Advanced Search Algorithm Implementation

Monika Sri Vyshnavi Nagalla
Student ID: 387338364

Weighted A* is an extension of A*. Here we add weights to the heuristic function and alter the solution according to the weight values. The cost function in WA* is $f(x)=g(x)+\varepsilon h(x)$. Where ε is the weight value which can range from 0 to infinity. Let's see the results for different ε values.

When $\varepsilon=0$ the search is uniform. Hence the below result is obtained. It took 62 steps to find the path

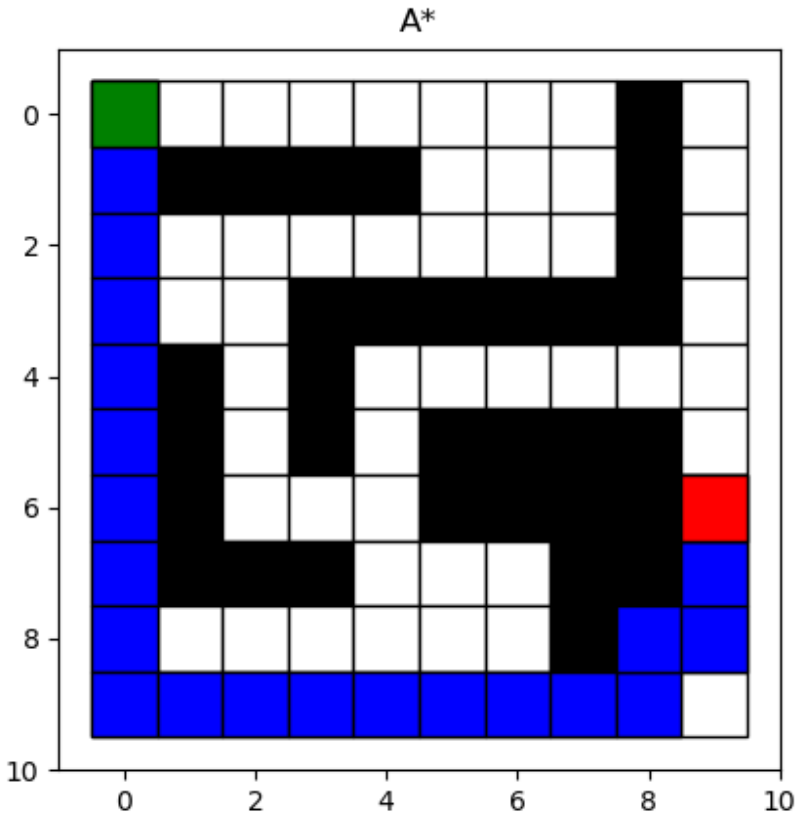

Figure1: $\varepsilon=0$ steps taken to find path=62 and path length =22steps

When ε=0.01 the algorithm still behaves similar to when ε=0. Because we aren't giving enough weight to heuristics that directs our algorithm towards goal.

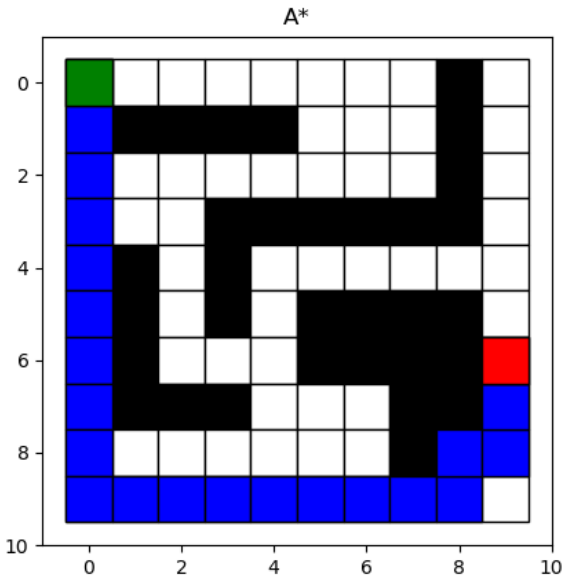

Figure2: ε=0.01 steps taken to find path =61 path length =20steps

When ε=1 the algorithm behaves similar to A*.

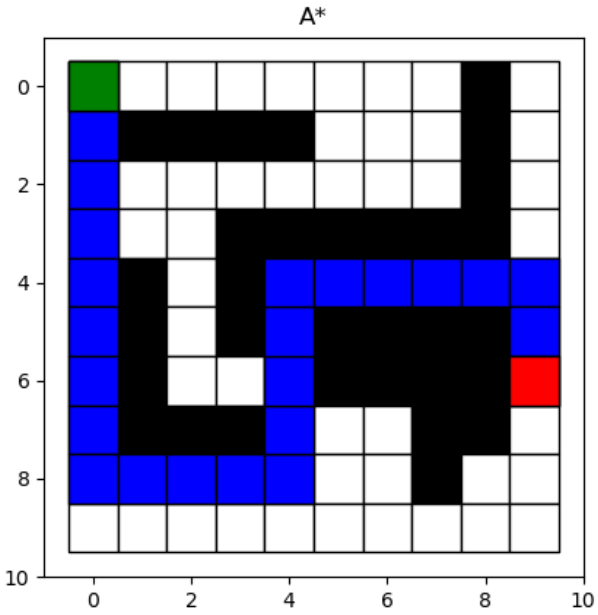

Figure3: ε=1 steps taken to find path=57 path length =22steps

When ε=10 the algorithm biases towards the goal because it is directed by heuristics.

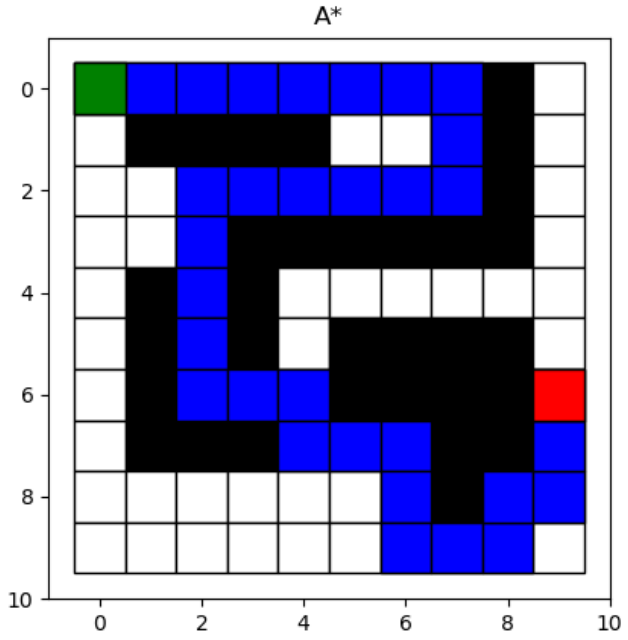

Figure4:ε=10 steps taken to find path =36 and path length=30 steps.

When ε=20 the algorithm does not change because it was saturated at ε=10.

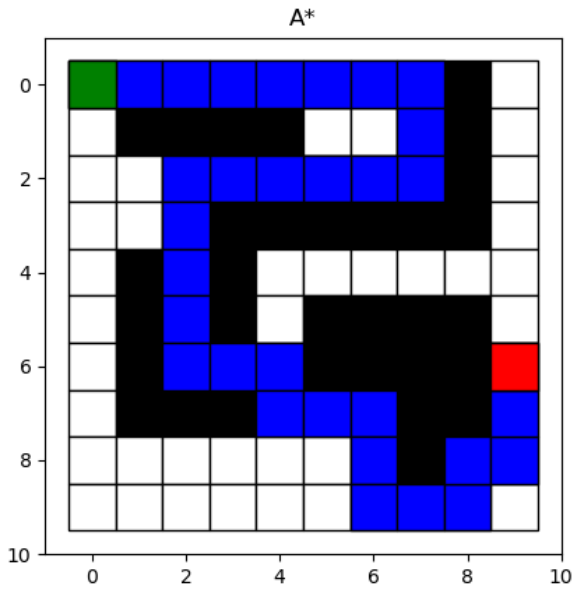

Figure 5:ε=20 steps taken to find path =36 and path length=30 steps.

Observation: As epsilon increases our algorithm tends to move in direction of goal this might result in less steps to find the path but as the algorithm is greedy it gets caught in local minima more often and takes more steps to achieve the goal. Similar patterns can be observed in the above pictures. When ε=10 the algorithm goes toward the goal and traverses through a narrow path.

RRG:

RRG can be treated as extension of RRT with similar RRT* features. The node creations and expansions are the same in RRT and RRG. Extending to node creation we have rewire function to RRG similar to RRT*. In this function we rewire the new found node to its nearest neighbors to form a web like structure. Hence each node has a list of parents instead of a single parent. We maintain the costs from these parents in an array of costs. During rewiring when we have to assign parents to a new_node we first sort the costs of neighbors and route the new_node through least cost path of the neighbor.

Steps in wiring of RRG:

1)`check the path cost to new_node through neighbor[i], take only minimum path through neighbor[i], use sorted array and first element`
2)`append the new path found through neighbour[i] and its costs to a list which can be again used to find shortest path and rewire the parents`
3)`sort the costs to find the best path through neighbors`
4)`wire the parents of new_node as neighbors, through their least cost path found above`
5)`append neighbors into parents list of new_node`
6)`append the particular path cost into costs list of new_ndoe`
7)`find if there exits a better path to neighbor through the new_node`
8)`if better path found then insert the new_node to neighbor[i]'s parents list as first item, because this is the low cost path`

`Below are some figures provided when the RRG algorithm was run over WPI_map.`

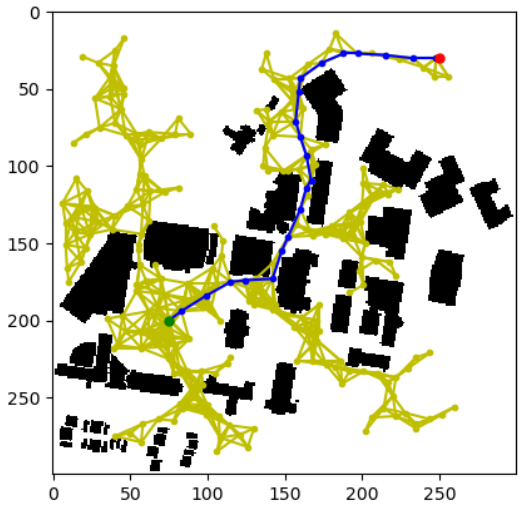

Figure 1: sub-optimal path, It took 248 nodes to find the current path. The path length is 307.13

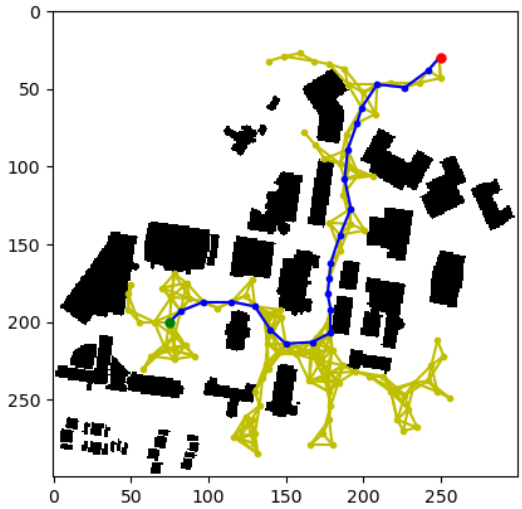

Figure 2:It took 123 nodes to find the current path The path length is 338.78

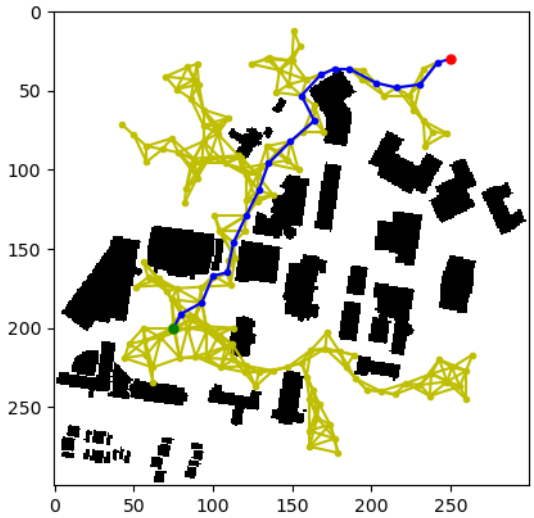

Figure 3:It took 157 nodes to find the current path. The path length is 294.62

Observation:

Here we can observe that the optimal path is not always guaranteed since the point generation and node generation are random. It ends as soon as a path is found unlike RRT* which tries to find the optimal path and keeps exploring even if a path is found