SHORTEST PATH ALGORITHMS: A COMPARATIVE ANALYSIS

R.Kiruthika M.C.A. M.Phil.*

Dr. R. Umarani M.C.A., M.Phil., Ph.D**

Abstract:

Now a days routing is the vital problem while forwarding information from one node to another node in communication networks. Routing in today's network is accomplished by shortest path routing algorithm. There are different algorithms for finding paths in the network. The studied algorithms are Bellmen Ford's, Dijkstra's algorithm, Floyd algorithm and Johnson's algorithm. In search algorithms, when a number of nodes get increased in the network, then the number of searches and complexity of the search algorithm also get increased. In this paper, analysis/comparison of shortest path algorithm is being done and it has been concluded that researchers have got remarkable success in designing better algorithms in the terms of space & time complexity to solve shortest path algorithms.

Keywords: Bellmen Ford's, Floyd algorithm, Dijkstra's, Johnson's.

* Assistant Professor in School of It and Science, Dr.G.R.Damodran College of Sciences, Coimbator.

** Associate Professor in Computer Science, Sri Sarada College for Women, Salem.

I. INTRODUCTION:

The shortest path computations are one of the problems in graph theory. If G(v,e) is directed weighted graph, where v represents the set of vertices of graph & e represents the set of edges of graph. |v | represents the total number of vertices in graph & |e| represents the total number of edges in the graph. In shortest path problems, a directed weighted graph is given & the goal is to determine the shortest path among vertices [1]. The shortest path problem can be categorized in to two different problems; single source shortest path problem and all pair shortest algorithm. In Single source shortest path problems, a graph G(v,e) is being given and the goal is to find a shortest path from a given fixed vertex to all other vertices of the graph. In all pair shortest path problems [2], the goal is to finding the shortest paths between all pairs of vertices of a graph.

II SHORTEST PATH ALGORITHMS:

A Bellman ford algorithm

Bellman ford algorithm can be used to solve the single source shortest path problems in which edges of a given diagraph can have negative weight as long as the graph contains no negative cycle. The computational time of this algorithm increases exponentially with the increase in the number of nodes. This algorithm returns a Boolean value TRUE if the given digraph contains no negative cycles that are reachable from source vertex otherwise it returns Boolean value FALSE.

```
BELLMAN-FORD (G, w, s)
    INITIALIZE-SINGLE-SOURCE (G, s)
    for each vertex i = 1 to V[G] - 1 do
        for each edge (u, v) in E[G] do
            RELAX (u, v, w)
     for each edge (u, v) in E[G] do
        if d[u] + w(u, v) < d[v] then
            return FALSE
        return TRUE
```

If G(V,E) be the graph, Where V represents the set of vertices & E represents the set of edges, then the time complexity [3] for Bellman Ford algorithm is O(|V||E|).

B Dijkstra's algorithm

Dijikstra algorithm can also be used to solve the single source shortest path problems on a given weighted, directed graph G(V,E) if and only if all the weights of edges are positive The time complexity of Dijikstra algorithm depends upon the implementation of min priority queue. If the Min priority queue is being implemented by using binary heap, then the time complexity [3] of Dijikstra algorithm is O((V+E)logV). But if the min priority queue is being implemented by using Fibonacci Heap, then the time complexity for Dijikstra algorithm is O(VlogV+E).

```
DIJKSTRA (G, w, s)
  INITIALIZE SINGLE-SOURCE (G, s)
     S ← { }     // S will ultimately contains vertices of
                    final shortest-path weights from s
    Initialize priority queue Q i.e., Q ← V[G]
    while priority queue Q is not empty do
        u ← EXTRACT_MIN(Q)    // Pull out new vertex
        S ← S ∪ {u}
         // Perform relaxation for each vertex v adjacent to u
         for each vertex v in Adj[u] do
           Relax (u, v, w)
```

C Floyd Warshall algorithm

The Floyd–Warshall algorithm (sometimes known as the WFI Algorithm or Roy–Floyd algorithm, since Bernard Roy described this algorithm in 1959) is a graph analysis algorithm for finding shortest paths in a weighted, directed graph. A single execution of the algorithm [4] will find the shortest paths between all pairs of vertices. The Floyd–Warshall algorithm is named after Robert Floyd and Stephen Warshall; it is an example of dynamic programming, a technique that takes advantages of overlapping sub problems, optimal substructure [5], and trade space for time to improve the runtime complexity of algorithms.

The algorithm considers the "intermediate" vertices of a shortest path, where an intermediate vertex [6] of a simple path p = v1, v2,..., vl is any vertex of p other than v1 or vl, that is any vertex in the set {v2, v3,..., vl-1}.The Floyd-Warshall algorithm is based on the following observation. The vertices of G are V = {1, 2,..., n}, let us consider a subset {1, 2,..., k} of vertices for some k. For any pair of vertices i, j Є V, consider all paths from i to j whose intermediate vertices are all drawn from {1, 2,..., k}, and let p be a minimum weight path from among them.

- If k is not an intermediate vertex of path p, then all intermediate vertices of path p are in the set {1, 2,..., k - 1}. Thus, a shortest path from vertex i to vertex j with all intermediate vertices in the set {1, 2,..., k - 1} is also a shortest path from i to j with all intermediate vertices in the set {1, 2,..., k}.
- If k is an intermediate vertex of path p, then we break p down into i→(p1) k→(p2)j. p1 is a shortest path from i to k with all intermediate vertices in the set {1, 2,..., k}. Because vertex k is not an intermediate vertex of path p1, we see that p1 is a shortest path from i to k with all intermediate vertices in the set {1, 2,..., k - 1}. Similarly, p2 is a shortest path from vertex k to vertex j with all intermediate vertices in the set {1, 2,... k - 1}.

Let $d_{ij}{}^{(k)}$ be the weight of a shortest path from vertex i to vertex j for which all intermediate vertices are in the set {1, 2,..., k}. When k = 0, a path from vertex i to vertex j with no intermediate vertex numbered higher than 0 has no intermediate vertices at all. Such a path has at most one edge, and hence $d_{ij}{}^{(0)} = w_{ij}$. A recursive definition is:

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0\,, \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{if } k \geq 1\,. \end{cases}$$

Because for any path, all intermediate vertices are in the set {1, 2,..., n}, the matrix gives the final answer:$d_{ij}^{(n)}$=shortest path between i and j for all i, j Є V.

The following returns the matrix D(n) of shortest-path weights.

```
FLYOD-WARSHALL (W)
  n ← rows [W]
  D (0) ← W
  for k ← 1 to n do
  for i ← 1 to n do
  for j ← 1 to n do
  $d_{ij}^{(k)} \leftarrow \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)},)$
  return D(n)
```

The running time of the Floyd-Warshall algorithm is determined by the triply nested for loops of lines 3-6. Because each execution of line 6 takes O(1) time, the complexity [7] of the algorithm is Θ(n3).

D Johnson's algorithm

Johnson's algorithm can be used to solve all pair shortest path problems within the time complexity of $O(V^2 \log V+VE)$ time. It allows some of the edge weights to be negative numbers, but no negative-weight cycles may exist. It works by using the Bellman–Ford algorithm to compute a transformation of the input graph that removes all negative weights, allowing Dijkstra's algorithm to be used on the transformed graph. It is named after Donald B. Johnson, who first published the technique in 1977.

If the graph contain negative cycle then Johnson‘s algorithm reports that the graph contains negative cycle. If the graph does not contain negative cycle then Johnson‘s algorithm returns a particular matrix [8][9] which shows the shortest distance among vertices.

Johnson's algorithm uses as subroutines both Dijkstra's algorithm and the Bellman-Ford algorithm. Johnson's algorithm uses the technique of re-weighting. Johnson's algorithm consists of the following steps:

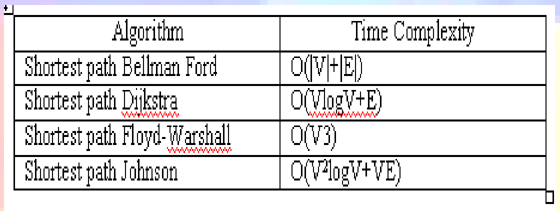
1. First, a new node q is added to the graph, connected by zero-weight edge to each other node.

2. Second, the Bellman-Ford algorithm is used, starting from the new vertex q, to find for each vertex v the least weight h (v) of a path from q to v. If this step detects a negative cycle, the algorithm is terminated.

3. Next the edges of the original graph are reweighted using the values computed by the Bellman-Ford algorithm: an edge from u to v, having length w(u,v), is given the new length w(u,v) + h(u) −h(v). (h: V -> R be any function mapping vertices to real Numbers.)

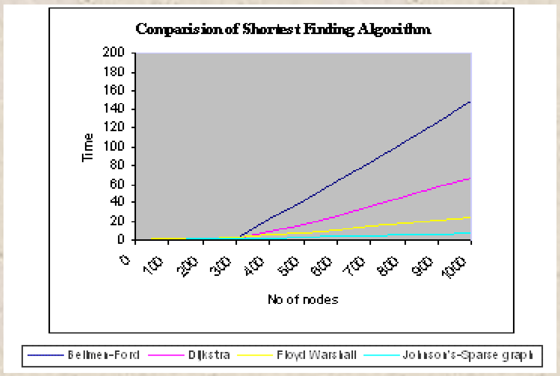
4. Finally, for each node s, Dijkstra's algorithm is used to find the shortest paths from s to each other vertex in the reweighted graph

II COMPARISION OF ALGORITHM:



Algorithm	Time Complexity
Shortest path Bellman Ford	O(|V|+|E|)
Shortest path Dijkstra	O(VlogV+E)
Shortest path Floyd-Warshall	O(V3)
Shortest path Johnson	O(V²logV+VE)

The time complexity of Bellman Ford algorithm is O(|V|+|E|), the time complexity of Dijkstra algorithm using Fibonacci heap is O(VlogV+E), the time complexity of Floyd-Warshall algorithm is O(V3), the time complexity of John’s algorithm using Fibonacci heaps in the implementation of Dijkstra algorithm O(V^2logV+VE), the algorithm uses O(VE) time for the Bellman-Ford stage of the algorithm, and O(V log V + E) for each of V instantiations of Dijkstra's algorithm.



III CONCLUSIONS:

In this paper, analysis of shortest path algorithms is being done and it has been concluded that researchers have success in designing better algorithms in the terms of time and space complexity to solve shortest path algorithms. Among Floyd Warshall and Johnson's, Floyd Warshall is quite simple, efficient and achieves a good running time of Θ(n3), but when the graph is sparse, the total time for Johnson's is faster than the Floyd-Warshall algorithm.

REFERENCES:

- Ravindra K. Ahuja, Kurt Mehlhorn, James B.Orlin, and Robert E. Tarjan. Faster algorithms for the shortest path problem. Journal of the ACM,37:213–223, 1990.
- Michael L. Fredman. New bounds on the complexity of the shortest path problem. SIAM Journal on Computing, 5(1):83–89, 1976.
- C. C. McGeoch. All pairs shortest paths and the essential subgraph. Algorithmica, 13(5):426– 441,1995.
- Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L. Introduction to Algorithms (2st ed.). MIT Press and McGraw-Hill.
- Ming-Yang Kao, Encyclopedia of Algorithms, SpringerLink (Online service)
- Jorgen Bang-Jensen, Gregory Gutin, Digraphs: Theory, Algorithms and Applications (2nd edition), Springer
- Jan van Leeuwen, Handbook of theoretical computer science,(Vol A Algorithms and Complexity),Elsevier

- Zvi Galil and Oded Margalit. All pairs shortest distances for graphs with small integer length edges.Information and Computation, 134(2):103–139,1997.
- Zvi Galil and Oded Margalit. All pairs shortestpaths for graphs with small integer length edges. Journal of Computer and System Sciences,54(2):243–254, 1997.