Bijou Detouring - A Dynamic Node Level Routing Algorithm

G.Reshma
Assistant Professor, Department of Information Technology PVPSIT, Kanuru (AP) INDIA

K.Swarupa Rani
Assistant Professor, Department of Information Technology PVPSIT, Kanuru (AP) INDIA

D.Leela Dharani
Assistant Professor, Department of Information Technology PVPSIT, Kanuru (AP) INDIA

D.Anusha
Assistant Professor, Department of Information Technology PVPSIT, Kanuru (AP) INDIA

Abstract— This paper holds a candle light on the Dijkstra algorithm and Distance Vector routing algorithm in detail, and illustrates the implementation technique of these algorithms and the drawbacks of these algorithms: the nodes require square-class memory, so it is difficult to calculate the shortest path of the major nodes in vast networks. On the other hand, it describes the Bijou detouring algorithm which is an optimization algorithm based on Dijkstra algorithm and Distance Vector algorithm. The Bijou detouring algorithm makes full use of relation of nodes in the network topology, and makes use of correlation matrix that contains substantial infinite value, making it more suitable analysis for the networks with mass data. It is proved that the algorithm saves a lot of memory and is more suitable to the networks having huge nodes.

Index Terms—Indexing, Hop count, insert, traversal.

I. INTRODUCTION

The shortest path problem is one of the fundamental problems with numerous applications. In this paper we study most common variants of the problem, where the goal is to find a point-to-point shortest path in a directed graph. Our goal is to find a fast algorithm for answering point-to-point shortest path queries. 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. Prior research does not provide a clear direction for choosing an algorithm when one faces the problem of computing shortest paths on real road networks. In this paper we are going through a case study of two algorithms to find the shortest path between two places and come at a new algorithm overcoming the drawbacks of these techniques and perform the shortest path calculation more efficiently.

This paper introduces the classical Dijkstra algorithm and Distance Vector routing algorithm in detail, and illustrates the method of implementation of the algorithms and the disadvantages of the algorithms: the network nodes require square-class memory, so it is difficult to quantify the shortest path of the major nodes. At the same time, it describes the Bijou detouring algorithm which is an optimization algorithm based on Dijkstra algorithm and Distance Vector algorithm. The Bijou detouring algorithm makes full use of connection relation of nodes in the network topology information, and makes use of correlation matrix that contains substantial infinite value, making it more suitable analysis of the network for mass data. It is proved that the algorithm can save a lot of memory and is more suitable to the network with huge nodes. The technique described in this paper can be specialized for the case where only shortest paths between a subset of nodes in the network have to be maintained.

A. *The shortest path problem*

A road traffic network is represented by a digraph G (N, A) that consists of a certain number of nodes N and a certain number of arcs A (or links used in this paper). Denote the count of nodes |N|=n and the count of links |A|=m. A link a = (i, j) $\in$ A is directed from node i to node j and has an associated generalized cost cij. This cost represents the impedance of an individual vehicle passing through that link and is generally described by link travel time, link length, tolls, etc. Without losing generality, the term link travel time is used mostly in this paper. A path from an origin (o) to destination (d) may be defined as a list of links in order: (o, j), (i, d) and the travel time of the path is the sum of travel times on the individual links. The difficulty is to find the path that has the

least total travel time from the origin node to the destination node. 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 minimum.

II. SOME SHORTEST PATH ALGORITHMS

We have gone through study of various algorithms for the computation of shortest path some of them are:

A. *DIJKSTRA ALGORITHM*:

Dijkstra's algorithm, introduced by Dutch computer scientist Edsger Dijkstra in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree. This algorithm is regularly used in the network routing protocols where all edge weights are non negative, all shortest-path weights must exist. It is also well-known as the single source shortest path problem. It computes length of the shortest path from the source to each one of the remaining vertices in the graph.

Assume you would like to find the shortest path between two intersections on a city map, a starting point and a destination. The order is theoretically simple: to start, mark the distance to every intersection on the map with infinity. This is done not to imply there is an infinite distance, but to note that intersection has not yet been visited; some variants of this method simply leave the intersection unlabeled. Now, at every iteration, select a current intersection. For the first iteration, the current intersection will be the starting point and the distance to it (the intersection's label) will be zero.

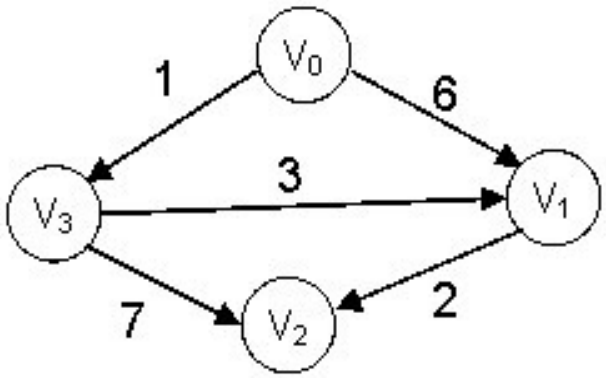

Fig. 1. Directed Weighted Graph

For following iterations (after the first), the current intersection will be the closest unvisited intersection to the starting point—this will be easy to find. From the current intersection, update the distance to every unvisited intersection that is directly connected to it. This is done by finding the sum of the distance between an unvisited intersection and the value of the current intersection, and renaming the unvisited intersection with this value if it is less than its present value. As a result, the intersection is relabeled if the path to it from the current intersection is shorter than the earlier known paths. To facilitate shortest path identification, in pencil, mark the road with an arrow pointing to the relabeled intersection if you label it, and delete all others pointing to it. After you have updated the distances to each adjacent intersection, mark the current intersection as visited and select the unvisited intersection with least distance (from the starting point) – or the lowest label—as the current intersection. Nodes marked as visited are labeled with the shortest path from the initial point to it and will not be revisited. Continue this process of updating the neighboring intersections with the shortest distances, then marking the current intersection as visited and moving onto the closest unvisited intersection until you have marked the destination as visited. Once you have marked the destination as visited (as is the case with any visited intersection) you have calculated the shortest path to it, from the starting point, and can trace back, following the arrows in reverse.

Let we name the node at which we are starting as the initial node. Let the distance of node Y be the distance from the initial node to Y. Dijkstra's algorithm will assign some initial distance values.

Step by Step Implementation of DIJKSTRA'S ALGORITHM

- Assign to all nodes a tentative distance value: set initial node to zero and the remaining as infinity.
- Mark all nodes unvisited. Take the initial node as current. Create a set of the unvisited nodes as the

unvisited set consisting of all the nodes.

- For the present node, take all of its unvisited neighbor's and calculate their tentative distances. For example, if the current node B is marked with a distance of 6, and the edge connecting it with a neighbor A has length 2, then the distance to A(through B) will be 6 + 2 = 8.
- When we are done considering the entire neighbor's of the present node, mark the present node as visited and erase it from the unvisited set. A visited node will never be checked.
- If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest unsure distance among the nodes in the unvisited set is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has completed.
- Select the unvisited node that is marked with the smallest unsure distance, and set it as the new "present node" then go back to step 3.

Disadvantages of Dijkstra's Algorithm

- None of the algorithms based on the Dijkstra's algorithm has achieved the linear time complexity.
- When the number of nodes (n) is huge, it is uphill task to apply Dijkstra's Algorithm as it uses square class memory for functionality.
- When the number of the nodes is very large, it occupies a lot of CPU memory.
- This algorithm is very difficult to apply in network analysis for more data.
- This algorithm uses three matrices to calculate the shortest path relayed on network cost matrix.

B. Distance Vector Routing Algorithm:

Distance-vector protocols are based on calculating the direction and distance to any link in a network. The less cost route between any two nodes is the route with less distance. Each node maintains a vector of minimum distance to every node. Based on the minimum distance the destination is selected.The Distance Vector algorithm is an algorithm that computes shortest paths from a single source node to all of the other nodes in a weighted digraph. It is less efficient than Dijkstra's algorithm for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative in number. A most successful attempt for implementation for the distance vector routing protocol is the Bellman-Ford algorithm.

The Bellman-Ford algorithm computes single-source shortest paths in a weighted di-graph. The algorithm is named after its developers, Richard Bellman and Lester Ford, Jr. For graphs which have non-negative weights, Dijkstra's algorithm is the solution. The Bellman-Ford algorithm is used mainly for graphs with negative weights. The algorithm can identify negative cycles and report their existence, but it cannot generate a correct "shortest path" if a negative cycle is accessible from the source.

It uses the same concept as the Dijkstra's algorithm but can handle negative edges as well. It has a improved running time than Dijkstra's algorithm. For graphs having only non-negative edge weights, the faster Dijkstra's algorithm also solves the problem. Though, if a graph contains a "negative cycle", i.e., a cycle whose edges sum to a negative value, then paths of arbitrarily low weight can be constructed by frequently following the cycle, so there may not be a *shortest* path. In such a case, the Bellman-Ford algorithm can detect negative cycles and report their existence, but it cannot generate a correct "shortest path" answer if a negative cycle is accessible from source.

Step by Step Evaluation of Distance Vector Routing:

```
Step 1: initialize graph
  for each vertex v in vertices:
  if v is source then distance[v]:= 0
  else distance[v]:= infinity
  predecessor[v]:= null

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"
```

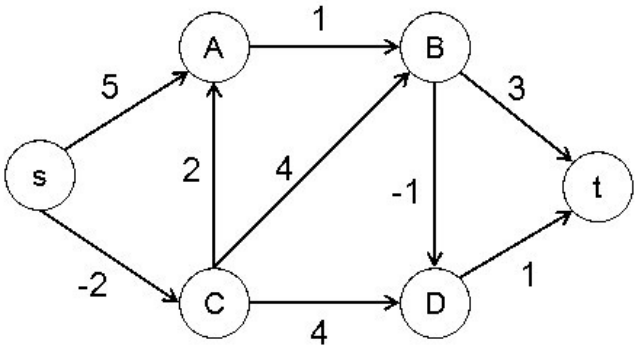

Fig. 2. A Graph with Negative Edges

The algorithm just relaxes all the edges, and does these |V|-1 times, where |V| is the number of vertices in the graph. The repetitions allow minimum distances to transmit perfectly throughout the graph, since in the absence of negative cycles; the shortest path can visit each node at most only once.

Disadvantages of Distance Vector Routing Algorithm

- Count to infinity occurs more often where one of the nodes losses its distance information to other node.
- Loops may be formed.
- Slow Convergence for copious data as it takes a lot of time to calculate and enhance.
- Has a very contracted Scalability as it complexity increases with the no of nodes.
- High Bandwidth Consumption for even small nodal values as the system use is more.

C. Bijou Detouring Algorithm:

We propose a new algorithm that performs node level routing implementation rather than route level implementation. Here we calculate the shortest traversal distances between all the nodes within the graph rather than in accordance to a single node .In this algorithm we sort all the nodes based on their hocount that is the number of neighboring vertices to the node.

We perform indexing of the nodes based on the hocount and node labels. We calculate the possible node traversals for every node both directly and with a single hop iteratively by increasing hocount value. By the end of function evaluation for every node we arrive at a updated cost matrix containing the least possible distance between any corresponding nodes. We also maintain a two dimensional array which stores the last accessed node for every node traversal in the cost matrix.

By using the updated cost matrix and last visited array we can retrieve the shortest path cost and its route for any two nodes.

Step by Step Evaluation of Bijou Detouring Algorithm:

- Take input from the user: The number of nodes ‘n’
- Take input from the user: The cost matrix ‘G[][]’
- Copy the cost matrix values to another matrix ‘E’
- Calculate the hopcount for all the nodes in the graph
- Sort the nodes based upon the hopcount
- Indexing of the nodes in the graph is done based upon the hopcount using a two dimensional array ‘A’

- Now, a function 'hop' is defined which calculates the possible traversal distances to the nodes directly and with one intermediate node
- The function 'hop' is executed iteratively for all the nodes in the graph based up on their index values from the two dimensional array 'A'
- After the execution of function hop for all the nodes the cost matrix gets updated with the least possible weights between the corresponding nodes
- We also maintain a 2D array 'N' which stores the previous visited node for every traversal path between any two nodes
- Now, we take input from the user any two nodes for which the least distance is to be updated cost matrix values we arrive at the minimum cost between two different nodes calculated
- By using and the path is traced back using the 2D array N
- Finally the minimum distance between the two nodes along the path is displayed to the use

III. CONCLUSION

Going through comparative study of various algorithms we came through various drawbacks of algorithm and also advantages of these approaches. Each and every algorithm is unique and plays specific role in obtaining best possible paths for the users. As per study we came to know that Dijkstra algorithm consumes lot of time and doesn't work well on dense networks and Distance Vector Routing algorithm has less scalability and convergence rate. It depends on the network and its complexity and also time dependency that to use the algorithm that best suits the networks. Our Bijou Detouring algorithm keep ups the merits from both these algorithms and overcome the drawbacks of them to the maximum possible and performs the short distance calculation using minimal data resources with lesser time complexity. Still researches have been going on for fastest algorithms for road networks to achieve best possible route and the optimal path.

IV. FUTURE ENHANCEMENTS

The future enhancements like the dynamic features can be made possible to our proposed algorithm like preferring a personalized route search say only highways, no toll gates etc., exclusion of certain points in the path, compulsory inclusion of certain points and also enhancements to get the second best possible path and any data losses like path between two nodes can be trace backed using the correlation matrix and last visited array which enables in error checking and hyphenation. This algorithm can also be used in standalone applications to specify path like local transit, malls, industries etc., also a graphical interactable interface and database updation and retrieval using php or any html pages.

REFERENCES

[1] Dijkstra, E. W. (1959). "A note on two problems in connexion with graphs". Numerische Mathematik 1: 269–271. Doi: 10.1007/BF01386390.

[2] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001). "Section 24.3: Dijkstra's algorithm". Optimization algorithms". 25th Annual Symposium on Foundations of Computer Science. IEEE. pp. 338–346. doi:10.1109/SFCS.1984.715934.

[3] "RFC1058 - Routing Information Protocol", C. Hedrick, Internet Engineering Task Force, June 1988

[4] "RFC1723 - RIP Version 2 Carrying Additional Information", G. Malkin, Internet Engineering Task Force, November, 1994

[5] "RFC2453 - RIP Version 2", G. Malkin, Internet Engineering Task Force, November, 1998

[6] Bellman, Richard (1958). "On a routing problem". Quarterly of Applied Mathematics 16: 87–90. MR 0102435.

[7] Ford Jr., Lester R. (August 14, 1956). Network Flow Theory. Paper P-923. Santa Monica, California: RAND Corporation.

[8] Moore, Edward F. (1959). "The shortest path through a maze". Proc. Internat. Sympos. Switching Theory 1957, Part II. Cambridge, Mass.: Harvard Univ. Press. pp. 285–292. MR 0114710.

[9] Wu Qishi, The Application of Genetic Algorithm in GIS Network Analysis, International Archives of Photogrammetry and Remo, 2000, 33, 1184-1191.

[10] Benjamin Zhan F. Three Fastest Shortest Path Algorithms on Real Road Networks. Journal of Geographic Information and Decision Analysis, 1997, 1 (1): 69-82.

[11] Liu Yanliang, Wang Pengtao, Optimization model of complicated network and shortest path algorithm, Journal of Tianjin University of Technology 2006, 22(1): 42-44.

[12] Wang Jiechen, Zhang Wei, Mao Haicheng, Study of Graph Simplifying Method in GIS Network Analysis, Acta Geodaeticaet Cartographica Sinica, 2001, 30(3): 263-268.

[13] Zeng Wen, Design of MAPGIS Pipeline Management Development Platform, Journal of China University of Geosciences, 2002, 27(3): 250-254.