RBE 550: Advanced Search Algorithms Implementation Report

Joel John
jjohn2@wpi.edu

Worcester Polytechnic Institute — April 8, 2022

1 D* Replanning

The D* Algorithm replans a path by processing a node based on whether it is a raise state or a lower state.

The D* Algorithm first finds a path on the static map. Then it moves towards the goal using the path. On each step of the path it senses the environment. If an obstacle is detected on the next step of the path, it means that a dynamic obstacle has appeared and a path needs to be replanned.

A list called the open list is used as a priority queue. Each node has a k value and a h value associated with it. The h value of the node is the cost to get to that node from the goal and the k value is the minimum h value that node has ever had. The nodes with the least k value is evaluated first from the priority queue. If a node is in the open list, it is tagged as open. If the node leaves the open list, it is tagged as closed. When a dynamic obstacle comes into the map, there will be node costs that are affected by this obstacle. These nodes have to be updated with their new costs. To do this, the obstacle and its neighbouring nodes are put back into the open list.
The next node in the priority queue is taken and each neighbour is evaluated, the neighbour node who has a backpointer pointing to the node in consideration but its h value is not equal to the sum of the h value of the node in consideration plus the transition cost is put into the priority queue with a very high h value. This is called a raise state as the nodes h value is greater than its k. The reason this happens is because the node which was sensed as an obstacle is now treated as an obstacle, this causes the transition cost to this node to become very high.

The raise state nodes propogate this cost increase to each node having a backpointer pointing to them. Note that the h value is what is being changed, the priority queue still evaluates nodes on their k value.
Again as nodes are popped off from the priority queue, the raise states keep propogating. The algorithm tries to find a node witch does not have a backpointer pointing to the raise state nodes, and whos h value is greater than the k value raise state in consideration. A new path could be found through this node as it could potentially lower the raise state nodes h value. This node is called a lower state. This lower state node will have its h value equal to its k value.
This lower state node is put back into the priority queue and when it is popped out, the algorithm checks if by passing through this node, the h cost of its neighbours can be lowered. If it can, it does it. The nodes whos values are lowered also become lower state nodes and are added back into the priority queue. Thus the cost reductions are propogated by the lower state nodes.

After all the costs are propogated completely, It means that the dynamic obstacle has been integrated into the map and by tracing the backpointers from the start node, we can get the new path.

2 Why does D* have faster replanning than A* or Dijkstra

Dijkstras Algorithm searches for the goal position in all directions, in an attempt to explore all nodes in the map till the goal is found. The A* Algorithm uses a heuristic to give information on how far it is from the goal, so it does not try to explore all nodes in the map and explores nodes closer and closer to the goal. Due to these reasons A* is faster than Dijkstra.

These algorithms do not account for any dynamic changes in the map. A way to use these algorithms in dynamic environments is to plan a new path each time an obstacle is sensed on the path planned in the static map.

This means the entire path is recomputed from scratch with a new start point.

D* Algorithm finds a path on the static map, but when an obstacle is detected on the path planned, it just updates the node costs. In other words it does not replan from scratch, it just updates the costs found during the first runthrough. This is why it has faster replanning than A* and Dijkstras algorithm. The costs of each node in D* are calculated with respect to the goal node to make updating less complicated and easier.

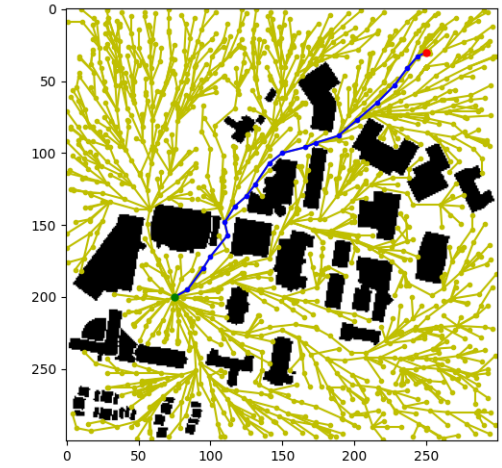
3 RRT* and Informed RRT*, The differences

In RRT*, a point is randomly generated in the map. Then the nearest node to the point in the tree is found and the tree is extended to the point using a step length. The neighbours of the new node extended are found and the path is optimized by rewiring the neighbours.

The random point sampled can be at any place in the map, this point can be very far away from the goal. This wastes lot of computational effort as this point which has no impact on the final path is being considered as a potential path point. It may be considered as a neighbour of another point and will even be rewired.

Informed RRT* solves this issue by sampling points in a region near the goal instead of the entire map. It behaves like the RRT* till an inital path is found. To improve the solution, it then constraints the space to sample points to an ellipsoid. This ellipsoid can be defined using the start and goal position, the length of the path found, and the euclidian distance between the start and the goal. The only variable here is the length of the path, when the length decreases, the ellipsoid becomes smaller till we get an optimal solution. This approach is better at finding paths through narrow spaces as well. By using Informed RRT*, the convergence rate can be improved and a path more optimal than regular RRT* can be found using less computational effort.

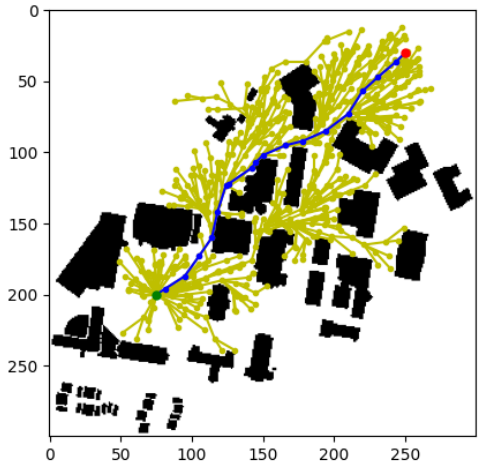
4 Results




```
It took 1494 nodes to find the current path
The path length is 255.51
```

Figure 1: RRT* Result for 2000 points

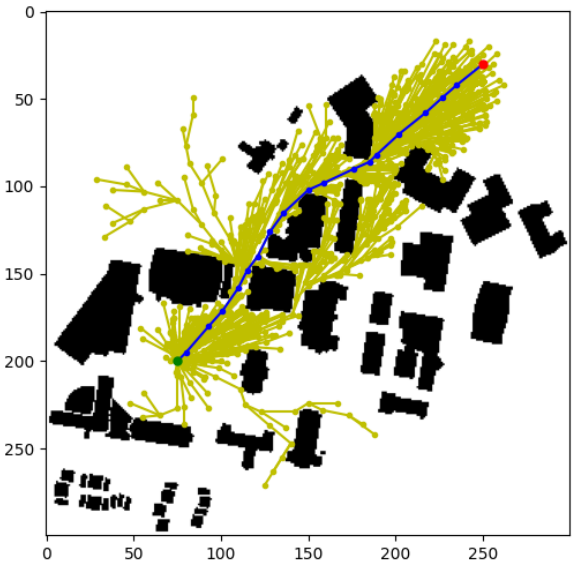
As stated before, Fig.1 shows that the RRT* samples points anywhere on the map and optimizes it. The algorithm has tried to optimize a path to each of these points sampled. This is not really efficient as we are only concerned with the path from start to goal.




```
It took 657 nodes to find the current path
The path length is 255.50
```

Figure 2: Informed RRT* Result for 1000 points

Fig.2 shows the Informed RRT* result after trying to sample 1000 points. We see that points are sampled only in the region that can improve the path. The ellipse shape is sort of visible in the figure. We see that less points are sampled than RRT* and we get a path length which is as good as the RRT* path length.




```
It took 1228 nodes to find the current path
The path length is 248.65
```

Figure 3: Informed RRT* Result for 2000 points

Fig.3 shows the Informed RRT* result after trying to sample 2000 points. Here our path is more optimal than the RRT* path.

Thus by using Informed RRT*, we can get better paths more efficiently than RRT*.