Finding Shortest Path By Using A* and Dijkstra Algorithm

Aye Thandar Oo, Dr. Mie Mie Khin
Computer University (Monywa)
ayethandaroo87@gmail.com,

Abstract

This paper is intended to present optimal route finding system for road network application. It is used to operate in complicated mapping situations and new unknown environments. This technique can be implemented to find the optimal path from source to destination in region of map. This system use A and Dijkstra algorithms to draw plan and develop the Geographic Information System(GIS) assisted optimal road network route finding and give shortest path. The optimal (or) shortest path based on distance or time or cost. And then evaluates the performance of these two algorithms.*

1. Introduction

The application of evolutionary computation techniques for the solution of optimization problems is now the major area of research. Optimization problems reveal the fact that the formulation of engineering design problems involves linear terms for constraints and objective function but certain other problems involve nonlinear terms for them. In some problem, the terms are not explicit functions of the design variables. Some algorithms perform better on one problem, but may perform poorly on other problems. That is why the optimization literature contains a large number of algorithms, each suitable to solve a particular type of problem.

Algorithms are very useful in real life applications. The real-world performance of any software system depends on only two things: (1) the algorithms chosen and (2) the suitability and efficiency of the various layers of implementation. The choice of a suitable algorithm for an optimization problem is, to a large extent, dependent on the user's experience in solving similar problems. An important part of computing is the ability to select algorithms appropriate to particular purposes and to apply them, recognizing the possibility that no suitable algorithm may exist. This facility relies on understanding the range of algorithms that address an important set of well-defined problems, recognizing their strengths and weaknesses, and their suitability in particular contexts. Efficiency is a pervasive theme throughout this area.

So, this system intended to develop the GIS assisted optimal road network route finding approach based on A* and Dijkstra algorithms. And then evaluates the performance of these two algorithms.

2. Geographic Information System(GIS)

Geographic Information System (GIS) is a computer system capable of capturing, analyzing, and displaying geographically referenced information; that is, data identified according to location. A GIS can also convert existing digital information, which may not yet be in map from; into forms it can recognize use. GIS technology integrates common database operations such as query and statistical analysis with the unique visualization and geographic analysis benefits offered by maps. This geographic information is information which can be related to a location (defined in terms of a point, area or blocks) on the Map. The positional data can be a specific set of spatial coordinates, or can cover less precise locations or areas.

These abilities distinguish GIS from other information systems and make it valuable to a wide range of public and private enterprises for explaining events, predicting outcomes, and planning strategies[1]. This system implements the GIS based information system of Sagaing Division. This system would contain functionality that allows users to point-and-click their location on the map.

3. Searching Strategies

Searches are divided into two main categories: uninformed searches (brute-force, blind), and informed (heuristic, directed) searches. Uninformed searches are done when there is no information about a preferred search path. Informed searches have some information to help pick search paths;

usually a rule of thumb is used to reduce the search area [6].

3.1 Uninformed Search

These search strategies that come under the heading of uninformed search or also called blind search. The term means that they have no additional information about states beyond that provided in the problem definition. This strategies can do is generate successors and distinguish a goal state from a non-goal state. All search strategies are distinguished by the order in which nodes are expanded [5].

3.2 Informed (Heuristic) Search

Heuristic search is a useful tool in solving planning problems. Reducing a planning problem into a graph search problem, the basic aim of heuristic search is to simplify the process of solving the problem by subtracting some unnecessary yet time-consuming computations, usually in the form of a state expansion. Informed search is the one that uses problem-specific knowledge beyond the problem definition. It can finds solution more efficiently than the uninformed search [5].

4. Shortest Path

It is based on search method. Search methods aren't the perfect solution for every problem, but with creative applications it can solve many. If search is an appropriate solution, then choose the one which is guaranteed to find the solution. Furthermore, pick the most efficient one. There are various search methods. These methods can use to solve path-finding problem. Path-finding problem need us to search a path or a way from the start point to the goal point through some constraint may exist [2]. This system provides users the option of selecting their origin or destination on the map of Sagaing Division. The routing algorithms finds the optimum path and output is presented to the user both in text and mark on the map.

4.1. A* Algorithm For The Shortest Path Problem

A* search is one kind of heuristically informed search strategy. A* search maintains the set open of so-called open nodes that have been generated but not yet expanded. This method always selects a node from open with minimum estimated cost, one of those it considers “best”. This node is expanded and cost of some node n with an evaluation function of the form f(n)=g(n)+h(n). A* is guaranteed to return an optimal (minimum-cost) solution (it is also said to be admissible) [4].

The most widely-known form of best-first search is called A* search. It evaluates nodes by combining g(n), the cost to reach the node , and h(n), estimated cost to the goal from n. f(n)=g(n)+h(n) .f(n), estimated total cost of path through n to the goal. To find the cheapest solution, a reasonable thing to try first is the node with the lowest value of g(n)+h(n). A* search is both complete and optimal. The optimality of A* is straight forward to analyze if it is used with TREE-SEARCH. In this case, A* is optimal if h(n) is an admissible heuristic_ that is, provided that h(n) never overestimates the cost to reach the goal. An admissible heuristic is the straight-line distance h_{SLD}. Straight-line distance is admissible because the shortest path between any two points is a straight-line. So, the straight-line cannot be an overestimate. Below is an outline of how A* uses these components to search for optimal solutions:

1. Create a search graph, **G**, consisting solely of the start node, n_0. Put n_0 on a list called OPEN.
2. Create a list called CLOSED that is initially empty.
3. If OPEN is empty, exit with failure.
4. Select the first node on OPEN, remove it from OPEN, and put it on CLOSED. Call this node ***n***.
5. If ***n*** is a goal node, exit successfully with the solution obtained by tracing a path along the pointers from n_o to n_0 in **G**. (The pointers define a search tree and are established in step 7.)
6. Expand node ***n***, generating the set, **M**, of its successors that are not already ancestors of ***n*** in **G**. Install these members of M as successors of ***n*** in **G**.
7. Establish a pointer to ***n*** from each of those members of **M** that were not already in **G** (i.e., not already on either OPEN or CLOSED). Add these members of **M** to OPEN. For each member, ***m***, of **M** that was already on OPEN or CLOSED, redirect its pointer to ***n*** if the best path to ***m*** found so far is through ***n***. For each member of **M** already on CLOSED, redirect the pointers of each of its descendants in **G** so that they point backward along the best paths found so far to these descendants.
8. Reorder the lost OPEN in order of increasing f(n) values. (Ties among minimal f(n) values are resolved in favor of the deepest node in the search tree.)

9. Go to step 3.

In step 7, Algorithm redirects pointers from a node if the search process discovers a path to that node having lower cost than the one indicated by the existing pointers. Redirecting pointers of descendants of nodes already on CLOSED saves subsequent search effort but at the possible expense of an exponential amount of computation. Hence, this part of step 7 is often not implemented. Some of these pointers will ultimately be redirected in any case as the search progresses [6].

4.1.1. Algorithm for A*

```
Begin
    A* search  (G , w , s )  {
          for(  each  u  Є  V  ){
          d[u]  =  ∞  ;
          pred[u]  =  nil ;
          }
          d[s]  =  0  ;
          f (V) =  0  ;
          Q  =  ( queue with all vertices )
          While  ( Non-Empty ( Q ) ) {
          u  =  Extract-Min ( Q  )
          for ( each v  Є  Adj[u] )  {
             if  ( d[u]  +  w ( u ,v )  <  d[v] ) {
               d[v] = d[u]  +  w ( u ,v )  ;
             Decrease-Key( Q ,v , d[v] ) ;
              pred[v]  =  u  ;
             f (v) = d [v] + h (v,D) } }
           } }
End;
```

4.2. Dijkstra Algorithm For The Shortest Path Problem

The first algorithm for solving shortest path length problems was discovered by a Dutch computer scientist named Dijkstra in 1959. The basic premise of this problem is to find the length of the shortest path between the starting vertex and a first vertex; then the length of the shortest path between the starting vertex and a second vertex; continuing until the length of the shortest path between the starting vertex and the ending vertex is found. The goal is to always be looking for the vertex closest to the starting vertex[3].

Dijkstra's Algorithm is one the most efficient algorithms for solving the shortest path problem. In a network, it is frequently desired to find the shortest path between two nodes. The weights attached to the edges can be used to represent quantities such as distances, costs or times. In general, if shortest path find the minimum distance from one given node of a network, called the source node or start node, to all the nodes of the network, Dijkstra's algorithm is one of the most efficient techniques to implement. In general, the distance along a path is the sum of the weights of that path. The minimum distance from node *a* to *b* is the minimum of the distance of any path from node *a* to *b*.

Dijkstra's algorithm is probably the best-known and thus most implemented shortest path algorithm. It is simple, easy to understand and implement, yet impressively efficient. By getting familiar with such a sharp tool, a developer can solve efficiently and elegantly problems that would be considered impossibly hard otherwise. Dijkstra's algorithm, when applied to a graph, quickly finds the shortest path from a chosen source to a given destination. In fact, the algorithm is so powerful that it finds all shortest paths from the source to all destinations. This is known as the single-source shortest paths problem. In the process of finding all shortest paths to all destinations, Dijkstra's algorithm will also compute, as a side-effect, a spanning tree for the graph. While an interesting result in itself, the spanning tree for a graph can be found using lighter (more efficient) methods than Dijkstra's [8].

Dijkstra algorithm use priority queue using a heap for shortest path and perform the operations Insert( ), Extract_Min( ), Decrease_Key( ) in the priority queue, each in O(log n) time. And then when processing a vertex u, the algorithm will examine all vertices v Є Adj[u]. If the length of the new path from s to v shorter than d[v], then update d[v] to the length of this new path. This work is called relaxation [7].

4.2.1. Algorithm for Dijkstra

```
Begin
Dijkstra  (G , w , s )  {
      for(  each  u  Є  V  ){
            d[u]  =  ∞  ;
            pred[u]  =  nil ;
            }
            d[s]  =  0  ;
            Q  =  ( queue with all vertices )
            While  ( Non-Empty ( Q ) ) {
            u  =  Extract-Min ( Q  )
            for ( each v  Є  Adj[u] )  {
            if  ( d[u]  +  w ( u ,v )  <  d[v] ) {
             d[v] = d[u]  +  w ( u ,v )  ;
            Decrease-Key( Q ,v , d[v] ) ;
            pred[v]  =  u  ;      } }
                              } }
End;
```

5. System Design

This system contains three main phases. They are modify map portions, view information and finding shortest path portion. In modify map portion, user can create/modify network map for the system and perform the shortest path algorithms. In finding shortest path portion, firstly user must choose the target and destination towns on the map and then user can search shortest path based on distance, time and cost by using A* or Dijkstra Algorithms. And then user can view the shortest path and other information. After calculating both A* and Dijkstra algorithm, the system will be display comparison chart for these two algorithms.

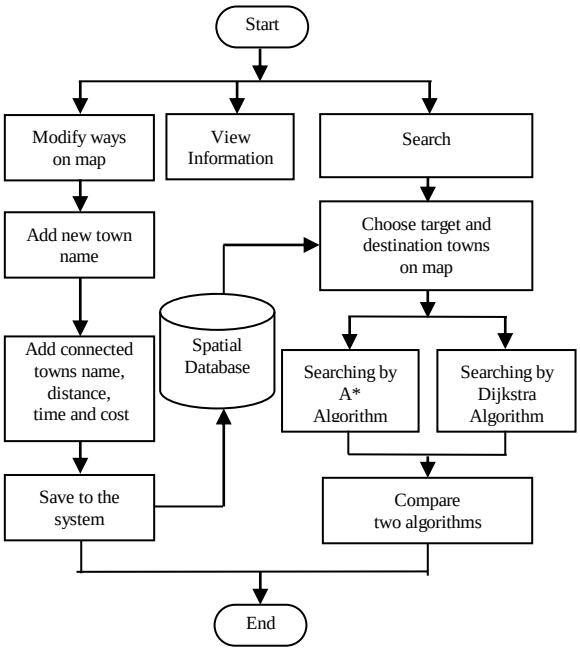

Figure 1. System Flow Diagram

6. Implementation and result

In this system, there are five main categories. They are view geographic information of Sagaing Division, searching for shortest path, modify paths in map, windows style function and system information menus. In "System" contains interesting places to view latitude and lontitude of interesting town and exit to exit the system. In "Search" contains A* and Dijkstra algorithm to search shortest path and comparison of processing time chart for these two algorithms. In "Modify" contains updating paths in map section such as adding new town name. In "Windows" contains style of opening function such as cascade, tile vertical and tile horizontal.

User can click interesting place on the map, latitude and longitude of the interesting place can be shown. The user can click source or start point and destination or target point on the map. And then user can search by using Dijkstra and A* algorithms by clicking the search button on the tool bar. These algorithms can perform to show the tree structure of search on possible track and shortest path on the chart respectively. If the user select "Information" button, the system will be displayed about the information of the distance mile, time and cost of source and destination towns and searching time of the algorithm as shown in Figure.2.

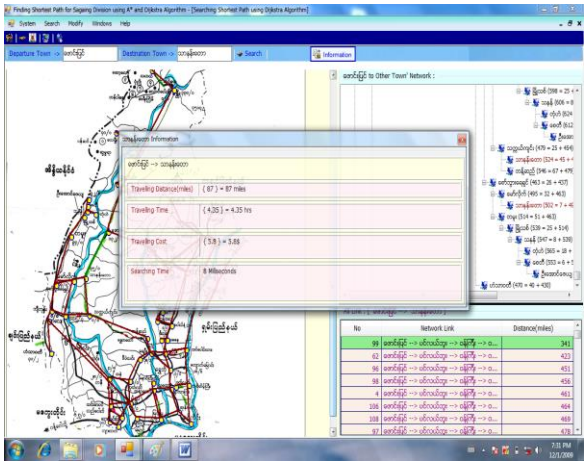

Figure 2. Possible ways and shortest path

After calculating Dijkstra algorithm, system will be displayed about the information of routing table (No, Network link and distance in miles). After claculating A* algorithm, user can also view the value of straight line distance to target waypoint and shortest path tree. There, the optimal path of the map can also be displayed for the method. The system also displayed the network tree view of source town to target town possible path and total distance. In this tree view, user can use expand facilities; expand current selected tree node and its all child nodes.User can view the comparison of processing time for these two algorithms (A* and Dijkstra algorithms) as show in Figure.3.

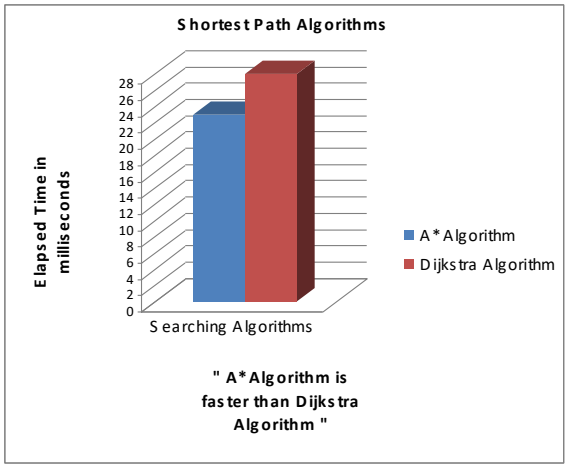

Figure 3. Comparison Graph for processing time.

The user can also set the location of the waypoint on the chart by using the one left click. If the user can type the waypoint's name in the waypoint name input box, these waypoint mark with waypoint name or mark position. If the user wants to set any other waypoint on the chart, the user can use the above procedure step by step.

The system allowed the user to connect the waypoints each other by the user's desire and calculate distance, time and cost. After creating the chart completely, the user can see the preview of the chart. Then, the chart can be confirmed from the confirm button, and saved the chart in current chart location by the use of save button.

7. Conclusion

The goal of this system was to explore different design issues associated with map-based itinerary-planning tools. In this way, a prototype was demonstrated to use and modify existing tools and software to create map-based finding shortest path system for Sagaing Division.

This system use A* and Dijkstra algorithms in the informed methods. A* algorithm expand node with an evaluation function of the form f(n)=g(n)+h(n), g(n) is the cost to reach the node and h(n), estimated cost to the goal from n and f(n), estimated total cost of path through n to the goal. This search algorithm is both complete and optimal. So A* algorithm is very suitable for this system. Dijkstra algorithm is frequently desired to find the shortest path between two nodes. It quickly finds the shortest path from a chosen source to a given destination. This is known as the single-source shortest paths problem. In the process of finding all shortest paths to all destinations, Dijkstra's algorithm will also compute, as a side-effect, a spanning tree for the graph. In fact, the algorithm is so powerful that it finds all shortest paths from the source to all destinations.

This system is used for two main portions such as create and modified map, points and distances, time and cost in map. And, the user can search shortest path by using these two algorithms on the chart and calculate distances between the points. And also evaluate the comparison of processing time.

8. References

[1] Vimalkumar A. Vaghani "Flood Impact Analysis using GIS" A case study for Lake Roxen and Lake Glan-Sweden"2005-06-08, ISRN: LIU-IDA-D20--05/016—SE

[2] Louis E.Frenzel, Jr. "Expert System and Applied Artificial Intelligence", California State University at Long Beach

[3] Maggie , "Shortest Path Algorithms", Johnson CS103B, Handout #17

[4] Dechter, R, Pearl, J: "Generalized Best_First Search Strategies and the Optimality of "A*", Journal of the ACM,

[5] Stuart Russell. Peter Norvig, "Artificial Intelligence a Modern Approach", Second Edition, Prentice Hall Series in Artificial Intelligence.

[6] Nils J.Nilsson, "Artificial Intelligence", Stanford University, Morgan Kaufmann Publishers, 1998.

[7] Emil Kelevedjiev, "Computational Approach for Assessment of Critical Infrastructure in Network Systems", Institute of Mathematics and Informatics Bulgarian Academy of Sciences.

[8] P. Biswas, P. K. Mishra and N. C. Mahanti , "Computational Efficiency of Optimized Shortest Path Algorithms", Department of Applied Mathematics, Birla Institute of Technology, Mesra (India)