Fuzzy Optim Decis Making (2008) 7:257–267
DOI 10.1007/s10700-008-9035-0

Multiple UAVs path planning algorithms: a comparative study

B. Moses Sathyaraj · L. C. Jain · A. Finn · S. Drake

Published online: 26 June 2008

Abstract Unmanned aerial vehicles (UAVs) are used in team for detecting targets and keeping them in its sensor range. There are various algorithms available for searching and monitoring targets. The complexity of the search algorithm increases if the number of nodes is increased. This paper focuses on multi UAVs path planning and Path Finding algorithms. Number of Path Finding and Search algorithms was applied to various environments, and their performance compared. The number of searches and also the computation time increases as the number of nodes increases. The various algorithms studied are Dijkstra's algorithm, Bellman Ford's algorithm, Floyd-Warshall's algorithm and the AStar algorithm. These search algorithms were compared. The results show that the AStar algorithm performed better than the other search algorithms. These path finding algorithms were compared so that a path for communication can be established and monitored.

Keywords Pathfinding · AStar · Dijkstra · Distance vector algorithms

1 Introduction

Unmanned aerial vehicle (UAV) is an aircraft with no onboard pilots. The UAVs generally fly in teams for achieving specific goal. The need for communication is very vital for the UAV team tasks. The information about the target should reach the human supervisors even when there are obstacles. The communication between the UAVs

B. M. Sathyaraj · L. C. Jain (✉)
Knowledge Based Intelligent Engineering Systems Center, University of South Australia, Adelaide, Australia
e-mail: Lakhmi.jain@unisa.edu.au

A. Finn · S. Drake
Defence Science & Technology Organisation, P. O. Box 1500, Edinburgh, SA 5115, Australia

and the human supervisors may be lost due to: communication being obstructed by intervening hills, line-of-sight obstruction owing to range constraints, the UAV does not communicate, or an UAV or a group of UAVs being shot down by enemy action.

The UAV have limited sensor capabilities. Cooperative control therefore relies heavily on communication between appropriate neighbours. Routing protocols use metrics to evaluate what path will be the best for a packet to travel. A metric is a standard of measurement, such as path bandwidth, sensor capabilities, physical distance between neighbours, time taken to communicate with the neighbours used by the routing algorithms to determine the optimal path for a particular destination. To aid the process of path determination routing algorithms are used to initialize and maintain routing tables, containing route information. Routing involves two basic activities. These are the determination of optimal routing paths and transformation of information groups through the network. The advantages of coordinating and collaborating UAV teams are: to accomplish the missions in a shorter period, to simultaneously accomplish many goals, teams of small aircraft can be both cheaper and less detectable than a single large vehicle; damage to a single UAV does not necessarily cause the entire mission to fault. Optimality refers to the capability of the routing algorithm to select the best route. This depends on the metrics and metric weightings used to make the calculation. For example, one routing algorithm may use a number of hops and delays. It may for example weigh delay more heavily in the calculation. Routing protocols must strictly define their metric calculation algorithms. Efficiency is particularly important when the software implementing the routing algorithm is used with a computer with limited physical resources.

Figure 1 shows that there is no direct radio channel between nodes A and F. Nodes B, D, E or G must serve as an intermediate router for communication between A and F. A distinguishing feature of networks is that all nodes must be able to function as routers on demand. A → B → C → F; A → D → F; A → G → F; A → E → G → F; A → B → E → G → F are different paths between A and F. The best or the shortest path is usually chosen for the communication between A and F. The source is A, and the intended destination is F. On the process of finding the path from A to F if A senses its neighbour is B, and B senses its neighbour is E and E senses its neighbour to be A then there is a loop formed. The communication intended for F fails to reach F instead go around the loop indefinitely.

There are various loop free routing protocols which avoid this type of loop formation. The recent developments in intelligent transportation systems (ITS) (Chabini and

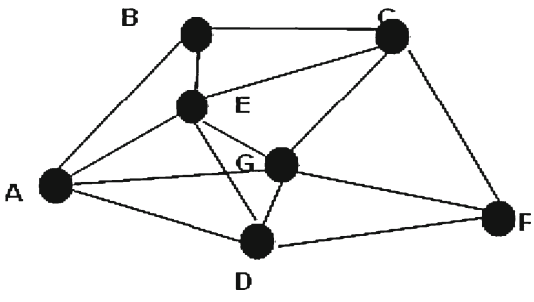

Fig. 1 Example network

Ganupati 2001), particularly in the field of in-vehicle route guidance system (RGS) (Li 2005) and real time automated vehicle dispatching system (AVDS) (Zhan and Noon 1998) where there is a definite need to find the shortest paths from an origin to a destination in a quick and accurate manner. The travel times are the basic input to the real-time routing and scheduling process and are dynamic in most urban traffic environments.

There is therefore requirement to use a minimum path algorithm repeatedly during the optimization procedure. Most of these heuristic search strategies originated in the artificial intelligence (AI) field (Hart et al. 1968; Nilsson 1971; Newell and Simon 1972; Pearl 1984), where the shortest path problem is often used as a testing procedure to demonstrate the effectiveness of these heuristics.

In this paper the shortest path algorithms are studied and compared. The comparison is done with respect to the number of searches required and the time taken for computation. This is done so that the path between the source and the destination is computed the shortest time so that the best path is selected. The information about the target has to reach the main station in time for any action on the target can be planned accordingly. The information of the target should reach in time and also be reliable for any actions to be taken. This is especially true when there are multimedia images are transmitted. The routes must be chosen has to be planned so that there is no loss in the information neither there is delay in the information reaching the control station. The presented target information enables the human supervisor to decide on further actions. This paper strives to achieve end-to-end information transfer even when there is communication barrier due to some of the above reasons. A scalable, reliable, efficient, low latency communication algorithm could be designed for a heterogeneous, high speed network with a shortest path algorithm with least number of searches and least computation time to reach the destination. Therefore these algorithms are compared so the appropriate path finding algorithm can be used for intelligent co-operation of UAVs to achieve required communication. Various path finding algorithms (Gallo and Pallottino 1984; Hung and Divoky 1988; Vuren and Jansen 1988; Cherkassky et al. 1996; Zhan and Noon 1998; Jacob et al. 1999; Fu 1996) are studied and compared. The main part of communication depends on the path or the route that the messages take to reach the destination. Once the shortest path is obtained then the communication can be achieved in short duration.

This paper is organized as follows. The next section deals with the background on Graphs, networks and path finding. Section 3 deals with the Path finding and search algorithms. The last section concludes the paper with the future work.

2 Background

2.1 Networks and graphs

A graph *G* is a set of vertex (nodes) v connected by edges (links) e. It can be represented as G = (v, e). Vertex (Node) is a node v which is a terminal point or an intersection point of a graph. Edge (Link) e is a link between two nodes. A network can be expressed as a graph with nodes and edges. Here A, B, C, D, E and F are the nodes of the graph.

The links between these nodes are given a value or weight based on: the physical distance between the nodes, the time for communication between the two nodes, the sensor capability of a node, the actual type of node, the availability of the nodes, the reliability of the node, the bandwidth constraints between the nodes, or a combination of some or all the above factors.

As weight values are given to each of the edges, it is called as weighted graph. The graph given is an undirected graph as there is no direction denoted in the graph. Therefore the cost for communication between A and D is the same as between D and A which equals 7. A path through a graph is a traversal of the consecutive vertices along a sequence of edges. The vertex at the end of one edge in the sequence must also be the vertex at the beginning of the next edge in the sequence. The vertices that begin and end the path are termed the initial vertex and terminal vertex, respectively. Each vertex within the path which has two neighbouring vertices must also be adjacent to the vertex. The length of the path is the sum of the weights of edges that are traversed along the path. The best path is the path which has the minimum sum of the weights from the source to the destination. The loop is a path in which the initial vertex of the path is also the terminal vertex of the path. By removing all the loops, the path can be considered an elementary path since there will no longer be any repeated vertices. An adjacency matrix for a graph G with "n" vertices is assumed to be ordered from v_1 to v_n. The n × n matrix A, is denoted by

A[i][j] or $A_{ij} = 1$ If there exists a path from v_i to v_j

A[i][j] or $A_{ij} = 0$ Otherwise

A graph can be represented as an adjacency matrix A in which each element (i, j) represents the edge between the elements i and j. $A_{ij} = 1$, if there is an edge between node i and j and otherwise by $A_{ij} = 0$. For the graph shown in Fig. 2 the adjacency matrix can be formed as given below (Fig. 3).

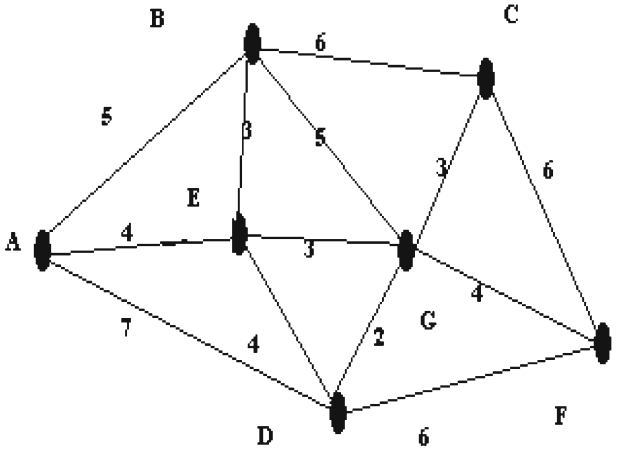

Fig. 2 A network represented as a graph

$$A[i][j] = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Fig. 3 Adjacency matrix

$$A[i][j] = \begin{pmatrix} \infty & 5 & \infty & 7 & 4 & \infty & \infty \\ 5 & \infty & 6 & \infty & 3 & \infty & 5 \\ \infty & 6 & \infty & \infty & \infty & 6 & 3 \\ 7 & \infty & \infty & \infty & 4 & 6 & 2 \\ 4 & 3 & \infty & 4 & \infty & \infty & 3 \\ \infty & \infty & 6 & 6 & \infty & \infty & 2 \\ \infty & 5 & 3 & 2 & 3 & 4 & \infty \end{pmatrix}$$

Fig. 4 Weighted cost matrix

A[i][j] = 1 in Adjacency matrix denotes that there is a path between the two nodes. The first row denotes the path existence between A and the rest of the nodes. The second row of the matrix denotes the existence of path between the node B and the rest of the nodes. Next the weighted cost matrix is obtained from the network graph (Fig. 4).

Here A[i][j] = actual cost denoted in the graph, for example A[1][2] = 5 denotes the cost of communication between A and B. If there is no direct communication link between the two nodes then the cost is shown as ∞. This is a matrix which represents a network. It is passed to the different path finding algorithms to test the capability of the algorithm to find the shortest path or to search the destination node.

2.2 Path finding and path planning

Path finding is the determination of all possible routes or paths from the source to the destination and path planning is the art of deciding which route to take and is based on the cost. Therefore path finding depends on factors including: how to get from source to destination, how to get around obstacles in the way, how to find the shortest possible path or, how to find the path quickly.

If the environment is flat with no obstacles, path finding is not a problem. The bot (a device or piece of software that can execute commands, reply to messages, or perform routine tasks, searches, either automatically or with minimal human intervention) can just move in a straight line to its destination. When there is an obstacle between the

bot and the destination then the bot has to choose another path. Of all the decisions involved in a computer game involving artificial intelligence, the most common is probably that of path finding when looking for a good route when moving an item from one place to another. The item can be a person, a vehicle, or a combat unit. The genre may be either an action game, a simulator, a role-playing game, or a strategy game. Any game in which the computer is responsible for moving items around has to solve the path finding problem.

Pathfinders enable you to look ahead and make plans instead of finding there is a problem at the last moment. Movement without using path finding is satisfactory in many situations, and can be extended to work in more situations, but path finding is a more general tool that can be used to solve a wider variety of problems. Most path finding algorithms can be developed using artificial intelligence.

3 Path finding and search algorithms

Given a connected graph G = (V, E) to find a shortest path from s to each vertex v in the graph, the following parameters are passed to the algorithm to find shortest path. G denotes the graph with weights in the form of matrix, s the source node from which the shortest path to all other nodes.

3.1 Distance vector algorithm

Distance vector routing (DVR) algorithm also known as Bellman Ford algorithm (Cavendish and Gerla 1998) is iterative as the process of exchanging information will continue until no more information is exchanged between the neighbourhoods, distributed as this algorithm enables each node receives some information from one or more of its directly attached neighbours. The Bellman-Ford distance-vector routing algorithm is used by routers on internetworks to exchange routing information about the current status of the network and how to route packets to their destinations. The algorithm basically merges routing information provided by different routers into lookup tables. It provides reasonable performance on small- to medium-sized networks, but on larger networks the algorithm is slow at calculating updates to the network topology. In some cases, looping occurs, in which a packet goes through the same node more than once.

Bellman-Ford algorithm solves the single-source shortest-path problem in the general case in which edges of a given digraph can have negative weight as long as G contains no negative cycles. This algorithm, like Dijkstra's algorithm, uses the notion of edge relaxation but does not use with greedy method. The algorithm progressively decreases an estimate cost on the weight of the shortest path from the source node to each node in the network until it achieves the actual shortest-path. The computational time of Bellman-Ford algorithm increases exponentially with the increase in the number of nodes. Routers that use this algorithm have to maintain the lookup tables, which gives the distances and shortest path to sending packets to each node in the network. The information in the lookup table is always updated by exchanging information with the neighboring nodes. The number of data in the table equals to that of all nodes in

networks. The columns of table represent the directly attached neighbours whereas the rows represent all destinations in the network. Each data contains the path for sending packets to each destination in the network and distance or time to transmit on that path or the cost. The measurements in this algorithm are the number of hops, latency, and the number of outgoing packets.

```
Bellman-Ford(G,w,s) G-Graph, w-weight, s-source
for each vertex ∈ Vertex[G]
  dist[v] ← ∞
   dist[s] ← 0
for i ← all vertices in G do
     for each edge (u,v) ∈ Edge[G] do
          Relax (u,v,w)
     for each edge (u,v) ∈ Edge[G] do
     if dist[v] > dist[u] + weight(u,v) then return false
                return true
```

3.2 Dijkstra's algorithm

Dijkstra's algorithm (Tan et al. 2006) creates labels associated with vertices. These labels represent the cost from the source vertex to that particular vertex. Within the graph, there exist two kinds of labels: temporary and permanent. The temporary labels are given to vertices that have not been reached. The value given to these temporary labels can vary. Permanent labels are given to vertices that have been reached and their cost to the source vertex is known. The value given to these labels is the cost of that vertex to the source vertex. For any given vertex, there must be a permanent label or a temporary label, but not both. The algorithm begins at a specific vertex and extends outward within the graph, until all vertices have been reached. More simply, Dijkstra's algorithm stores a summation of minimum cost edges whereas Prim's algorithm stores at most one minimum cost edge. Dijkstra's algorithm determines the costs between a given vertex and all other vertices in a graph. This may be useful to determine alternatives in decision making.

The algorithm begins by initializing any vertex in the graph (vertex A, for example) a permanent label with the value of 0, and all other vertices a temporary label with the value of 0. The algorithm then proceeds to select the least cost edge connecting a vertex with a permanent label (currently vertex A) to a vertex with a temporary label (vertex B, for example). Vertex B's label is then updated from a temporary to a permanent label. Vertex B's value is then determined by the addition of the cost of the edge with vertex A's value. This process is repeated until the labels of all vertices in the graph are permanent. Dijkstra's algorithm solves the single-source shortest path problem in weighted graphs. Dijkstra's algorithm starts from a source node and in next iteration add another vertex to the shortest-path spanning tree. This vertex is the point closest to the root which is still outside the tree. Watch as the tree grows by radiating

out from the root. As it is not a breadth-first search; we do not care about the number of edges on the tree path, only the sum of their weights.

```
Dijkstra(G, w, s) G-Graph, w-weight, s-source
    for each vertex in Vertex[G]
     dist[v] ← ∞
        dist[s] ← 0
 S ← empty set
 while Vertex[G] is not an empty set
    u ← Extract_Min(Vertex[G])
     S ← S union {u}
  for each edge (u,v) outgoing from u
      if dist[u] + weight(u,v) < dist[v]
        dist[v] ← dist[u] + weight(u,v)
```

3.3 Floyd-Warshall's algorithm

The Floyd-Warshall Algorithm (Gayraud and Authie 1992) is an application of dynamic Programming (DP) (Murray et al. 2002) is a technique that takes advantage of overlapping sub problems, optimal substructure, and trades space for time to improve the runtime complexity of algorithms. In bottom up dynamic programming, we start from smaller cases and store the calculated values in a table for future use, an effective strategy to most dependency-based problems. This avoids calculating the sub problem twice. Dynamic programming generates all enumerations, or rather; cases of the smaller breakdown problems, leading towards the larger cases, and eventually it will lead towards the final enumeration of size n this will give the shortest distances between any two nodes, from which shortest paths may be constructed (Fig. 5).

```
The Floyd-Warshall Algorithm
Floyd-Warshall (G,w,s) G-Graph, w-weight, s-source
    init path[ ][ ];
 for k = 1 to n
     for each (i, j) in (1 . . . n)
         path[i][j] = min(path[i][j],
 path[i][k]+path[k][j]);
 end
```

3.4 A star algorithm

The A* (called as 'A-star') (Cvetanovic and Nofsinger 1990) is like other graph-searching algorithms in that it can potentially search a huge area of the map. It's like Dijkstra's algorithm in that it can be used to find a shortest path. It's like breadth first search (BFS) in that it can use a heuristic to guide itself. In the simple case, it is as fast as BFS. Since in the worst case breadth-first search has to consider all paths to all possible nodes the time complexity of breadth-first search is increased depending on the number of nodes and edges in the graph (Fig. 6).

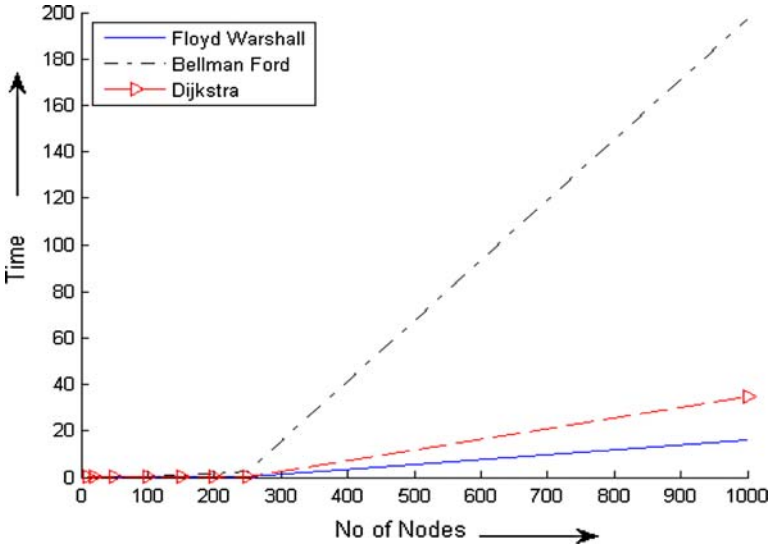

Fig. 5 Computation efficiency of search algorithms

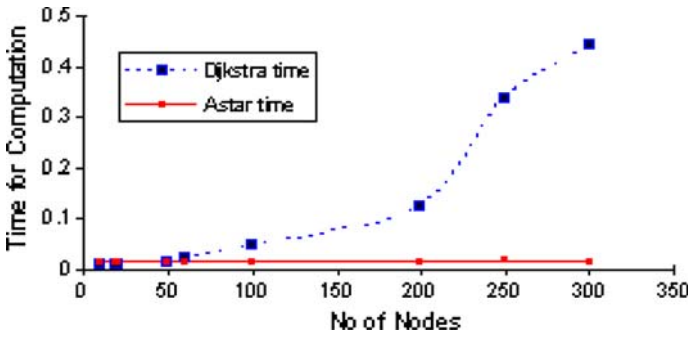

Fig. 6 Astar versus Dijkstra search

The best-established algorithm for the general searching of optimal paths is A*.

The AStar Algorithm

```
A*(G,s,w) G-Graph, w-weight, s-source
  var closed=the empty set
  var q=make_queue(path(start))
  while q is not empty
    var p=remove_first(q)
    var x=the last node of p
    if x in closed
       continue
     if x=goal
     return p
```

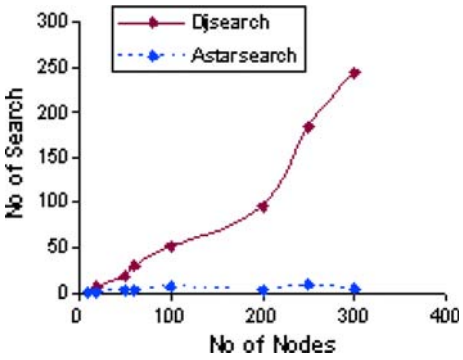

Fig. 7 Dijkstra versus astar

add x to closed
foreach y in successors(p)
enqueue(q, y)
return failure

The best-established algorithm for the general searching of optimal paths is A*. This heuristic search ranks each node by an estimate of the best route that goes through that node.

The typical formula is expressed as:

$$f(n) = g(n) + h(n) \tag{1}$$

where f(n) is the score assigned to node n, this is the estimate of the best solution that goes through n, g(n) is the actual cheapest cost of arriving at from the start node to n and h(n) is the heuristic estimate of the cost to the goal from n (Fig. 7).

4 Conclusion and future work

Different path finding algorithms were studied in order to formulate and study a search algorithm in intelligent environment. The path finding algorithm used for search depends on the number of nodes. The various algorithms were compared. For example, if the goal is to the south of the starting position, BFS will tend to focus on paths that lead southwards. It shows that BFS can find paths very quickly compared to Dijkstra's algorithm. The trouble is that BFS is greedy and tries to move towards the goal even if it's not the right path. Since it only considers the cost to get to the goal and ignores the cost of the path so far, it keeps going even if the path it's on has become really long. A* was developed to combine the virtues of the heuristic approaches like BFS and formal approaches like Dijkstra's algorithm. However, A* is built on top of the heuristic, and although the heuristic itself does not give a guarantee, A* can guarantee a shortest path. The experiments were done in MATLAB environment. These search algorithms were studied and compared in order to be implemented in various environments. Once the destination is found using the path finding algorithm the information flow path can be decided. This can be further improved to formulate a complete reliable and a low

latency communication algorithm (Sathyaraj and Chellappa Doss 2005; Royer and Toh 1999; Chabini and Ganupati 2001) which can be used in an intelligent environment.

References

Cavendish, D., & Gerla, M. (1998). Internet QoS routing using the Bellman-Ford algorithm. In *International Conference on High Performance Networking* (HPN'98).

Chabini, I., & Ganupati, S. (2001). *Design and implementation of parallel dynamic shortest path algorithms for intelligent transportation systems applications*, Transportation Research Record No. 1771. Washington, DC: National Academy Press.

Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest paths algorithms: Theory and experimental evaluation. *Mathematical Programming, 73*(2), 129–174.

Cvetanovic, Z., & Nofsinger, C. (1990). Parallel Astar search on message-passing architectures. *IEEE Computer Society, 1*, 82–90.

Finn, A., Kabacinski, K., & Drake, S. P. (2007). *Design challenges for an autonomous cooperative of UAV's*. Defence Science & Technology Organisation, Department of Defence, Australia, IDC 2007.

Fu, L. (1996). Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks. Unpublished Ph.D. Dissertation, University of Alberta, Edmonton, Alberta.

Gallo, G., & Pallottino, S. (1984). Shortest path methods. In M. Florian (Ed.), *Transportation planning models* (pp. 227–256). Amsterdam: Elsevier Science Publishers.

Gayraud, T., & Authie, G. (1992). A parallel algorithm for the all pairs shortest path problem. In *Proceedings of the International Conference on Parallel Computing*.

Hart, E. P., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transaction, System Science and Cybernetics, SSC-4*(2), 100–107

Hung, S. M., & Divoky, J. J. (1988). A computational study of efficient shortest path algorithms. *Computers and Operations Research, 15*(6), 567–576.

Jacob, R., Marathe, M., & Nagel, K. (1999). A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics, 4*, 1–19.

Li, W.-W. (2005). New trends in route guidance algorithm research of intelligent transportation system. *Journal of Zhejiang University, 39*, 819–824.

Murray, J. J., Cox, C. J., Lendaris, G. G., & Saeks, R. (2002). Adaptive dynamic programming. *IEEE Transactions on Systems, Man and Cybernetics, 32*, 140–153.

Newell, A., & Simon, H. A. (1972). *Human problem solving*. Englewood Cliffs, NJ: Prentice-Hall.

Nilsson, J. N. (1971). *Problem-solving methods in artificial intelligence*. NewYork: McGraw-Hill.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving*. Addison-Wesley Publishing Company.

Royer, E. M., & Toh, C.-K. (1999). A review of current routing protocols for ad-hoc mobile wireless networks. *IEEE Personal Communications Magazine*, April 1999, 46–55.

Sathyaraj, B., Chellappa Doss, R. (2005). Route maintenance using mobility prediction for mobile ad hoc networks. *IEEE international conference: Mobile adhoc and sensor systems conference*, MASS 2005.

Tan, G.-Z., He, H., & Sloman, A. (2006). Global optimal path planning for mobile robot based on improved Dijkstra algorithm and ant system algorithm. *Journal of Central South University of Technology, 13*, 80–86.

Vuren, V. T., & Jansen, G. R. M. (1988). Recent developments in path finding algorithms: A review. *Transportation Planning and Technology, 12*, 57–71.

Zhan, F. B., & Noon, C. E. (1998). Shortest path algorithms: An evaluation using real road networks. *Transportation Science, 32*(1), 65–73.