algorithm, Bellman-Ford Algorithm, A* Search Algorithm, Johnson's Algorithm.

II. ALGORITHMS FOR SOLVING SHORTEST PATH PROBLEM

Following subsections summarize the algorithms that determine the shortest path between single root node and all the remaining nodes of the graph. Shortest path trees are made when shortest paths of remaining nodes are calculated from on single root node. The algorithms determining shortest path that are summarized in the following subsections have been classified under an efficiency analysis. Each algorithm has particular features that make it different from other algorithms in terms of their properties and performance.

A. Dijkstra's Algorithm

Dijkstra's algorithm [4, 5] calculates the shortest path between a given initial node to a single destination node and all other nodes of the graph when all edges have positive weight values. Algorithm selects any node *i* as a starting node. Then the algorithm builds a tree *T* that ultimately spans all vertices reachable from *S1*. Vertices are inserted to tree *T* according to the distances i.e., first *S1*, then the vertex closest to *S1* and so on. Initially the distance between each node is set to infinity. At the beginning of the algorithm the source node *i* is set as the current node. From this node the tentative distances of all unvisited neighbour nodes are calculated that are connected by edge. The current node is then marked as the visited node and the next current node is selected which has the minimum tentative distance from the previous current node. The algorithm is terminated when all nodes have been visited and the set of unvisited nodes is empty. The complexity time for simplest Dijkstra's algorithm having N nodes is $O(N^2)$. This time can be reduced by using different variants of Dijkstra's algorithm.

function Dijkstra(*Graph*, *source*): create vertex set Q

```
 for each vertex v in Graph:           // Initialization
 dist[v] ← INFINITY                    // Unknown distance
from source to v
 prev[v] ← UNDEFINED                   // Previous node in
optimal path from source
 add v to Q                            // All nodes initially in Q
(unvisited nodes)

dist[source] ← 0                       // Distance from source
to source

while Q is not empty:
u ← vertex in Q with min dist[u]       // Source node will
be selected first
remove u from Q

for each neighbor v of u:              // where v is still in Q.
alt ← dist[u] + length(u, v)
if alt < dist[v]:                      // A shorter path to v has been
found
dist[v] ← alt
prev[v] ← u

return dist[], prev[]
```

B. Bellman Ford Algorithm

Bellman Ford [8] computes shortest paths between a single source node to all other node in a weighted digraph. It is more versatile but slower than Dijkstra's Algorithm since it can compute the shortest path of the graphs that have negative edge which makes it useful in the applications having negative edge graphs. It works on relaxation procedure by taking two nodes and edge connecting them. If the distance of first node j from the source plus the edge length is less than that of second node, then the first node is denoted as the predecessor and the distance of second node k is recalculated by:

$$Dist(k) = Dist(j) + edge\ length$$

When a cycle comes having negative edges no cheapest value can be found thus the algorithm yields falsehood. In a graph having N nodes, maximum number of paths from the source node can be N-1 edge long provided that there is no negative cycle. For E edges this algorithm does N-1 iterations thus the complexity time of Bellman Ford algorithm is *O(N.E).*

```
function BellmanFord(list vertices, list edges, vertex
source)

Step 1: initialize graph, At the beginning, all vertices have a
weight of infinity

  for each vertex v in vertices:
   distance[v] := inf

// And a null predecessor
     predecessor[v] := null

// Except for the Source, where the Weight is zero
  distance[source] := 0
Step 2: relax edges repeatedly
  for i from 1 to size(vertices)-1:
     for each edge (u, v) with weight w in edges:
      if distance[u] + w < distance[v]:
           distance[v] := distance[u] + w
           predecessor[v] := u

Step 3: check for negative-weight cycles
  for each edge (u, v) with weight w in edges:
     if distance[u] + w < distance[v]:
        error "Graph contains a negative-weight cycle"
  return distance[], predecessor[]
```

C. Floyd-Warshall Algorithm

Floyd Warshall Algorithm [9] finds the shortest path in a graph with negative costs of edges but the graph does not contain any negative cycle. It is an example of dynamic programming. It is also known as Roy–Warshall algorithm, Floyd's algorithm, WFI algorithm or Roy–Floyd algorithm.

The Floyd-Warshall algorithm uses a different approach and has a lower computational complexity equal to Θ (N3).
Let we have a graph *G* having *N* number of vertices. Also we have a function named $Short_{Path}$ (*x, y, k).* $Short_{Path}$ () function will return the shortest path from vertex *x* to vertex *y* using vertices from set {1, 2, …, *k*}. So given the information about the graph, our goal is to find the shortest path among vertex 1 and *k+1*. If *w(x, y)* is the weight of the edge between vertex *x* and vertex *y*, then we can define the shortest path in terms of base case and recursive case as follows:

Base case is:

$$Short_{Path}(x, y, 0) = w\ (x\ , y)$$

Recursive case is:

$$Short_{Path}(x, y, k+1\ ) = \min(Short_{Path}(x, y, k),$$

$$Short_{Path}(x, k+1, k), Short_{Path}(k+1, y, k))$$

This is the basic formula for Floyd Warshall Algorithm. Pseudocode for this basic algorithm is given as follows:

let distance is an array of |T|×|T|,consists of distances that are initialized to ∞

for each node z distance[z][z] ← 0

for each edge (z,g) distance[*g*][z] ← w(*g*,z)

for *h* **from** 1 **to** |T|
for *f* **from** 1 **to** |T|
for *s* **from** 1 **to** |T|

if distance[f][*s*] > distance[*f*][s] + distance[*h*][*s*]

distance[*f*][*s*] ← distance[*f*][*s*] + distance[*h*][s]

end if

D. A Search*

A* Search [6, 7] is an algorithm to find the shortest path between two locations. It is the most popular and widely used algorithm in graph theory. A* algorithm was developed to improve the efficiency of Dijkstra's algorithm. It is also called Informed Search Algorithm. It finds the solution of the problem by analyzing all possible routes that leads to the solution. It finds the path with minimum travelled distance or minimum cost. It was formulated on the basics of Weighted Graph. Steps that involved in A* Search is:

i. Choose a specific node from the graph as starting point.
ii. Build a tree of routes starting from this node.
iii. Expand the paths per step.
iv. Continue expanding the path until the goal node reached.
v. At each path or route, this algorithm determines that which path has to be expanded based on the estimated cost in reaching the destination node.
vi. A* Search selects the route that reduces the total cost $T(m)$ in reaching the destination from source.

$$T(m) = K\ (m) + P\ (m)$$

Where *m* is the last node on the route, *K(m)* is the cost from the starting point to the present node *m* and *P(m)* is the heuristic that estimates the lowest cost path from *m* to destination. The heuristic is called *monotone* if it is able to satisfy a condition i.e. $P(a) \leq d\ (a,\ b) + P\ (b)$ for each

edge (a, b) in graph. Here *d(a, b)* is length of edge from node *a* to node *b*. In this case A* algorithm can be implemented in a more efficient way. It is a complete algorithm like Breadth First Search.

E. Johnson's Algorithm

Donald Johnson developed a shortest path algorithm named Johnson's Algorithm [10]. Johnson's algorithm considers the edges with negative weights. It removes the negative weights of the graph and transforms the graph by using Bellman Ford Algorithm. Then the Dijkstra's Algorithm is used to compute the shortest path on the transformed graph. Johnson's algorithm consists of the following steps:

i. Insert a new node *t* in graph that is connected by edges of zero weights.
ii. Apply Bellman–Ford algorithm to find minimum weight path from new added node *t* to each node *y*. If a negative cycle exists this algorithm terminates.
iii. Graph is reweighted with new values that are computed in step ii.
iv. At the end, *t* is deleted, and the shortest path from each node *s* to every other vertex is determined using Dijkstra's algorithm.

The time complexity of Johnson's Algorithm is $O(V^2 \log V + VE)$. Algorithm utilizes O (VE) for Bellman Ford and $O(V \text{ Log } V + E)$ for implementing Dijkstra's algorithm. For a sparse graph the total time of this algorithm is faster.

III. CONCLUSION

In this paper, we discussed a few of shortest path algorithms i.e. Dijkstra's, Bellman Ford, Johnson's, A* Search. We analyzed behavior of each algorithm with positive and negative costs. Time complexities of each algorithm are also discussed in this paper. Dijkstra performs best among all the algortihms.

REFERENCES

[1] Sergiy A. Vorobyov, Shuguang Cui, Yonina C. Eldar, Wing-Kin Ma, Wolfgang Utschick; Optimization Techniques In Wireless Communications. EURASIP Journal On Wireless Communications And Networking 2009, 2009:567416

[2] Kairanbay Magzhan, Hajar Mat Jani, "A Review And Evaluations Of Shortest Path Algorithms". International Journal Of Scientific & Technology Research Volume 2, Issue 6, June 2013

[3] S.G.Shirinivas, S.Vetrivel, Dr. N.M.Elango; APPLICATIONS OF GRAPH THEORY IN COMPUTER SCIENCE AN OVERVIEW. S.G. Shrinivas et. al. / International Journal of Engineering Science and Technology Vol. 2(9), 2010, 4610-4621

[4] Dijkstra E W. A node on two problem in connexion with graphs. Numerische Mathemartik. vol. 1, 1959, pp.269-271.

[5] Paige R, Kruskal C. Parallel algorithms for shorest path problem. IEEE Transactions on Computer, vol.C34, 1985, pp.14-20.

[6] Ismail Chabini,Shan LAN. Adaptationd of the A* Alorithm for the Computation of Fastest Paths in Deterministic Discrete-Time Dynamic Network. IEEE Transations on Intelligent Transportion Systems, vol.4, 2002, pp.121-125.

[7] Chabini I, Lan S. Adaptations of the A*algorithm for the computation of fastest paths in deterministic discrete-time dynamic networks, IEEE Transations on intelligenttransportation systems, vol.3, 2002, pp.60-74.

[8] Vaibhavi Patel, Prof.ChitraBaggar; A Survey Paper of Bellman-Ford Algorithm and Dijkstra Algorithm for Finding Shortest Path in GIS Application. International Journal of P2P Network Trends and Technology (IJPTT) – Volume 5 – February 2014 ISSN: 2249-2615

[9] Stefan Hougardy , ―The Floyd-Warshall Algorithm on Graphs with Negative Cycles ―,Information Processing Letters 110 (2010), 279-281

[10]https://en.wikipedia.org/wiki/Johnson%27s_algorithm

[11] Himanshu Garg, Paramjeet Rawat; "An Improved Algorithm for Finding All Pair Shortest Path". International Journal of Computer Applications (0975 – 8887) Volume 47– No.25, June 2012.