CS571/561 – Artificial Intelligence

End Semester Examination

Name: **M Maheeth Reddy**	Roll No.: **1801CS31**	Date: **24-Nov-2021**

Documents Submitted:

i. Codes with appropriate documentation: Please see the file **hill_climbing.py** for the code with documentation. I have also put the start state file called **start_state.txt**.

ii. Outputs as mentioned in (c) for each of the heuristics referred in (d)

iii. Detailed outputs with proper explanations for (i)-(v) of (e).

iv. Miscellaneous explanation

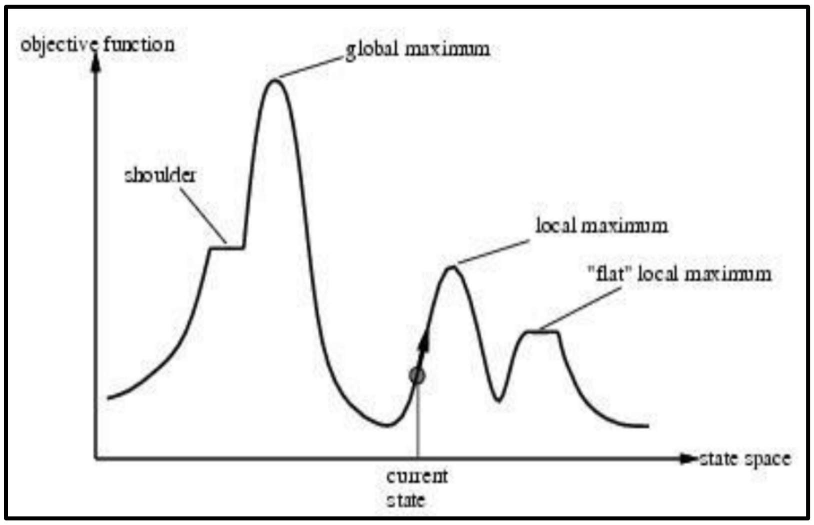
iv. Miscellaneous explanation

Hill Climbing Algorithm, also called **Greedy Local Search**, is a local search algorithm which continuously moves in the direction of increasing value until it finds a peak where no neighbor has a higher value i.e., or the best solution to the problem.

Drawbacks of Hill Climbing:

- **Local Maxima**: These are the peak states in the problem space, that are higher than the neighbouring states but below the global maxima. The algorithm is likely to get stuck in such cases.
- **Plateau**: The region where all the neighbours of the current state have the same value is termed as a **Plateau**. There is no uphill exit for the algorithm and it gets stuck.
- **Ridge**: A region which is flat like a plateau, but with drop-offs to the sides, has a slope and hence cannot be reached in a single step.

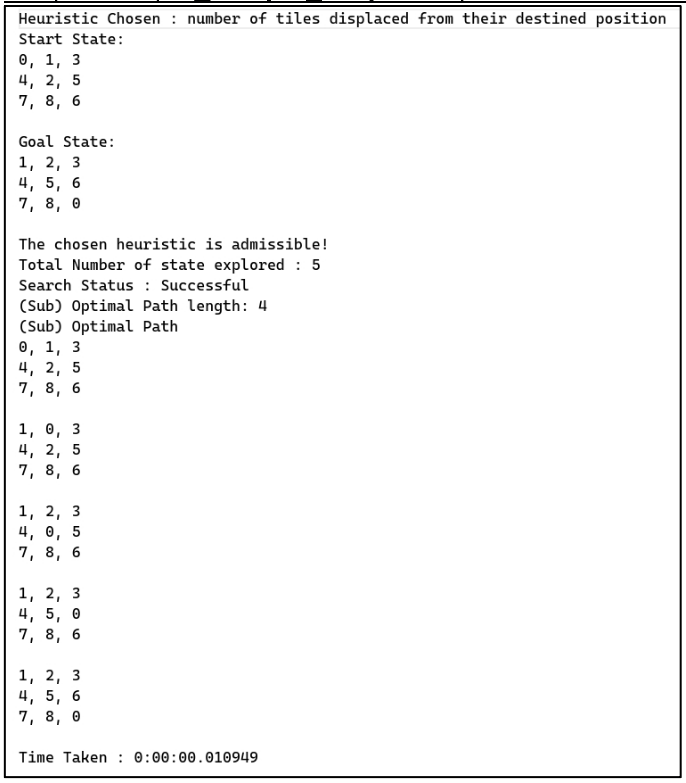
State Space Diagram of Hill Climbing



Credits: Slides

ii. Outputs as mentioned in (c) for each of the heuristics referred in (d)

Output file (**h1_sample_output.txt**) for **Heuristic h1 = number of displaced tiles**

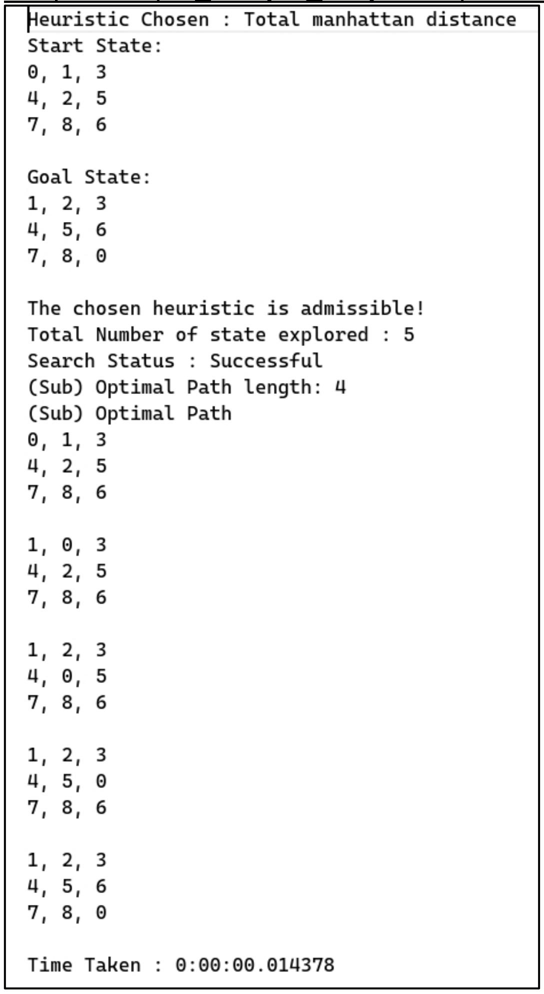


```
Heuristic Chosen : number of tiles displaced from their destined position
Start State:
0, 1, 3
4, 2, 5
7, 8, 6

Goal State:
1, 2, 3
4, 5, 6
7, 8, 0

The chosen heuristic is admissible!
Total Number of state explored : 5
Search Status : Successful
(Sub) Optimal Path length: 4
(Sub) Optimal Path
0, 1, 3
4, 2, 5
7, 8, 6

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

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

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

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

Time Taken : 0:00:00.010949
```

The output file consists of the following:

1. Success or Failure Message
2. Heuristics chosen, Start state and Goal state
3. (Sub)Optimal Path (on success)
4. Total number of states explored
5. Total amount of time taken

Output file (**h2_sample_output.txt**) for **Heuristic h2 = Total Manhattan Distance**



```
Heuristic Chosen : Total manhattan distance
Start State:
0, 1, 3
4, 2, 5
7, 8, 6

Goal State:
1, 2, 3
4, 5, 6
7, 8, 0

The chosen heuristic is admissible!
Total Number of state explored : 5
Search Status : Successful
(Sub) Optimal Path length: 4
(Sub) Optimal Path
0, 1, 3
4, 2, 5
7, 8, 6

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

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

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

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

Time Taken : 0:00:00.014378
```

Note

In case this program is run with a different start state and is stuck in a local maxima, then the number of states explored is not displayed in the output because that number can vary everytime the program is run with the same input.

iii. Detailed outputs with proper explanations for (i)-(v) of (e).

i. Check whether the heuristics are admissible.

Answer:

The heuristic h1(n) = Number of displaced tiles, **is always admissible**.

Reason: A heuristic h is admissible if h(n) ≤ h*(n) where h*(n) is the true cost to a nearest goal. In words, this means that, heuristic h should not overestimate the cost of reaching the goal state for being admissible. In the 8-puzzle problem, each displaced tile must be moved at least once to reach the goal state. So, the actual cost of reaching the goal state i.e., h*(n) in the notation above, will be greater than or equal to the number of displaced tiles, h1(n). Mathematically, h1(n) ≤ h(n). Since, h1 is not overestimating the cost of reaching the goal state, it is admissible.

The heuristic h2(n) = Total Manhattan distance, **is always admissible**.

Reason: In the 8-puzzle problem, we can only move one block at a time and in only one of the four directions. The least cost for moving each block to its position in the goal state is 1 provided each block has an unobstructed path to its goal state. In other words, the Manhattan Distance in such a case is 1. In every other scenario, it will take more moves than the Manhattan Distance to get the blocks in the right place. Since this heuristic does not overestimate the cost of reaching the goal state, it is admissible.

ii. What happens if we make a new heuristics h3 (n) = h1 (n) + h2 (n)

Answer:

The newly created heuristic h3(n) **is not** admissible.

Reason: We have checked that h1(n) and h2(n) are admissible in the previous answers. So, h1(n) ≤ h*(n) and h2(n) ≤ h*(n).

Now, h3(n) = h1(n) + h2(n) does not guarantee that h3(n) must lesser than or equal to h*(n). So, the admissibility of the heuristic h3(n) cannot be deduced.

Consider the following **example**,

Start State		Path		Goal State
1 2 3		1 2 3		1 2 3
4 5 6	→	4 5 6	→	4 0 6
7 8 0		7 0 8		7 5 8

In this case, h1 = 2 and h2 = 2. As per Heuristic h3's definition, h3(n) = 2 + 2 = 4. But we are clearly able to reach the Goal state in 2 steps only. In other words, actual cost, 2 < h3, 4. Hence h3 is proven to be a not admissible heuristic for the 8-puzzle problem.

iii. What happens if you consider the blank tile as another tile?

Answer:

The heuristics values may increase now as the heuristic values of even the blank tile is included. This might affect the admissibility of the heuristic as I will show below.

Let us define two more heuristics:

- Heuristic h'(n) = **Number of displaced tiles(h1)** when blank tile is considered another tile
- Heuristic h''(n) = **Total Manhattan Distance(h2)** when blank tile is considered another tile

Consider the same **example** as above,

Start State		Path		Goal State
1 2 3		1 2 3		1 2 3
4 5 6	$\longrightarrow$	4 5 6	$\longrightarrow$	4 0 6
7 8 0		7 0 8		7 5 8

By their definitions, we can see that h'(n) = 3 and h''(n) = 4. So, it is clear that actual cost, 2 is lesser than both h'(n), 3 and h''(n), 4. Hence, if we consider blank tile then both the heuristics h1 and h2 become non-admissible.

iv. What if the search algorithm got stuck into Local optimum? Is there any way to get out of this?

Answer:

In case of Hill Climbing (no modifications or workarounds), if the algorithm is stuck in a Local Optimum, it is not possible to get out of it. In such case, we can use modified hill climbing algorithms:

Stochastic hill-climbing:

- Random selection among the uphill moves.
- Converges slowly than steepest ascent (best first search) but finds better solution in some cases.

First Choice Hill Climbing:

- Stochastic hill climbing by generating successors randomly until a better one is found.
- Good strategy when a state has many successors

Random restart Hill Climbing:

- Conducts a series of hill-climbing searches from randomly generated initial states and stops when the goal is found.
- Works well when there are very few local optima

v. What happens when all the neighbours of the current state have the same value? How to get rid of this situation?

Answer:

In the Hill Climbing Algorithm, the situation when all the neighbours of the current state have the same value is termed as a **Plateau**. There is no uphill exit for the algorithm and it gets stuck.

To get rid of a Plateau situation:

- Either very long steps or very short steps must be taken, or
- Choose a state randomly, far from the current state, so that the algorithm can find a region wthout plateau.