A SURVEY ON NODE BASED PATH PLANNING ALGORITHMS

[1]FATHIMA NIJIYA K, [2]JINI RAJU

[1]Mtech Student, [2]Assistant Professor
[1] Computer Science and Engineering,
[1] T K M College of Engineering, Kollam, India.

Abstract: Recent advancement in research and technology has led to the development of various self-adaptive systems like unmanned ground vehicles, unmanned aerial vehicles and robots. The autonomous navigation of these systems is an important area of concern. There exists various node based path planning algorithms that can find a collision free path from an initial location to a destination location avoiding obstacles. In this paper, we provide a comparative study of some node based algorithms based on the time efficiency, solution quality and type of data structure used. In addition, we conduct an applicable analysis of the studied algorithms after considering the merits and demerits.

Index Terms **- Path planning, Heuristic search, incremental search, Multi-objective path planning and Directed acyclic state expansion graph (DAG).**

I. INTRODUCTION

Path planning is an important research area that has been studied for many years and it has extensive applications in areas of autonomous navigation, robotics, virtual simulations and computer games. We use different path planning algorithms based on the observability and dynamics of the environment. Based on the dynamics, the environments can be divided into static and dynamic. In real world problems, the environments are chosen to be dynamic due to the presence of moving and stationary obstacles. Moreover, the observability factor of the environment is also to be considered. We may need to perform the path planning in fully observable, partially observable or unknown environments. The observability of the environment depends on the sensor range of the mobile agent. Existing path planning algorithms can also be classified based on the objectivity optimization. The path planning algorithms can be either single objective or multi-objective. There exist multiple objectives to be optimized in case of real world problems. Multi objective path planning problems demands the need for decision making under multiple objectives.

In this paper, a study is conducted to analyze the merits and demerits of some node based search algorithms. Here five algorithms are examined: Dijkstra's, A*, D*, MOA* and MOD* Lite. The comparison study of this algorithms will help to find the time efficiency and optimality of the methods studied. Section 2 describes the path planning algorithms in static and dynamic environments. Section 3 provides a comparative study of different algorithms used in path planning for both static and dynamic environments. Section 4 gives a concluding remark on why we need a new technique in path planning.

II. PATH PLANNING IN STATIC AND DYNAMIC ENVIRONMENTS

In static environment, the obstacles are always stationary. So path planning in static environment is the process of finding an optimal path from start to destination location avoiding stationary obstacles with the help of sensors. Path planning in dynamic environments is the process of finding an optimal path from start to destination location avoiding stationary and moving obstacles. Following are few path planning algorithm used in static environments.

2.1 Dijkstra's Algorithm

The algorithm was named after its creator, Edsger Dijkstra [1] and was proposed in 1959. Dijkstra's Algorithm is a special form of dynamic programming and it is also a breath first search method. It finds shortest path which depends purely on local path cost. The basic working of this algorithm is as follows. The cost of all nodes except the initial node is set to be infinity. The cost of initial node is set to be zero. The initial node is marked as current node and all other nodes are marked as unexplored nodes. Now, the initial node is examined to find its eight nearest neighbors [figure 1].

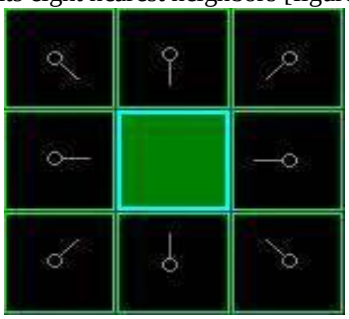

Figure 1. 8 nearest neighbors

The g values of these 8 nearest neighbors are calculated. The g value of a node p is the distance from the initial node to the node p through current node. When all the neighboring nodes of the current node are examined, the current node is marked as explored and the node with the lowest g value is chosen to be the next current vertex. If the new g value of a node is less than the previously recorded g value, the new g value replaces the old one. This process repeats until the target node is marked as explored. In case where the target node is unreachable, this algorithm is repeated until all the nodes are marked as explored.

Here we maintain 2 lists: an open list and a closed list. The closed list contains the explored nodes and open list contain unexplored nodes. Here the open list is implemented using a priority queue. The node with the lowest g value has the highest priority. An improved Dijkstra's algorithm was proposed by adding a center constraint; it works well in tubular objects. But Dijkstra's algorithm relies much on the priority queue Q data structure type, which influences the total exploring time.

Dijkstra's algorithm is considered as a special case of A* algorithm which has no heuristic function. Due to the absence of this heuristic function, it searches by expanding out equally in all directions. Due to this reason, Dijkstra's usually ends up exploring a much larger area before the target if found. So a new algorithm that introduces a heuristic function was introduced named A*.

2.2 A-Star (A*)

A-star [2] is an off-line path planning algorithm that uses heuristic to guide the search and computes the path with minimum cost. This algorithm is an extension of Dijkstra's algorithm and was developed in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael. It was able to limit the unnecessary node explorations with the help of heuristic functions.

Here the heuristic function gives a cost estimation from current node to the goal node. The heuristic function h of a node s is denoted as h(s). Thus, h(s) gives the cost estimation from the node s to the goal node. The term g(s) is the actual cost for moving from start to current node s. The figure 2 shows the pictorial representation of g and h functions.

A* uses an evaluation function f (n) = g(n) + h(n) where g(n) is the actual distance from start node to node n and h(n) is the estimated cost from node n to goal node. This evaluation function consists of a post calculation towards the initial state and a heuristic estimation towards the goal state. A* is one of the best path planning algorithm that has the same uses as that of Dijkstra's algorithm. Dijkstra's algorithm is essentially the same as A* algorithm, except there is no heuristic (H is always 0). Since it has no heuristic, it searches out by expanding equally in every direction. Because of this, Dijkstra's usually ends up exploring a much larger area before the target is found. This generally makes it slower than A*.

The A* algorithm starts from start vertex and finds its 8 adjacent neighbors and adds them to the open list. The start node is the parent for these 8 nodes. The start vertex is inserted into the closed list. From the open list, we choose the vertex with minimum f value and this is the next vertex to be processed. For each vertex, a parent is assigned. Once the processing of goal vertex is completed, the algorithm stops and backtrack the parent chain and reach the start vertex. In order to improve the efficiency of A*, the open list is maintained using a priority queue from which the minimum f value can be easily accessed. The evaluation function is given by f(n) = g(n) + h(n) where g(n) is the actual cost for moving from start vertex to the current vertex and h(n) is the heuristic distance from current to goal node.

By the introduction of heuristic, the algorithm seems to converge very fast and ensures optimality. On comparison with Dijkstra's algorithm, A* was found to converge at a much faster rate. A∗ has been widely implemented in many environments. Amato et al. [8] constructed the feasible road map by applying PRM and then adopted A∗ to execute best route exploration. Authors in [7] implemented A∗ with UAV platform with an octree based PRM. Niu and Zhuo [9] introduced "cell" and "region" conception to enhance the environment understanding of A∗, thus enabling flexible representation of 3D environments. Koenig and Likhachev proposed an environment-representation varying adaptive A∗, that is, Lifelong Planning A∗ (LPA) [10]. LPA∗ can adapt to environment changes by using previous information as well as iterative replanning. Williams and Ragno [11] propounded a conflict-direct A∗; it accelerates the exploring process by eliminating subspaces around each state that are inconsistent. It is proposed in [12] that A∗ can choose any state to be parent state, thus resulting in a more flat turning angle, named Theta∗. The algorithm has the ability to obtain system constraints; thus it can find shorter and more realistic paths. De Filippis et al. [13] implemented both Theta∗ and A∗ in 3D environment, and an experimental comparison is given to prove that Theta∗ reduces the searching compared to A∗. Although Theta∗ acts well compared to A∗, but when applied to 3D environment, it consumes much time to check unexpected neighbors. Line-of-sight check method, called lazy Theta∗ [14], is proposed to avoid unnecessary check of unexpected neighbors. In [15], a method is introduced to reuse information from previous iterations and update information of the affected and relevant portions of the exploring space. This may cause extra computational consumption, but it can tackle dynamical threat and converge fast.

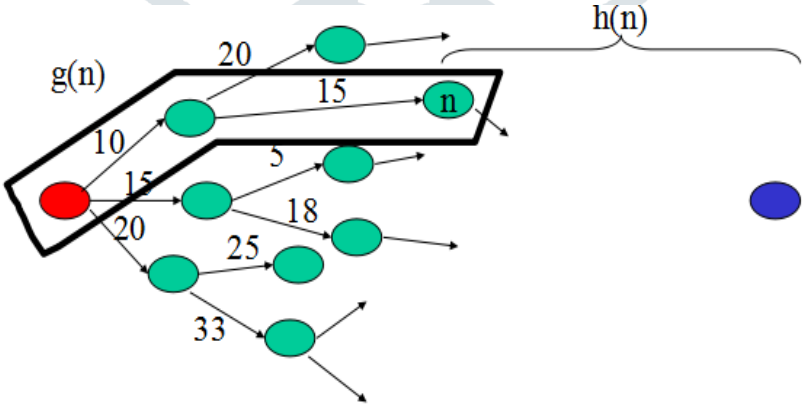

Figure 2. Pictorial representation of g and h functions

2.3 D-Star (D*)

D-star (D∗) [3], short for dynamic A∗, is famous for its wide use in the DARPA unmanned ground vehicle programs. D∗ is a sensor based algorithm that deals with dynamic obstacles by real time changing its edge's weights to form a temporal map and then moves the robot from its current location to the goal location in the shortest unblocked path.

D* is an incremental version of A*algorithm that is it uses the information gained from previous iterations. Consider the navigation of an unmanned ground vehicle. In case of D*, the shortest path from start node to goal node is found initially and the agent follows this path. During this navigation, if the agent faces any obstruction, it re-plans the path by re-computing the shortest path from the point of change as the new start.

D∗, As well as A∗, evaluates the cost by considering the post calculation and forward estimation. D∗ maintains a list of states which is used to propagate information about changes of the arcs cost function. The evaluation function is represented as

$$f'(n) = g'(n) + h'(n) \quad (1)$$

Unlike A∗, $h'(x)$ is not necessarily the shortest path length to the goal; also computation of $h'(x)$ assumes that the robot can pass through obstacles. Each time it encounters a new obstacles, the minimum heuristic function is updated, thus enabling efficient searching in dynamic environments.

Smirnov [16] considered the worst travel distance case of D∗ and restrained its lower bound as $\Omega((\log |V|/\log \log |V|)|V|)$ steps and upper bound as $O(|V|2)$.The bound is used to solve the problem that D∗ uses unrealistic distance in its graph, and there is a large gap between the lower bounds and upper bounds. Based on [16], Tovey et al. [17] put it forward by adding more tighter bounds for D∗. Koenig and Likhachev [18] extended LPA∗ to the case where the goal changes between replanning episodes; it is like a simplified version of D∗, called D∗ Lite, but it is proposed based on LPA∗.

D* algorithm was able to provide optimal solutions in dynamic environment with good time efficiency. Data structure used for implementation of this algorithm is Priority queue. In case of rea world problems, there exist multiple objectives in finding an optimal path. So the need for a multi-objective path planning algorithm was felt and A* algorithm was extended MOA*.

2.4 Multi-Objective A-Star (MOA*)

Multi-Objective A* [5] was extended from classical A* algorithm to handle multiple objectives that inherently exist in many application domains. It uses the evaluation function *f (n) = g(n) + h(n)* similar to A* but functions return vectors instead of scalar values. Size of the vector is the number of objectives to be optimized. If there is only one objective MOA* becomes standard A*. Like A*, it provides complete and optimal solutions when heuristic function is admissible which means the heuristic estimation of every objective is not overestimated.

MOA* keeps track of state expansions using OPEN (explored nodes) and CLOSED (expanded nodes) sets. Non-dominated states are maintained in a subset of OPEN named ND which is formed by the elements that are not dominated by any other element of this set and any of the discovered solutions.

At each iteration of the algorithm, first the best alternative node is selected from ND. Then, the selected node is checked whether it is in the set of goal nodes or not. Then, the current node and its path cost vector are added to the solution set and the iteration continues with selection of a new node. Otherwise, the adjacent nodes of current node are generated. At this step, each newly generated node *n* is checked for being generated for the first time. Then, its path cost estimate vector *f (n)*, traversed path cost vector *g(n)* and the heuristic estimate vector *h(n)* are computed, and the newly generated node is added to OPEN set. If the node is not explored for the first time, there is a possibility that a path passes through this node with non-dominated costs to other candidates. Then the node and its non-dominated cost vectors are taken into consideration in the following steps of the solution. The algorithm repeats over the above steps until the ND set becomes empty. Finally, solution paths are generated by backtracking from goal to start.

The main limitation with this algorithm was that it takes a larger execution time to execute in dynamic and partially observable environments.

2.5 Multi-Objective D-Star (MOD*)

MOD* [6] is an incremental derivation of D* which is in turn an incremental derivation of A*. This algorithm is able to handle the existence of multiple objectives like finding the shortest and the safest path. The algorithm is also able to work just like D* in case of dynamic and partially observable environments.

Here we keep track of 3 values: g value, h value and rhs value. Rhs value is one step look-ahead value of g value and is better informed when compared to g values. Rhs value helps to find the best predecessor with the lowest cost. Multiple objectives were implemented by using an objective vector that consist of n elements, each representing an objective. A key value is found out to determine the priority of a state. In this case, the vectors need to be compared. So we need to solve the issue of non-domination of two vectors.

This issue of choosing a state in the presence of non-dominating keys is solved by the use of Directed Acyclic State Expansion Graph as the data structure. DAD provides a topological ordering of the states based on the key domination. In this model, DAG consists of a set of nodes and their associated state and key values. While inserting a new vertex, we check whether the vertex shows any domination with the nodes in the DAG. In case of domination, an edge is introduced from that node to the nodes that are dominated by it.

The first part of the algorithm initializes the g and rhs values of all nodes as [infinity; infinity] except the goal node. The rhs value of goal node is set to be [0, 0].This inconsistency in g and rhs value results in the insertion of goal node to DAG. In the DAG, the nodes are topologically arranged on the basis of key values. Now we continue to find the g and rhs values of the neighboring nodes and inset the inconsistent vertices into DAG. This process is repeated until the key of start node dominates the key values of all other nodes. After this step, the parent are assigned to each of the non-dominated successors and the solution path is found by backtracking the parents. This algorithm shows an improved execution time when compared to MOA*. There are more challenges existing in this algorithm like the moving target.

III. ANALYSIS

An efficient algorithm is one that provides an optimal solution with better execution time. Initial analysis was done to find which algorithms are better for static environments and the ones that are better for dynamic environment. Their merits and demerits were analyzed and presented as follows:

Table 1. Algorithms and type of environment.

Algorithms	Environment	Advantages	Disadvantages
Dijkstra	Static	Easy to implement for various environments	High time complexity
A*	Static	Fast searching ability in static environments	Limited to static threats only
D*	Dynamic	Fast searching ability in dynamic environments	Unrealistic distance.
MOA*	Static	Multi objective fast searching in static environments	Limited to static threats only
MOD*	Dynamic	Multi objective fast searching in dynamic environments	None

We have also compared the execution times of the two multi-objective algorithms using 20x20, 40x40, 60x60, 80x80, 100x100, 120x120, 140x140 and 160x160.

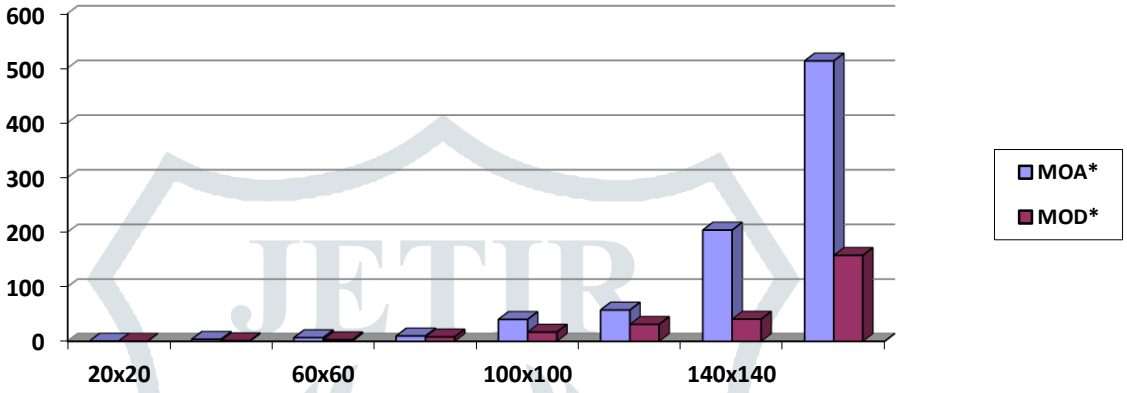

Figure 3. Execution time comparison of MOD* and MOA*

The comparison between MOA* and MOD* shows that MOD* has better execution time when compared to MOA*. Results for larger mazes show significant differences.

IV. Conclusion

In this paper, we have presented a review on different path planning algorithms in static as well as dynamic environment. It is observed from the review that planning path in dynamic environment is complicated as compared to static due to the existence of dynamic obstacles. Many studies have been implemented in robot path planning but very few of them have reported convincing experimental results. There is a gap between theoretical work and experimental results, although a great amount of theoretical work has been validated in simulations and off-line processing. Most current research activities provide effective solutions to robot path planning in static environments. Current robot path planning algorithms have not yet reached the level of robustness and reliability required for real-world applications. Path planning in alien dynamic environments with moving obstacles and moving target is still a challenging aspect for researchers.

The incorporation of multiple objectives into the path planning problem converge it closer with real world problems. The merits and demerits of the algorithms are studied and presented. The execution time comparison of multi objective algorithms were done to study the best node based algorithm under multiple objective and use it to implement the moving targets as a future work.

V. Acknowledgment

We would like to express our sincere gratitude to Dr. S. Ayoob, Principal of T.K.M College of Engineering, Dr. Ansamma John, Associate Professor and Head of the Department, CSE, TKMCE, for their constant support and encouragement throughout the successful completion of this paper. We would also like to express sincere gratitude to all our friends and well-wishers who has helped throughout this work.

References

[1] E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. Numerical Mathematics, vol. 1, no. 1, pp. 269–271.

[2] P. Hart, N. Nilsson and B. Raphael. 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans.Syst. Sci. Cybernetics, vol. 4, no. 2, pp. 100–107.

[3] A. T. Stentz. 1994. Optimal and efficient path planning for partially-known environments. Proceedings IEEE InternationalConference on Robotics Automata. (ICRA), vol. 4, pp. 3310–3317.

[4] S. Koenig and M. Likhachev. 2002. D* lite. Proc. 18th Nat. Conf. Artif.Intell., Menlo Park, CA, USA, pp. 476–483.

[5] B. S. Stewart and C. C. White. 1991. Multiobjective A*. J. ACM, vol. 38,pp. 775–814.

[6] Tugcem Oral and Faruk Polat. 2015. MOD* Lite: An Incremental Path Planning Algorithm Taking Care of Multiple Objectives. IEEE Transactions on Cybernetics.

[7] F. Yan, Y.-S. Liu, and J.-Z. Xiao. 2013. Path planning in complex 3D environments using a probabilistic roadmap method. International Journal of Automation and Computing, vol. 10, no. 6, pp. 525–533.

[8] N. M. Amato, O. B. Bayazit, and L. K. Dale. 1998. OBPRM: an obstacle-based PRM for 3D workspaces. Proceedings of the 3rd Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic

Perspective (WAFR '98), pp. 155–168.

[9] L. Niu and G. Zhuo. 2008. An improved real 3d a∗ algorithm for difficult path finding situation. Proceeding of the International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. 37, Beijing, China.

[10] S. Koenig and M. Likhachev. 2002. Improved fast replanning for robot navigation in unknown terrain. Proceedings of the IEEE International Conference on Robotics and Automation (ICR'02), vol. 1, pp. 968–975.

[11] B. C. Williams and R. J. Ragno. 2007. Conflict-directed A∗ and its role in model-based embedded systems. Discrete Applied Mathematics, vol. 155, no. 12, pp. 1562–1595.

[12] A. Nash, K. Daniel, S. Koenig, and A. Felner. 2007. Theta∗: Any angle path planning on grids. Proceedings of the National Conference on Artificial Intelligence, vol. 22, pp. 1177–1198, AAAI Press, MIT Press, Vancouver, Canada.

[13] L. De Filippis, G. Guglieri, and F. Quagliotti. 2012. Path planning strategies for UAVS in 3D environments. Journal of Intelligent and Robotic Systems, vol. 65, no. 1–4, pp. 247–264.

[14] A. Nash, S. Koenig, and C. Tovey. 2010. Lazy theta∗: any-angle path planning and path length analysis in 3D. Proceedings of the Third Annual Symposium on Combinatorial Search, vol. 2, pp. 153–154, Atlanta, Ga, USA.

[15] M. Likhachev and D. Ferguson. 2009. Planning long dynamically feasible maneuvers for autonomous vehicles. The International Journal of Robotics Research, vol. 28, no. 8, pp. 933–945.

[16] Y. V. Smirnov. 1997. Hybrid algorithms for on-line search and combinatorial optimization problems. Carnegie-Mellon Univ Pittsburgh Pa Dept of Computer Science, vol. 142, p. 141.

[17] C. Tovey, S. Greenberg, and S. Koenig. 2003. Improved analysis of D∗. Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3371–3378, Taipei, Taiwan.

[18] S. Koenig & M. Likhachev. 2005 Fast replanning for navigation in unknown terrain. IEEE Transactions on Robotics, vol. 21, no. 3, pp. 354–363.