Standard Search Algorithms Report: Jash Mehta

WA*

Pseudocode:

#Initialise start parameters
Row, Col, visited, prev=...
# initialize cost as infinity of all nodes
cost=inf
# initialize queue to store values of nodes and its corresponding cost
queue=....
Assign value “B” to goal point =...
#If condition to check if start is already at goal
If start==goal:
    Stop the search

While elements still left in queue
    If Loop over the nodes in the queue to find new neighbors (avoiding obstacles, out of bound conditions and already visited nodes)
        cost g(x) to neighbor node= Manhattan distance from start
        Heuristic h(x) = distance in x + distance in y from goal
        f(x)= g(x) + epsilon * h(x) (**Weighted heuristic)**
        Add new neighbors along with corresponding cost f(x) to queue
        Mark current node as visited
        Save value of parent node in prev
        Check if goal “B” is reached
        Exit loop
    Sort queue based on cost and run while loop again
    Exit loop

#backtrack the path to get to start
at=coordinates of goal
While at not equal to zero:
    Add at value to path
    Update at with parent node to till it becomes zero
#This path is the path from goal to start so we reverse the order
Reverse path

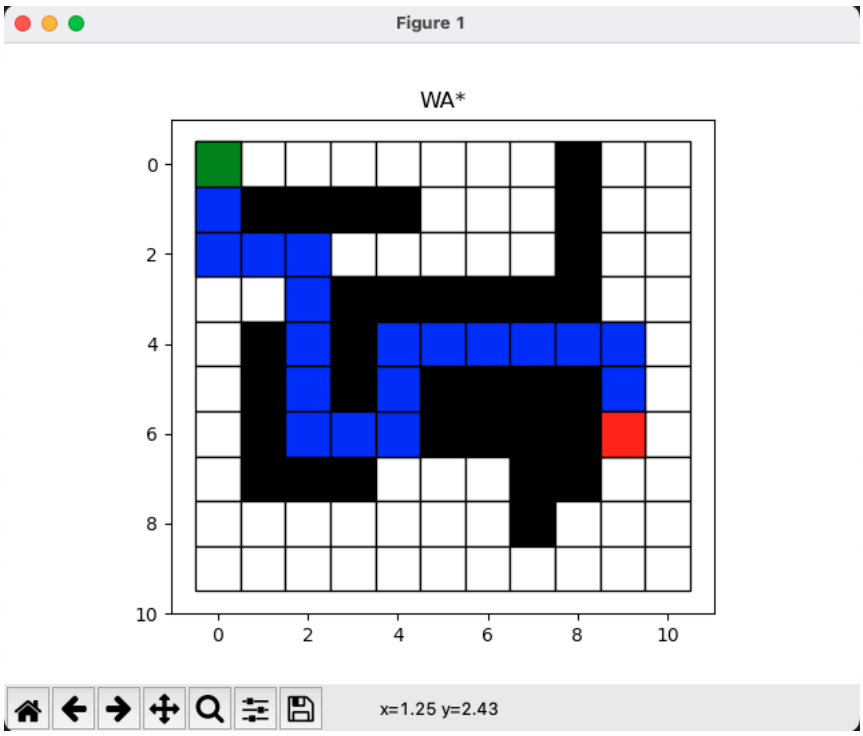
Working of code:

We initialize all the necessary variables and cost to all nodes as infinity and heuristic and create a queue which stores the cost and the corresponding nodes that are still left to explore. We add the start nodes to this queue with cost 0. We run a while loop over all nodes in the queue to find

their neighbors. Boundary conditions are tested and if all are satisfied, we add the new neighbors along with the cost (manhattan distance from the start) and weighted heuristic (manhattan distance from goal to node) to the queue in FIFO order and save their parent node in a prev queue. We make that node as visited and increase the number of steps. We then sort the queue based on the cost (g(x)+epsilon*h(x)) and explore neighbors again. If a node is already visited we compare cost and make parent which has the lower cost. After all the nodes are explored in the queue we backtrack and find the path from the goal to start and then reverse the order.

In weighted A* we give a higher weight to heuristic if epsilon > 1 which makes the search biased towards the goal. This sometimes results in a suboptimal path.

Results:



For simple A* we find that it takes 44 steps to search for a path with a length of 19 steps

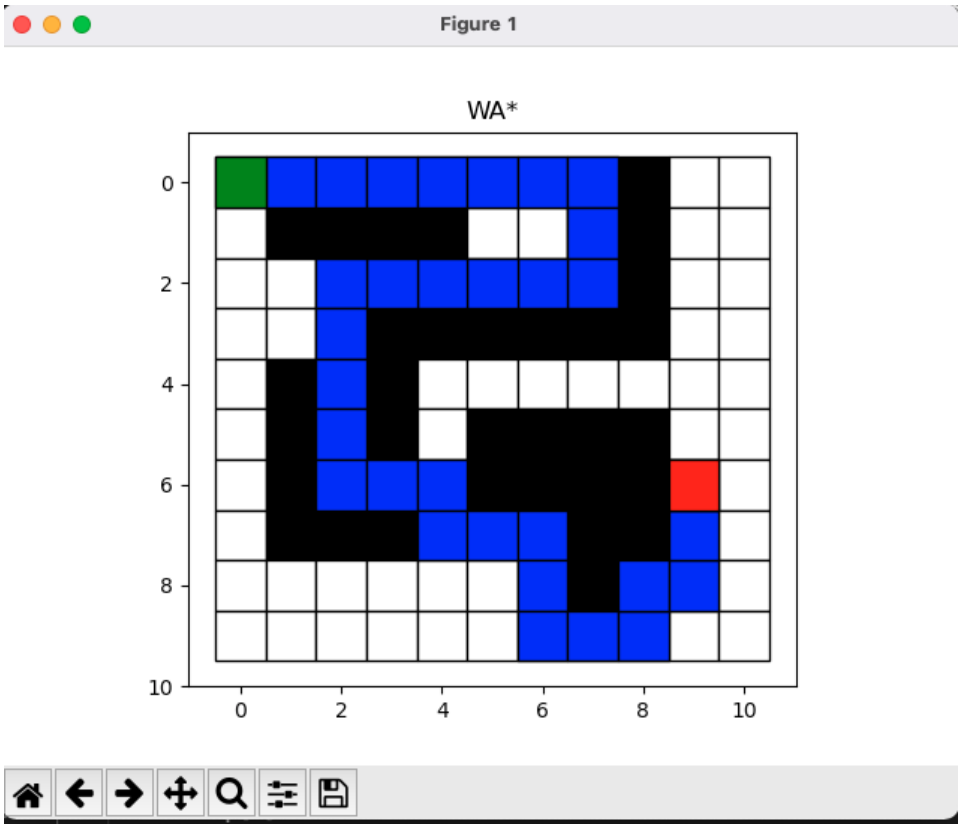
```
(base) jashmehta@Jashs-MacBook-Pro Basic Search Algorithms1 % python3 main.py
It takes 44 steps to find a path using WA*
It takes 19 steps to find traverse path using WA*
```

Epsilon = 3, we find the same path with fewer steps

```
(base) jashmehta@Jashs-MacBook-Pro Basic Search Algorithms1 % python3 main.py
It takes 39 steps to search for a path using WA*
It takes 19 steps to find traverse path using WA*
```

Epsilon = 0.5, takes more steps to get to the same path

```
(base) jashmehta@Jashs-MacBook-Pro Basic Search Algorithms1 % python3 main.py
It takes 58 steps to search for a path using WA*
It takes 19 steps to find traverse path using WA*
```



Epsilon = 6, we find a path in even fewer steps but the path is suboptimal as it takes more steps to traverse to the goal

```
(base) jashmehta@Jashs-MacBook-Pro Basic Search Algorithms1 % python3 main.py
It takes 36 steps to search for a path using WA*
It takes 31 steps to find traverse path using WA*
```

Informed RRT*

Informed RRT* steps

1. Generate random point outside of obstacle
2. Get nearest node to point
3. Get new node in direction of new point
4. Check collision between new node and nearest node
5. Get neighbors of new node
6. Rewire all neighbors
7. Update cost of new node
8. Update parent of new node as neighbor node with lowest cost
9. If close to goal, goal parent = new node
10. Once path is found sample within ellipse with c_max and cmin

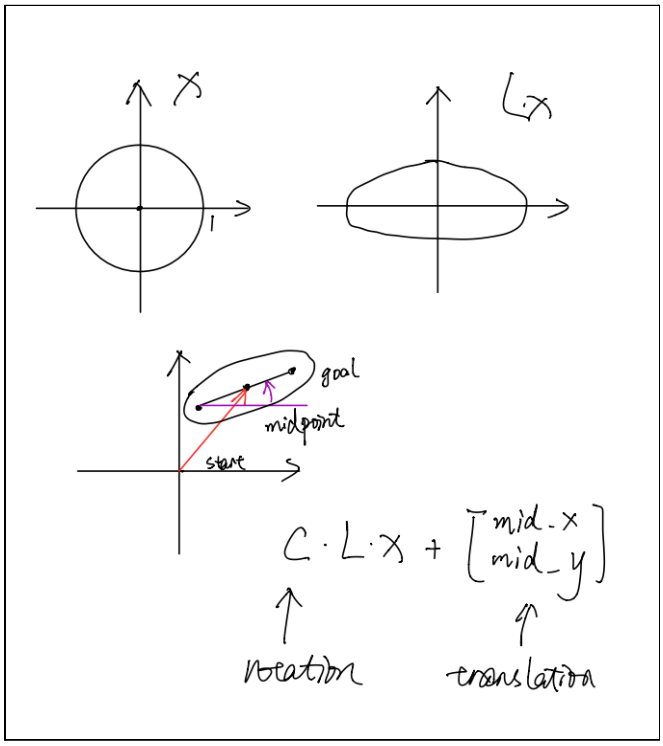
We use informed RRT* over RRT* as it samples in the region very close to goal instead of the entire map once a path is found.

Algorithm 2: $\texttt{Sample}\left(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, c_{\max}\right)$

1 **if** $c_{\max} < \infty$ **then**
2 $\quad c_{\min} \leftarrow \left|\left|\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{start}}\right|\right|_2$;
3 $\quad \mathbf{x}_{\text{centre}} \leftarrow \left(\mathbf{x}_{\text{start}} + \mathbf{x}_{\text{goal}}\right)/2$;
4 $\quad \mathbf{C} \leftarrow \texttt{RotationToWorldFrame}\left(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}\right)$;
5 $\quad r_1 \leftarrow c_{\max}/2$;
6 $\quad \{r_i\}_{i=2,\ldots,n} \leftarrow \left(\sqrt{c_{\max}^2 - c_{\min}^2}\right)/2$;
7 $\quad \mathbf{L} \leftarrow \text{diag}\left\{r_1, r_2, \ldots, r_n\right\}$;
8 $\quad \mathbf{x}_{\text{ball}} \leftarrow \texttt{SampleUnitNBall}$;
9 $\quad \mathbf{x}_{\text{rand}} \leftarrow \left(\mathbf{C}\mathbf{L}\mathbf{x}_{\text{ball}} + \mathbf{x}_{\text{centre}}\right) \cap X$;
10 **else**
11 $\quad \mathbf{x}_{\text{rand}} \sim \mathcal{U}\left(X\right)$;
12 **return** $\mathbf{x}_{\text{rand}}$;

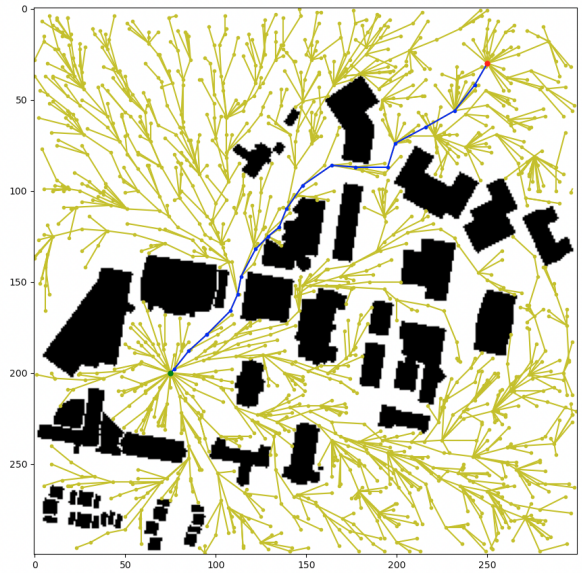
We find a rotation matrix between start and goal. Then we find a diagonal matrix with major and semi-major axes of the ellipse. We then find samples within a unit circle and then implement line 9 of the algorithm mentioned above.

Effectively we perform:



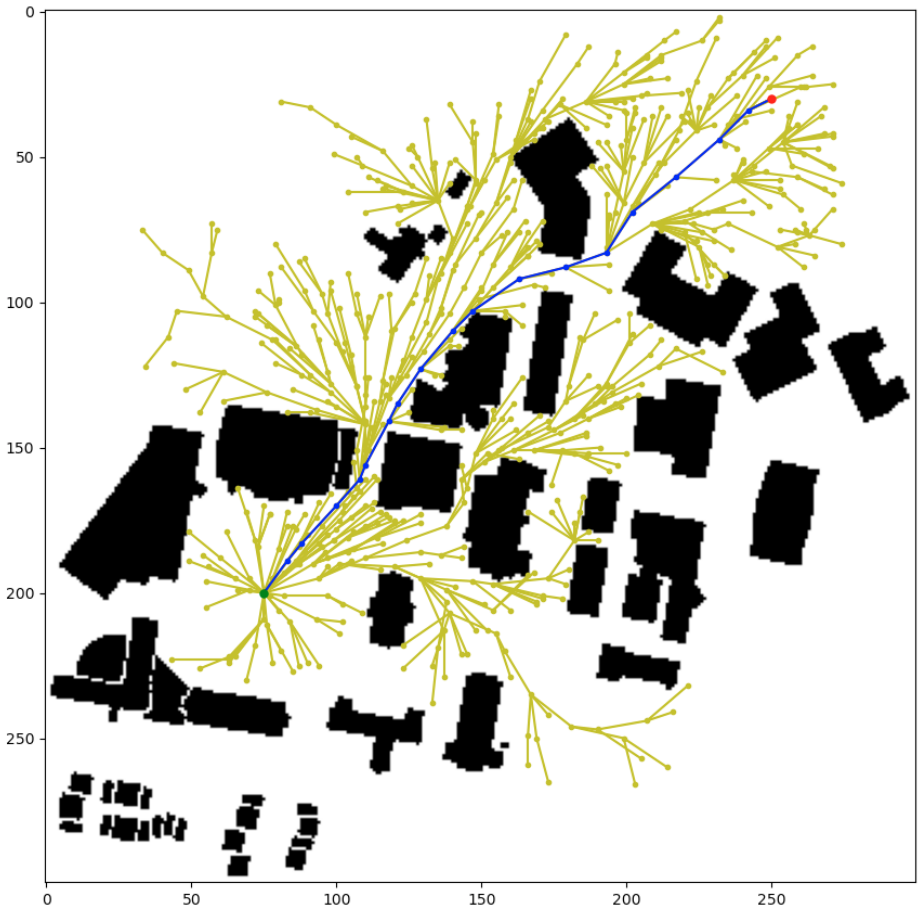
A full explanation of every step of the code is given in the comments of the Code itself.

Results:
A* result



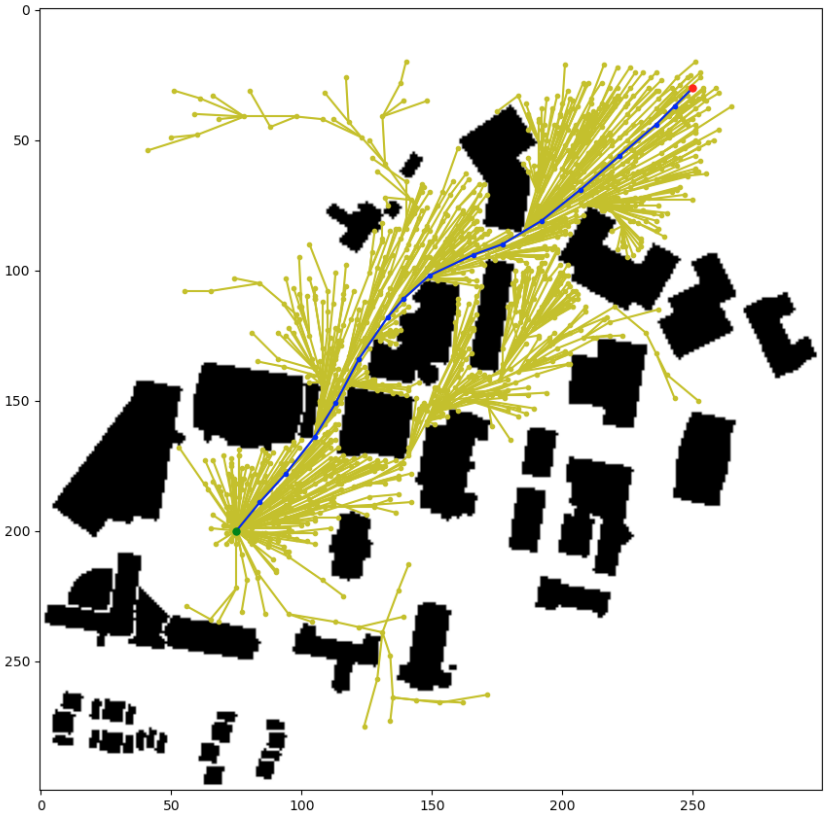

```
It took 1513 nodes to find the current path
The path length is 258.00
```

Informed RRT* with 1000 samples




```
(base) jashmehta@Jashs-MacBook-Pro Standard Search Algorithms % python3 main.py
It took 667 nodes to find the current path
The path length is 258.31
```

Informed RRT* with 2000 samples




```
(base) jashmehta@Jashs-MacBook-Pro Standard Search Algorithms % python3 main.py
It took 1417 nodes to find the current path
The path length is 249.18
```

We can see that as we increase the number of samples, we get a better path to goal and instead of sampling in the entire graph, we sample only in the regions close to the goal.

References:

1. https://theclassytim.medium.com/robotic-path-planning-prm-prm-b4c64b1f5acb
2. https://theclassytim.medium.com/robotic-path-planning-rrt-and-rrt-212319121378
3. https://www.youtube.com/watch?v=Ob3BIJkQJEw&t=302s
4. Reference code provided by Joey (TA)
5. https://arxiv.org/abs/1404.2334