# INFO - 550
# Artificial Intelligence

*Dr Tyler Millhouse*

# Project Report for
# Letter of Recommendation

By

**Abhay Kumara Sri Krishna Nandiraju**

**SID: 23888722**

## Introduction:

The project aims to compare the performance of a random agent, Uniform Cost Search(UCS) algorithm and A* algorithm with two different heuristics on the Grid Search problem. Uniform Cost Search does not use a heuristic whereas A* is a heuristic based algorithm and the random agent picks the next valid move randomly with all the valid moves being equally-likely. The project also aims to find the effect of different heuristics while using A* in solving the grid-search problem and explores the advantages of sophisticated heuristics in the context of efficiency.

## Algorithms/Agent:

1. Random Agent: The random agent does not use any specific steps in picking the next move. It chooses the next valid move out of all the possible valid moves with all the valid moves being equally-likely.

2. Uniform Cost Search(UCS): This algorithm expands the node solely based on the lowest path cost from the start node to the node to be visited and doesn't use any heuristic. The cost function is denoted as $g(n)$. This is an uninformed search algorithm that uses no additional information about states beyond that provided in the problem definition. In simple words, it picks the next node that has the lowest $g(n)$.

3. A*: This algorithm expands the node by combining the path cost to reach the node($g(n)$) and the cheapest path cost to get from the node to the goal node($h(n)$). 'h(n)' is called the heuristic function and this is an informed search algorithm since the heuristic function uses additional knowledge about the states beyond that provided in the problem definition. In simple words, it picks the next node that has the lowest $g(n) + h(n)$ and $f(n)$ is used to represent the expression $g(n) + h(n)$ in short.

## Heuristics and Cost functions:

The cost function(g(n)) used for both the algorithms is the straight-line which is calculated using the Euclidean distance formula. For the A* algorithm, the euclidean distance and octile distance are used as heuristics. The detailed explanation of each heuristic is given below:

a. Euclidean distance: It is the shortest straight line distance between the current position and the goal. This heuristic is admissible and consistent since the grid search problem allows diagonal movement too.

   $h(n) = \sqrt{\left(x_{goal} - x_n\right)^2 + \left(y_{goal} - y_n\right)^2}$

b. Octile distance: This distance takes into account the costs of diagonal and non-diagonal movements. The cost of diagonal movement is √2 times the cost of non-diagonal movement. This heuristic is admissible and consistent too.

$h(n) = (D_2 - D_1) * min(|x_{goal} - x_n|, |y_{goal} - y_n|) + D_1 * max(|x_{goal} - x_n|, |y_{goal} - y_n|)$,
where $D_2$ (cost of diagonal move) = $\sqrt{2}D_1$ (cost of non-diagonal move). ($D_1$== 1)

## Grid-Search Problem Setting:

A 32x32 size grid is initialized such that the start node is set as the top-left node/cell and the goal node is set as the bottom-right cell/node. Then the complexity of the grid is quantified by a difficulty level which randomly makes a certain percentage of the nodes inactive or invalid. The complexity of the grid can be thought of as a random obstacle generation. At difficulty level 0, all the nodes are active/valid whereas at difficulty level 90, only a mere 10 percent of the nodes are active/valid. As the difficulty level increases in steps of 10, the number of paths to reach the goal node decreases.

## Implementation Details:

a. The code is implemented following the principles of object oriented programming so that it is error free and easy to update in the future.
b. The random agent is implemented in a RandomAgent class in the random_agent.py file and it randomly selects valid moves from 8 possible directions(includes diagonals). It performs no path optimization.
c. The Uniform Cost Search algorithm is implemented in the UCSAgentGrid class in the ucs.py file and uses a priority-queue called frontier to track the nodes to be explored based on the path cost and uses a visited set to track the already visited nodes. It also keeps track of the cost and the path followed during the process along with the number of nodes expanded before reaching the goal node. A function called reconstruct_path() is used to retrace the path from the initial node to the goal node. The UCS agent computes the cost using the straight line distance. It calculates the cost between the initial node and the next node to be visited by keeping track of the step-costs and adding them to find the total cost.
d. The A* algorithm is implemented in the AStarAgentGrid class in the astar.py file and uses a priority-queue called frontier to track the nodes to be explored based on the sum of path cost and heuristic value,and uses a visited set to track the already visited nodes. It also keeps track of the cost and the path followed during the process along with the number of nodes expanded before reaching the goal node. It uses the straight line distance to find the cost and a heuristic function that calculates the distance between the goal node and the node to be visited next. Euclidean distance and Octile distance

heuristics are used to compute the heuristic values. The euclidean distance is a simple heuristic whereas the octile distance is a sophisticated one which expands the nodes in a more optimized way.

e. Performance metrics such as number of nodes expanded and cost are compared over different levels of difficulties.
f. First a grid is generated and the grid configuration is saved only when it has a solution. The generated grid is verified using the A* algorithm with the euclidean heuristic to confirm that it has a solution.
g. In the above mentioned way, 100 solvable grids are generated for each difficulty level from 0 to 90.
h. Then the 100 solvable grids for each difficulty level are solved using each algorithm and metrics like number of steps, number of nodes expanded, path cost and number of times path found are calculated and stored in JSON files. Finally the average values are also computed for each difficulty level over the 100 grids.
i. These JSON files are used to plot the visualizations and compare the respective algorithm performances.

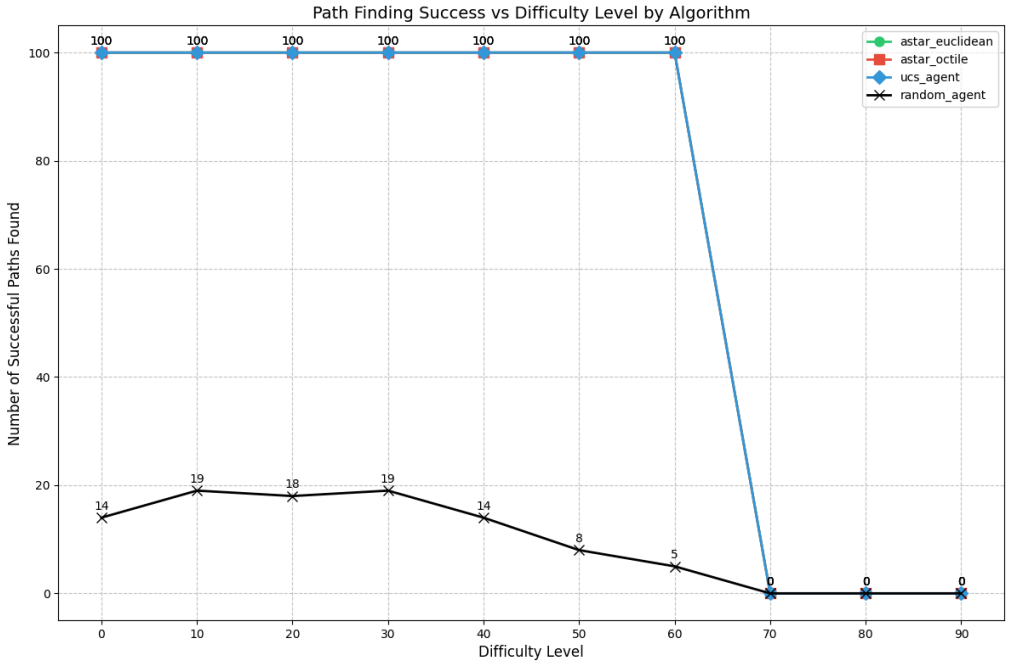
## Problems during implementation:

During the initial implementations of the UCS and A* algorithms I did not consider different costs for cardinal and diagonal moves which lead to the number of steps and total cost being identical. Later, I corrected this by incorporating cost 1 for non-diagonal moves and $\sqrt{2}$ for diagonal moves. Moreover, I was verifying whether the grid configuration was solvable, but I forgot to save the grid state so it could be passed to the respective algorithm for solving. I resolved this problem by adding preset_grid, preset_goal and preset_initial in the GridSearch class located in the grid_search.py file. I am familiar with saving results in a pandas dataframe and later using it for plotting visualizations but the project instructions included that pandas cannot be used so, I found a way to solve this by storing the result values in a json file and use them later for visualization purposes.

## Hypotheses:

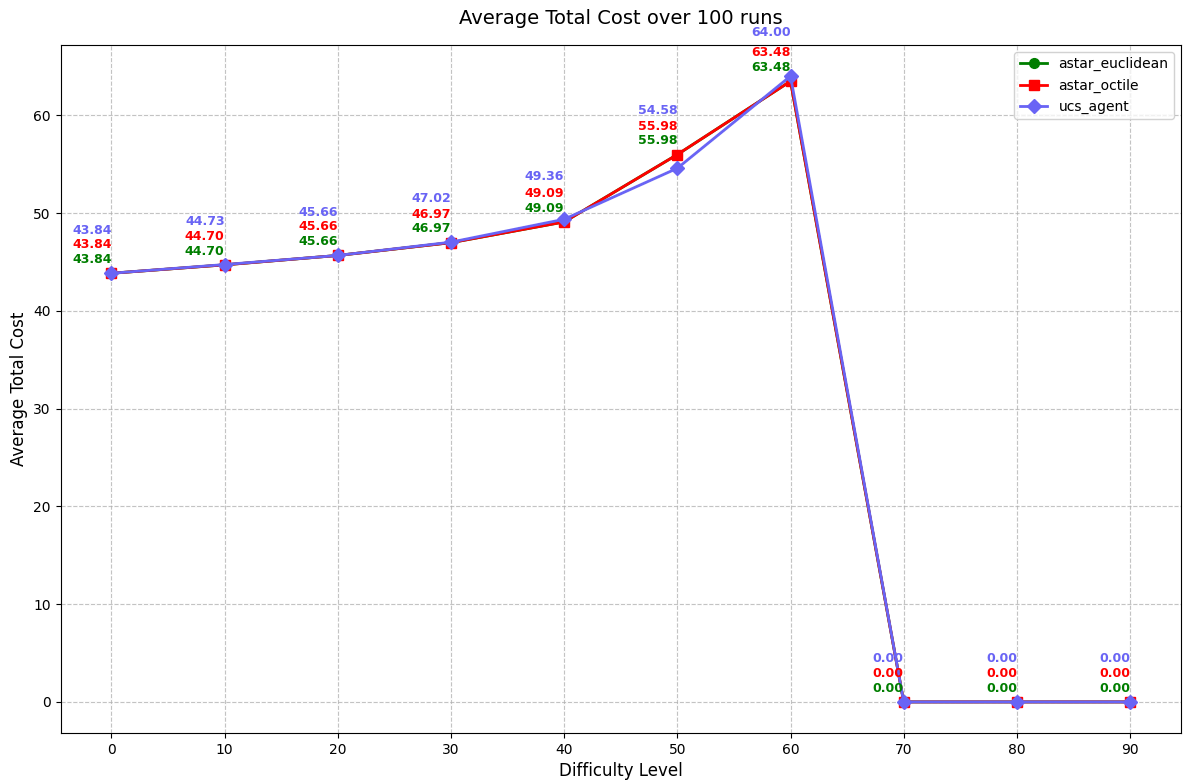
a. Random agent may not find a path to reach the goal node even at lower difficulty levels in a fixed number of moves.
b. The number of nodes expanded by the A* algorithm is less compared to those of UCS algorithm.
c. A* with euclidean heuristic expands more number of nodes compared to A* with octile distance heuristic.

## Plots and Conclusions:



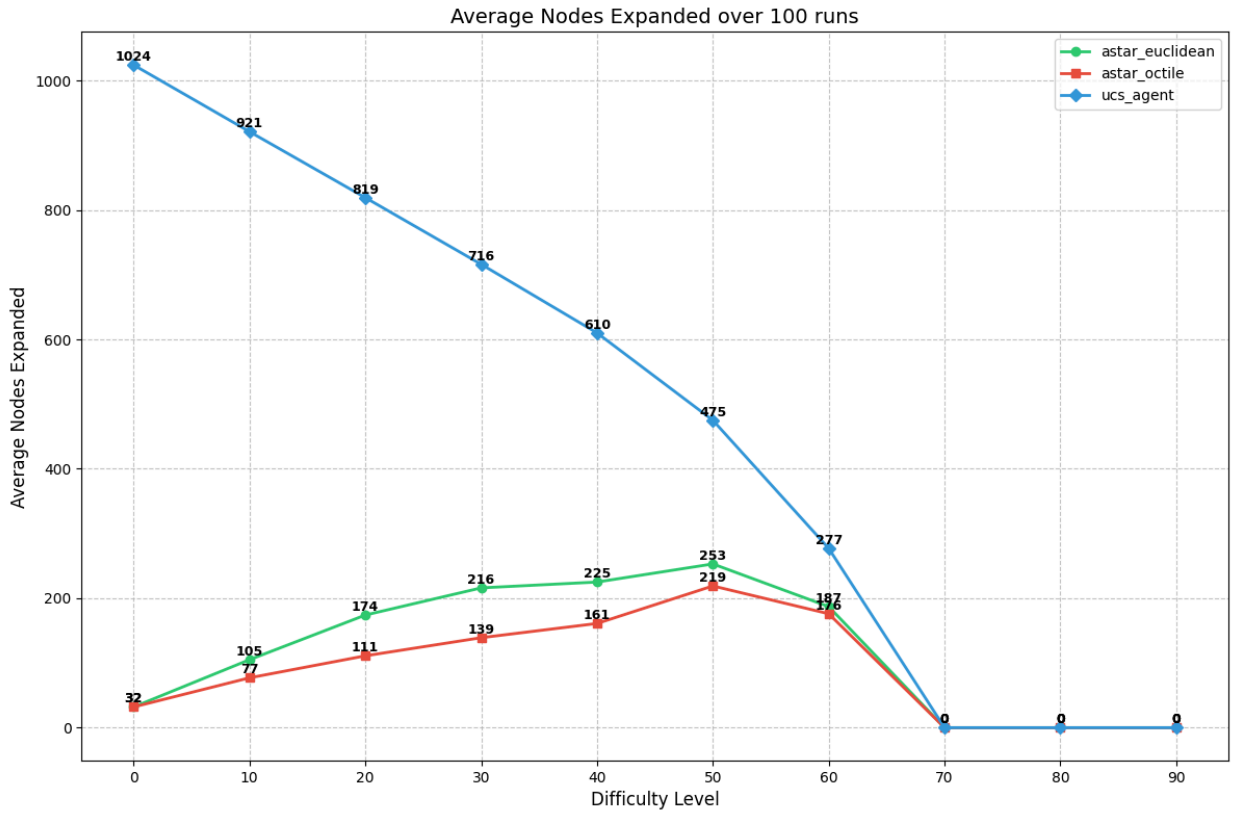

The above plot illustrates the number of times an algorithm found a path to the goal node over 100 solvable grids at each difficulty level. From this plot, we can conclude that the random agent does not always find a path to the goal node at lower difficulty levels too. The UCS, A*-Euclidean and A*-Octile algorithms always find a path to the goal node if the grid is solvable at that difficulty level. Moreover, from the graph we can see that the values match for UCS, A*-Euclidean and A*-octile algorithms.

In addition to this, for difficulty levels more than 60 no algorithm finds a path to the goal node(bottom right) from the initial node(top-left). This indicates that the generated grids of size 32 by 32 have no solvable path to reach the goal node for difficulty levels above 60.




The above graph depicts the average total cost over 100 solvable grids at each difficulty level. At each difficulty level, 100 solvable grids are generated and the mean of the costs over 100 grids are computed to find the average total cost at each difficulty level. As expected the total cost to reach the goal node is same for A*-euclidean and A*-octile since A* algorithm uses the straight line distance to compute the cost between the initial node and the node to be visited next.

The UCS algorithm has a slightly more cost compared to the A* algorithm since it expands the nodes solely based on the cost whereas A* expands nodes based on the sum of the cost and the heuristic value. A* is more goal aware but UCS is not, resulting in exploration of redundant paths by UCS. For difficulty levels beyond 60 there are no solvable grids that have a valid path to reach the goal node. Hence, the values are zero.




The above graph illustrates the average number of nodes expanded over 100 solvable grids for each difficulty level. From the plot, we can see that the UCS algorithm explores more number of nodes compared to A* at a given difficulty level. It is due to UCS being unaware about the goal and exploring redundant paths before finding a path to the goal node.

Moreover, A* with octile heuristic expands less number of nodes compared to A* with euclidean heuristic at a given difficulty level. This is because euclidean heuristic underestimates the cost of diagonal movements. Even though both heuristics are admissible and consistent, octile heuristic provides a closer estimate or tighter bound on the true cost compared to the euclidean heuristic in grid problems where diagonal movements are allowed.