Assignment-2

2023-02-05

Name: Varun Gampa
ID : 773686296

Four algorithms to find the goal (which is on the bottom right corner of the world) from source (top left corner) are implemented. Their paths generated are shown along with the number of iterations it took to find the goal.

The entire grid world is organized as graph. For each vertex, the adjacent **8** cells are directions that the robot can go to. As a result. each unoccupied cell is treated as a vertex, and edges to it are those cells which are surrounding it and are unoccupied by obstacles.

This is represented as an adjacency list in python. That is an array is created, where the argument of array is the vertex, and in each element of array is a linked list, which contains those nodes/ vertices, which are connected to vertex corresponding to the argument of the array, (i.e set of edges or children). The first element of the linked list is the vertex itself, the subsequent elements are its children. Each vertex is represented as node data type, which contains that vertex number, parent node info and the next node info, which are the siblings of that node.

Since the graph is cyclic, the parent of a node is kept as none. The parent is decided depending on how we traverse the graph, note that in BFS,DFS algorithms we visit every node only once.

In Dijkstra algorithm, whenever an edge distance is updated by looking at a node, that node is added to a **priority** queue instead of a normal queue to speed up the algorithm. Whenever a node is added to the priority queue its parent is updated as well.

For the above three algorithms, since we establish a parent node for each node visited, the path generated is simply the list pf parents of the goal node, till the source node is reached.

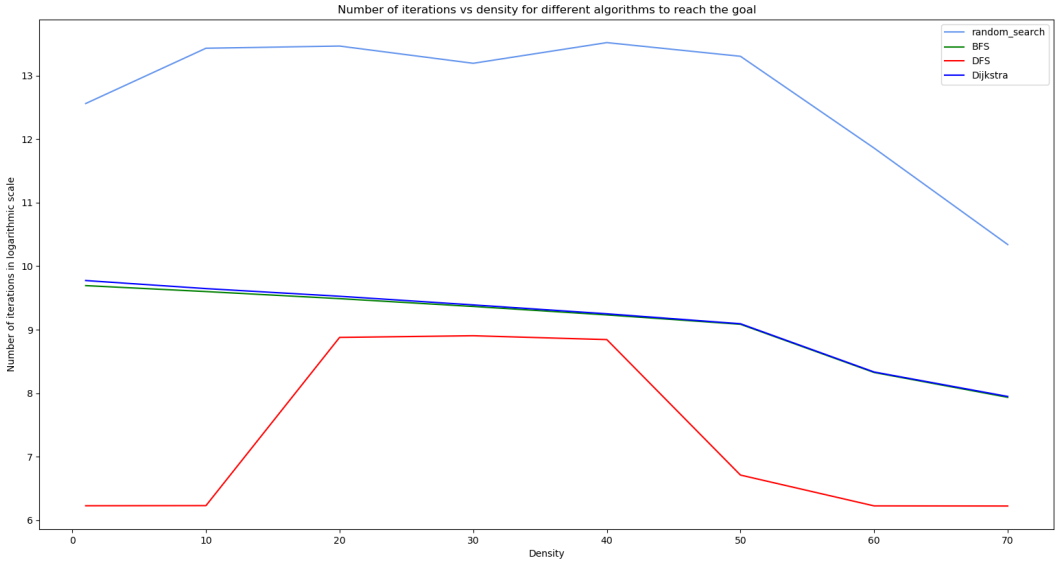
In random search algorithm, the point robot at a vertex (cell) moves in any of the direction specified by the adjacency list corresponding to that vertex. Any direction that the robot moves in is added in the path list. This is the path reported.

Whenever any node is visited, the iteration is incremented for all the algorithms.

The four algorithms were tested in a 2d world with tetromino obstacles, with density of obstacles going from 1 to 70 percent.

Note that in 60 and 70 percent coverage path couldn't exist going from source to goal. Hence the borders of the grid in these cases only are kept unoccupied so that a path could exist.

The plot of iterations versus density:



The plot has logarithmic scale in the number of iterations (y-axis)

It can be seen that DFS finds the solution fastest (it does depend on how the children of a node are organized as).

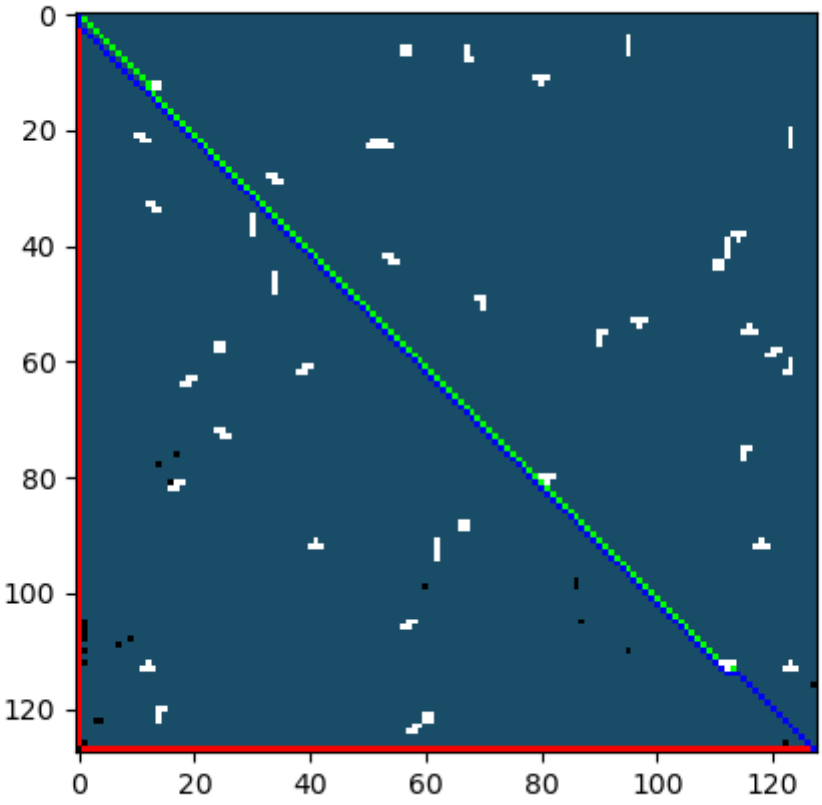
Dijkstra and BFS take almost the same iterations, Dijkstra takes just a little bit longer.

The random search takes the largest number of iterations.

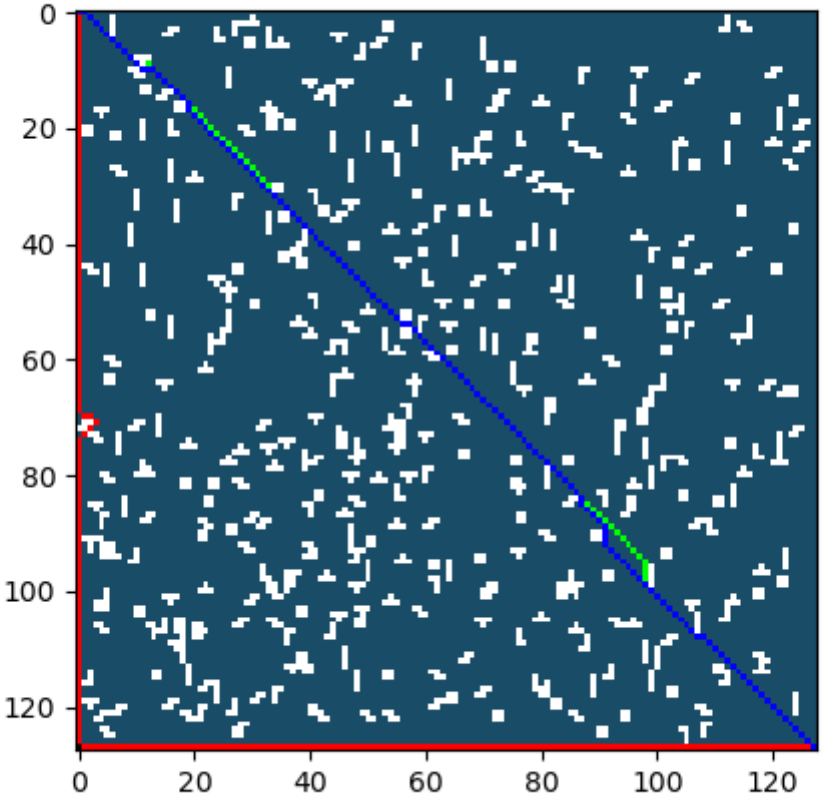
The graphs along with solution paths:

Note that Red path is DFS search, Green is BFS, Blue is Dijkstra, the Greyish-Blue cells are the points visited by random search.

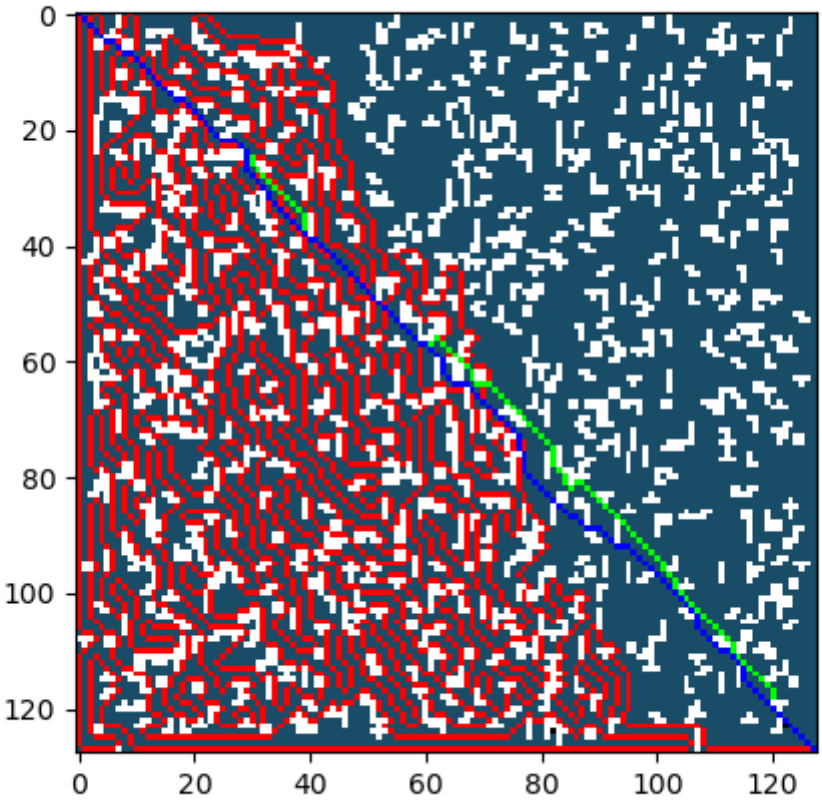
1% obstacles



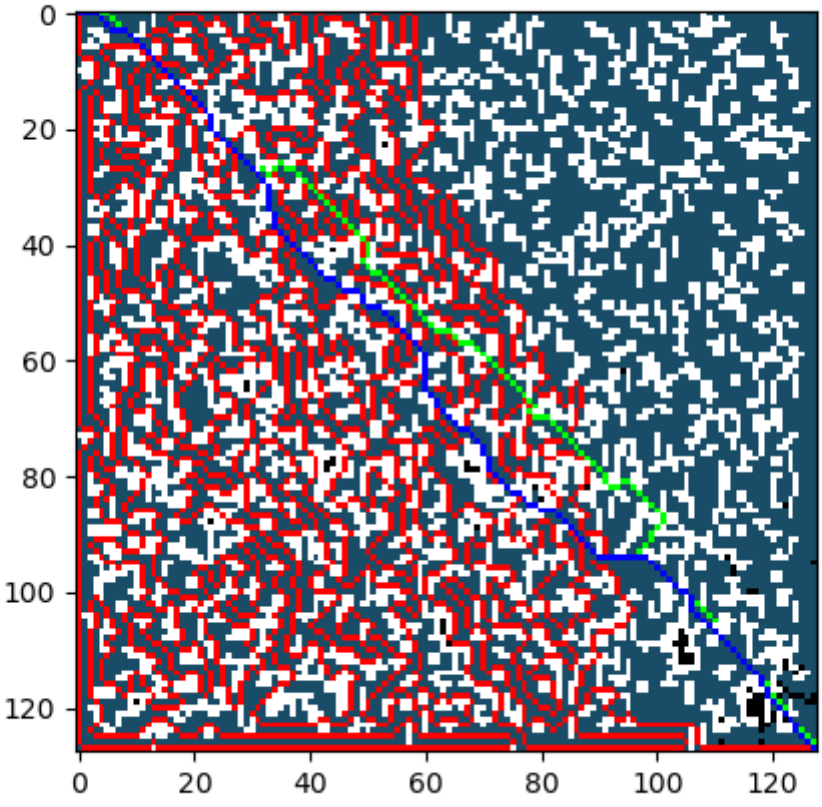
10% obstacles



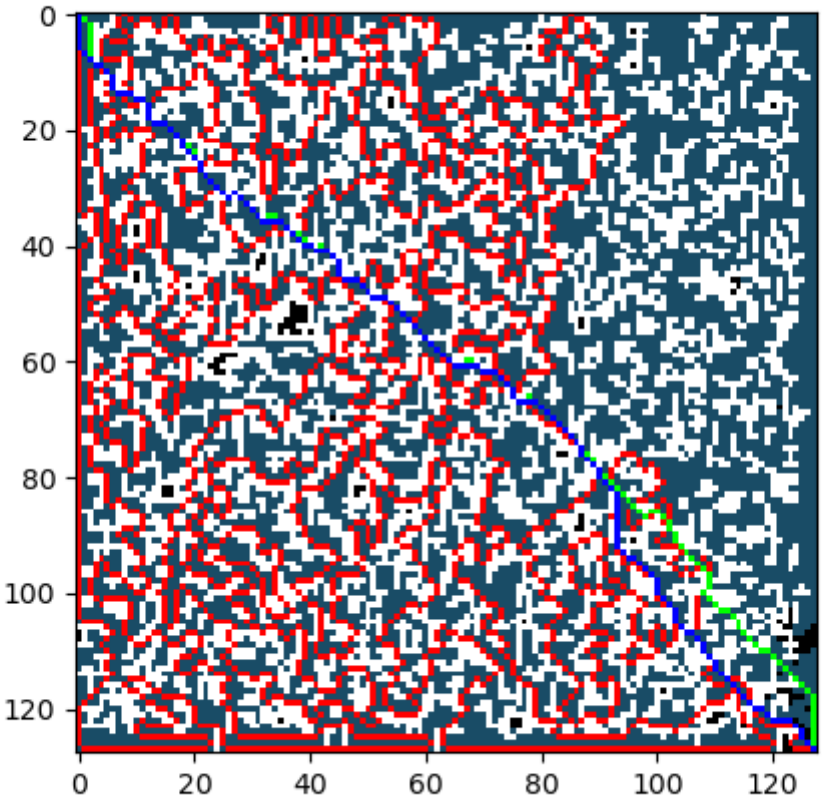
20% obstacles



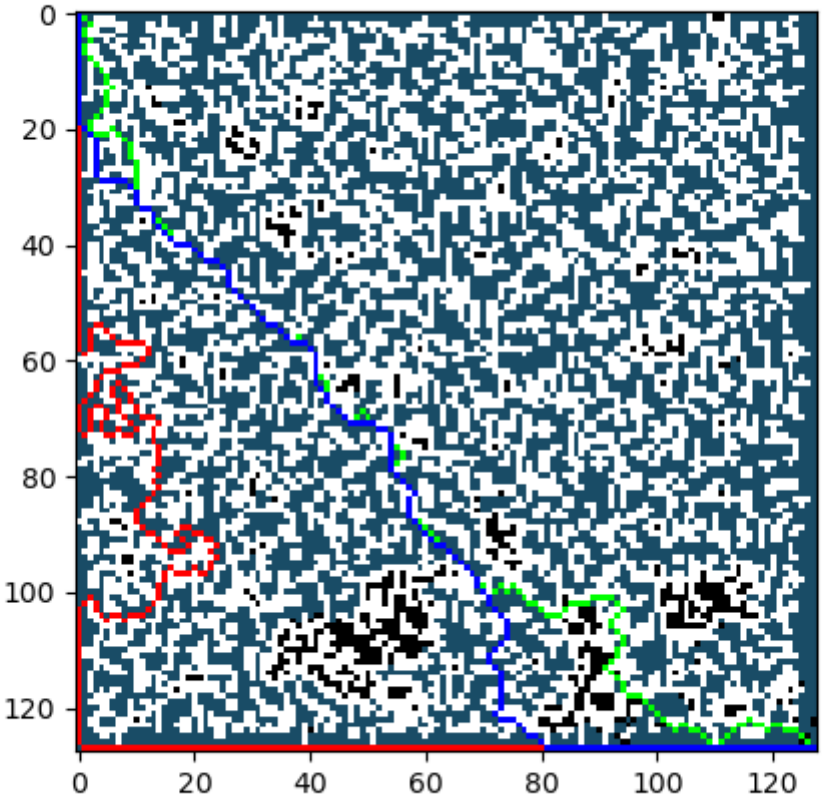
30% obstacles



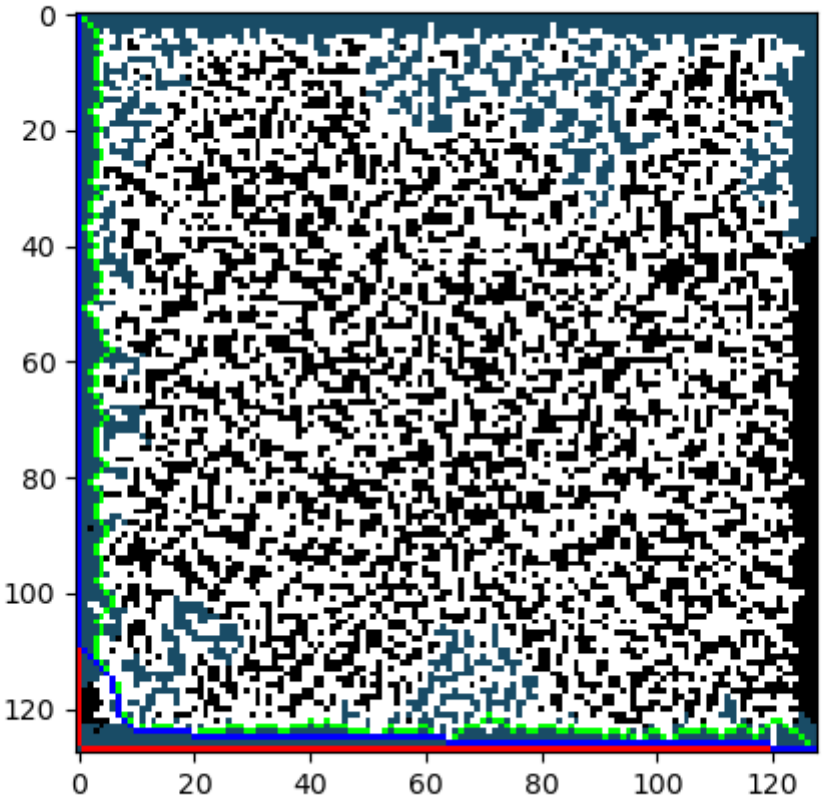
40% obstacles



50% obstacles



60% obstacles (edges are kept unblocked)



70% obstacles (edges are kept unblocked)