**University of California, Riverside**

**CS205-**

**Artificial Intelligence**

**Winter 2024**

**CS205 Assignment 1: The Eight Puzzle**

**05/07/2024**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **SID** | **Email** | **Group Number** |
| **Parth Shinde** | **862466119** | [**pshin022@ucr.edu**](mailto:pshin022@ucr.edu) | **-** |

I consulted the following in completing this assignment:

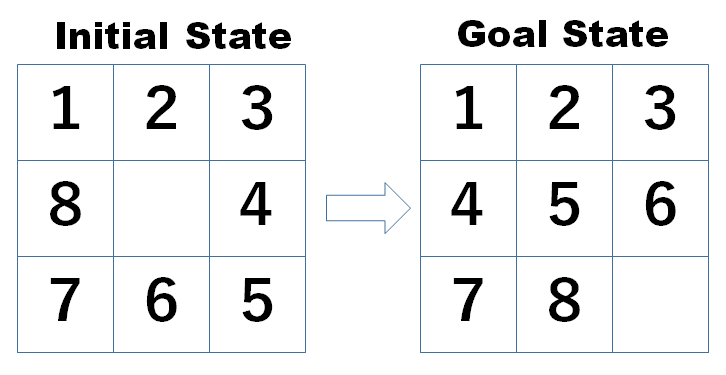
* CS 205 Lecture slides on [heuristic search](https://www.dropbox.com/sh/ftzvcnntl2j5eiu/AABcF7bSG1Na5cUrP3yICgyra/3__Heuristic%20Search.pptx?dl=0) by Dr. Eamonn Keogh
* [Content](https://realpython.com/python-heapq-module/) - I consulted for python library heapq (Unimportant – part of python in-built library)
* [Content](https://www.cs.princeton.edu/courses/archive/spring18/cos226/assignments/8puzzle/index.html#:~:text=The%208%2Dpuzzle%20is%20a,vertically%20into%20the%20blank%20square.) - I consulted for structuring the game format and the board(specifically the input) Npuzzle
* I also consulted the following links for understanding heuristic search and the different cost functions and their implementation.
  + <https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/>
  + <https://www.educative.io/answers/what-is-uniform-cost-search>

List of contents:

* + Cover page: (this page)
  + Description of assignment: Pages 2-3
  + Example Traceback: Pages 3-4
  + Summary of results: Page 5
  + Code: Pages 6-9

**Introduction**

The sliding puzzle, particularly in its 3x3 (8 puzzle) form, presents a classic problem of rearranging shuffled tiles through sliding movements to reach a predefined goal state (for our case is the goal state is the normal ascending order in a row-major format in a the 3x3 matrix of tiles).



We address this problem using the A\* algorithm, enhanced with three distinct heuristic functions to evaluate the potential cost of moves. I have used Python 3 for this implementation, and it guides the puzzle towards the solution efficiently.

**Description of heuristics**

The A\* algorithm is algorithm that considers the state with the lowest cost. The cost can be described by the following:

Here is the cost for going from the initial state to the current state. This is generally used to describe the total cost of operations applied to arrive at the current state. is an estimate of the cost to reach the goal, calculated using a heuristic function. For this assignment, we are asked to implement three heuristics. If the considered state is not the goal state, then it is expanded and the next state with lowest cost is considered next.

**Uniform Cost Search**

It is a part of A\* algorithm that has some with to the Djikstra Algorithm. Uniform cost search considers hardcoded as 0. It is an uniformed search that consists of simple cost function g(n) which is described as the total number of operations required to get to the current state. So , in our case ,it would be the cumulative number of moves that it takes to reach from the initial state to the current state(i.e the next state to choose from).

**Manhattan Heuristic**

It is considered an fully informed heurstic as it uses the - Manhattan heuristic function considers the cost as the sum of the Manhattan distances of location of each tile in the current state with the location of the corresponding tile in the goal state.

**Misplaced Tile Heuristic**

Misplaced tile heuristic function considers cost as the total number of tiles that are not in the correct location as per the desired goal state. Thus it is considered a partially informed heuristic as we do not know the exact location of each tile.

Algorithm

As per the assignment, the pseudocode of the algorithm is described in figure 3

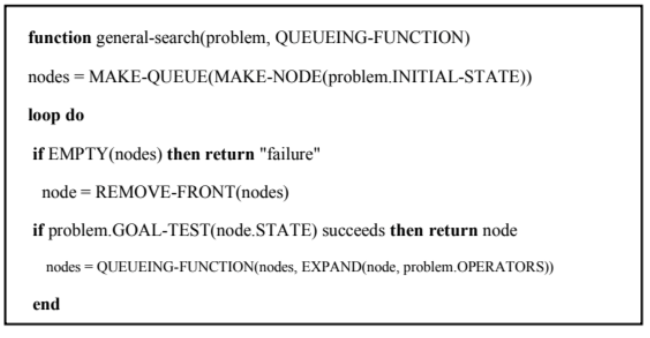


Figure 3: Pseudocode

In the assignment we have 7 puzzle inputs to verify and check the validity of our code. The code allows the user the following choices:-

1. It allows the user to input their NxN puzzle board (can be 3x3 or 4x4,etc)
2. It allows the user to input their initial state of the board .
3. It allows the user to this in a easy to understand string format ,so the input is entered in a string format and represented in a row-major matrix format
4. In the

One difference from the standard heuristic function I have implemented is that the queuing function considers the node with the lower if two states have the same .

The code can be accessed [here](https://colab.research.google.com/drive/13sj6i9GiW6mmS7Br-LNFxzL1llTsAofO?usp=sharing)

The following is the traceback of a depth 8 puzzle, the user inputs are highlighted:

Welcome to puzzle solver!

Press enter to solve from a list of default puzzles, or input a number N to solve a custom NxN-puzzle:

\_

Choose among a list of predefined puzzles, with depths 0,2,4,8,12,16,20,24 by inputting one of these depths:

8

Select Heuristic, input 1 for Uniform Cost, 2 for Manhattan and 3 for Misplaced Tile:

2

Performing Search, initial\_grid:

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

------

goal\_grid:

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

Goal State!

enter 'y' to perform traceback, press enter to skip:

y

Considering Node with g(n)=0, h(n)=8 :

[1, 3, 6]

[5, 0, 2]

[4, 7, 8]

Considering Node with g(n)=1, h(n)=7 :

[1, 3, 6]

[5, 2, 0]

[4, 7, 8]

Considering Node with g(n)=2, h(n)=6 :

[1, 3, 0]

[5, 2, 6]

[4, 7, 8]

Considering Node with g(n)=3, h(n)=5 :

[1, 0, 3]

[5, 2, 6]

[4, 7, 8]

Considering Node with g(n)=4, h(n)=4 :

[1, 2, 3]

[5, 0, 6]

[4, 7, 8]

Considering Node with g(n)=5, h(n)=3 :

[1, 2, 3]

[0, 5, 6]

[4, 7, 8]

Considering Node with g(n)=6, h(n)=2 :

[1, 2, 3]

[4, 5, 6]

[0, 7, 8]

Considering Node with g(n)=7, h(n)=1 :

[1, 2, 3]

[4, 5, 6]

[7, 0, 8]

Considering Node with g(n)=8, h(n)=0 :

[1, 2, 3]

[4, 5, 6]

[7, 8, 0]

Summary of results:

I ran the example puzzles provided in the assignment document. I tracked metrics including time taken, number of nodes expanded and the max size of the frontier. These are summarized below. I have included semilog plots also for a better understanding of the observations.



Conclusion

* The Manhattan heuristic led to lower time and memory consumption than the misplaced tile heuristic. This seems intuitive since the Manhattan heuristic is “dominant” compared to misplaced tile.
* The results overall seem intuitive based on the content discussed in the lectures.

The implementation was done in Google colab. The code is enclosed below.

