# **Indian Institute of Technology Patna**

CS561 AI & ML Laboratory

Assignment #2: A \* Search

Submission Date: 7th April 2024

Group of 2:

1. Baskar Natarajan - 2403res19(IITP001799)
2. Jyotisman Kar – 2403res35(IITP001751)

Semester-1

Mtech AI & DSC

IIT Patna.

[Indian Institute of Technology Patna 1](#_Toc1662841797)

[1. Problem Definition: 2](#_Toc2108963224)

[2. Python Libraries used are, 2](#_Toc1164430642)

[3. Common logics used are, 3](#_Toc162253554)

[a. H1: 4](#_Toc1723068925)

[b. H2: 4](#_Toc1253378267)

[c. H3: 4](#_Toc1102023981)

[e. Calculate\_cost: 4](#_Toc84660044)

[f. Generate Ramon Initial State: 4](#_Toc2136449976)

[g. Astar\_Search: 4](#_Toc789555376)

[h. Astar\_Search: 5](#_Toc1598296703)

[3. Evaluate\_heuristics: 5](#_Toc781976352)

[4. Advantages of A Star Search over BFS and DFS 6](#_Toc1395710056)

[5. Statistics and Comparision between Heuristic functions: 6](#_Toc1057114816)

[6. Output: 6](#_Toc443501697)

## Problem Definition:

The assignment targets to implement A\* search for the 8-puzzle problem

**Question**:

* In a general search algorithm, each state (n) maintains a function f(n) = g(n) + h(n) where g(n) is the least cost from the source state to state n found so far and h(n) is the estimated cost of the optimal path from state n to the goal state.
* Implement a search algorithm for solving the 8-puzzle problem with the following assumptions for the given start and end state. If unreachable, start with a random state and retry until the Target State (shown above) is reached ()
* Random state can be taken as input as well.

1. g(n) least cost from the source to the current state.
2. Heuristics
   1. h1(n) = 0.
   2. h2(n) = number of tiles displaced from their destined position.
   3. h3(n) = sum of the Manhattan distance of each tile from the goal position.
   4. h4(n) = Devise a heuristic such that h(n) > h∗(n)
   5. Goal State selected is:

[[1, 2, 3],

[4, 5, 6],

[7, 8, 0]]

## Python Libraries used are,

* **Random** - The random library in Python provides functionalities to generate random numbers, choose random elements from sequences, and shuffle sequences
  + In Our case we are using to generate random initial state.
* **Datetime** – It provides various datetime functions, we are using now() function to calculate the time taken to attain the goal state.
* **Numpy** - provides a more convenient way to work with arrays and matrices
  + numpy.array - create numpy array to fully use of its features
  + numpy.where - to find the indices of the given element in the array
  + numpy.array\_equal – find the equality matrix between two array
  + Numpy.sum - adding the values of an array
* **Heapq** – Priority Queue
  + Efficient implementation of priority queue using a binary heap
  + After heapq.heappush, the heap will be adjusted such that the node with the lowest total cost will be at index 0 of the heap list.
  + This allows us to efficiently retrieve the node with the lowest total cost when needed for expansion.

## 3. Common logics used are,

### H1:

* 1. Heuristic value is always 0

### H2:

* 1. Calculates total number of misplaced tiles (excluding the empty space)

### H3:

* 1. Find Manhattan distance:
  2. Sum of absolute value of distance between indices of each element to the goal state index.
  3. Manhattan Distance for tile=∣x−goalx ∣+∣y−goaly ∣

1. H4:
   1. Just add a constant value to h3, so that the following criteria match.
   2. h(n) > h\*(n)

### Calculate\_cost:

* 1. Returns heuristic value and steps, above are heuristic functions.

### Generate Ramon Initial State:

* 1. Generate random values between 0-9, array will be shaped as 3x3

### Astar\_Search:

* 1. Initialize the state and variables required for searching and statistics.
     1. visited = set() # To track visited states  
        open\_list = [] # Priority queue for the open list  
        iterations = 0 # Count iterations
     + Flatten the start\_state for set operations so that its hashable.
     + Add the flat start\_State to the visited set.
     + Add the current state and cost value to the Priority queue
     + Check current state equal to goal state, if yes return
     + Else continue to check the Neighbours.
       - Calculate the Cost for each Neighbour and add it to the list visited and priority Queue.
     + Continue the parent until the Neighbours are available in the priority Queue, If Queue is underflow, return none.
     + Generate Random state again until the solution found

## Evaluate\_heuristics:

Following are the values we need as part of the evaluation and statistics

min\_cost # *Minimum Cost across Heuristic Fns*  
min\_heuristic\_cost\_fn = None # *Fn name of Minimum Cost*   
min\_time # *Minimum time taken across Heuristic Fns*  
min\_heuristic\_time\_fn = None # *Fn Name of the above value*  
min\_iterations = # *Minimum iterations taken across* *Heuristic Fns*  
min\_heuristic\_iterations\_fn = None # *Fn Name of the above value*

* + *Call the Astar\_Search\_retry* 
    - *From there it calls the Astar Search*

## Advantages of A Star Search over BFS and DFS

* A\* search is advantageous over BFS and DFS when we want to find the shortest path to a goal state in a graph or search space.
* It is optimal, efficient, memory-efficient, and flexible due to its use of **heuristics to guide** the search.
* However, it does require an admissible heuristic, which may not always be available or easy to define for some problems.

## Statistics and Comparision between Heuristic functions:

* In Real time scenario mostly H3 should be the best solution. But in the random state, we got the following statistics.
  + Minimum Total Cost among Heuristics:9 by Heuristic Fn :h1  
    Minimum Time Taken: 0.0093 seconds (Heuristic: h4 )  
    Minimum Iterations: 74 (Heuristic: h4 )

## Output:

Starting Heuristic:h1  
Initial State:  
[[1 5 2]  
 [7 4 0]  
 [8 6 3]]  
Goal State:  
[[1 2 3]  
 [4 5 6]  
 [7 8 0]]  
Heuristic:h1 took 9 steps, 336 Iterations and took 3.8228 seconds to reach the goal  
Total Cost (Steps + Heuristic Value): 9  
h1 completed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Starting Heuristic:h2  
Initial State:  
[[3 1 2]  
 [0 8 7]  
 [6 5 4]]  
Goal State:  
[[1 2 3]  
 [4 5 6]  
 [7 8 0]]  
Heuristic:h2 took 23 steps, 9348 Iterations and took 0.2736 seconds to reach the goal  
Total Cost (Steps + Heuristic Value): 31  
h2 completed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Starting Heuristic:h3  
Initial State:  
[[2 8 3]  
 [7 0 1]  
 [6 4 5]]  
Goal State:  
[[1 2 3]  
 [4 5 6]  
 [7 8 0]]  
Heuristic:h3 took 18 steps, 147 Iterations and took 34.6395 seconds to reach the goal  
Total Cost (Steps + Heuristic Value): [32]  
h3 completed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Starting Heuristic:h4  
Initial State:  
[[8 2 4]  
 [1 7 3]  
 [6 5 0]]  
Goal State:  
[[1 2 3]  
 [4 5 6]  
 [7 8 0]]  
Heuristic:h4 took 20 steps, 74 Iterations and took 0.0093 seconds to reach the goal  
Total Cost (Steps + Heuristic Value): [44]  
h4 completed \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
Minimum Total Cost among Heuristics:9 by Heuristic Fn :h1  
Minimum Time Taken: 0.0093 seconds (Heuristic: h4 )  
Minimum Iterations: 74 (Heuristic: h4 )