Foundation of Intelligent System – Project 1 Write Up

Sanyukta Sanjay Kate (ssk8153) Amit Magar (ajm6745)

Problem Definition

In this project we have implemented the A* search algorithm to find a path in the maze along with the rolling dice. In order to find an optimal path in the maze, we make use of two cost functions – the node cost (g(n) which is the cost from the start to the current node) and the heuristic (h(n) which is the cost from the current node to the goal). The node selection for the path is determined by g(n)+h(n) which is the total cost (f(n)). We have used three heuristic functions – Euclidean, Manhattan, and Diagonal. We use these three heuristic functions in order to find the efficiency of the algorithm, but the A* will always give an optimal as well as a complete path from the start state to the goal state. There are various restrictions of the dice while finding the path from start to goal. When we start from the start location, we have to keep our dice top as 1, we cannot walk over obstacles, we cannot consider a state when 6 appears on top of the dice, cannot consider the goal state when 6 comes over the top of the dice, and further, we can consider the goal state when reached only when the dice stop has the top face as 1.

State Space

The state space for the given maze/puzzle is defined by the total number of possible states which one can go to. As there are obstacles in the maze and the length and width of the maze is defined by the "number of rows" and the "number of columns", the total number of states for the maze is (NumberOfRows + NumberOfColumns - NumberOfObstacles). As a dice is used to select a state and as 6 is not allowed to come of the top face at anytime, we consider only 5 faces of the dice at anytime. Since, we only travel in 4 directions (north, south, east, west), the total number of dice states is 4*5. The complete state space for the problem statement is (4*5)(NumberOfRows + NumberOfColumns - NumberOfObstacles).

Initial State

The initial state of the state space is the start state on the maze which is represented by the letter "S" on the maze. When the dice is placed on the start state, then the start state's dice top value should be 1, east/right value of the dice would be 3 and the north/up value should be 2.

State Transition

The moves or rather actions can be determined by moving the maze in four different directions (north, south, east, west), but only when there are no obstacles and when the dice top is not 6. Every movement from one location to another immediate location is an action which is determined by the top face of the dice. When we start moving from one square to its neighboring square, we roll the dice in its respective direction (straight (north), back(south), right(east), left(west)). If 6 does not appear on top then consider the neighboring position. The further selection of the path is controlled by the lowest path cost function (f(n)). We continue to move in the maze till we reach the goal state with the dice top as 1.

Goal State

We keep moving in the maze till we stumble upon the "G" value on the maze. But, we consider this goal state only when the dice top value has "1" on it. If we come across the goal at any point and the dice top

value is not 1, then we ignore this goal state and continue with the path finding till we reach the goal again but with the dice top value as 1. The path acquired for all there heuristics will be the same.

Path Costs

The path cost is determined using the two costs, the first being the "node cost (g(n))" from the start node to the current node (every node/sate with its neighboring node/state is 1, this is similar to Dijstra's) and the second one being the cost from the goal state to the current state which is calculated using the three different heuristics (h(n)). Therefore, the total cost function would be f(n) = g(n)+h(n). G(n) would be the g(n) cost of the parent node + 1. Initially, g(n) for the start node is 0. One may take a bigger/costlier path depending upon the maze restrictions and the dice restrictions.

Heuristics

We have implemented three heuristic functions, which are both consistent and admissible for finding the path using the A* algorithm. The following heuristics are explained below and why are they considered as admissible and consistent.

Euclidean Distance

The Euclidean distance is measured using the formula sqrt((x1-x2)^2+(y1-y2)^2), where, x1 and y1 is the current node coordinates and x2 and y2 is the goal node's coordinates. This is the straight line distance (hypotenuse of the a right angle'd triangle) from the current node to the goal node. By taking a straight line distance from the current state to the goal state we can say that this particular heuristic does not over-estimate the path cost from the current node to the goal node. Hence, this heuristic is admissible. The node cost for every node would be the parent's node cost plus 1. As all the neighbors of a node would have the same g(n), what would matter the most would be h(n). We notice that the total path cost would always be greater than the heuristic's value (here, is the Euclidean distance), as g(n) is added to h(n), therefore we can say that this heuristic is consistent too.

Manhattan Distance

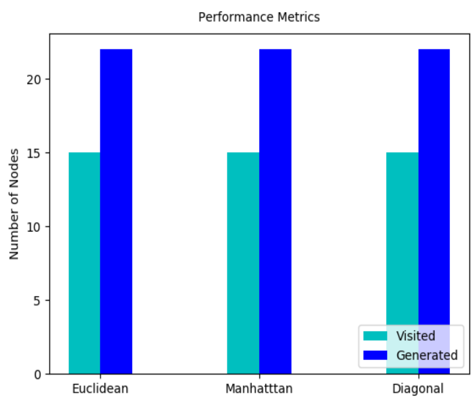
The Manhattan distance is determined by the formula abs(x1-x2)+abs(y1-y2) which gives the absolute distance from the current node to the goal node. We can say that the manhattan distance just considers the empty horizontal and vertical spots/squares on the maze in order to move. Thus, this is the shortest distance between the empty blocks in the maze. While calculating the manhattan distance, we do not consider the fact that there are obstacles. But, as we are moving in the maze, there are obstacles and hence, we may get the actual manhattan distance and get an optimized cost from the current node to the goal node. Hence, we can say that the manhattan distance is admissible. As there are multiple dice constraints, we would always get a path which may be longer than the original path (without constraints), hence, this is also a consistent heuristic.

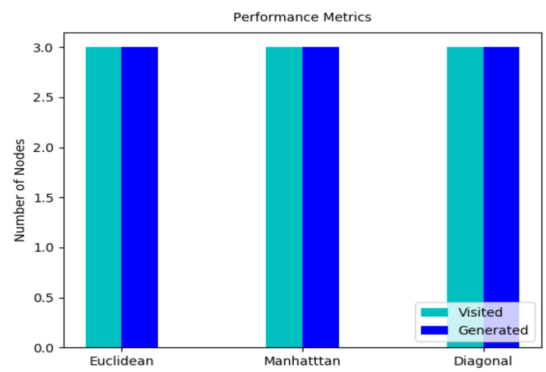
Diagonal Distance

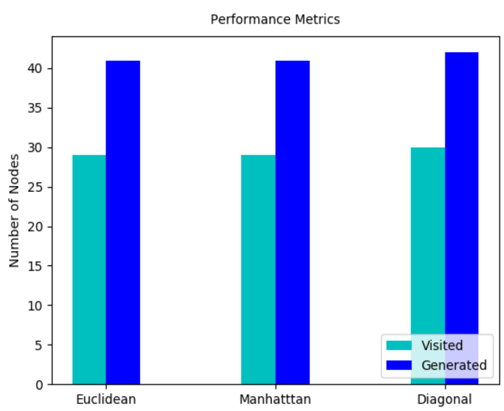
The diagonal distance is calculated using the formula max(abs(x2-x1), abs(y2-y1)), which means that this distance counts the horizontal, vertical and the diagonal squares. This distance is a combination of manhattan and the euclidean distance and as both of them are admissible, we also get diagonal distance

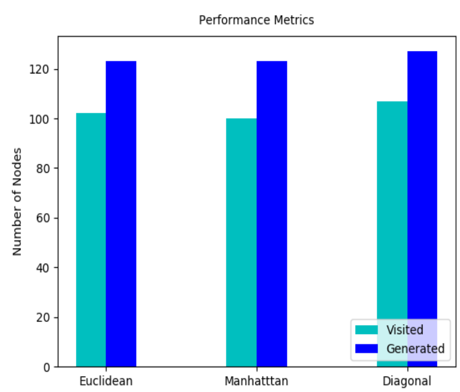
as admissible. As the diagonal empty blocks are also counted for calculating the cost, which is the euclidean cost from the current node to the goal node and the horizontal and the vertical blocks which is the manhattan distance, we always get a cost which is better than manhattan and euclidean. Hence, diagonal is considered as admissible. Further, we get diagonal distance better than the manhattan and euclidean distance because we consider no obstacles.

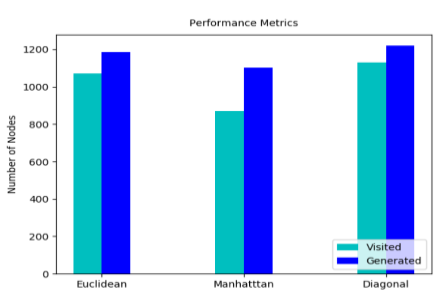
Performance Metrics











Puzzle	No. Of Paths	Visited	generated	No. Of paths	Visited	generated	No. of paths	Visited	genertaed
1.	7	15	22	7	15	22	7	15	22

2.	**-1**	**3**	**3**	**-1**	**3**	**3**	**-1**	**3**	**3**
3.	**13**	**29**	**41**	**13**	**41**	**29**	**13**	**42**	**30**
4.	**22**	**102**	**123**	**22**	**100**	**123**	**22**	**107**	**128**
5.	**27**	**1072**	**1187**	**27**	**871**	**1104**	**27**	**1131**	**1220**

In the performance metrics we observe that the manhattan, euclidean and the diagonal distance perform with the same efficiency for the 1st puzzle which is the smallest puzzle given. We can see from the given information, that the diagonal fails to perform better than euclidean and manhattan in the puzzles(4 and 5), this is because we have ignored the diagonal blocks which may have given a better path cost. We only consider the empty squares which are to the north, south east and west. Plus, there are obstacles and dice constraints due to which we may leave out the path costs which would give a better path or shortest path.

Discussion

As the implementation of the project had multiple constraints, understanding the state space and the presentation of the problem was the first step in implementing this project. Before starting to code, we understood, how the dice should work, that is the movement of the dice along the maze and its face/top value and the entire dice configuration importance. We then had to understand how the A* algorithm works along with the dice constraints and the obstacles. Understanding how different heuristics work was the one of the essential parts of the project ad hence, three different heuristics were implemented just in order to see how efficient they were in finding the path. We observe that in the end, the all three heuristics give the same path which is the most optimal path with the constraints given.

We designed our project into three different classes. We worked on the representation of the state/node and then worked on the frontier. Dice was the other class which had the entire dice configuration and the rolling movements of the dice. Every node had different attributes such as the row, column, dice (configuration), pathCost, gCost, hCost.

After implementation, came the testing part, where we tested our code on different puzzles. While observing the observations, we saw that the smaller puzzle(puzzle1) gave the same results (visited, generated nodes) as the same. This shows us that the efficiency of all the three heuristics was the same on small puzzles. Later, we see that the when big puzzles are used, we get different number of visited and generated nodes. This is because, every heuristic used calculates the cost in different ways. Also, our break a tie method, for the same heuristic cost is considered in such a way, that the node which comes in last with the same Heuristic cost is considered first (FILO). This may cause few nodes which were generated, visited to vary a little bit. We also observed that when we performed break the tie using stack, the performance was very good and with FIFO it was not very efficient.

Ideally, diagonal heuristic should work the best, but it does not as the diagonal blocks are ignored which may have a better cost. Hence, we expected diagonal distance to work the best but it does not due to the maze direction considerations. Hence, we observe that the manhattan and the euclidean perform the best. This we observed by looking at the nodes generated and visited for the three different heuristics.