AI Project

(N-Puzzle, Group 4)

Abhigyan Aggarwal - 18UCC035
Archit Khandelwal -18UCC164
Eshaan Rathi -18UCC089
Kartikey Sharma -18UCC158

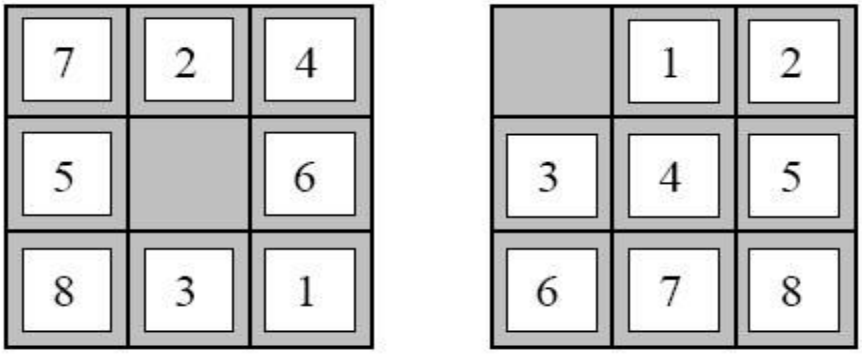
Demo Link : Google Drive
Code Link: Google Drive

Requirements:

1. Python version 3.6 or above
2. Python packages : numpy, matplotlib
3. C++ compiler for C++14 and C++17 (preferably GNU Compiler Collection)

Introduction

N-Puzzle : It is a sliding puzzle that consists of N tiles where N can be 8,15,24 and so on. Here N is of the form $X^2 - 1$. It consists of a grid of X columns and X rows. One of the tiles in the grid is blank and can be used to move the tiles around to solve the puzzle.



This is an 8 puzzle with a given start and goal state. The purpose of the puzzle is to find moves such that the goal state can be found by moving the blank tile.

Solvability of an n-Puzzle Problem :
All states of the n puzzle problem are not not solvable, in general, we can find out if a n puzzle with a K*K grid (for example an 8 puzzle has a 3*3 grid) is solvable or not by following below simple rules:

- If K is odd, then the puzzle instance is solvable if the number of inversions is even in the given state.
- If K is even, puzzle instance is solvable if
    - blank is on an even row counting from the bottom (second-last, fourth-last, etc.) and the number of inversions is odd.
    - blank is on an odd row counting from the bottom (last, third-last, fifth-last, etc.) and the number of inversions is even.
- For all other cases, the puzzle instance is not solvable.

Here inversions refers to the number of pairs a, b where a occurs before b in the array and a>b. The array for a given state of n puzzle consists of linearly arranged N numbers on the tiles.

In our project we have tried to solve the 8 puzzle problem using various algorithms including hill climbing, A* Search, Hill Climbing with Random Restart and Simulated Annealing, etc.

Methods and Algorithms

1) Heuristic Used:
   Manhattan Distance : It is calculated by summing the absolute value of difference in coordinates for the points. For example the manhattan distance between two points (x_1,y_1) and (x_2,y_2) is given by $|(x_2-x_1)|+|(y_2-y_1)|$. In our n-puzzle the Manhattan distance for a state is calculated as the sum of the Manhattan distance for a tile in the current state and it's position in the goal state for all the tiles.
- Manhattan distance is an admissible heuristic in our case because every tile will have to move at least the given number of steps to reach it's correct position.

2) Hill Climbing:
   It is a local search algorithm that uses a greedy approach to find a solution. In hill climbing we use a heuristic function to evaluate how close a state is to the goal state. The algorithm evaluates all of the possible neighboring states and moves on to the best possible one. It terminates when the current state is better than all it's neighboring states or when the goal state has been found. When we have more than one possible state, as best it chooses randomly among them.

   Hill climbing algorithm has some shortcomings which are :
   A. Local Maximum : If the current state is better than all its neighboring states the algorithm will terminate even if there is a better state to be found.
   B. Plateau : If the current state and the neighboring states all have equal values the algorithm will not perform well and might not take the correct move.
   C. Ridge : When the algorithm reaches a peak on a ridge it will terminate because the neighboring states have lower values.

3) Random Restart and Simulated Annealing

In random restart we do hill climbing for a fixed number of retries and pick the best ending state of all the retries. If we retry too many times, search cost will increase as well as the probability of finding a solution will increase. This increases the time taken by our algorithm.

4) Simulated annealing is based on the metallurgical process of annealing. It is used to increase ductility and reduce hardness in metals to make them workable. Simulated annealing is used to pick moves based on a probability function. Instead of always picking the better move we can also choose a worse move based on this probability function. We start with an initial value of temperature and reduce it in every step, as the temperature gets lower the probability of selecting a worse move also reduces. So we are more likely to evaluate more states as well as get to a global maximum in the end. The performance depends on how slowly we reduce the temperature function with time, if we do it slowly we have higher chances of success but the search_cost increases.

5) A* Search -
   In A* search, we are using an underestimated heuristic h(n) = manhattan distance between the tile position and goal position.

   Proof for admissible heuristic by contradiction :-

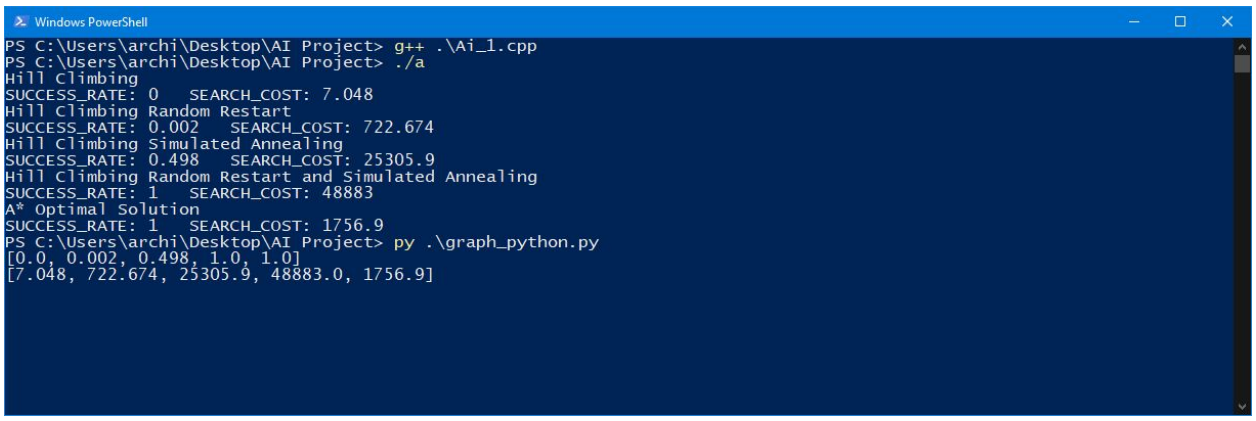
   Let's take s* = goal state and s_0 = some initial state.
   Now, suppose the heuristic is overestimated ie. $h(s) > h^*(s)$, so $h(s_0) > h^*(s_0)$.
   And since in each step the tile can make one only one move in any four directions. So the heuristic decreases by 1 in each step in the best scenario.
   If we reach the goal position after some C* steps then,
   $\Rightarrow h(s^*) \geq h(s_0) - C^* > 0$.
   $\Rightarrow h(s^*) > 0$.
   But h(s*) should be 0, so we get a contradiction. Hence $h(s) \leq h^*(s)$, ie h(s) is always an underestimated heuristic.

   Terms :
   a) gscore - how far we have come from our initial state (ie. g(n)).
   b) fscore - sum of gscore and heuristic value of the state (ie. f(n)).
   c) TentativeGscore - gscore of parent state + 1.
   d) Openset - priority queue of fscore and state which is sorted in increasing order of fscores.

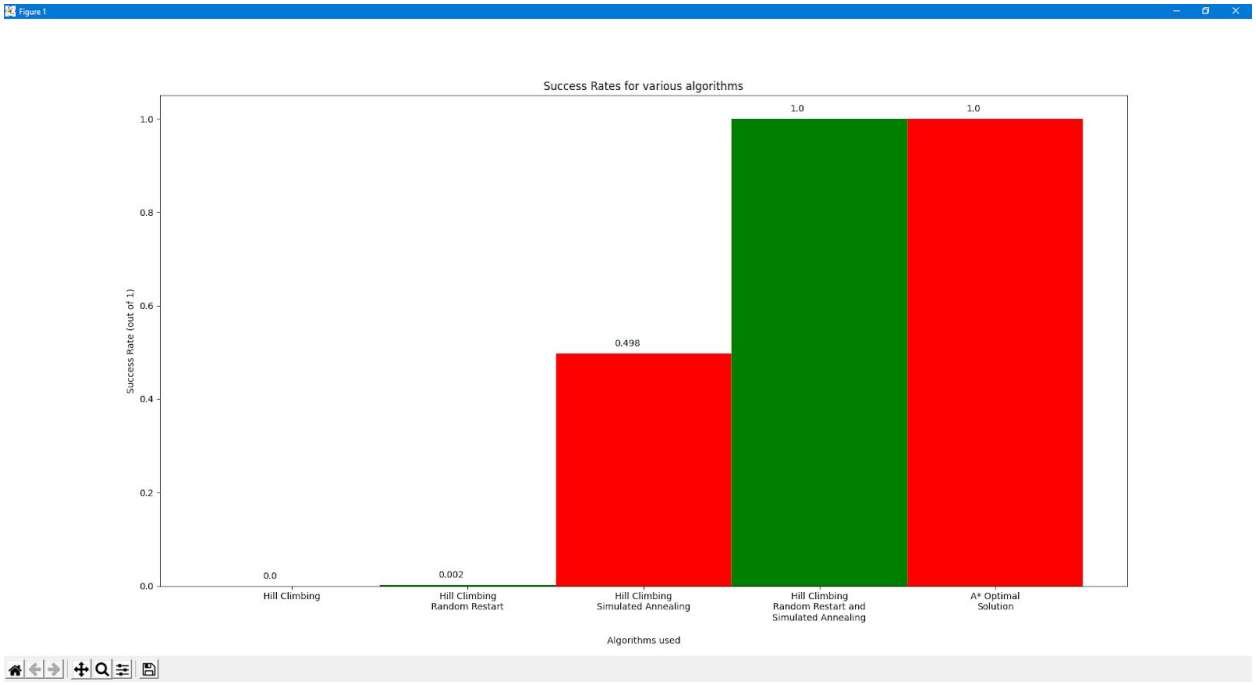
   An openset is used to store pairs of fscore and corresponding states. Initially only the start state is present in openset which has a gscore of 0. While the openset is not empty, in each loop we select the state with the smallest fscore, using openset which has O(1) time complexity and if the state is our goal state we return with a success code, otherwise this state is popped from the openset and all the next possible states (these

being if the blank tile can make a valid move in up, down, left and right directions) are evaluated. For each of these neighboring states, we calculate their TentativeGscore. If this score is less than the current gscore the state, it is assigned the TentativeGscore, therefore a new fscore and is pushed into the openset, and the loop continues.

Results

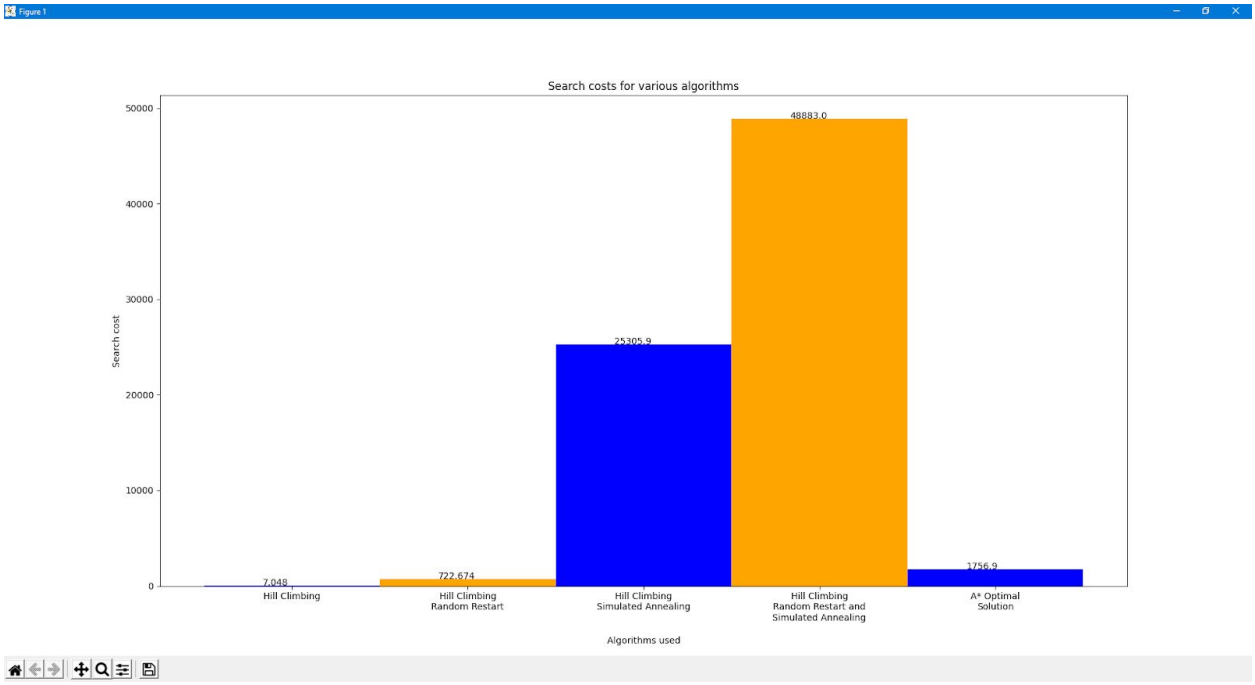


Results for running the program for 1000 Instances of the 8-puzzle problem.



Graph showing average search cost for various algorithms after running on 1000 instances of 8-puzzle problem.

Graph showing success rate for various algorithms after running it on 1000 instances of the 8-puzzle problem.



Observations

Some observations :-

1) If we decrease the rate of change in temperature in simulated annealing, it increases the success rate as well as the search cost. For example, we used a change rate of 0.01 per move in our experiment. If we change it to 0.001. We can get a success rate close to 1, but it also increases the search cost by 5-6 times, thus taking much more time to execute.
2) On combining simulated annealing with restart we get a success rate of 1, as compared to close to 0.5 for only simulated annealing. This also increases the search cost to nearly double of simulated annealing. But the improvement in success rate is more significant as compared to the increase in search cost in terms of execution time.
3) Hill Climbing Algorithm alone or combined with random restart, terminates on a plateau or local maxima most of the time, and rarely reaches the goal state.
4) A* algorithm always reaches the goal state when it is possible to reach the goal state.
5) Hill Climbing with annealing with random restart has a very high success rate and reaches goal state most of the time but has a high search cost.

Conclusion

Among all the variations of hill climbing that we have implemented, Hill climbing with random restart and simulated annealing has the best performance, it is able to reach the goal state most of the time and the search cost is also not too high. Basic hill climbing has a lot of issues and almost always gets stuck in local maxima, so it cannot be used to solve 8 puzzle problems with efficiency. Random restart and simulated annealing provide very exceptional improvements with basic hill climbing.

Contribution of each group member

Abhigyan Aggarwal

- Coded basic struct state and constructor to initialize goal state.
- Added function to return possible moves taking value of blank tile in the puzzle.
- Coded function hill_climbing_smulated_annealing to perform hill climbing with simulated annealing algorithm from a state and return success, search cost
- Coded function hill_climbing_randomrestart_simmulated_annealing to perform hill climbing with random restart and simulated annealing from a state and perform success, search cost.
- Created this document and wrote Introduction.
- Added Heuristic used in Methods and Algorithms within this document.
- Added results in this document.
- Added Observations in this document.

Archit Khandelwal

- Added random state initialization constructor in struct state
- Added inversion count function to check if a state is solvable and changed random state initialization constructor to generate only solvable states.
- Coded function hill_climbing to perform hill climbing algorithm from a state and return success, search cost.
- Added auxiliary function get_probability() for simulated annealing.
- Created a sharable link to the project code files.
- Added Hill Climbing in Methods and Algorithms within this document.
- Posted requirements for running the code.
- Added conclusion in this document.

Eshaan Rathi

- Added function definition for get_heuristic() within struct state.
- Defined function move() to change state return a new state after performing a move on a state.
- Added function to overload == operator to check if two states are equal.
- Coded function hill_climbing_random_restart to perform hill climbing with random restart algorithm from a state and return success, search cost.

- Added code to write results in a csv file for plotting graphs.
- Created video demo of the project.
- Added Simulated Annealing and Random Restart in Methods and Algorithms within this document.
- Coded a python script to read csv files and plot graphs using matplotlib.

Kartikey Sharma

- Coded function posof_blank_tile() to return a pair with position of blank tile.
- Coded function A_star to run A_star algorithm for the optimal solution returning search_cost ,success.
- Defined function get_hash() to struct state for use in map to store visited states.
- Defined struct Compare and Node for use in A* priority queue and map.
- Added code in int main() to integrate results of each algorithm.
- Added A* search in Methods and Algorithms within this document.
- Added contributions of each member in the document

References

https://www.cs.bham.ac.uk/~mdr/teaching/modules04/java2/TilesSolvability.html

http://www.cs.cmu.edu/afs/cs.cmu.edu/project/learn-43/lib/photoz/.g/web/glossary/anneal.html

https://www.cs.princeton.edu/courses/archive/spring18/cos226/assignments/8puzzle/index.html

https://www.cs.ucdavis.edu/~vemuri/classes/aiclass_old/heuristicsearch.pdf