International Journal of Multidisciplinary and Current Research *ISSN: 2321-3124*
Research Article *Available at: http://ijmcr.com*

Multiple UAVs communication in Intelligent Environment

Sameer

Shah Satnam Ji P.G Boys College, Sirsa, India

Accepted 26 Dec 2016, Available online 30 Dec 2016, **Vol.4 (Nov/Dec 2016 issue)**

Abstract

Seamless mobility of the nodes in the multi-UAV cooperative causes the network topology to change frequently. As the UAVs have limited sensor capabilities, cooperative control relies heavily on communication with appropriate neighbours. The leashing problem is to physically or electronically constrain radio nodes within the network so that communication can take place between designated end points. This problem is of interest in UAV applications, as communication is often required between nodes that would not otherwise be able to communicate for instance because of range constraints or line-of-site obstructions. Efficient, reliable, low latency communication is required to fully realize and utilize the benefits of multi- vehicle teams. The need for communication is very vital in the performance of the UAV team tasks. This survey paper deals with the comparison of shortest path finding algorithms for intelligent environment

Keywords: *Seamless mobility of the nodes etc.*

1. Introduction

As the UAVs have limited sensor capabilities, cooperative control relies heavily on communication with 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, that is used by routing algorithms to determine the optimal path to a destination. To aid the process of path determination, routing algorithms initialize and maintain routing tables, which contain route information. Routing involves two basic activities: determining optimal routing paths and transporting information groups through an internetwork.

A. Example Network

Routing algorithms often have one or more of the design goals such as optimality, simplicity and low overhead, robustness, stability, rapid convergence and flexibility.

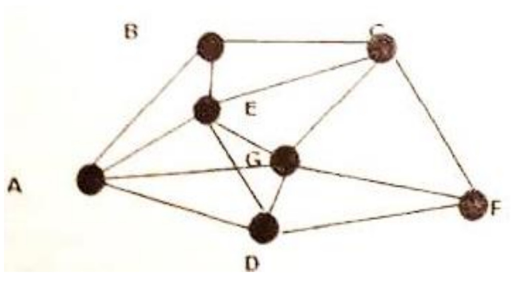


Figure 1 Example Network

The advantages of coordinating and collaborating UAV teams are: to accomplish the missions in a shorter period, to accomplish many goals simultaneously, teams of small aircraft can be cheaper and less detectable than a single, larger vehicle; or damage to a single UAV does not necessarily affect the entire mission. Achieving the leashing goal for instance in a more optimal way by knowledge sharing is one of the research goals. Optimality refers to the capability of the routing algorithm to select the best route, which 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, but it may weigh delay more heavily in the calculation. Naturally, routing protocols must define their metric calculation algorithms strictly. Efficiency is particularly important when the software implementing the routing algorithm must fun on a computer with limited physical resources.

As shown in Figure 1, there is no direct radio channel between 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.

As the UAVs have limited sensor capabilities, cooperative control relies heavily on communication with 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, that is used by routing algorithms to determine the optimal path to a destination.

B. Path Determination

To aid the process of path determination, routing algorithms initialize and maintain routing tables, .which contain route information. Routing involves two basic activities: determining optimal routing paths and transporting information groups through an internetwork. The information about the target should reach the human supervisors even in the presence of obstacles. A scalable, reliable, efficient, low latency communication algorithm is required for a heterogeneous, high speed network. For a low latency communication the shortest path finding algorithm is vital. The fastest, cheapest, or easiest route to take is oftentimes more important than finding just any path. That is where optimal search comes in

Depending on the relevance of information the information should be routed such that the information reaches the control station in time and also uses the bandwidth more efficiently. The weight of the path depends on various factors like the distance, end-to-end time delay, reliability of foe link etc. In wireless networks where foe nodes seamlessly move, the weights can be defined dynamically using learning technique. These weights are passed on to foe router which decides foe route so that foe target information reaches foe source without any error and also in time. When there is an image of foe source to be transmitted to foe control station, then foe message or foe information could be divided into packets which reach foe destination which is foe control station with human supervision through foe available shortest path.

2. Path Finding Algorithms

A. Literature Review

Path finding is to find foe possible routes from foe source to the destination while path planning is deciding which route to take based on foe terrain. The complete route to be taken is created as a sequence of moves from foe initial position. At each step, foe algorithm must decide which way to move next. An efficient dynamic implementation is obviously required if foe overhead is to be minimized. The properties of foe path can be taken into account by a deliberative thinking module, responsible for making founded decisions. For example, path lengths can be used to estimate travel-time, which influences foe agent's decision to move to a particular location. The various path finding algorithms[5-12] are studied and compared

The recent developments in Intelligent Transportation Systems (ITS), particularly in foe field of in-vehicle Route Guidance System (RGS) and real time Automated Vehicle Dispatching System (AVDS) where there is a definite need to find foe shortest paths from an origin to a destination in a quick and accurate manner. Because foe travel times are foe basic input to foe real-time routing and scheduling process and are dynamic in most urban traffic environments, there is an implicit 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 (1-4), where the shortest path problem is often used as a testing mechanism to demonstrate foe effectiveness of these heuristics.

Distance Vector is a decentralized routing algorithm that requires that each router simply inform its neighbours of its routing table. For each network path, foe receiving routers pick foe neighbour advertising foe lowest cost, then add this entry into its routing table for re-advertisement. To find foe shortest path, Distance Vector is based on one of two basic algorithms: the

Bellman-Ford and foe Dijkstra's algorithms (OSPF). In RIP (The Routing Information Protocol), Distance Vector is known as the Bellman-Ford algorithm. Open Shortest Path First (OSPF) is a routing protocol developed for Internet Protocol (IP) networks by foe Interior Gateway Protocol (IGP) working group of the Internet Engineering Task Force (IETF).

Distance Vector algorithm is iterative as foe process of exchanging information will continue until no more information is exchanged between foe neighborhoods, distributed as this algorithm enables each node receives some information from one or more of its directly attached neighbours and asynchronous as this algorithm does not require all of the nodes to operate in lock step with each other. Dijkstra's algorithm creates labels associated with vertices.

These labels represent foe cost from foe source vertex to that particular vertex. Within foe graph, there exists 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 foe source vertex is known. The value given to these labels is foe 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 foe 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 foe 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 foe addition of foe 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 foe single-source shortest path problem in weighted graphs. Dijkstra's algorithm start from a source node and in each iteration adds another vertex to the shortest-pafo spanning tree. This vertex is the point closest to foe 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 foe tree path, only the sum of their weights. The time required by Dijkstra's algorithm is O (|V|2).

A graph can be represented as an adjacency matrix A in which each element (i, j) represents the edge between element i and j. A y = 1, if there is an edge between node i and j, otherwise, A y = 0. 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.

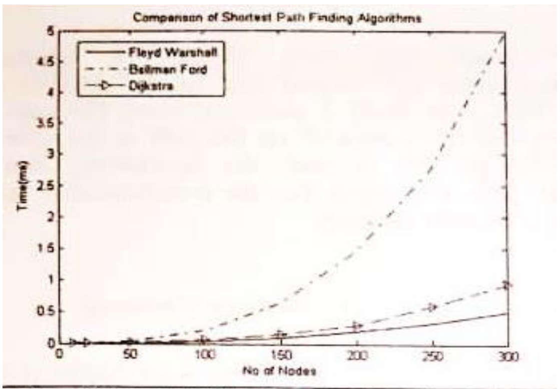
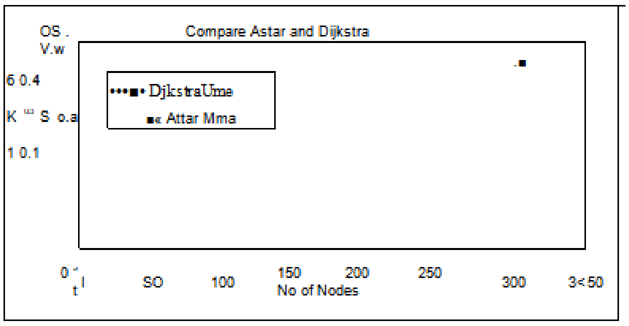


Figure 2 Shortest Path Finding Algorithms Comparison

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 Bellman-Ford algorithm runs in O (E) time. Routers that use this algorithm have to maintain the distance tables, which tell the distances and shortest path to sending packets to each node in the network. The information in the distance 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, the number of outgoing packets, etc.

Pathfinders let you look ahead and make plans rather than waiting until the last moment to discover there's a problem. Movement without path finding works 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 from Artificial Intelligence or Algorithms research are designed for arbitrary graphs.

The Best-First-Search (BFS) algorithm works in a similar way, except that it has some estimate called a heuristic of how far from the goal any vertex is. Instead of selecting the vertex closest to the starting point, it selects the vertex closest to the goal. BFS is *not* guaranteed to find a shortest path. However, it runs much quicker than Dijkstra's algorithm because it uses the heuristic function to guide its way towards the goal very quickly.



For example, if the goal is to the south of the starting position, BFS will tend to focus on paths that lead southwards. In the following diagram, yellow represents those nodes with a high heuristic value (high cost to get to the goal) and black represents nodes with a low heuristic value (low cost to get to the goal). 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 heuristic approaches like BFS and formal approaches like Dijsktra's algorithm. Ifs a little unusual in that heuristic approaches like BFS usually give you an approximate way to solve problems without guaranteeing that you get the best answer. However, A* is built on top of the heuristic, and although the heuristic itself does not give you a guarantee, A* can guarantee a shortest path.

3. Results

The A* 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 BFS in that it can use a heuristic to guide itself. In the simple case, it is as fast as BFS.

Dynamic Programming [16] 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 (DP) 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. The Floyd-Warshall Algorithm is an application of Dynamic Programming. Given a directed graph, the Floyd-Warshall All Pairs Shortest Paths algorithm computes the shortest paths between each pair of nodes in O (n^A3).

References

[1] Halt EP, Nilsson NJ, Raphael B.A formal basis for the heuristic determination of minimum cost paths. IEEE Transaction, System Science and Cybernetics 1968;SSC-4(2): 100-7.
[2] Nilsson JN. Problem-solving methods in artificial intelligence. NewYoik: McGraw-Hill; 1971.
[3] Newell A, Simon HA Human problem solving. Englewood Oifls, NJ: Prentice-Hall; 1972.
[4] Pearl J. Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Publishing Company; 1984.
[5] Gallo G, Pallottino S. Shortest path methods. In: Florian M, editor. Transportation planning models. Amsterdam: Elsevier Science Publishers; 1984. p. 227-56.
[6] Hung SM, Divoky JJ. A computational study of efficient shortest path algorithms. Computers and Operations Research 1988;15(6):567-76.
[7] VurenVT, JansenGRM.Recent developments in path finding algorithms: a review.Transportation Planning andTechnology 1988;12:57-71.
[8] Cherkassky BV, Goldberg AV, Radzik T. Shortest paths algorithms: theory and experimental evaluation. Mathematical Programming 1996;73(2): 129-74.
[9] Zhan FB, Noon CE. Shortest path algorithms: an evaluation using real road networks. Transportation Science 1998;32(1): 65-73.