Name: *Priyank Shah* **NetID:** *pks170030*

Topic: *Optimal and Efficient Path Planning for Partially-Known Environments*

Abstract:

Incremental heuristic search methods use heuristics to focus their search and reuse information from previous searches to find solutions to series of similar search tasks much faster than is possible by solving each search task from scratch. The problem here is to find the most optimal and efficient path from start position to goal position for a moving object in an unknown environment. There are various algorithms like D*, LPA*, D* Lite which solves this problem. I have implemented D* Lite algorithm in this project which is the most efficient and this algorithm uses the concepts of different algorithms like D* and LPA*.

Problem Description:

The problem here is to find the optimal path for the robot moving through the grid which consists of obstacles from start position to goal position. The robot moves in a dynamic environment in which user can add new obstacles anywhere in the grid at any point of time.

Advantages of D* Lite:

The main advantage of this algorithm is we do not have to calculate everything from the scratch. Popular path finding algorithms like Dijkstra's and A* will find an optimal path only if the structure of the search space does not change. A* typically re-plan the entire path when the environment changes. But, in real time applications there can be possibility that path changes during the runtime. These algorithms will fail in such scenarios. D* Lite algorithm is a dynamic path finding algorithm. A dynamic path finding algorithms are the one which hold their search data. If some of the connections are modified then the nodes which are affected will be recalculated, we need not start from scratch.

The D* Algorithm solves the issue of route planning with partially known environments. The name is because it behaves like A* except it is dynamic in nature. The objective of this algorithm is to move the robot from the start position to the goal position such that it avoids all obstacles and minimizes the cost. This algorithm is sound, optimal and complete. Sound because once a state has been visited, a finite sequence of back pointers to the goal has been constructed. Optimal because it defines the condition under which the chain of back pointers to the goal is optimal. Complete because if a path exists from current position to the goal it will be constructed.

It solves goal-directed navigation problems in unknown terrain. It computes the shortest path between current robot position to goal position. D* Lite is an incremental heuristic search algorithm that is used by agents where the surrounding terrain is not completely unknown. D*

Lite finds the shortest route from the start to the goal as it encounters new obstacles. These obstacles are considered and only part of the path is modified.

D* lite is the better algorithm as it introduces a new value in the algorithm. This value is known as the Right-Hand Side (RHS) value. This value is equal to the cost to the parent of a node plus the cost to travel to that node. By comparing this value to the cost of the node we can detect inconsistencies.

D* algorithm is efficient because it uses a heuristic to restrict attention to only those states that could possibly be relevant in repairing the current solution path from current state to goal state.

Example:

Consider the Google maps application where you enter your current position i.e. start position and your destination i.e. goal position. There might be an accident or major road block on your way. In this case, map should be dynamically updated and provide the most optimal path. All the changes should happen dynamically. D* Lite algorithm helps in these situations.

D* Lite Algorithm:

procedure CalculateKey(s)
{01'} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;

procedure Initialize()
{02'} $U = \emptyset$;
{03'} $k_m = 0$;
{04'} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{05'} $rhs(s_{goal}) = 0$;
{06'} U.Insert(s_{goal}, CalculateKey(s_{goal}));

procedure UpdateVertex(u)
{07'} if $(u \neq s_{goal})$ $rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{08'} if $(u \in U)$ U.Remove(u);
{09'} if $(g(u) \neq rhs(u))$ U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath()
{10'} while (U.TopKey() $<$ CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start})$)
{11'} $k_{old} =$ U.TopKey();
{12'} $u =$ U.Pop();
{13'} if ($k_{old} <$ CalculateKey(u))
{14'} U.Insert(u, CalculateKey(u));
{15'} else if $(g(u) > rhs(u))$
{16'} $g(u) = rhs(u)$;
{17'} for all $s \in Pred(u)$ UpdateVertex(s);
{18'} else
{19'} $g(u) = \infty$;
{20'} for all $s \in Pred(u) \cup \{u\}$ UpdateVertex(s);

procedure Main()
{21'} $s_{last} = s_{start}$;
{22'} Initialize();
{23'} ComputeShortestPath();
{24'} while $(s_{start} \neq s_{goal})$
{25'} /* if $(g(s_{start}) = \infty)$ then there is no known path */
{26'} $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{27'} Move to s_{start};
{28'} Scan graph for changed edge costs;
{29'} if any edge costs changed
{30'} $k_m = k_m + h(s_{last}, s_{start})$;
{31'} $s_{last} = s_{start}$;
{32'} for all directed edges (u, v) with changed edge costs
{33'} Update the edge cost $c(u, v)$;
{34'} UpdateVertex(u);
{35'} ComputeShortestPath();

D* Lite Working:

It will work by basically running A* search in reverse, starting from the goal and attempting to work back to the start. It will find the most optimal path from start to goal. As the solver searches it also updates permanent cells to hold the values that they were assigned on this run of the algorithm. The solver then gives out the current solution and waits for change in the weights or obstacles that it is presented with. This process is repeated until the robot reaches the goal position.

My Approach:

CalculateKey:

This function calculates the key of the node. The key is also called as the priority of the node.

Initialize:

This function initializes all the nodes with their g value and rhs value. The rhs value of the goal node is set to 0.

UpdateVertex:

For the node passed in this function, it will recalculate its rhs value based on all its successor nodes. The rhs value of a node is computed based on the g value of its successors in the graph and the transition costs of the edges from node to its successors.

ComputerShortestPath:

This is the function where the most optimal path is found out. We keep an open list which contains all the nodes to be explored. We examine the nodes in the open list and if the node is inconsistent, the algorithm makes the node consistent.

Main:

This is the main function where all the above functions are called. We initialize the graph and compute the most efficient path according to the current scenario.
Once the calculation is done, we ask if the user wants to modify the environment by adding more obstacles. If there are any changes in the graph, we update the vertex of that node and its neighbors.

Data and Experiments:

I have implemented the D* Lite algorithm.

Input: size of the 2D matrix, start and goal position of the robot, a list of obstacles which are placed initially.

Output: location of the robot at each step, until it reaches the goal. If cannot reach the goal, no path found

Experiment 1: No obstacles

```
Enter the size of the Board
5
Initial Positions
---------------------------------------
|   1   |   2   |   3   |   4   |   5   |
---------------------------------------
|   6   |   7   |   8   |   9   |   10  |
---------------------------------------
|   11  |   12  |   13  |   14  |   15  |
---------------------------------------
|   16  |   17  |   18  |   19  |   20  |
---------------------------------------
|   21  |   22  |   23  |   24  |   25  |
---------------------------------------
Enter the cell number for the Start and Goal
10
22
Enter the number of obstacles
0
```

```
---------------------------------------
|   1   |   2   |   3   |   4   |   5   |
---------------------------------------
|   6   |   7   |   8   |   9   |   S   |
---------------------------------------
|   11  |   12  |   13  |   C   |   15  |
---------------------------------------
|   16  |   17  |   18  |   19  |   20  |
---------------------------------------
|   21  |   G   |   23  |   24  |   25  |
---------------------------------------
```

```
---------------------------------------
|   1   |   2   |   3   |   4   |   5   |
---------------------------------------
|   6   |   7   |   8   |   9   |   S   |
---------------------------------------
|   11  |   12  |   13  |   14  |   15  |
---------------------------------------
|   16  |   17  |   C   |   19  |   20  |
---------------------------------------
|   21  |   G   |   23  |   24  |   25  |
---------------------------------------
```

```
---------------------------------------
|   1   |   2   |   3   |   4   |   5   |
---------------------------------------
|   6   |   7   |   8   |   9   |   S   |
---------------------------------------
|   11  |   12  |   13  |   14  |   15  |
---------------------------------------
|   16  |   17  |   18  |   19  |   20  |
---------------------------------------
|   21  |   G   |   23  |   24  |   25  |
---------------------------------------
Reached Goal
```

Experiment 2: Obstacles initially

```
Enter the size of the Board
5
Initial Positions
-------------------------------------
|   1   |   2   |   3   |   4   |   5   |
-------------------------------------
|   6   |   7   |   8   |   9   |   10  |
-------------------------------------
|   11  |   12  |   13  |   14  |   15  |
-------------------------------------
|   16  |   17  |   18  |   19  |   20  |
-------------------------------------
|   21  |   22  |   23  |   24  |   25  |
-------------------------------------
Enter the cell number for the Start and Goal
1
25
```

```
Enter the number of obstacles
4
Enter the cell number for the Obstacles
12
Enter the cell number for the Obstacles
13
Enter the cell number for the Obstacles
14
Enter the cell number for the Obstacles
19
```

```
-------------------------------------
|   S   |   2   |   3   |   4   |   5   |
-------------------------------------
|   C   |   7   |   8   |   9   |   10  |
-------------------------------------
|   11  |   O   |   O   |   O   |   15  |
-------------------------------------
|   16  |   17  |   18  |   O   |   20  |
-------------------------------------
|   21  |   22  |   23  |   24  |   G   |
-------------------------------------
```

```
-------------------------------------
|   S   |   2   |   3   |   4   |   5   |
-------------------------------------
|   6   |   7   |   8   |   9   |   10  |
-------------------------------------
|   C   |   O   |   O   |   O   |   15  |
-------------------------------------
|   16  |   17  |   18  |   O   |   20  |
-------------------------------------
|   21  |   22  |   23  |   24  |   G   |
-------------------------------------
```

```
-------------------------------------
|   S   |   2   |   3   |   4   |   5   |
-------------------------------------
|   6   |   7   |   8   |   9   |   10  |
-------------------------------------
|   11  |   O   |   O   |   O   |   15  |
-------------------------------------
|   16  |   17  |   18  |   O   |   20  |
-------------------------------------
|   21  |   22  |   23  |   C   |   G   |
-------------------------------------
```

```
-------------------------------------
|   S   |   2   |   3   |   4   |   5   |
-------------------------------------
|   6   |   7   |   8   |   9   |   10  |
-------------------------------------
|   11  |   O   |   O   |   O   |   15  |
-------------------------------------
|   16  |   17  |   18  |   O   |   20  |
-------------------------------------
|   21  |   22  |   23  |   24  |   G   |
-------------------------------------
Reached Goal
```

Conclusion:

D* Lite algorithm solves the goal directed navigation problem in unknown environment and is also known as fast re-planning algorithm. It helps in computing the shortest path from start to goal in dynamic conditions. D* Lite algorithm works with real cost.

References:

https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.65.5979

http://idm-lab.org/bib/abstracts/papers/aaai02b.pdf

https://en.wikipedia.org/wiki/D*

https://www.cs.cmu.edu/~motionplanning/lecture/AppH-astar-dstar_howie.pdf

http://www.cs.cmu.edu/~maxim/files/dlite_tro05.pdf