Carleton University
School of Computer Science
COMP 4905
HONOURS PROJECT
Fall Term, 2005

Fast Shortest Path Algorithm for Road Network and Implementation

Author:	Liang Dai
Student #:	100302459
E-mail address:	liang_dai@yahoo.com
Supervisor:	Dr. Anil Maheshwari

Summary

In the road network application, the A* algorithm can achieve better running time than Dijkstra's. The restricted algorithm can find the optimal path within linear time but the restricted area has to be carefully selected. The selection actually depends on the graph itself. This algorithm can be used in a way that allowing search again by increasing the factor if the first search fails.

Acknowledgments

I would like to thank Professor Anil Maheshwari, who provides me the opportunity to study on this topic, directs and leads me on the right track. I also would like to thank the Greenley & Associates Co. for all helps I got during my co-op work term in the summer of 2005.

Table of Contents

1. Introduction

Shortest Path problems are inevitable in road network applications such as city emergency handling and drive guiding system, in where the optimal routings have to be found. As the traffic condition among a city changes from time to time and there are usually a huge amounts of requests occur at any moment, it needs to quickly find the solution. Therefore, the efficiency of the algorithm is very important [1, 3 and 5]. Some approaches take advantage of preprocessing that compute results before demanding. These results are saved in memory and could be used directly when a new request comes up. This can be inapplicable if the devices have limited memory and external storage. This project aims only at investigate the single source shortest path problems and intends to obtain some general conclusions by examining three approaches, Dijkstra's shortest path algorithm, Restricted search algorithm and A* algorithm.
To verify the three algorithms, a program was developed under Microsoft Visual C++ environment. The three algorithms was implemented and visually demonstrated. The road network example is a graph data file containing partial transportation data of the Ottawa city.

2. Three Algorithms for Finding Path

2.1 Dijkstra's Shortest Path Algorithm

The Dijkstra's shortest path algorithm is the most commonly used to solve the single source shortest path problem today. For a graph G(V, E), where V is the set of vertices and E is the set of edges, the running time for finding a path between two vertices varies when different data structure are used. This project uses binary heap to implement Dijkstra's algorithm although there are some data structures that may slightly improve the time complexity, such as Fibonacci heap that can purchase time complexity of O(V*log(V)) [6].

```
Dijkstra's shortest path algorithm
for each u ∈ G:
        d[u] = infinity;
        parent[u] = NIL;
End for
        d[s] = 0;  // s is the start point
        H = {s};  // the heap
 while NotEmpty(H) and targetNotFound:
        u = Extract_Min(H);
        label u as examined;
        for each v adjacent to u:
                 if d[v] > d[u] + w[u, v]:
                d[v] = d[u] + w[u, v];
                parent[v] = u;
                DecreaseKey[v, H];
```

Time Complexity:
The run time of first for loop is O(V). In each iteration of the while loop, Extract_Min of the heap is logV. The inner for loop iterates each adjacent node of the current node, the total run time is O(E). Therefore, the time complexity of this algorithm is O((V + E)*log(V) = O(E* log(V)). The correctness of this algorithm is well proved in [6].
As the number of nodes in a graph increases, the running time of the applied algorithm will become longer and longer. Usually, a road network of a city has more than 10^4 nodes. A fast shortest path algorithm becomes more desirable.

2.2 Restricted Search Algorithm

Basically, the structure of road networks is relative simple. They are large scaled, sparse and connected graph. When the Dijkstra algorithm is used to find the shortest path, it starts search from the start point and spreads as a circle until the radius arrives the destination. Most searches at the area opposite the direction of destination are useless. M. Fu et al. [4] described an optimal approach to find shortest path for Vehicle Navigation System by physically cutting off area within which the shortest path is not supposed to appear.

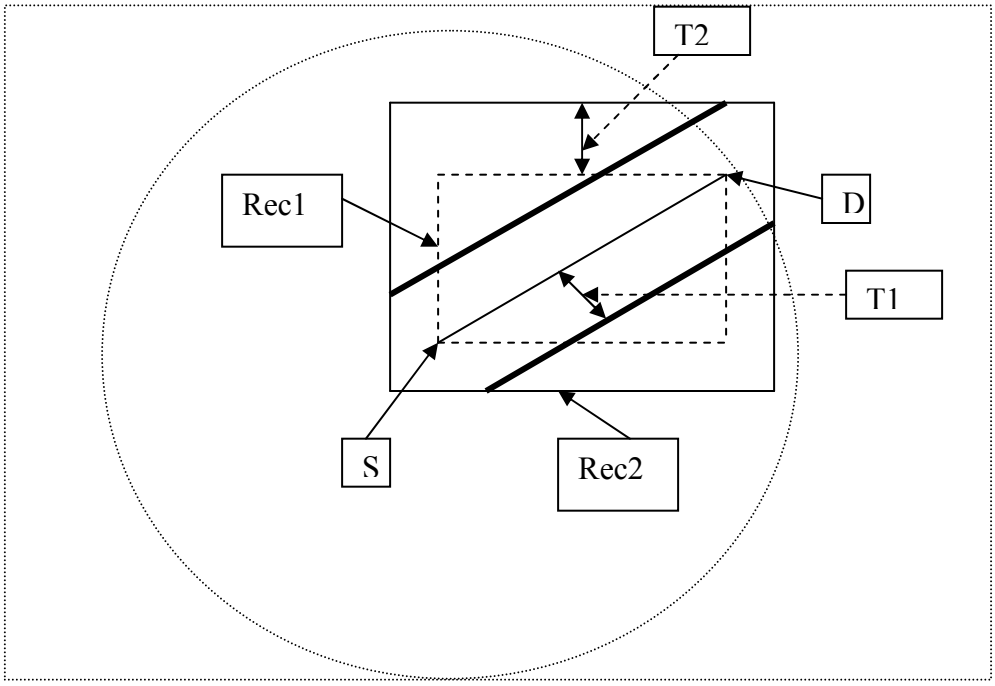

Figure 1, Search Area of Dijkstra and Restricted Search Algorithm
(S: Start point, D: Destination)

Instead of search the entire circle, the Restricted Search Method only search with the small area of the remaining part of rectangle Rec2 cutting off by the two bold straight lines. The rectangle Rec1 has the straight line of S and D as a diagonal and Rec2 is a rectangle extended from Rec1 by a threshold T2. The two straight bold lines parallel the

straight line of SD with a distance of T1. T1 and T2 are two variables that need to be decided to ensure that the optimal path is included within the restricted area. They usually range from 500m to 1500 m.

This project uses the following algorithm to achieve this goal.
Restricted Search Algorithm:

```
for each u ∈ G:
        d[u] = infinity;
        parent[u] = NIL;
End for
        d[s] = 0;
        H = {s};
while NotEmpty(H) and targetNotFound:
        u = Extract_Nin(H);
        label u as examined;
        for each v adjacent to u:
                if outOfRange(v), then continue;
                if d[v] > d[u] + w[u, v], then
                        d[v] = d[u] + w[u, v];
                        parent[v] = u;
                        DecreaseKey[v, H];

Procedure outOfRange(Constraint Area A, Vertex v):
        //A is a polygon given;
        //v is a Vertex being checked;
        Make a straight-line L from v to the right of v;
        Counter = 0;
        For each edge e of A
                if L intersects with e
                        increase Counter by one;
        if Counter is even
                return true;
        else
                return false;
```

The Dijkstra algorithm is used as the same way. However, instead of relaxing all adjacent nodes in each iteration, the algorithm filters out the nodes beyond the restricted area by checking if they are out of range. The checking procedure is implemented by counting the number of intersection of the horizontal line starting from the node to the right with the restricted area. The figure2 illustrates this method. If the number of intersection is odd, the node is within this area, otherwise it is not in the area.

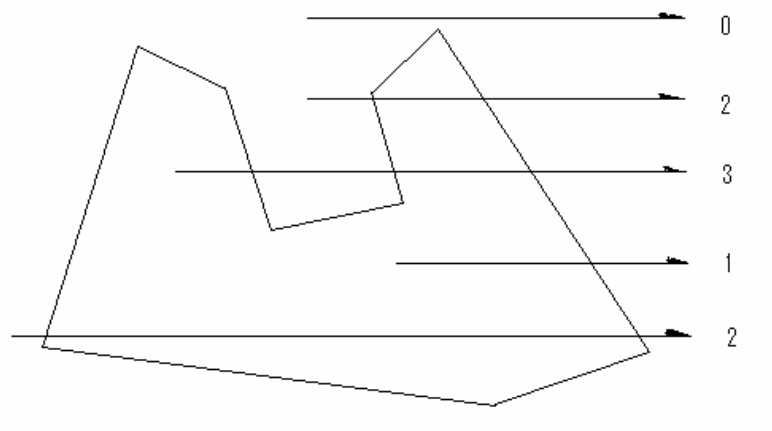

Figure 2, Check point in polygon

Time Complexity:

Suppose the nodes in the road network are distributed evenly.

Let density be C,

V*: the number of vertices examined by using Dijkstra algorithm.

V: the number of vertices examined by using this algorithm.

Soptimal: searched area by using this algorithm.

Sdij: searched area by using Dijkstra algorithm.

K1, K2: the factors of threshold.

Then, the ratio P can be used to describe the improvement of efficiency,

$P = V/V^*$

$= C^* Soptimal / C^* Sdij$

$<= 2 * T1 (R + T2 * \sqrt{2}) / \pi R^2$

$= 2 * K1*R(R + K2* R*\sqrt{2}) / \pi R^2$

$=2 * K1(1 + \sqrt{2} * K2) / \pi$

Since K1 and K2 are small number, the great improvement can be achieved. For instance, if $K2 = 0.4$, $P \approx K1$.

Correctness:

There are two problems exist in this approach.

(1) The fixed threshold may not get the solution properly. As the distance from start point to the destination increases, the shortest path more likely spreads wider from the straight line of SD.

(2) The city traffic lines have different categories with different driving speed. The high way may be beyond the restricted area but in the shortest path.

To achieve a better solution, this project uses relative thresholds instead of the fixed ones, called factor K1, K2 (threshold / length of SD). To simplify the problem, they have same value in the implementation. The threshold proportionally increases with the distance from start point to the destination in a selected factor. The second problem could be solved by using logical position instead of physical location for each node. For example, upon a node being examined, its logical position will be the accumulation from the logical location of its parent by the cost of the edge between them. The cost of an edge is logical distance that relative to physical distance and road category (see section 3.1). The

logical location of start point is the same as its physical location. All nodes being examined use their logical position to decide if they are within the restricted area. This ensures that the nodes on different roads can be treated equally in searching. However, due to the time constraint, this project will not implement it in the program.
This algorithm actually uses the Dijkstra within the restricted area. It is obvious that a shortest path in this area will be found if the path exists. However, this shortest path may not be the shortest one in the whole area. Thus, it is optimal path with restricted area.

2.3 A* Search

The A* algorithm integrates a heuristic into a search procedure. Instead of choosing the next node with the least cost (as measured from the start node), the choice of node is based on the cost from the start node plus an estimate of proximity to the destination (a heuristic estimate). F. Engineer [2] described this approach to solve the problem of optimal path finding.
This project uses Euclidean distance as estimated distance to the destination. In the searching, the cost of a node V could be calculated as:

f(V) = distance from *S* to V + estimate of the distance to *D*.
= d(V) + h(V,D)
= d(V) + sqrt($(x(V) - x(D))^2 + (y(V) - y(D))^2$)

where x(V), y(D) and x(V), y(D) are the coordinates for node V and the destination node D.

The A* Search algorithm:

```
for each u ∈ G:
        d[u] = infinity;
        parent[u] = NIL;
End for
        d[s] = 0;
        f(V) = 0;
        H = {s};
 while NotEmpty(H) and targetNotFound:
        u = Extract_Min(H);
        label u as examined;
        for each v adjacent to u:
                if d[v] > d[u] + w[u, v] , then
                        d[v] = d[u] + w[u, v];
                        p[v] = u;
                        f(v) = d[v] + h(v, D);
                        DecreaseKey[v, H];
```

Time Complexity:
This algorithm does not improve worst case time complexity, but it improves average time complexity. The shortest path search starts from start point and expands node that goes towards the destination. Therefore, the run time is much shorter than the Dijkstra's algorithm.

Correctness:
The algorithm uses the same approach as Dijkstra's except that it uses accumulated cost of edge plus the Euclidean distance from current node to the destination. This value is used to decide the position of a node in the min heap. The one with smallest value will be selected and removed from the heap. In the implementation, this value only affects the searching order. It doesn't modify the edge weights and accumulated distance. The accumulated distance is updated as the same way as Dijkstra when a node relax. Therefore, this algorithm is same as Dijkstra and it is correct.

3. Implementation

This program is developed under Microsoft Visual C++ environment. The three algorithms was implemented and visually demonstrated. The road network example is a graph data file containing partial transportation data of Ottawa city.

3.1 Brief Description

The map of Ottawa city is a directed graph with about 26000 vertices. There are four categories of road in the graph: minor road, regional road, major road and highway. Different type of road has different maximum driving speed. Therefore, physical distance of two points is not enough to describe the path. Instead, a logical distance is used to represent the real distance as well as shortest path. The logical distance is the physical distance times the type of the road segment. Assume:

Speed of highway / speed of minor = 3;
Speed of major / speed of minor = 1.5;
Speed of regional / speed of minor = 2.

Then, the type of minor is 6, regional 3, major 4 and highway 2. For example, if a segment of road is 10, the logical distance is 10 * 2 = 20 for highway, 10 * 6 = 60 for minor. Thus, if two paths have same length, the traveling times are the same. The shortest path is the path has shortest traveling time.

3.2 Main Feature

There are following main features in this program:
1, the window loads and displays the Ottawa city road network. This map is stored in the file "Ottawa_city.gph".
2, the user can drag and move the start point and destination on the window screen by using the mouse.
3, the window can display the shortest path as the user demands.
4, there are four categories of road in this graph: minor, regional, major and highway. They are represented in different colors.

3.3 Run program

1. Start up the program
2. From the menu bar, choose “File”, “Open”, “Open exist file”. From the pop up dialog, select the file “ottawa_city.gph” in the “data” directory. Then, push “ok” button.
3. After the graph is loaded, use the mouse dragging the blue square and two squares show up, which represent start point and destination point respectively.
4. The graph can be zoomed out, zoomed in, moved left, moved right, moved up and moved down by using “page down”, “page up”, “left”, “right”, “up” and “down” keys respectively.
5. Press “p” key to toggle showing path between the two points.
6. From the “method” menu, choose either “Dijkstra”, “astar”, or “restricted” to apply the three algorithms to show the current path. The “restricted” menu has five selections of the factors respecting the absolute distance from start to destination, 0.2, 0.4, 0.6, 0.8 and 1.0.
7. Drag the point again to show the path on current setting.
8. Exit the program by selecting “exit” under “File” menu or closing the window.

Figure 3, the snap shot of the screen

Figure 4, the view of zoom in

4. Experimentation

The first part compares the run time and accuracy of the three algorithms. The distance is the path length which is represented by logical distance as discussed in section 3.1. The accuracy of a algorithm for a particular distance is the path length found by this algorithm divided by the path length found by Dijkstra.

1). the running time and accuracy of the algorithms with respect to different distances

Distance	5009	7527	11262	13957	15164	19366	21498	25901	30657
Dijkstra	25/20	50/45	55/50	71/70	100/80	160/140	180/120	220/190	350/300
A*	24/19	40/38	45/40	50/45	70/60	110/100	130/100	140/120	250/212
Accuracy	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
Restricted	12/10	15/10	20/15	20/18	40/30	60/54	68/40	60/40	60/40
Accuracy	1.28	1.26	1.02	1.25	1.26	1.1	1.2	1.1	1.06

Distance	34250	40903.	44752	50227
Dijkstra	401/300	410/380	410/350	410/300
A*	310/250	360/350	410/350	410/325
Accuracy	1.0	1.0	1.0	1.0
Restricted	100/70	120/50	120/90	170/150
Accuracy	1.2	1.26	1.04	1.05

Note: the restricted algorithm uses factor as 0.4. Ignore the data if it can not find a path.

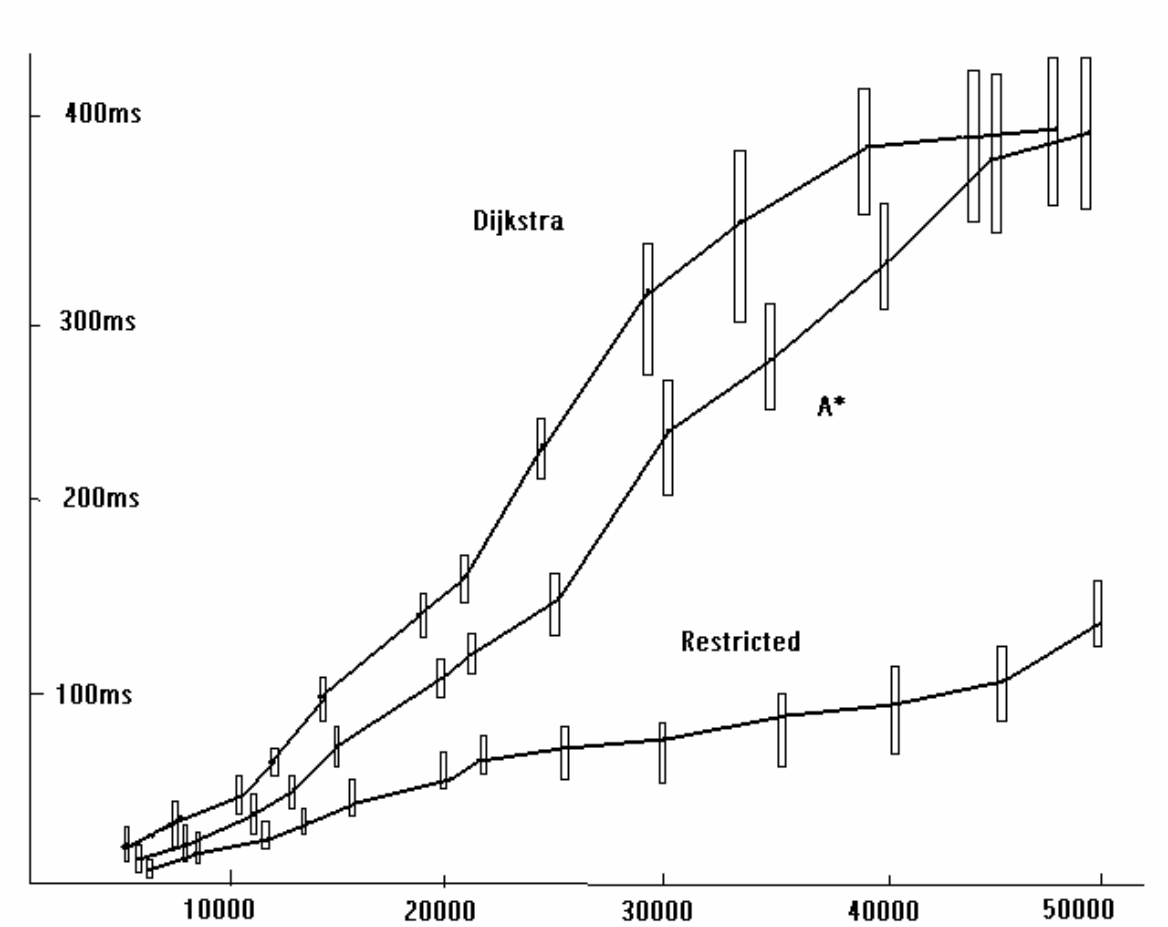

Figure 5, running time and accuracy of the three algorithms

The second part shows the effect of the value of factor on the search result. This is only for the restricted algorithm. Two measurements are used here: the search time of restricted algorithm divided by the search time of Dijkstra, and the path length of restricted algorithm divided by the path length of Dijkstra. The data was obtained by running the algorithm on different distance. So, it is a range from min to max. The “no” in the table means that there is no path can be found at most of time under a particular factor.

2). the effect of the factor from the restricted algorithm

factor	0.2	0.4	0.6	0.8	1.0
Time / Dijkstra time	no	0.23/0.64	0.33/0.69	0.6/1.0	0.8/1.15
Path / Dijkstra path	no	1.5/1.0	1.15/1.0	1.01/1.0	1.0

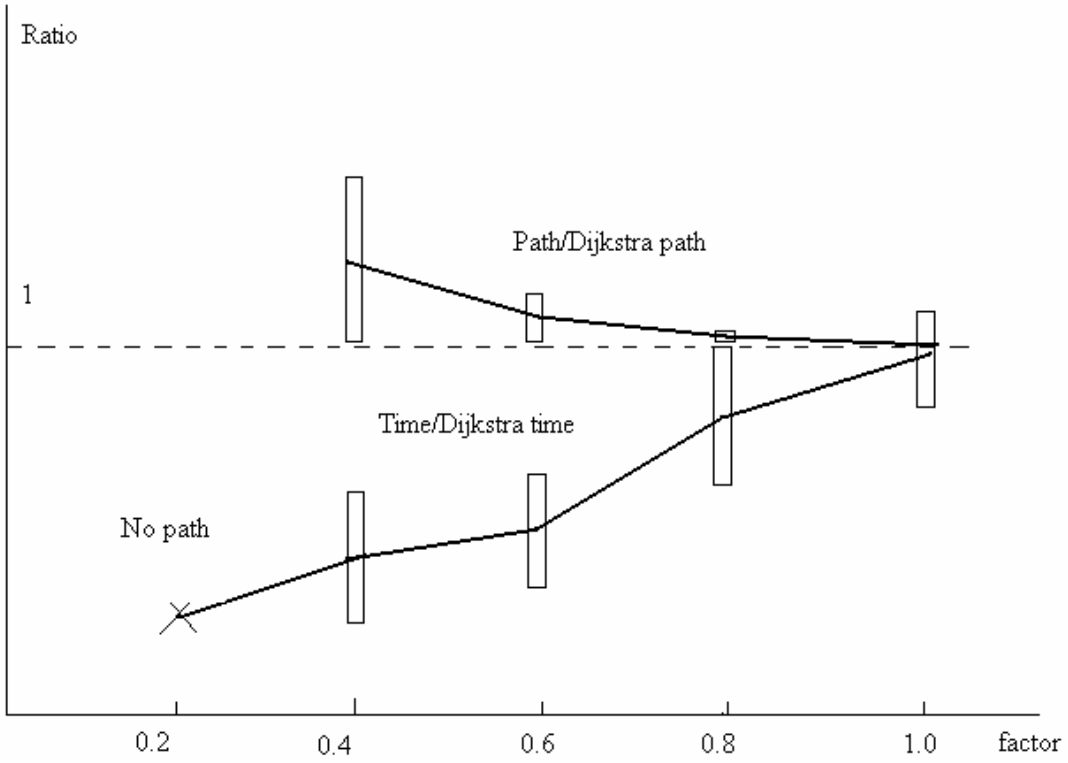

Figure 6, the effect of the factor

In the Figure 5, the running times of both Dijkstra and A* grow up fast until the distance reaching to 30000. This matches the theoretical time complexity. However, since the city map is limited, when the distance over 30000, the searching space dose not increase too much corresponding to the distance because there are no more unsearched points available beyond the searching area. So, the growing rate decreases.
The Figure 5 also shows that the A* algorithm is more efficient than Dijkstra algorithm. It roughly cuts running time to half because it restricts its search area towards destination by using Euclidean heuristic function. Again, as the distance increases, the cutting rate decreases due to the same reason discussed above.
The restricted algorithm shown in Figure 5 uses the factor of 0.4. This factor works in most of time. The result indicates that it running time is linear as long as it can find the path.

Actually, the factor selection is graph dependent. Different graph or different area of the graph have different smallest factor. In this graph, the factor of 0.2 doesn't works at almost all of the times. The Figure 6 gives the effect of the factor on the running time and accuracy. As long as the path can be found, the smaller factor gets faster running time. However, the tradeoff is the less accuracy. As an experiment result, when the factor is up to 1.0, the algorithm tends to find the shortest path but the running time is also the same as Dijkstra's.

5. Conclusion

The A* algorithm can achieve better running time by using Euclidean heuristic function although it theoretical time complexity is still the same as Dijkstra's. It can also guarantee to find the shortest path. The restricted algorithm can find the optimal path within linear time but the restricted area has to be carefully selected. The selection actually depends on the graph itself. The smaller selected area can get less search time but the tradeoff is that it may not find the shortest path or, it may not find any path. This algorithm can be used in a way that allowing search again by increasing the factor if the first search fails.

6. Reference

[1], Yu-Li Chouy H. Edwin Romeijnz Robert L. Smithx. Approximating Shortest Paths in Large-scale Networks with an Application to Intelligent. Transportation Systems. September 27, 1998
[2], Faramroze Engineer. Fast Shortest Path Algorithms for Large Road Networks. Department of Engineering Science. University of Auckland, New Zealand. http://www.esc.auckland.ac.nz/Organisations/ORSNZ/conf36/papers/Engineer.pdf
[3], Roozbeh Shad, Hamid Ebadi, Mohsen Ghods. Evaluation of Route Finding Methods in GIS Application. Dept of Geodesy and Geomatics Eng. K.N.Toosi University of Technology, IRAN.
http://www.gisdevelopment.net/technology/gis/ma03202.htm
[4], Fu, Mengyin, Li, Jie, Deng, Zhihong. A practical route planning algorithm for vehicle navigation system. Proceedings of the World Congress on Intelligent Control and Automation (WCICA), v 6, WCICA 2004 - Fifth World Congress on Intelligent Control and Automation, Conference Proceedings, 2004, p 5326-5329
[5], Thomas Willhalm. Speed Up Shortest-Path Computations. January 27, 2005. http://www.mpi-sb.mpg.de/~sanders/courses/algen04/willhalm.pdf
[6] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest and Charles E. Leiserson. Introduction to Algorithms, 2nd edition, McGraw-Hill Higher Education, 2001, ISBN: 0070131511