Intelligent Systems Project 1

Ankit Bhankharia (atb5880)

Uday Wadhone (uw1919)

1. Problem Definition

Objective:

To solve rolling-die maze using A-star search algorithm. A die will roll along its edges through a grid until goal location is reached. The initial state of the die is at the starting location S with 1 facing up (visible), 2 facing north, and 3 facing east. Also, during the duration of the maze, 6 should never be face up on the die and number 1 must be on top when die reaches goal location in the maze.

State Space:

Considering n to be the width, m to be the height and w as number of walls, the total number of possible states would be (*n*m – w).* The die having six sides and for possible orientations, by exclusion of having the sixth side not allowed to be face up, the number of states for the die would be (6 - 1)*4. Combining the two we get a total state space of 20(*n*m - w*).

Problem Representation:

Initial State:	Initial state is the die on the start point S with 1 on the die facing up, 2 facing north and 3 facing east.
Actions:	An action can involve rolling north, south, east or west and depending on orientation of dice, different combinations of these will be selected
Transition Cost:	A transition consists of a state and action which give a resulting state. The transition cost is measured by number of moves.
Goal State:	The goal space is defined as G block on the maze with the constraint that the die rests on this block with 1 facing up.

2. Heuristics

As we are using A* algorithm for this problem, we need to determine evaluation of the heuristic function h(n). We have implemented three different heuristic functions as defined below:

Euclidean Distance

This heuristic uses the Euclidean distance from the current state to the goal state. The admissibility of this heuristic is supported because it calculates the shortest distance from the current state to the goal state. There is no overestimation in this as there is no other shorter distance between two points. The heuristic function for Euclidean distance can be calculated as:

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

Where node is the current position(x, y) of the die on the maze and goal(x,y) is the target node.

Manhattan Distance

The second heuristic is based on the Manhattan distance, which by definition is the sum of absolute differences of Cartesian coordinates. To calculate this heuristic, we work on the assumption that there are no obstacles. It calculates the Manhattan distance from current state to the goal state. This heuristic is admissible as it calculates the shortest amount of blocks from the current state to the goal state. Thus it is not possible to overestimate because there is no path through these blocks which is shorter. Consider two points P (p1, p2) and Q (q1, q2). The Manhattan distance will be calculated as follows:

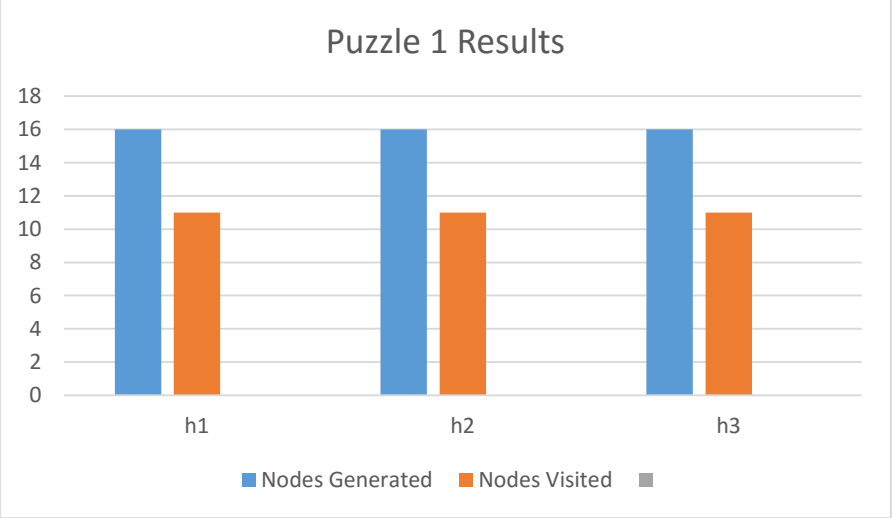
$$|p_1 - q_1| + |p_2 - q_2|.$$

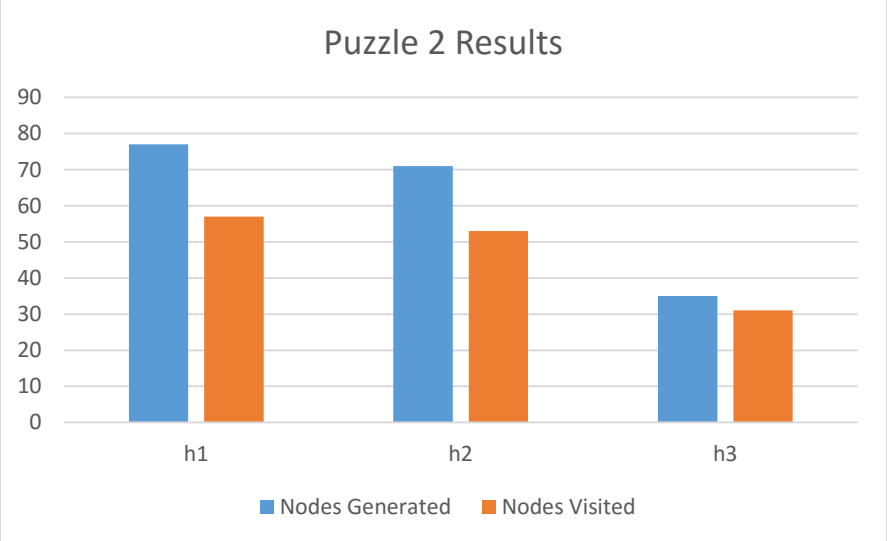
Improved Manhattan Distance

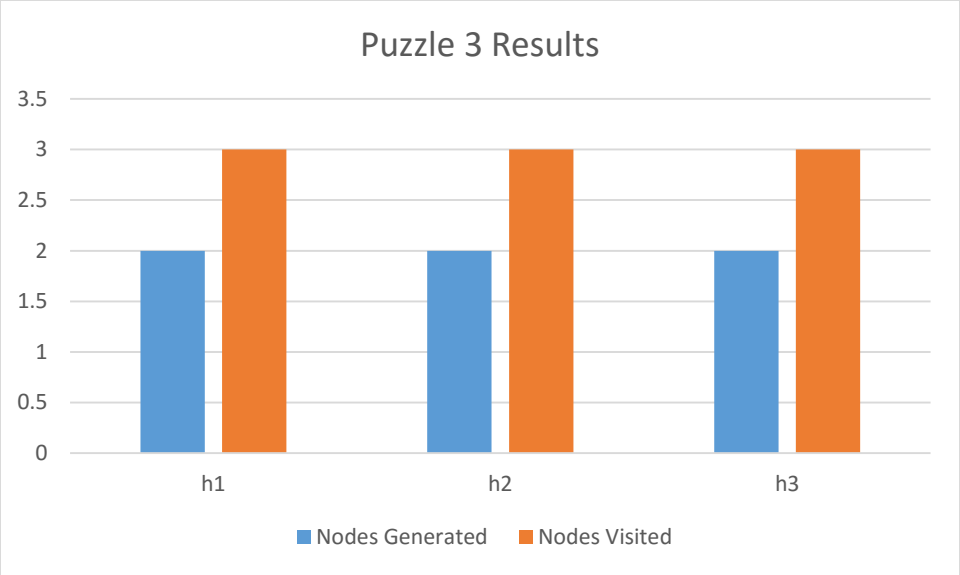
The third heuristic could be seen as an improved version of the Manhattan distance rule. This heuristic actually considers and takes advantage of how the die rolls. We have added a couple of conditions which uses the fact that the die when is on its side, that is has 2, 3, 4 or 5 facing up, it can roll as many spaces as it needs to in a straight line. The first condition states that if a die is on the goal block but the side facing up is not 1, then it would take four moves to resolve this and get 1 on top. Thus, this operation will cost 4. The second condition states that if the die is not on the goal block and the distance between the die and goal is greater than one. Then in the case of the die not being on its side, the cost to reach the goal would be calculated as the Manhattan distance plus four transitions to reach the goal. In case of these two conditions not being satisfied, the cost will be calculated as regular Manhattan distance evaluation as described in the second heuristic. Using these conditions, either the exact distance to the goal or just short of exact distance is calculated. Also, none of the conditions can overestimate the actual distance between the current and target node.

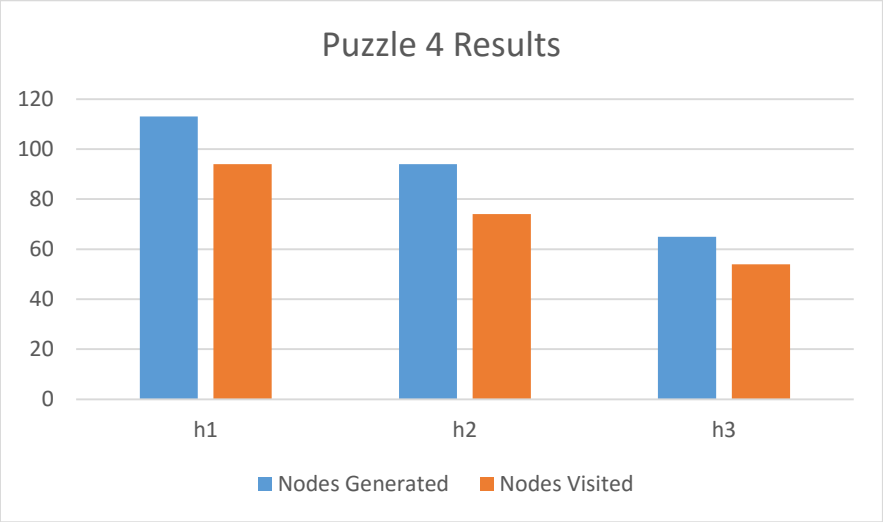
3. Performance Metrics

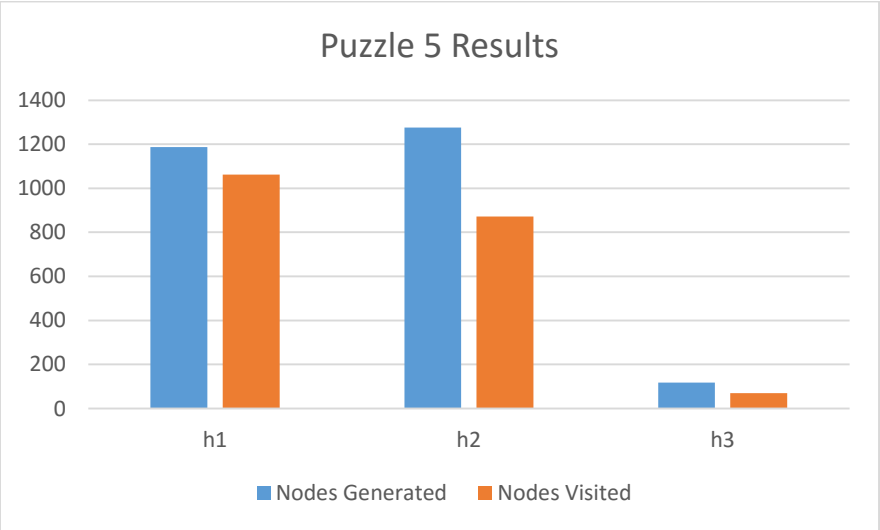
	Euclidean Distance		Manhattan Distance			
States	Generated	Visited	Generated	Visited	Generated	Visited
Puzzle 1	16	11	16	11	16	11
Puzzle 2	77	57	71	53	35	31
Puzzle 3	2	3	2	3	2	3
Puzzle 4	113	94	94	74	65	54
Puzzle 5	1187	1062	1276	872	117	70






Puzzle 3 Results
3.5
3
2.5
2
1.5
1
0.5
0
h1
h2
h3
Nodes Generated
Nodes Visited


Puzzle 4 Results
120
100
80
60
40
20
0
h1
h2
h3
Nodes Generated
Nodes Visited


Puzzle 5 Results
1400
1200
1000
800
600
400
200
0
h1
h2
h3
Nodes Generated
Nodes Visited

4. Discussion

When using the heuristic Euclidean distance, we always calculate the absolute shortest path which will always be less than the actual cost of the path from current state to the goal state. As we can see from the performance metrics, Euclidean has the highest number of nodes visited and generated. Euclidean distance will give us the shortest paths but will still take A* longer to run. Using Manhattan distance for A* can sometimes calculate values near the actual cost. It performs really well when the die is on its side and the distance to travel is one. This might result to be an underestimation in case of Manhattan distance heuristic.

The third heuristic can be observed to be a big improvement over the second heuristic. This is due to the fact that the heuristic helps us take advantage of die orientation and handle many scenarios where the die is on its side. For simple mazes, all heuristics perform mostly the same, but the performance of third heuristic over others can be seen in complex mazes.

In the third maze, number of nodes generated is less than number of nodes visited as there is no solution for this maze. The first node is visited and not expanded further, while only two nodes are expanded before the die will be blocked in the maze.

We can see that heuristic function is an important part of the A* algorithm and that the performance of the algorithm is hugely affected by it.