International Journal of Science and Engineering Research (IJ0SER),

Vol 7 Issue 2February -2019
3221 5687, (P) 3221 568X

Efficient And Dynamic Route Planning Algorithm For Navigation And Path Finder Application

[1]K.Sruthi,[2]B.Vinothkumar
[1] Department of Electronics and communication Engineering ,SSIET College,
[2]Associate Professor and Head, Department of Electronics and communication Engineering ,
SSIET College,
Coimbatore, India.

Abstract-Aim of this paper is, finding the best possible route between the source and the destination in a road network taking into consideration the optimal path that is found by using the Shortest Path Algorithm, Multipath Algorithm and the Spatial Distribution features. These systems are capable of performing some of the tasks traditionally performed by user, such as determining the best route to the destination from the source. This process of finding shortest path from one point to another is called Routing. In this paper, a new shortest path algorithm is proposed. The suggested algorithm is a modified version of Dijkstra's Shortest Path Computation algorithm, with Multipath & Spatial Distribution Concept. With the help of spatial distribution the searching time can be reduced by limiting the search space.These methods can improve run time and memory usage because the nodes and edges they visit are limited. the set of considered intersections, multipath routing finds the closest intersection from source to destination.

Keywords: Directed Network, Node, Activity, Dijkstra's Algorithm,Permissible route,Multipath Routing, Path Finding

1.0 INTRODUCTION

A practical path planning algorithm for the shortest path between two nodes should have the characteristic of rapidness and optimality; furthermore the rapidness is more important. The Dijkstra's algorithm can solve the shortest path problem, but algorithm's time complexity is O (N^2), N is the number of network nodes. So the Dijkstra algorithm is difficult to satisfy the rapidness of path planning when the number of nodes in the road network is large. Therefore, it is necessary to study the rapid path planning algorithm that includes the concept of Multi Path Algorithm. Considering the geographical characteristic of the road network, a rapid path planning for vehicle navigation is proposed.

The spatial distribution concept is used to reduce the searching space by reducing the number of nodes to be searched. This functionality is done by assigning topography with co-ordinate points covering the source and destination co-ordinates. It includes the features of minimizing the search space. The expected results are achieved in the displayed results. For example, in the shortest path finding, the source and destination is added to the program and the shortest path is displayed which is the expected result. In the alternate path finding, the congestion nodes should be eliminated in finding the alternate path and the alternate path eliminates the congestion nodes which are displayed as the expected result.

PROPOSED ALGORITHM

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex. It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined [4]. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm[9] can be used to find the shortest route between one city and all other cities. As a result, the shortest path is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First).

Let the starting of the node be called an initial node. Let a distance of a node Y be the distance from the initial node to it. Dijkstra's algorithm will assign some initial distance values and will try to improve them step-by-step.

1. Assign to every node a distance value. Set it to zero for our initial node and to infinity for all other nodes.
2. Mark all nodes as unvisited. Set initial node as current.
3. For current node, consider all its unvisited neighbors and calculate their distance (from the initial node). For example, if current node (A) has distance of 6, and an edge connecting it with another node (B) is 2, the distance to B through A will be 6+2=8. If this distance is less than the previously recorded distance (infinity in the beginning, zero for the initial node), overwrite the distance.
4. When we are done considering all neighbors of the current node, mark it as visited. A visited node will not be checked ever again; its distance recorded now is final and minimal.
5. Set the unvisited node with the smallest distance (from the initial node) as the next "current node" and continue from step 3.

In graph theory, the shortest path problem is the problem of finding a path between two vertices (or nodes) such that the sum of the weights of its constituent edges is minimized. An example is finding the quickest way to get from one location to another on a road map; in this case, the vertices represent locations and the edges represent segments of road and are weighted by the time needed to travel that segment.

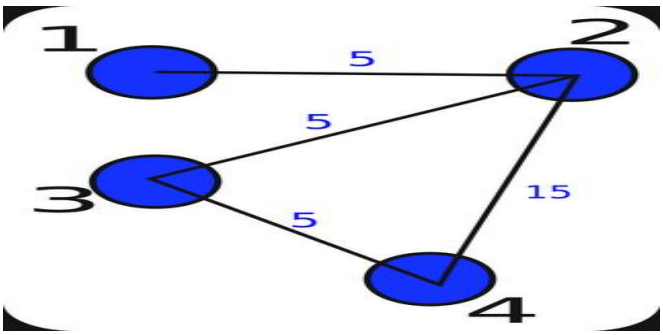
Formally, given a weighted graph (that is, a set V of vertices, a set E of edges, and a weight function $f: E \to \mathrm{R}$), and one element v of V, find a path P from v to a v' of V so that

$\sum_{p \in P} f(p)$ is minimal among all paths connecting v to v'.

The problem is also sometimes called the **single-pair shortest path problem**, to distinguish it from the following generalizations:

- The **single-source shortest path problem**, in which we have to find shortest paths from a source vertex v to all other vertices in the graph.
- The **single-destination shortest path problem**, in which we have to find shortest paths from all vertices in the graph to a single destination vertex v. This can be reduced to the single-source shortest path problem by reversing the edges in the graph.
- The **all-pairs shortest path problem**, in which we have to find shortest paths between every pair of vertices v, v' in the graph.

These generalizations have significantly more efficient algorithms than the simplistic approach of running a single-pair shortest path algorithm on all relevant pairs of vertices. This can be understand from the below figure



DIJKSTRA'S ALGORITHM

A priority queue stores the vertices outside the cloud

- Key: distance
- Element: vertex

Locator-based methods

- ***insert*(*k*,*e*)** returns a locator
- ***replaceKey*(*l*,*k*)** changes the key of an item we store two labels with each vertex:
- Distance (d (v) label)
- Locator in priority queue

Algorithm *DijkstraDistances*(*G, s*)
Q ← new heap-based priority queue
for all *v* ∈*G.vertices*()
If *v* = *s*
***setDistance*(*v,* 0)**
else
***setDistance*(*v,* ∞)**
***l* ← *Q.insert*(*getDistance*(*v*), *v*)**
***setLocator*(*v*,*l*)**
while ¬*Q.isEmpty*()
***u* ← *Q.removeMin*()**
for all *e* ∈*G.incidentEdges*(*u*)
{ relax edge ***e*** }
***z* ← *G.opposite*(*u*,*e*)**
***r* ← *getDistance*(*u*) + *weight*(*e*)**
if *r* <*getDistance*(*z*)
***setDistance*(*z*,*r*)**
***Q.replaceKey*(*getLocator*(*z*),*r*)**

ANALYSIS

Graph operations

- Method incident Edges is called once for each vertex Label operations
- Set/get the distance and locator labels of vertex ***z*** ***O***(deg(***z***)) times
- Setting/getting a label takes ***O*** (1) time Priority queue operations
- Each vertex is inserted once into and removed once from the priority queue, where each insertion or removal takes ***O*** (log ***n***) time
- The key of a vertex in the priority queue is modified at most deg(***w***) times, where each key change takes ***O***(log ***n***) time Dijkstra's algorithm runs in ***O***((***n*** + ***m***) log ***n***) time provided the graph is represented by the adjacency list structure
- Recall that Σ***v***deg(***v***) = 2***m*** The running time can also be expressed as ***O***(***m*** log ***n***) since thegraph is connectedAll-Pairs Shortest PathsFind the distancebetween every pair of vertices in a weighted directed graph G. We can make n calls to Dijkstra's algorithm (if no negative edges), which takes O(nmlog n) time. Likewise, n calls to Bellman-Ford would take O(n2m) time. We can achieve O(n3) time using dynamic programming (similar to the Floyd-Warshall algorithm).

METHODOLOGY

The Shortest Route Problem

This particular problem determines the route of minimum weight that connects two vertices namely a source and a destination in a weighted graph in a transportation network. Other situation can be represented by the same model like the Very Large Scale Integrated (VLSI) design, equipment replacement, and others. Different types of shortest path algorithm are used to determine the shortest path of a graph. The most frequently encountered path are the shortest path between two specified vertices, the shortest path between all pairs of vertices, and the shortest path from a specified vertex to all others.

The Dijkstra's algorithm is the most efficient algorithm used to find the shortest path between a known vertex to other vertices. Some improvements on Dijkstra's algorithm are done in terms of efficient implementation and cost matrix. In this paper, we propose to implement the Dijkstra's algorithm to determine the shortest route from the production plant of the company to any of the other location in the network.

Dijkstra's algorithm

The problem of finding the shortest path from a specified vertex *s* to mother *t* can be stated as follows: A simple weighted digraph G of n vertices is described by an n by n matrix D = [d_{ij}], where d_{ij} = length (or distance or weight) of the directed edge from vertex *i* to vertex *j*:

Dijkstra's algorithm labels the vertices of the given digraph, at each stage in the algorithm some vertices have permanent labels and others temporary labels. The algorithm begins by assigning a permanent label 0 to the starting vertex s, and temporary label infinity to the remaining *n-1* vertices. Then, another vertex sets a permanent labelin each iteration, according to the following rules:

a. Every vertex *j* that is not yet permanently labelled gets a new temporary label whose value is given by min [old label of *j*, (old label of $i + d_{ij}$)], where *i* is the latest vertex permanently labelled, in the previous iteration, and $d_{ij}$is the direct distance between vertices *i* and *j*. if *i* and *j* are not joined by an edge, the d_{ij} = infinity.

b. The smallest value of all the temporary labels is found, and this becomes the permanent label of the corresponding vertex. In a case of more than one shortest path, select any one of the candidates for permanent labelling. Steps **a** and **b** are repeated alternately until the destination vertex t gets a permanent label. The first vertex to be permanently labelled is at a distance of zero from *s*. The second vertex to get a permanent label (out of the remaining *n-1* vertices) is the vertex closest to *s* from the remaining *n-2* vertices, the next one to be permanently labelled is the second closest vertex to *s*. And so on. The permanent label of each vertex is the shortest distance of that vertex from *s*.

Simply, the Dijkstra's Algorithm can be stated as: Let u_i be the shortest distance from source node 1 to node *i*, and definedd_{ij}(> 0) as the length of the arc (*i*, *j*). Then the algorithm defines the label for an immediatelysucceeding node *j* as

$$[u_j, i] = [u_i + d_{ij}, i],\ d_{ij} > 0$$

That is the label for the node is [0, --], indicating that the node has no predecessor.

Node labels in Dijkstra's algorithm are of two types: *temporary* and *permanent.* A temporary label is modified if a shorter route to a node can be found. If no better route can be found, the status of the temporary label is changed to permanent.

Step 0: Label the source node (node 1) with the permanent label [0, --], set i = 1.

Step i:

a) Compute the temporary labels [$u_i + d_{ij}$, *i*] for each node j that can be reached from node *i*, provided *j* is not permanently labelled. If node *j* is already labelled with [$u_j + k$] through another node *k* and if $u_i + d_{ij} < u_j$, replace [u_j, *k*] with [$u_i + d_{ij}$, *i*].

b) If all the nodes have permanent labels, stop. Otherwise, select the label [u_r, *s*] having the shortest distance (=u_r) among all the temporary labels (break tie arbitrarily). Set *i* = *r* and repeat step i.

2.0 RELATED WORK

Dijkstra (1959) proposed a graph search algorithm that can be used to solve the single-source shortest path problem for any graph that has a non-negative edge path cost.This graph search algorithm was later modified by Lee in 2006 and was applied to the vehicle guidance system. Thisvehicle guidance system is divided into two paths; namely, the shortest path and the fastest path algorithms(Chen et al., 2009).While the shortest path algorithm focuses on route length

parameter and calculates the shortest route between each OD pair, the fastest path algorithm focuses on the path with minimum travel time. The future travel time can be predicted based on prediction models using historical data for link travel time information which can be daily, weekly or even a session.

Meghanathan (2012) reviewed Dijkstra's algorithm and Bellman-Ford algorithm for finding the shortest path ina graph. He concluded that the time complexity of Dijkstra's algorithm is O (|*E*|**log* |*V*|) while the time complexity of the Bellman-Ford algorithm is O (|*V*||*E*|).

Lili Cao et al (2005)concluded that the search for the shortest path is an essential primitive for a variety of graph-based applications, particularly those on online social networks. An example is the LinkedIn platform where users perform queries to find the shortest path "social links" connecting them to a particular user to facilitate introductions. This type of graph query is challenging for moderately sized graphs but becomes computationally intractable for graphs underlying today's social networks, most of which contain millions of nodes and billions of edges. They propose *Atlas*, a novel approach to scalable approximate shortest paths between graph nodes using a collection of spanning trees. Spanning trees are easy to generate, compact relative to original graphs, and can be distributed across machines to parallelize queries. They demonstrate its scalability and effectiveness using 6 large social graphs from Facebook, Orkut, and Renren, the largest of which includes 43 million nodes and 1 billion edges. They describe techniques to incrementally update Atlas as social graphs change over time. They capture graph dynamics using 35 daily snapshots of a Facebook network and show that Atlas can amortize the cost of tree updates over time. Finally, they apply Atlas to several graph applications and show that they produce results that closely approximate ideal results.

Wadhwa *(*2000)stated that researchers have targeted a Network Design Problem (Cable and Trench Problem), which involves a trade-off between utilization costs and capital costs for network construction. A larger network, (the shortest path tree) may cost more to build but may reduce utilization costs by including more attractive origin-destination paths. Conversely, a smaller network, (minimum spanning tree) may increase the utilization costs. A heuristic has been provided which gives us optimal or near optimal solutions. This heuristic is an adaptation of the Savings algorithm given by Clarke and Wright in 1964, for solving a vehicle routing problem. The heuristic provides us good solutions which can be used as upper bounds for branch and bound methods, giving us the optimal solutions in lesser times than that given by branch and bound without the upper bounds.

Pallottino and Scutella (1997)reported on Shortest Path Algorithms in Transportation models: classical and innovative aspects. They reviewed theshortest path algorithms in transportationsin two parts. The first part includes classical primal and dual algorithms which are the most interesting in transportation, either as a result of theoretical considerations or as a result of their efficiencies, and in view of their practical use in transportation models. They discussed the Promising re-optimization approaches involved. The second part includes dynamic shortest path problems that arise frequently in the transportation field. They analyzed the main features of the problems present under suitable conditions on travel time and cost functions, a general "chronological" algorithmic paradigm,called*Chrono-SPT*.

3.0 OUR SYSTEM MODEL

1) Dijkstra's Algorithm

The tracing and finding of the shortest path between the Source and Destination is implemented by using the Dijkstra's Shortest Path Computation Algorithm. Congestion takes place when there occurs a traffic block or a road block. The specialty of the algorithm is that it retraces to the node that is safely travelled without congestion and from that place it will find the optimal route to the destination. This reduces the time and increases the efficiency of the optimal path when compared to that of the classical Dijkstra's algorithm.

2) Multi Path Routing

The Multi Path Algorithm works with the following concept .The order is conceptually simple at each iteration; create a set of intersections consisting of every unmarked intersection that is directly connected to marked intersections. From that set of considered intersections, find the closest intersection to the destination and highlight it and mark that node to that intersection, draw an arrow with the direction, then repeat.

3) Spatial Distribution

The Third module includes the Spatial Distribution concept. This concept is used to reduce the searching space by reducing the number of nodes to be searched. This functionality is done by assigning topography with co-ordinate points

covering the source and destination co-ordinates. It includes the features of minimizing the search space. Instead of checking the entire search space for finding the shortest path between the source and the destination, a minimum area can be restricted with the help of fixing co-ordinate points from the source to destination.

4.0 CONCLUSION AND FUTURE SCOPE

In this paper, a practical method for Efficient Route Planning Algorithm for Vehicle Navigation System is described that it increases the efficiency of the paper by using the Multipath algorithm and spatial distribution concept. This method is effective in solving the existing problem. It increases the efficiency of the algorithm by using the new modified version of Dijkstra's Algorithm. This method is superior to the Dijkstra Algorithm as a method for practical vehicle navigation devices.

Obviously by comparison, the proposed effective route planning algorithm has less search nodes and shorter search time than the existing Dijkstra's algorithm. This work can be effectively used in other optimization problems. Compared with Dijkstra algorithm, the proposed algorithm reduces the seeking space and raises the seeking speed greatly.

REFERENCES

[1]. Ahmat, K. A. (n.d.). *Graph Theory and Optimization Problems for Very Large Networks.* New York: City University of New York/Information Technology.

[2]. Brendan, H. a. (2005). Generalizing Dijkstra's Algorithm and Gaussian Elimination for solving MDPs. *International Conference onAutomated Planning and Scheduling/Artificial Intelligence Planning System*, (pp. 151 - 160).

[3]. Chen, K. M. (2009). A real-time wireless route guidance system for urban traffic management and its performance evaluation. *Papers of the 70th Vehicular Technology Conference Anchorage VTC*, (pp. 1 -5).

[4]. Ebrahimnejad, S. A. (2011). Find the Shortest Path in Dynamic Network using Labelling Algorithm. *Journal of Business and SocialScience*.

[5]. Goldberg, A. V. (n.d.). *Point - to - Point Shortest Path Algorithms with Pre-processing.* Silicon Valley:MicrosoftResearch.

[6]. Goyal, S. (n.d.). *A Survey on Travelling Salesman Problem.* North Dakota: Department of Computer, University of North Dakota.

[7]. Lee, D. C. (2006). Proof of a modified Dijkstra's algorithm for computing shortest bundle delay in networks with deterministically time-varying links. *IEEE Communications Letters*, 734 -736.

[8]. Lieberman, F. S. (2001). *Introduction to Operations research seventh edition.* Mc Graw Hill.

[9]. Lili Cao, X. Z. (n.d.). *Approximating Shortest Path in Social Graph.* U.C. Santa Barbara: Computer Science Department.

[10]. Meghanathan, D. N. (n.d.). *Review of Graph Theory Algorithms* . MS: Department of Computer Science , Jackson State University.

[11]. Nar, D. (1997). *Graph theory with applications to engineering and computer science.*Prenctice Hall.

[12]. Sadavare, A. (2012). *International Journal of Computer Science and Information Technologies*, 5296 - 5300.

[13]. Scutella, S. P. (1997). *Technical report on Shortest Path Algorithms in Transportation models, Classical and innovative aspects.*

[14]. Sommer, C. (2010). *Approximate Shortest Path and Distance Queries in Networks.* Tokyo: Department of Computer Science, Graduate School of Information Science and Technology.

[15]. T. Li, Q. a. (2008). An efficient Algorithm for the single source Shortest Path Problem in Graph Theory. *International Conferenceon Intelligent System and Knowledge Engineering*, (pp. 152 - 157).

[16]. Taha, H. (2011). *Operations research an introduction, ninth edition.* Pearson Publisher. Wadhwa, S. (n.d.). *Analysis of a networkdesign problem.* 2000: Lehigh University

[17] Bi Jun, Fu Meng-yin, Zhou Pei-de. A quick path planning algorithm for vehicle position and navigation. Beijing. Journal of Beijing Institute of Technology,Vol.22(2): 188-191.

[18] Chen.K and Miles J.C, ITS Handbook 2000 – Recommendations from the World Road Association (PIARC), Artech House, 1999.

[19] Dijkstra E.W, A note on two problems in connection with graphs, NumerischeMathematik1(1959), 269–271.

[20] Fuji.M,J. Liand P.Zhau. "Design and implementation ofbidirectionalDijkstra algorithm," Joumalof Beijing InStiNte of Technology, vol. 12,no. 4, pp. 366-370, Dec. 2003.