Tejas Murali

Dr. Sivakumar Rathinam

MEEN 485 Directed Studies

4/15/20

Path Planning Search Algorithms

EXECUTIVE SUMMARY

The objective of *Path Planning Search Algorithms* is to find a path between two nodes in a preset or changing grid of nodes. In particular, the two Path Planning Search Algorithms that are discussed in this report are the *A* Algorithm*, and the *D* Lite Algorithm*. Both algorithms use an evaluation function and find the optimal path between any two nodes in a grid with or without obstacles. However there is a restriction for the A* Algorithm. While the A* Algorithm only works when the obstacles are static, the D* Lite Algorithm is also works when they are dynamic. Also, if the obstacles are altered or moved during the process, the D* Lite Algorithm replans involving only the changed nodes in order to find the new shortest path, whereas the A* Algorithm would have to start all over. Thus, the D* Lite Algorithm's performance proves to be more useful overall. In this report, pseudocode was devised for each algorithm to form an implementation of code in Python. The difficulties that were encountered mainly dealt with writing the implementation for the D* Lite Algorithm, because it solves a more complex problem.

METHOD

In this section, the steps that were taken to reach the objective and solve the problem are listed. The tables below show the important assumptions, nomenclature and

formulas that were used to write a pseudocode for both algorithms.

Path Planning Search Algorithms	
Objective	To find a path between two nodes in a preset or changing grid
Search Performance Factors	● Completeness ● Optimality (Operating cost) ● Space Complexity - Must take a reasonable amount of space to run as a function of the length of the input ● Time Complexity - Must take a reasonable amount of time to run as a function of the length of the input
Uninformed Search	● Searches blindly without obtaining knowledge about the grid of nodes ● Ex. Dijkstra's Algorithm
Informed Search	● Has an evaluation function that is used to define attributes of the node class ● Evaluation function includes a heuristic which makes search more efficient ● Ex. A* Algorithm, D* Lite Algorithm
Assumptions	Transitions will be euclidean

A* Algorithm	
Objective	In a given grid of nodes with or without obstacles, the A* Algorithm finds the shortest path between any two nodes using an evaluation function
Node	State at a point on grid
Backpointer	Previous node, usually used in backtracking to find path
Start Node	Starting Node
Goal Node	Ending Node
Current Node	Node that is switched from the Open List to the Closed List due to having the lowest f cost
Open List	Set of nodes to be explored
Closed List	Set of nodes already explored
Neighbor	Nodes adjacent to the current node
Child	Next node used in the path (from current)
Evaluation Function	• $f = g + h$ • g is the operational cost, from start to current node • h is the heuristic cost, from current node to goal node

D* Lite Algorithm	
Objective	In a given grid of nodes with or without obstacles, the D* Lite Algorithm finds the shortest path between any two nodes using an evaluation function. Additionally, if the nodes or obstacles are altered or moved during the process, the D* Lite Algorithm will replan the path to find the new shortest path between the two nodes
Inconsistent	g value is not equal to rhs value
Consistent	g value is equal to rhs value
Open List (Priority Queue)	Receives inconsistent nodes for consideration
Successor	Given a node, the node that an edge is directed to
Predecessor	Given a node, the node that an edge is directed from
Evaluation Function	• $rhs = (c + g)$ minimum of successors' • c is the transition value from a node to its successor • g is the g value of a successor
Evaluation Function #2	• $key = (g,\ rhs)$ minimum of a node's • g is the g value of a node • rhs is the rhs value of a node

Furthermore, the pseudocodes that were used to prepare an implementation of code for both algorithms are located in the *Final Report* folder. Please see *Pseudocode Astar* and *Pseudocode Dstar Lite*.

PROCEDURE

This section is crux of the report. The planning step before in the *Method* section of writing pseudocodes was executed to complete the objective and tackle the problem. Inside the Final Report folder are also the implementation of codes for both algorithms that were developed. Please see *MyAstar* and *MyDstarLite*. The advantages of the A* Algorithm were that since it is simpler to understand, implement, and debug, no issues were encountered when testing the code, and it was overall very accurate and consistent. However, as mentioned earlier, shortcomings of the code may include the fact that it does not allow the robot to replan the path when obstacles have changed or moved, and will not be able to satisfy the objectives in such conditions. On the other hand, for the D* Lite Algorithm, there are no such shortcomings, and it has the advantage of being capable of tackling more complex problems with obstacles that have been altered before or during the process because it can instantly replan the path. However, because of the complexity of the algorithm, it is slightly less accurate and required a lot more debugging to fix the errors.

RESULTS AND DISCUSSION

In this section, the results of the procedure will be discussed, and the user will see how to run the code. The A* Algorithm code is easy to run, as the user only needs to simply download the *py* file, open it, and run it. See *Astar Screenshots* located in the Final Report folder. In the grid seen in the code, nodes with a value of *1* are obstacles, and it can be seen how the robot evades them when moving from start to finish. As for the D* Lite Algorithm code, follow the instructions given in the command prompt. See *Dstar*

Lite Screenshots. First enter the start and end node in the format shown. Next, *c* must be pressed for each iteration the obstacles are wished to remain constant, until they are desired to be changed, in which case the obstacle coordinates should be inputted. Repeat this until the final statement with the path is outputted. After understanding how both algorithms are run, the statements made about the two in the *Procedure* are clear and obvious. The A* Algorithm code is straightforward to run but doesn't account for any moving obstacles, whereas the D* Algorithm requires a few inputs by the user and replans the path if necessary. These points will be touched on once again in the last section.

CONCLUSION

Once again, the purpose of Path Planning Search Algorithms is to determine a path between a Start Node and an End Node in a static or dynamic grid of nodes. The two algorithms used in this report were the A* Algorithm and D* Lite Algorithm, both of which had a pseudocode written to later plan and implement a code into Python as seen above. Both algorithms solve the same problem of finding the shortest optimal path between any two given nodes, but A* Algorithm is less complicated to understand and implement which causes it to be more accurate than D* Lite. However, the performance of D* Lite is better for most applications, because it can automatically handle changing obstacles unlike A*, which are more similar to the ones dealt with in real life.

REFERENCES

[1]Choset, Howie. "Robotic Motion Planning: A* and D* Search." Path Planning Search Algorithms. *Robotics Institute 16-735*, 2019, https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf