DESIGN AND IMPLEMENTATION OF MULTI-PARAMETER DIJKSTRA'S (MPD) ALGORITHM: A SHORTEST PATH ALGORITHM FOR REAL-ROAD NETWORKS

*NISHTHA KESSWANI,#DINESH GOPALANI
*ASSISTANT PROFESSOR, CENTRAL UNIVERSITY OF RAJASTHAN, INDIA
#ASSISTANT PROFESSOR, MALAVIYA NATIONAL INSTITUTE OF TECHNOLOGY, JAIPUR.

ABSTRACT

Several shortest path algorithms have been suggested by the researchers and Dijkstra's shortest path algorithm is the most appropriate one when there is a single source-single destination problem. Though Dijkstra's algorithm is targeted towards single source-single destination problem but it considers only the weights or distance between the nodes as a criterion for selecting the shortest path. Taking the real road networks into consideration, we have suggested a modification of the Dijkstra's algorithm, the multi-parameter Dijkstra's algorithm (MPD) that considers multiple parameters into consideration. Apart from the distance between any two nodes, it considers factors such as time taken to travel from the source to the destination, congestion of the route etc. so that the user can select the desired route based on his/her preferences.

In the implementation part of the algorithm, we have designed a navigation system for Jaipur that incorporates the multi-parameter aspect. It has been designed for the lay man, so that he is able to view the shortest path between a source-destination pair, and also the available bus routes across the specified path. The user can also view the traffic congestion across the selected route. The navigation system has been designed so that it aids the common man in navigating across the city.

I. INTRODUCTION

With the development of geographic information systems (GIS) technology, network and transportation analysis within a GIS environment have become a common practice in many application areas. A key problem in network and transportation analysis is the computation of shortest paths between different locations on a network. Sometimes this computation has to be done in real time. For the sake of illustration, let us have a look at the case of a 108 call requesting an ambulance to rush a patient to a hospital. Today it is possible to determine the fastest route and dispatch an ambulance with the assistance of GIS. Because a link on a real road network in a city tends to possess different levels of congestion during different time periods of a day, and because a patient's location can not be expected to be known in advance, it is practically impossible to determine the fastest route before a 108 call is received. Hence, the fastest route can only be determined in real time. In some cases the fastest route has to be determined in a few seconds in order to ensure the safety of a patient. Moreover, when large real road networks

are involved in an application, the determination of shortest paths on a large network can be computationally very intensive [3]. Because many applications involve real road networks and because the computation of a fastest route (shortest path) requires an answer in real time, a natural question to ask is: *Which shortest path algorithm runs fastest on real road networks*?

There are several algorithms mentioned in the literature [1,2] like Dijkstra's algorithm which is a single source- single destination shortest path algorithm, Bellman-Ford algorithm aimed to solve single source shortest path algorithm, A* search algorithm solves single pair shortest path problems using heuristics, Floyd Warshall algorithm and Johnson's algorithm find all-pairs shortest path and Perturbation algorithm finds locally shortest path.

Several modifications of Dijkstra's algorithm like Dijkstra's algorithm with buckets, Dijkstra's algorithm with double buckets, Dijkstra's algorithm with approximate buckets have also been suggested in [3]. We have proposed a modified version of Dijkstra's algorithm, a multi-parameter Dijkstra's algorithm (MPD).As compared to other state-of-the-art shortest path algorithms, this algorithm use multiple parameters such as distance, cost and congestion across the routes. The overhead of using buckets in other versions of Dijkstra's has been overcome in the proposed algorithm.

Using the proposed algorithm, a navigation system for Jaipur city has been suggested by us. The database for Jaipur city was created and graph has been generated from the database. From the graph, the shortest path was calculated and the results were displayed. In the navigation system that has been designed by us, the user can view the shortest path, the bus routes that are available for Jaipur Bus and the congestion across the routes. The congestion factor varies on a scale of 1 to 10. Higher the value of congestion factor, higher is the traffic congestion across the specified route.

II. SHORTEST PATH ALGORITHMS: A STATE-OF-THE ART

There are several shortest path algorithms. The shortest path algorithms can be classified into following categories:

1. *Single source shortest path algorithms:* Find shortest path from source vertex to all other vertices in the graph
2. *Single destination shortest path algorithms:* Find shortest path from all vertices to a single destination. This can be reduced to single source shortest path problem by reversing the edges.
3. *All-Pairs shortest path algorithms:* Find shortest path between every pair of vertices.

During the initial stages of the project, following shortest path algorithms were reviewed:

- *Dijkstra's algorithm[1]:* which is a single source-single destination shortest path problem
- *Bellman-Ford algorithm [1]:* aimed to solve single source shortest path problem in which edge weights may be negative.
- *A* search algorithm [4]:* single pair shortest path algorithm using heuristics.
- *Floyd Warshall algorithm [2]:* solves all-pairs shortest paths.
- *Johnson's algorithm [4]:* solve all-pairs shortest path problem.
- *Perturbation [4]:* finds locally shortest path.

Dijkstra's algorithm [1] solves the single source shortest path problem. It finds the path with lowest cost between a vertex and every other vertex. Another very common algorithm is Bellman Ford algorithm [1] that computes single source shortest path in a directed graph. If a graph contains a negative cycle i.e. a cycle with sum of edges equal to a negative value, then walks of arbitrarily low value can be constructed. Bellman ford algorithm can detect negative cycles, but cannot produce a correct answer if a negative cycle is reachable from the source. This algorithm is similar to Dijkstra's algorithm but it relaxes all the edges and the minimum distance is propagated throughout the graph.

Another algorithm A* search [4] uses greedy best first search and finds least cost path from a given initial node to a goal node. This algorithm uses distance and cost heuristic function f(x) to determine order in which search visits nodes in the tree. The distance-plus-cost heuristic is the sum of two functions:

(i) Path-cost function g(x) – cost from starting node to current node
(ii) Admissible "heuristic estimate" h(x) of the distance to the goal

h(x) part of f(x) must be admissible heuristic i.e. it must not overestimate the distance to the goal. Thus h(x) might represent straight line distance to goal since it is physically the smallest possible distance between any two nodes. If heuristic h satisfies the additional condition

$$h(x) \leq d(x,y) + h(y)$$

for every edge x,y of the graph (where d(x,y) denotes the length of that edge, then h is called monotone or consistent.

No node needs to be processed more than once and A* is thus equivalent to Dijkstra's algorithm with reduced cost

$$d'(x,y) = d(x,y) - h(x) + h(y)$$

A* takes into account distance already travelled, not simply the local cost from previously expanded node. Starting with the initial node, it maintains a priority queue of the nodes to be

traversed known as an open set. Lower the f(x) for a given node x, higher its priority. At each

step, the node with lowest f(x) is removed from the queue. The values of f and h of the neighbors are updated accordingly and these neighbors are added to the queue. The algorithm continues until a goal node has lower f value than any node in the queue. The algorithm may also update each neighbor with its predecessor; this information can be used to reconstruct the path by working backwards from the goal node.

Another shortest path algorithm is Floyd Warshall algorithm [2] for finding shortest path in a weighted graph with positive and negative edge weights. Single execution of the algorithm finds lengths of all the shortest paths between all pairs of vertices. This algorithm incrementally improves the estimate on shortest path between two vertices, until the estimate is optimal. There can be two candidates for shortest path:

(i) True shortest path that uses vertices in the set {1,2,….k} or

(ii) There exists some path that goes from I to k+1 then from k+1 to j. Best path from I to j uses vertices 1 through k is defined by the shortest path(I,j,k) defined by the following formula:

$$shortestPath(i,j,k)$$

$$= \min \begin{Bmatrix} shortestPath(i,j,k-1), \\ shortestPath(i,k,k-1) + shortesPath(k,j,k-1) \end{Bmatrix}$$

$$shortestPath(i,j,0) = edgeCost(i,j)$$

Johnson's algorithm [4] finds the shortest path between all pairs of vertices in a sparse directed graph. It allows some edge weights to be negative numbers but no negative weight cycles may exist. Perturbation theory [4] comprises mathematical methods that can be used to find an approximate solution to a problem which cannot be solved exactly by starting from exact solution of a related problem. The problem can be formulated by adding a small term to the mathematical description of the exactly solvable problem. In this theory, desired solution can be expressed in some power series. The leading term is the solution of the problem while further terms describe the derivation of the solution.

A study of fastest shortest path algorithms has been given in [3]. The researchers have identified a set of three shortest path algorithms that run fastest on real road networks. These algorithms are 1) the graph growth algorithm implemented with two queues, 2) Dijkstra algorithm implemented with double buckets and 3) Dijkstra algorithm implemented with approximate buckets. Recent evaluations of shortest path algorithms like Zhan and Noon's evaluation have also been mentioned.

III. MULTI-PARAMETER DIJKSTRA'S (MPD) ALGORITHM

Several algorithms available in literature [1, 2, 3, 4] provide solution to the shortest path problem. After conducting extensive research on the existing shortest path algorithms, it was observed that Dijkstra's shortest path algorithm is the most appropriate for calculating shortest paths in real-road networks as it involves calculation of shortest path between single source-single destination pair. But it needs to be modified to introduce several other factors in the real-life scenario. Factors such as the time taken to travel, the monetary cost and congestion factor must also be considered before implementing the algorithm to calculate the optimal paths. The proposed algorithm incorporates these parameters for an efficient calculation of the shortest path.

Algorithm1: Multi-parameter Dijkstra's algorithm

Algorithm MPD(Graph,source,destination,choice)

```
1.  begin
2.  for each vertex v in Graph do
3.  alternate_path[i]:=NULL;
4.  dist[v] := infinity;
5.  weight_update(choice);
6.  for each vertex v in Graph do
7.  if v= source or v = destination then
8.  for each neighbour u of v do
9.  if alternate_path[i] > dist[u] + distance(u,v) then
10. alternate_path[i]:= dist[u] + distance(u,v);
11. end if
12. end for
13. end if
```

14. end for

15. end

//choice indicates parameters such as time, distance, cost, congestion

Algorithm 2: Incorporate the multiple parameters

Algorithm weight_update(choice)

1. begin
2. if choice = distance then;
3. else if choice = time then
4. distance := distance * time factor;
5. else
6. distance := distance * congestion factor;
7. return distance;
8. end;

As compared to other state-of-the-art algorithms this multi-parameter version allows the user to select one or more parameters such as distance, time or congestion. The distance factor indicates the actual distance between any two nodes in real-time scenario. The time factor indicates the time taken to travel from one node to another and congestion factor varies on a scale of 1 to 10 where 1 indicates a low and 10 indicates a high level of congestion. The flow of the algorithm is indicated in Figure 1.1.

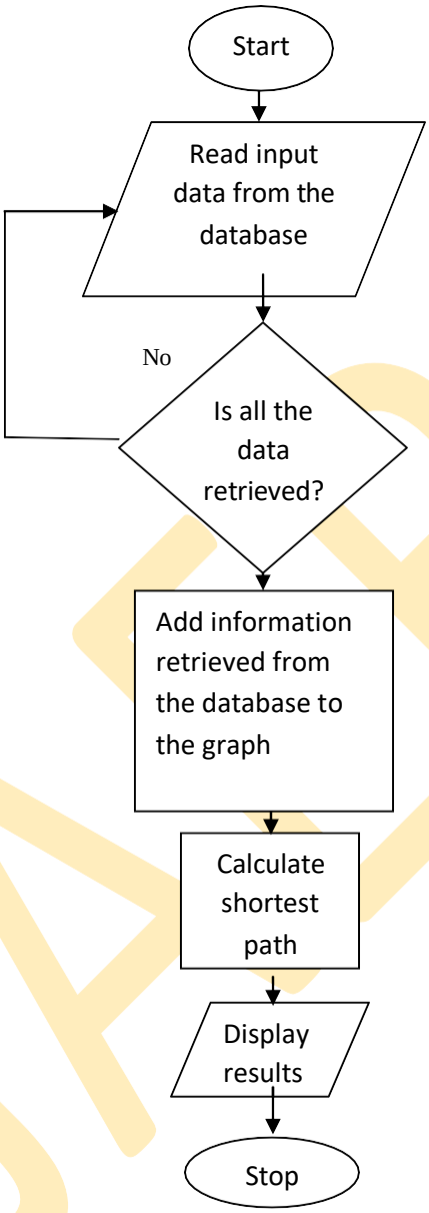

Figure 1: Implementation of the multi-parameter Dijkstra's algorithm

IV. EXPERIMENTAL RESULTS

The Multi-parameter Dijkstra's algorithm has been tested on Jaipur city database and it has been found that it is capable of displaying not only multiple routes but the optimal path between any source-destination pair. Also, the incorporation of congestion factor provides greater flexibility to the user so that he/ she can choose a route that is less congested.

The proposed algorithm has been compared to the existing algorithms. Several modifications of Dijkstra's algorithm have also been proposed by the researchers [3] such as

Dijkstra's algorithm with buckets (DKB), Dijkstra's algorithm with double buckets (DKD),

Dijkstra's algorithm with overflow bag implementation (DKM), Dijkstra's algorithm with approximate buckets implementation (DKA). In the study of the modifications of Dijkstra's algorithms, it was found that:

- Nodes may be: unlabelled, temporarily labelled or permanently labelled in all of the proposed algorithms.
- The bottleneck was that all the nodes have to be visited to select the node with min distance
- In Dijkstra's algo with buckets (DKB) nodes are sorted by distance labels. Bucket k stores all temporarily labelled nodes whose distance is within a certain range. Requires nC+1 buckets where C= max arc length of a network
- Memory requirement in DKB can be reduced using either Overflow bag implementation (DKM) or approximate buckets implementation(DKA).
- DKM maintains a<C+1 buckets. Only temporarily labeled nodes whose distance labels fall in the range[a(i),a(i)+a-1] are contained in ith bucket. Other nodes are maintained in overflow bag
- DKA: a bucket i contains temporarily labelled nodes with distance in the range [i*b,(i+1)*b-1] where b= constant. Approximate means that values of distance labels in a bucket are not exactly the same as DKB
- Dijkstra's algorithm with double buckets (DKD): maintains two levels of buckets, high level and low level. Nodes in low level buckets are examined and when all low level buckets are scanned, high level buckets are moved to low level.

When compared to these algorithms, the proposed multi-parameter version was found to be more efficient due to the following reasons:

- *Naïve Dijkstra's algorithm:* uses only one parameter i.e. distance, whereas the multi-parameter version provides multiple parameters such as distance, cost and congestion.
- *Dijkstra's algorithm with Buckets:* maintains nC+1 buckets. It requires extra overhead of sorting the nodes by distance labels so that bucket k stores all the nodes whose distance labels fall within a certain range. There is no such overhead in the proposed algorithm.
- *Dijkstra's algorithm with Double Buckets:* requires extra space for storing the two levels of buckets whereas no such requirement is there in the proposed algorithm.
- *Dijkstra's algorithm with overflow bag:* requires maintenance of the overflow bag. No such overhead is there in the proposed algorithm.

In order to test the algorithm, a database for Jaipur city with 500 nodes has been created. The schema of the database has fields source, destination, distance and congestion. The arc to node ratio typically varies between 2 and 3. Though, due to the limitation of the availability of the data, more than two routes have not been considered. A navigation system was designed that used the Multi-parameter Dijkstra's algorithm to calculate the shortest path.

The user interface provides following facilities to the user:

- ✓ Browse all routes
- ✓ Browse bus routes
- ✓ Display Traffic congestion among all the routes
- ✓ Display Traffic congestion across some specific routes
- ✓ Get the shortest path between a source-destination pair and also the bus routes that are available across the shortest path

Table1: Comparison of MPD with other modifications of Dijkstra's algorithm

S.No.	Criteria	Algorithm	Comparison with MPD
1.	Number of parameters	Dijkstra's Algorithm	Considers only one parameter as compared to MPD that uses multiple parameters
2.	Space and time	Dijkstra's Algorithm with Buckets (DKB)	1. More space required for nC+1 buckets 2.Extra time required for sorting buckets MPD does not have any such overhead
3.	Space	Dijkstra's Algorithm with Double Buckets(DKD)	Requires extra space for two levels of buckets. No such requirement in MPD
4.	Accuracy	Dijkstra's algorithm with Approximate Buckets (DKA)	Makes approximation about the distance labels stored in a bucket that are within a certain range and this can lead to approximate results about the shortest path.
5.	Space	Dijkstra's Algorithm with Overflow bag (DKM)	Requires maintenance of the overflow bag. No such overhead is there in the proposed algorithm.

The dependence of various parameters was studied and it was observed that the distance travelled is proportional to the number of nodes traversed as shown in Figure 2.

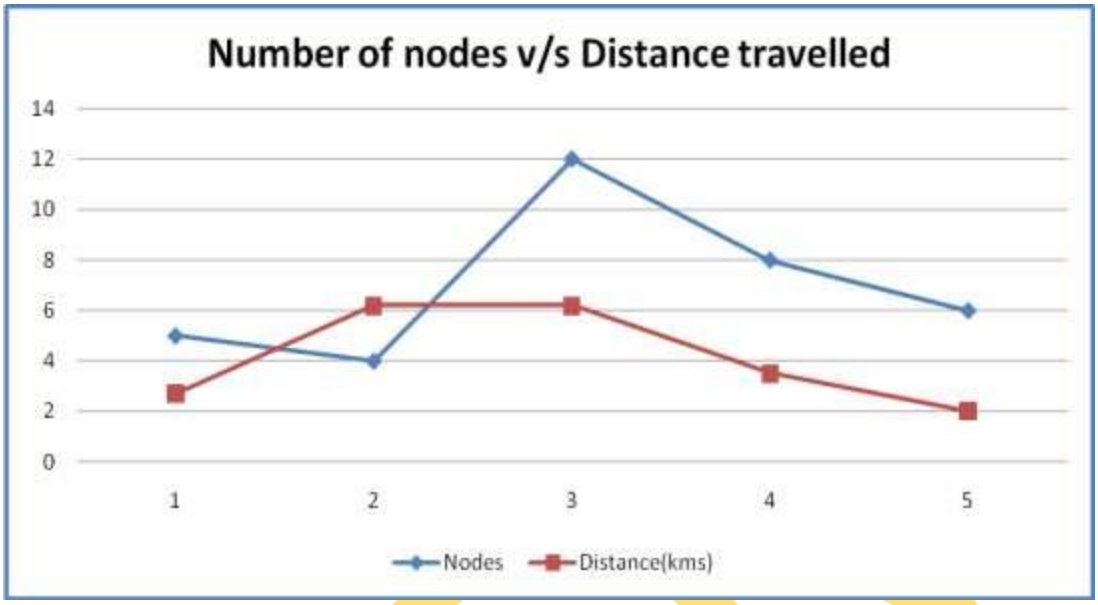

Figure 2: Number of nodes v/s distance travelled

Again it was observed that the traffic congestion across various routes was independent of the number of nodes as some nodes were more congested than others. The results are indicated in Figure 3.

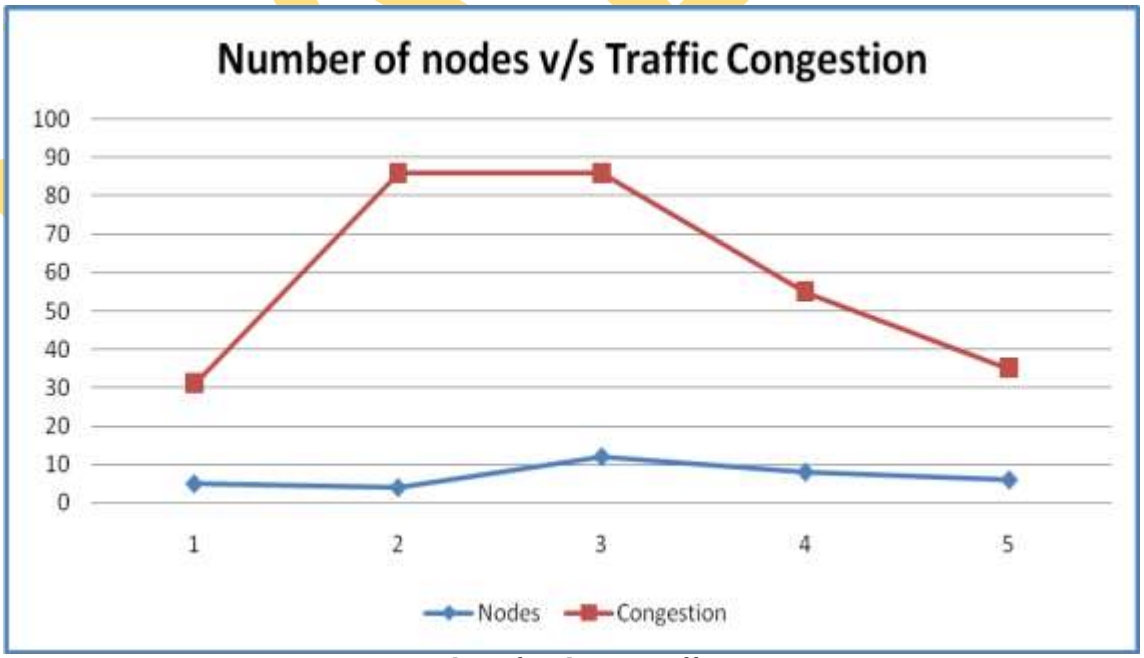

Figure 3: Number of nodes v/s traffic congestion

V. CONCLUSIONS AND FUTURE WORK

With the advancement of GIS technology and the availability of high quality road network data, it is possible to conduct transportation analysis concerning large geographic regions within a GIS environment. Sometimes, this type of analysis has to be completed in real time. As a consequence, these analysis tasks demand high performance shortest path algorithms that run fastest on real road networks.

The navigation system provides a user-friendly interface. It not only allows the user to browse most of the routes in the city, but also provides details about the distances and congestion of the routes. As compared to other contemporary algorithms, it provides multiple parameters like distance, time and congestion, out of which the user can choose one. The user can also view the best available route between a source-destination pair. During the design process some assumptions have been made such as the congestion factor may vary in real time scenario and may be affected by traffic jams etc. Also while calculating the time it is assumed that the user would move towards the desired destination, without any stoppages in between. In the bus routes, it is assumed that buses are always available at the node when the passenger arrives, the arrival time of the buses has not been taken into consideration.

REFERENCES

[1] Thomas H. Cormen, Charles E.Lieserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms", Prentice Hall of India, 2009.

[2] Anany Levitin, "Introduction to the design & analysis of algorithms", Pearson Education, Second Edition, 2009.

[3] F.Benjamin Zhan,"Three Fastest shortest path algorithms on Real-road networks", Journal of Geographic information and decision analysis, 2010, Vol. 1, No.1, pp.69-82.

[4] Shortest path algorithms, Wikipedia, the free encyclopedia, http://www.en.wikipedia.com.

[5] The Dublin Bus Navigator, http://www.iol.ie/~aidanh/dubbus/onebus/a.htm#1

[6] Mumbai Navigator, http://www.cse.iitb.ac.in/navigator1/index.html

[7] Ioannis Delikostidis & Corné P.J.M. van Elzakker,"User-Centered mobile navigation system interface development for improved personal Geo-identification and navigation".

[8] David Betaille," Creating enhanced Lane level vehicle navigation", IEEE Transactions on Intelligent transportation systems 2010, Vol. 11, Issue 3.

[9] Maps at www.mapsofindia.com Google maps www.maps.google.co.in

[10] Java Server Programming, Black Book, Kogent Solutions Inc., 2009.

[11] The Complete Reference, Java, Seventh Edition, Herbert Schildt, Tata McGrawHill Publishing Company, 2009.

[12] Giorgio Gallo, Stefano Pallottino , "Shortest path algorithms", Annals of Operations

research, Springer, Volume 13, Number1, pp. 1-79, 2009.

[13] Stuart E. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms", Operations Research, 2009, Vol.17, No. 3 pp. 395-412.

[14] F.Benjamin Zhan, Charles E. Noon, "Shortest path algorithms: an evaluation using real road networks", Transportation Science, Vol.32, No. 1, February 2008.

[15] Narsing Deo, Chi-Yin Pang, "Shortest path algorithms: Taxonomy and Annotation, Networks, Wiley, Vol.14, Issue 2, pp.275-323, 2006.

[16] Steffano Pallotino, "Shortest path algorithms in transportation models: classical and innovative aspects", Technical Report: TR 97-06.