CSE 401: Artificial Intelligence

8 Puzzle Single Player Game (A* Algorithm)

Utkarsh Gupta
A2305217557
7CSE 8Y

August 01st, 2020

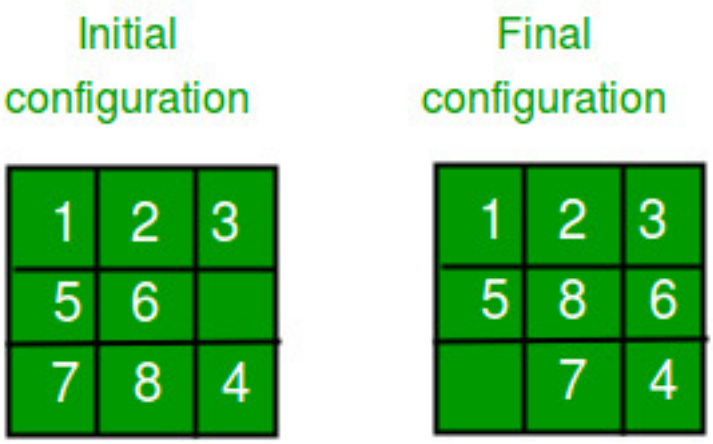
1 Introduction

An instance of the n-puzzle game consists of a board holding $n^2 - 1$ distinct movable tiles, plus an empty space. The tiles are numbers from the set $1, .., n^2 - 1$. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0. Given an initial state of the board, the combinatorial search problem is to find a sequence of moves that transitions this state to the goal state; that is, the configuration with all tiles arranged in ascending order $0, 1, .., n^2 - 1$.

The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions {'Up', 'Down', 'Left', 'Right'}, one move at a time.

In this 8 puzzle problem is discussed.

Given a 3×3 board with 8 tiles (every tile has one number from 1 to 8) and one empty space. The objective is to place the numbers on tiles to match final configuration using the empty space. We can slide four adjacent (left, right, above and below) tiles into the empty space. For example:



A* Algorithm: It is a searching algorithm that searches for the shortest path between the initial and the final state. It is used in various applications, such as maps. In maps the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state).

---

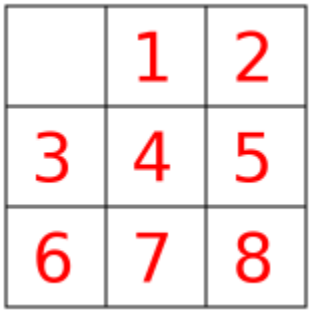
2 Explanation

A* algorithm has 3 parameters:

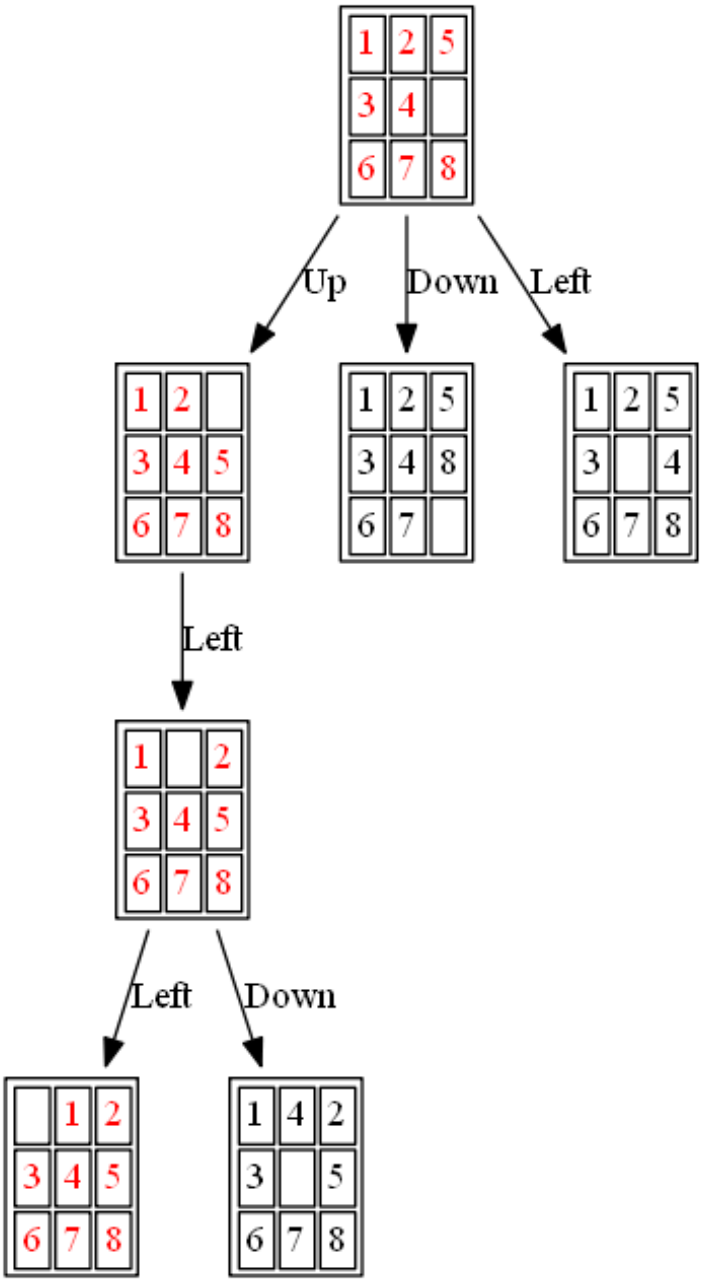
1. **g** : the cost of moving from the initial cell to the current cell.
2. **h** : also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.
3. **f** : it is the sum of g and h. So, f = g + h

Example: A*

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



The path to the goal node with A* (also the nodes expanded by A*) is shown in the following figure:



Dear students, in the code below there are few *TODO* task that you have to complete in this lab session.

---

3 Code

Below is the code (along with the **bonus task**), compiled by Utkarsh Gupta (A2305217557).

[1]:
```
from time import time
from queue import PriorityQueue
import math
```

[2]:
```
#Creating a class Puzzle
class Puzzle:
    # Setting the goal state of 8-puzzle
    goal_state=[1,2,3,8,0,4,7,6,5]
    # Setting up the members of a class
    heuristic=None
    evaluation_function=None
    needs_hueristic=False
    num_of_instances=0
    # constructor to initialize the class members
    def __init__(self,state,parent,action,path_cost,needs_hueristic=False):
        self.parent=parent
        self.state=state
        self.action=action
        if parent:
            self.path_cost = path_cost + parent.path_cost
        else:
            self.path_cost = path_cost
        if needs_hueristic:
            self.needs_hueristic=True
            self.generate_heuristic()
            self.evaluation_function=self.path_cost+self.heuristic

        Puzzle.num_of_instances+= 1

    # method used to display a state of 8-puzzle
    def __str__(self):
        return str(self.state[0:3])+'\n'+str(self.state[3:6])+'\n'+str(self.
 ↪state[6:9])

    # method used to generate a heuristic value
    def generate_heuristic(self):
        self.heuristic=0
        for num in range(1,9):
            distance=abs(self.state.index(num)-self.goal_state.index(num))
            i=int(distance/3)
            j=int(distance%3)
            self.heuristic=self.heuristic+i+j
```

```
    def goal_test(self):
        if (self.state==self.goal_state):
            return True
        return False

    @staticmethod
    def find_legal_actions(i,j):
        # find the legal actions as Up, Doum, Left, Right based on each cell of␣
↪state
        legal_action = ['U', 'D', 'L', 'R']
        if i == 0:  # up is disable
            # if row is 0 in board then up is disable
            legal_action.remove('U')
        elif i == 2:
             legal_action.remove('D')
        if j == 0:
             legal_action.remove('L')
        elif j == 2:
             legal_action.remove('R')
        return legal_action

    # method to generate the child of the current state of the board
    def generate_child(self):
        children=[]
        x = self.state.index(0)
        i = int(x / 3)
        j = int(x % 3)
        legal_actions=self.find_legal_actions(i,j)

        for action in legal_actions:
            new_state = self.state.copy()
            # if the legal action is UP
            if action == 'U':
                # Swapping between current index of 0 with its up element on the␣
↪board
                new_state[x], new_state[x-3] = new_state[x-3], new_state[x]
            elif action == 'D':
                new_state[x], new_state[x+3] = new_state[x+3], new_state[x]
            elif action == 'L':
                new_state[x], new_state[x-1] = new_state[x-1], new_state[x]
            elif action == 'R':
                new_state[x], new_state[x+1] = new_state[x+1], new_state[x]

            children.append(Puzzle(new_state,self,action,1,True))

        return children
```

```
    # method to find the solution
    def find_solution(self):
        solution = []
        solution.append(self.action)
        # implementing the Bonus Task: @utkarsh2102
        solution_path = []
        solution_path.append(self)
        path = self
        while path.parent != None:
            path = path.parent
            solution.append(path.action)
            solution_path.append(path)
        solution = solution[:-1]
        solution.reverse()
        solution_path.reverse()
        # iterating the list to print the path as we go
        for i in solution_path:
          print(i)
          print()
        return solution
```

[3]:
```
# method for A-star search
def Astar_search(initial_state):
    count=0
    explored=[]
    start_node= Puzzle(initial_state, None, None, 0, True)
    print("-----------------------------------------")
    print()
    print("Initial state:")
    print(start_node)
    print()
    print("Solution path in the A-star algorithm is: ")
    q = PriorityQueue()
    q.put((start_node.evaluation_function, count, start_node))

    while not q.empty():
        node=q.get()
        node=node[2]
        explored.append(node.state)
        if node.goal_test():
            return node.find_solution()

        children=node.generate_child()
        for child in children:
            if child.state not in explored:
                count += 1
                q.put((child.evaluation_function, count, child))
```

```
    return
```

[4]:
```
# Start executing the 8-puzzle with setting up the initial state
# Here we have considered 3 initial state intitalized using state variable
state=[[1, 3, 4,
        8, 6, 2,
        7, 0, 5],

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

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

# Iterate over number of initial_state
for i in range(0,3):
    Puzzle.num_of_instances = 0
    # Set t0 to current time
    t0 = time()
    astar = Astar_search(state[i])
    # Get the time t1 after executing the breadth_first_search method
    t1 = time() - t0
    print("A*:    ", astar)
    print("Space: ", Puzzle.num_of_instances)
    print("Time:  ", t1)
    print("Bonus task done by Utkarsh Gupta (A2305217557)")
    print()
    print("----------------------------------------")
```

```
----------------------------------------

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

Solution path in the A-star algorithm is:
[1, 3, 4]
[8, 6, 2]
[7, 0, 5]

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

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

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

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

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

A*:     ['U', 'R', 'U', 'L', 'D']
Space:  16
Time:   0.0008704662322998047
Bonus task done by Utkarsh Gupta (A2305217557)

----------------------------------------
----------------------------------------

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

Solution path in the A-star algorithm is:
[2, 8, 1]
[0, 4, 3]
[7, 6, 5]

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

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

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

[8, 1, 3]
[2, 4, 0]
```

```
[7, 6, 5]

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

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

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

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

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

A*:     ['U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
Space:  42
Time:   0.0008480548858642578
Bonus task done by Utkarsh Gupta (A2305217557)

----------------------------------------
----------------------------------------

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

Solution path in the A-star algorithm is:
[2, 8, 1]
[4, 6, 3]
[0, 7, 5]

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

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

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

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

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

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

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

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

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

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

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

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

A*:     ['R', 'U', 'L', 'U', 'R', 'R', 'D', 'L', 'L', 'U', 'R', 'D']
Space:  95
Time:   0.005315542221069336
Bonus task done by Utkarsh Gupta (A2305217557)

----------------------------------------
```