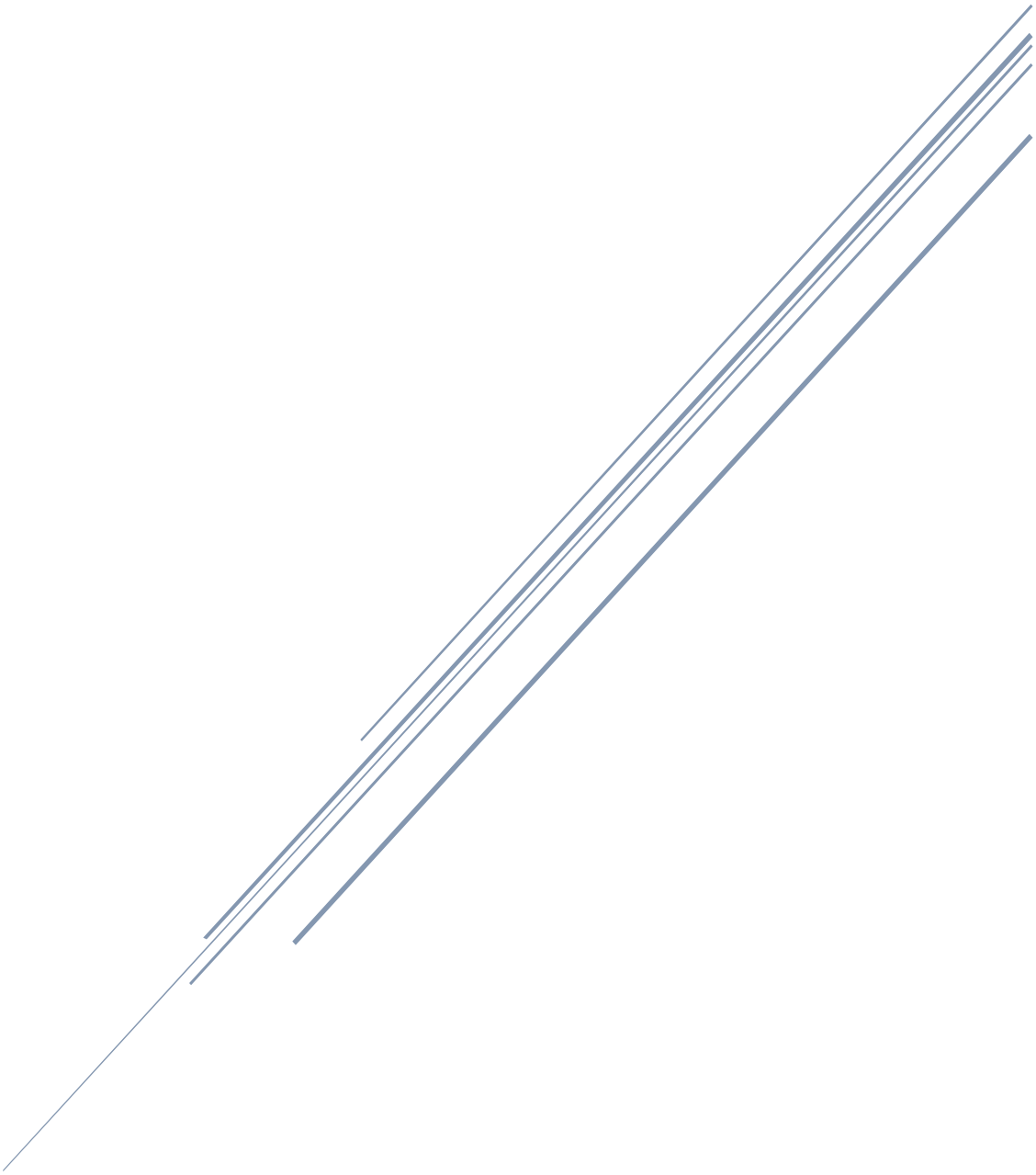
AI ASSIGNMENT 2 – A*

Harsh Jha – 2311AI14

Kumar Satyam – 2311AI16

Supriya Maji – 2311AI20



Understanding for A* –

For A* algorithm, we are trying to find a cost of each node, which will give us a better understanding of which path to expand as it will lead to a smaller number of nodes explored.

To calculate cost, we are using –

f(n) = g(n) + h(n)

where, g(n) = cost till that node

h(n) = heuristic cost

With the heuristic value we are trying to estimate which node is closer to goal.

A good heuristic value must be **admissible**, i.e., **h(n) <= h*(n)**,

where h(n) = our estimated heuristic

and h*(n) = optimal cost

Time and Space complexity of A* in the worst case can be exponential.

However, time complexity, mainly depends on what heuristic value is used and data structure, used to maintain open and closed stated.

In worst case Time complexity and space complexity is - **O(b^d)**

Where b = branching factor

d = depth of optimal solution

NOTE: For Below test these are the heuristic functions

1. h(n) = 0
2. h(n) = misplaced tile
3. h(n) = Manhattan distance
4. h(n) = suboptimal result, below code is used to calculate this, where costTillNow contains g(n) for that node

```
# to calculate incorrect heuristic
1 usage
def calculateIncorrectCosts(node):
    cost = node.costTillNow + ManhattanDistanceHeuristic.manhattanDistance(node) + random.randint(10, 100)
    return cost
```

Observation –

Goal Grid Configuration-

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

1. First Test-

Initial Grid Configuration-

[2, 5, 7]

[3, 0, 4]

[8, 6, 1]

Nodes Searched and Time-

1. Manhattan Distance -
   Nodes Searched - 98, Time Taken - 0.003
2. Misplaced Tile-
   Nodes Searched - 274, Time Taken - 0.009
3. H(n) = 0 -
   Nodes Searched - 2237, Time Taken - 0.062
4. Suboptimal Heuristic-
   Nodes Searched - 81982, Time Taken - 3.206

Number of elements matched between -

1. h(n) = Manhattan distance and h(n) = Misplaced tile - 97
2. h(n) = Manhattan distance and h(n) = 0 - 98
3. h(n) = Misplaced and h(n) = 0 - 273
4. h(n) = Manhattan distance and h(n) = sub-optimal - 98

2. Second Test –

Initial Grid Configuration-

[0, 4, 6]
[1, 8, 3]
[5, 7, 2]

Nodes Searched and Time-

1. Manhattan Distance -
   Nodes Searched - 895, Time Taken - 0.032
2. Misplaced Tile-
   Nodes Searched - 13905, Time Taken - 0.579
3. H(n) = 0 -
   Nodes Searched - 112916, Time Taken – 5.177
4. Suboptimal Heuristic-
   Nodes Searched - 180597, Time Taken - 7.316

Number of elements matched between -

1. h(n) = Manhattan distance and h(n) = Misplaced tile – 889
2. h(n) = Manhattan distance and h(n) = 0 – 895
3. h(n) = Misplaced and h(n) = 0 – 13905
4. h(n) = Manhattan distance and h(n) = sub-optimal - 895

3. Third Test –

Initial Grid Configuration-

Note: For below configuration no solution was found

[5, 2, 6]
[0, 7, 8]
[4, 1, 3]

Nodes Searched and Time-

1. Manhattan Distance -
   Nodes Searched - 181440, Time Taken - 9.216
2. Misplaced Tile-

Nodes Searched - 181440, Time Taken - 9.920

3. H(n) = 0 -
   Nodes Searched - 181440, Time Taken – 9.376
4. Suboptimal Heuristic-
   Nodes Searched - 181440, Time Taken - 7.824

Number of elements matched between -

1. h(n) = Manhattan distance and h(n) = Misplaced tile – 181440
2. h(n) = Manhattan distance and h(n) = 0 – 181440
3. h(n) = Misplaced and h(n) = 0 – 181440
4. h(n) = Manhattan distance and h(n) = sub-optimal - 181440

4. Fourth Test –

Initial Grid Configuration-

[2, 5, 7]
[8, 1, 4]
[6, 0, 3]

Nodes Searched and Time-

1. Manhattan Distance -
   Nodes Searched - 8717, Time Taken - 0.406
2. Misplaced Tile-
   Nodes Searched - 61985, Time Taken - 3.050
3. H(n) = 0 -
   Nodes Searched - 149949, Time Taken – 6.177
4. Suboptimal Heuristic-
   Nodes Searched - 142603, Time Taken - 6.316

Number of elements matched between -

1. h(n) = Manhattan distance and h(n) = Misplaced tile – 8668
2. h(n) = Manhattan distance and h(n) = 0 – 8712
3. h(n) = Misplaced and h(n) = 0 – 61977
4. h(n) = Manhattan distance and h(n) = sub-optimal - 7992

Solutions of provided Questions –

1. Based on above observations, we can conclude that a better heuristic expands a smaller number of states and we can say h(manhattanDistance) is better heuristic that h(misplacedTile)
   i.e., h(manhattanDistance) >= h(misplacedTile) >= h(0) >= h(subOptimalHeuristic)\
   From test 1, states explored-
   1. Manhattan Distance -> 98
   2. Misplaced Tile -> 274
   3. H(n) = 0 -> 2237
   4. Suboptimal Heuristic- 81982

2. Based on above observations, we can say every node explored by better heuristic is also explored by suboptimal heuristic
   From Test 1-
   Manhattan Distance - Nodes Searched – 98
   Misplaced Tile- Nodes Searched – 274
   H(n) = 0 - Nodes Searched - 2237
   Suboptimal Heuristic- Nodes Searched - 81982
   Number of elements matched between -
   1. h(n) = Manhattan distance and h(n) = Misplaced tile -> 97
   2. h(n) = Manhattan distance and h(n) = 0 -> 98
   3. h(n) = Misplaced and h(n) = 0 -> 273
   4. h(n) = Manhattan distance and h(n) = sub-optimal -> 98

   For this we can see that nodes matched in a better heuristic is almost same as number of nodes generated using that heuristic.
   NOTE: To check if nodes are same or not, we are comparing all the nodes generated in one heuristic with all nodes generated in other list.

3. **When h(n) = 0**

   If h(n) is equal to 0 for all states n, this essentially means that the heuristic estimate for every state is that it requires no additional cost to reach the goal state. In other words, the heuristic is assuming that every state is already at the goal state. This situation effectively removes any guidance from the heuristic in terms of choosing the best paths for reaching the goal state, as it doesn't provide any differentiation between states.
   With h(n)=0 for all states, the monotonic restriction h(n) ≤ cost(n,m) + h(m) simplifies to: **0 ≤ cost(n,m) + 0**

   Therefore, this heuristic trivially satisfied monotone restriction because no matter the cost of moving from state n to m, the inequality holds. It doesn't provide any useful information or constraint on the search process because h(n)=0 always.

   When h(n) = Misplaced Tile

   Number of tiles displaced from their destined position
   For this heuristic, as the puzzle gets closer to the goal state, the number of tiles displaced from their destined positions can only decrease or remain the same, since we are trying to solve the puzzle by minimizing the number of misplaced tiles. Therefore, **this heuristic follows the monotone restriction**.

   When h3(n) = Sum of the Manhattan distance of each tile from the goal position

   The Manhattan distance heuristic calculates the sum of the distances each tile needs to travel to reach its goal position. As the puzzle gets closer to the goal state, the Manhattan distances for tiles can only decrease or remain the same. This is because tiles are moved closer to their goal positions, reducing

the distance they need to travel. Therefore, this heuristic also **follows the monotone restriction**.

When h(n): Sub-Optimal

If h(n) is a suboptimal heuristic, it means that the heuristic estimates it provides are not always accurate and may overestimate the cost to reach the goal state. In the context of the monotonic restriction h(n) ≤ cost(n,m)+h(m), using a suboptimal heuristic can have implications on the search process.

When **h(n) is suboptimal, it may violate the monotonic restriction**. Let's break down the restriction: h(n) ≤ cost(n,m)+h(m)

If h(n) is suboptimal, it can potentially overestimate the true cost to reach the goal from state n. This means that h(n) might be larger than the actual cost required to get to the goal state. If this happens, the left side of the inequality is higher than it should be.

On the right side of the inequality, cost(n,m) + h(m) represents the actual cost to move from state n to state m plus the true cost from state m to the goal. If h(m) is also a suboptimal heuristic, it might overestimate the cost to the goal from state m, further increasing the value on the right side of the inequality. Because h(n) and h(m) are overestimating the true costs, the inequality h(n) ≤ cost(n,m) + h(m) might not hold in certain cases.

4. The 8-puzzle has a 3x3 grid with 8 numbered tiles and 1 blank tile. Any configuration of the puzzle can be represented as a permutation of the numbers 1 to 8 (plus the blank tile, which is represented as 0). The "goal state" of the puzzle is usually represented as:

   [1 2 3]
   [4 5 6]
   [7 8 0]

   A configuration is considered solvable if it is possible to transform it into the goal state through a series of legal moves. Otherwise, it's unsolvable.

   To determine whether a given configuration is solvable, you can use the concept of permutation inversions. An inversion occurs whenever a tile precedes another tile with a lower number but appears after it in the current configuration. For example, in the configuration:

   [2 1 3]
   [8 0 4]
   [7 6 5]

   The tile 2 precedes tile 1 and tile 2 has a lower number, so there's an inversion. Counting all such inversions in the puzzle configuration can provide a measure of its "parity."

The parity of a configuration (whether it's even or odd) plays a key role in determining its solvability. The formal proof involves a few steps:

1. If the blank tile is in an even row counting from the bottom and the number of inversions is odd, the puzzle is solvable.
2. If the blank tile is in an odd row counting from the bottom and the number of inversions is even, the puzzle is solvable.
3. In all other cases, the puzzle is unsolvable.

5. **Monotone restriction: h(n) <= cost (n, m) + h(m)**
   In this inequality, h(n) represents the heuristic value for state n, cost(n,m) represents the cost of moving from state n to state m, and h(m) represents the heuristic value for state m.

   Let's analyze the two given heuristics:
   Heuristic 2 (h2(n)): Number of tiles displaced from their destined position
   For this heuristic, as the puzzle gets closer to the goal state, the number of tiles displaced from their destined positions can only decrease or remain the same, since we are trying to solve the puzzle by minimizing the number of misplaced tiles. Therefore, this heuristic follows the monotone restriction.

   Heuristic 3 (h3(n)): Sum of the Manhattan distance of each tile from the goal position
   The Manhattan distance heuristic calculates the sum of the distances each tile needs to travel to reach its goal position. As the puzzle gets closer to the goal state, the Manhattan distances for tiles can only decrease or remain the same. This is because tiles are moved closer to their goal positions, reducing the distance they need to travel. Therefore, this heuristic also follows the monotone restriction.

   In both cases, the heuristics follow the monotone restriction $h(n) \leq cost(n,m)+h(m)$, where the heuristic values h(n) are the number of tiles displaced from their destined positions (h2) and the sum of the Manhattan distances of each tile from the goal position (h3). The monotone restriction is satisfied because both heuristics only decrease or remain the same as the puzzle progresses towards the goal state.

6. Let's analyze the situation where the cost of the empty tile is considered, and we treat the empty tile as just another tile with a certain cost associated with it. We'll use the same monotone restriction **h(n) ≤ cost(n,m) + h(m)** to verify whether monotonicity is violated.
   When we consider the cost of the empty tile as just another tile, and we treat it like any other tile with a cost, let's denote this new heuristic as h_modified(n).

   Now, let's observe what happens when we apply the monotone restriction to this modified heuristic:
   Monotone restriction: h_modified(n) ≤ cost(n,m)+ h_modified(m)

For the monotone restriction to hold, the heuristic value for a state n should be less than or equal to the sum of the cost of moving from n to m and the heuristic value of state m.
Consider the empty tile in this scenario. Since we are treating the empty tile as another tile with a cost, it would be included in the calculation of the heuristic value. However, the empty tile does not contribute to the actual distance or cost in a meaningful way like the other tiles do. It's only used to ensure that the heuristic value satisfies the restriction.
When we add the cost of the empty tile in the heuristic calculation, it could potentially introduce scenarios where the monotone restriction is violated. This is because the empty tile doesn't have a meaningful movement cost associated with it like the other tiles (they are swapped without any cost), and it could lead to situations where the heuristic value violates the restriction, especially when combined with other tiles' movement costs.

Therefore, by adding the cost of the empty tile and treating it as just another tile with a cost, we could indeed violate the monotonicity of the heuristic and the corresponding restriction h_modified(n) ≤ cost(n,m)+ h_modified(m) might not hold in certain cases.