A* Search Pathfinding Algorithm implementation on an 8-Puzzle game

Wilbert M. De la Rosa

University of Massachusetts Lowell

Final Project - COMP 4200 Artificial Intelligence

Professor João Luís Garcia Rosa

Summary:

A* is a pathfinding algorithm that has been used for many years as one of the top options for its optimal approach to solving problems. In this paper, I implemented the algorithm to solve an 8-puzzle game. This consists of a 3x3 grid with numbers that have an initial state, the user needs to move a white square by moving it left, right, up, or down to match it to a provided final state. In this paper, I explain how A* search works and an implementation with a python programming language script that completes the game providing an optimal solution for the 8-puzzle game.

A* search algorithm:

A* search is a computer algorithm that is widely used in pathfinding and graph traversal. This algorithm was developed in the Shakey project, a robot that calculated a set of movements. This graph traverser algorithm finds the shortest path of nodes within a particular region, this can be chosen as two specific points that can be described as nodes. This algorithm was first introduced at Stanford University in 1969 and is also known as an extension of the Dijkstra algorithm.

A* search uses the following formula: f(n) = h(n) + g(n) where h(n) is the heuristic cost and g(n) is the cost of the node, or if it is on a graph representation is also known as the depth of the nodes. The heuristic function does an estimation of how far a point from another is. This can be represented with the following formula in some cases: h(v) = √ (Vx − Tx) + √ (Vy − Ty). This is an example of an equation to estimate the distance between two points. One of the goals with A* is to find the least-cost paths when the cost of a path is the sum of its edge nodes. This algorithm is an example of an informed search, in which it already knows the final state. One of the benefits is that it will keep track of the previous states already visited and will not check on paths that are not necessary, so it will go down the optimal path. This allows the algorithm to save time which will make this faster than some other search algorithms.

8-puzzle game:

The 8-puzzle game is known as a sliding game where the user needs to complete a 3x3 grid with 9 squares where the purpose is to move a blank square through the grid to complete it. In my python program, I used a 3x3 matrix as the representation of the game with a "_" symbol for the blank space. As the following images show:

Example of 8-puzzle game

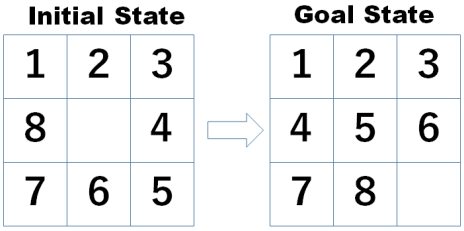

Figure 1.1 - Image from researchgate.net

Representation of state in my program

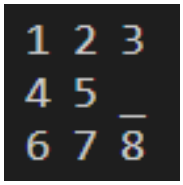

Figure 1.2 - State representation

Implementation of A*seach on the game:

To complete the 8-puzzle with A* the function f(n) = g(n) + h(n) is used in this case. h(n) is the number of misplaced positions of a node compared to the final state position. So for this, the implementation would be to reach the final state position of all the nodes with the starting state and count all of them to obtain the "h" value. Then the "g" will be the depth of the node, this is the number of how far the positions compare to the final one. "N" in this case is the input where it will be the number of nodes in the states. One of the benefits of this is that with this game we already have the final state which allows the algorithm to do a comparison between the current state and the final state. This implementation can be represented with best-first search which demonstrates and selects the next path using the evaluation function f(n) as shown in figure 2.1. The good thing about using this algorithm is that by having the final state it will go through the matrices get the function for each one and select the best one.

Example of implementation Using A* search to solve 8-Puzzle

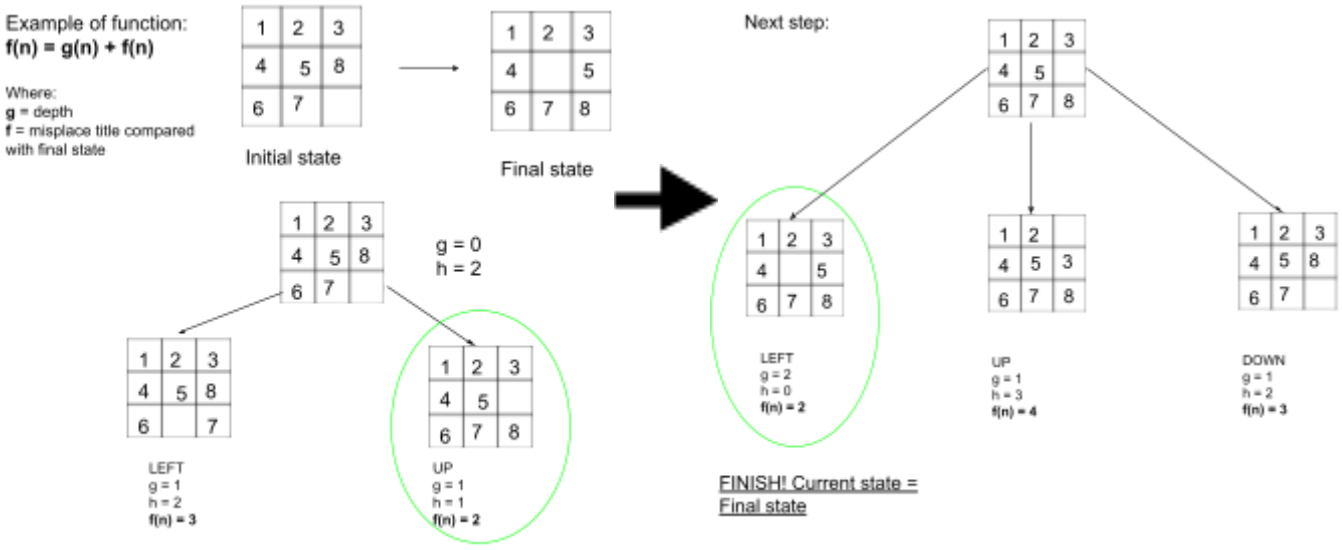

Figure 2.1 - A* search, best-first search solving 8-puzzle

I created this representation of the A* function application on the 8-puzzle to obtain the optimal path. For this, the smallest value of f(n) is the best option to continue through that path. This will continue until we have the final state. For this example, the f(n) will be selected as the next path to continue with the A* search approach. As the image shows going up will be the best option to get to the final state. The best state in the depth is selected with the green circle. On the right side, you can see I started with the result on the left side and as a result, moving the blank to the left made it to the final state.

Explanation of Python Computational Implementation:

I created the final state, with the purpose of testing the solutions in an easier and faster way. For this, I created a 3x3 matrix with strings so that I could create a list of the same data type. The program will ask the user to insert the Initial state of the 8-puzzle game and it will convert it to matrices that will be used on the functions to obtain the f(n) = g(n) + h(n).

```
Please insert your Initial state 3x3 matrix:
1 2 3
4 5 _
6 7 8
```

```
FinalState = [['1','2','3'],
              ['4','_','5'],
              ['6','7','8']]
```

To obtain the f(n) function I first wrote the h(n) function which consists of comparing the initial matrix with the final one. This function will return an “h” value that will be added to the “g” value. After that, I store the f(n) values with their matrix on a hashmap. This function selects the lowest value of f(n), prints it, and continues doing this until the current state is the same as the final state. The data structures I used in the code are arrays (Matrix), and hashmaps (Dictionaries in Python).

Steps implemented in the code:

1. Get the input state from the user and store it in a 3x3 matrix.

```
#Accepts and returns user input of Desire starting state Puzzle
def startPuzzle():
        print("Please insert your Initial state 3x3 matrix: ")
        matrices = []
        for i in range(0,3):
            numbers = input().split(" ")
            matrices.append(numbers)
        print("\n")
        return matrices
```

2. Create a function that will return the “h” value. I did this by comparing the current with the final state matrix and counting the number of misplaced nodes.

```
#This function counts the h number and will compare the current puzzle against the final
def hFunction(matrix, FinalState):

        h = 0
        for x in range(0,3):
            for y in range(0,3):
                if matrix[x][y] != FinalState[x][y] and matrix[x][y] != '_':
                    h += 1
        return h
```

3. Copy the current matrix and create new ones for (UP, DOWN, LEFT, RIGHT) and store them in a hashmap as a key pair. I have a function for the 4 movements where I used a copy of a matrix and used a temporary position to do swaps. This return the h value and corresponding matrix. This is an example of the up function.

```
def up(Matrix, positionOfBlank):
    tempMatrix = copyMatrix(Matrix)
    if positionOfBlank[0] == 1  or positionOfBlank[0] == 2:
        tempPosition = tempMatrix[positionOfBlank[0]-1][positionOfBlank[1]]
        tempMatrix[positionOfBlank[0]-1][positionOfBlank[1]] = '_'
        tempMatrix[positionOfBlank[0]][positionOfBlank[1]] = tempPosition

    h = hFunction(tempMatrix,FinalState)
    return tempMatrix, h
```

4. Add matrices to a dictionary, select the min value of the key, and select that one as the current matrix.

```
dictionaryofMatrices = {}
dictionaryofMatrices[Dh] = Dm
dictionaryofMatrices[Uh] = Um
dictionaryofMatrices[Rh] = Rm
dictionaryofMatrices[Lh] = Lm


minval = min(dictionaryofMatrices.keys())
currentState = dictionaryofMatrices[minval]
```

5. For the “g” first set a value to 0, with a while loop that will stop until the f(n) = 0 which means that it has completed the puzzle.

Results: Examples of the program executing

e.g. 1

```
Please insert your Initial state 3x3 matrix:
1 2 3
4 5 8
6 7 _


next A* state generated:
1 2 3
4 5 8
6 7 _


next A* state generated:
1 2 3
4 5 _
6 7 8
f(n) =  2


next A* state generated:
1 2 3
4 _ 5
6 7 8
f(n) =  2


FINISH!
```

e.g. 2

```
Please insert your Initial state 3x3 matrix:
1 _ 3
4 2 5
6 7 8


next A* state generated:
1 2 3
4 _ 5
4 5 3
6 7 8


next A* state generated:
1 2 3
4 5 _
6 7 8
f(n) =  1


next A* state generated:
1 2 3
4 _ 5
6 7 8
f(n) =  0


FINISH!
```

e.g. 3

```
Please insert your Initial state 3x3 matrix:
1 _ 3
4 2 5
6 7 8


next A* state generated:
1 2 3
4 _ 5
6 7 8
f(n) =  0


FINISH!
```

In this output, the program printed the next generated state, the matrices, and the f(n) value printed. If the program ends it prints the final matrix and says “FINISH!”. As you can see here I used the same example as figure 2.1 and for the other two, I used simple examples where I tested both with that only have 2 steps.

Problems Encountered - Things I would improve if I had more time/python knowledge:

One of the hardest things is that many of the youtube videos and codes about the solution have complex approaches and most of them don't explain the solution or how to write the code. In this case, I wrote a code with an already final state, the user just needs to input the initial state with the same numbers and a blank in a different place which helped me to do test cases in a faster way. If I want the user to insert a final state I just need to create a function similar to the insert first state which is easy.

In terms of code, finding a way of creating the branches and storing them is the hardest part. I did not know what was the best way to accomplish this since I had many months without using Python. If I had more time I would implement the algorithm using classes with nodes, but for the time I had, I preferred doing it in a way that I could do without having to do complex things. But I was able to find many resources on StackOverflow and similar websites to write matrices, dictionaries, and more functions.

Conclusion:

In conclusion, I was able to implement an A* search pathfinder algorithm using python on an 8-puzzle game. The program will find a solution to most 8-puzzles, some puzzles don't have solutions, and in that case, the program will continue running infinitely or will output an error. It was fun learning about this algorithm and implementing it for the first time with Python.

GitHub link that includes and readme and code with comments:
https://github.com/Wilbertdelarosa/8-puzzle_solver_with_A_search

Bibliographies:

YouTube. (2021, November 2). *1. A Star Search algorithm to move from start state to Final State 8 puzzle problem by dr. Mahesh H*. YouTube. Retrieved November 20, 2022, from https://www.youtube.com/watch?v=dvWk0vgHijs

Solving the 8 puzzle in a minimum number of moves: An application of ... (n.d.). Retrieved November 25, 2022, from https://web.mit.edu/6.034/wwwbob/EightPuzzle.pdf

Old Dominion University. (n.d.). *A* search algorithm*. Retrieved November 22, 2022, from http://www.ccpo.odu.edu/~klinck/Reprints/PDF/wikipediaNav2018.pdf

Duke University. (n.d.). *The A* search algorithm*. Retrieved November 27, 2022, from https://courses.cs.duke.edu/fall11/cps149s/notes/a_star.pdf

Informed search algorithms - Portland state university. (n.d.). Retrieved November 20, 2022, from https://web.pdx.edu/~arhodes/ai6.pdf

Python matrices and NumPy Arrays. Programiz. (n.d.). Retrieved November 20, 2022, from https://www.programiz.com/python-programming/matrix

8 puzzle problem using branch and bound. GeeksforGeeks. (2022, July 29). Retrieved November 22, 2022, from https://www.geeksforgeeks.org/8-puzzle-problem-using-branch-and-bound/