

Solving 8 – Puzzle using A* Search Algorithm

PROJECT DOCUMENTATION REPORT

PROGRAMMING PROJECT 1

ITCS 6150 – Intelligent Systems

Submitted to

Prof. Dewan T. Ahmed

Submitted by

Shimpli Kalse
801202853

Devyani Barde
801208619

Table of Contents

1.	PROBLEM FORMULATION	3
1.1	Introduction.....	3
	8 Puzzle Problem	3
	A* Search Algorithm.....	3
2.	PROGRAM STRUCTURE.....	4
2.1	Global Variables.....	4
2.2	Functions and Procedures	4
2.3	Source Code	6
2.4	Result Summary.....	13
	CONCLUSION.....	30
	REFERENCES.....	30

1. PROBLEM FORMULATION

1.1 Introduction

8 Puzzle Problem

The 8 puzzle problem is a puzzle invented by Noyes Palmer Chapman. It is played on a 3-by-3 grid with 8 square blocks labeled from 1 to 8 and a blank square. The goal is to rearrange the blocks so that they are in order. We are permitted to slide the blocks horizontally or vertically into the blank space. The following shows the sequence of legal moves from initial to goal position.

1 3	1 3	1 2 3	1 2 3	1 2 3
4 2 5	=>	4 2 5	=>	4 5
7 8 6	7 8 6	7 8 6	7 8 6	7 8 6

A* Search Algorithm

A* search is an algorithm that searches for the shortest path from the initial to goal node. A* search combines two evaluation functions from Greedy search algorithm and uniform cost search algorithm. $h(n)$ is the minimized estimated cost to the goal used by greedy search and $g(n)$ is the minimized cost of path used in uniform cost search. A* search combines the two evaluation functions simply by summing them:

$$f(n) = g(n) + h(n)$$

Since $g(n)$ gives the path cost from the start node to node n , and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have

$f(n)$ = estimated cost of the cheapest solution through n

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of f . [1] The explanation of three functions used in A* search:

- $h(n)$: 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. [2]
 - $g(n)$: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell. [2]
 - $f(n)$: it is the sum of $g(n)$ and $h(n)$. So, $f(n) = g(n) + h(n)$ [2]

Different heuristics can be used to find solution of one particular problem. In this project we will be solving the 8 puzzle problem with A* search algorithm. We will be using two heuristics namely Manhattan Distance and Misplaced Tiles.

2. PROGRAM STRUCTURE

2.1 Global Variables

Variable name	Variable type
Initial_state	Integer list (user input for initial state from [0-8])
goal	Integer list (user input for goal state from [0-8])
bestpath	Integer list of state space after every optimal move
expanded	Integer value containing total number of expanded nodes.
tot_steps	Integer value that holds the total number of moves made to reach the solution

2.2 Functions and Procedures

Function/procedure	Description
inputo()	This function takes input for initial_state ranging from 0-8 where 0 is treated as blank tile. If the user put any number out of range (0-8), it gives out the error and asks user to put the input again
goal()	This function takes input for desired goal state ranging from 0-8 where 0 is treated as blank tile. If the user put any number out of range (0-8), it gives out the error and asks user to put the input again
Best_path(state)	This function takes all the states as a and traverses through each of the states to find the best solution and returns the best optimal solution.
mh_dis(initial_state, goal)	This function computes the mahanttan distance of each element in the initial state by calculating the optimal moves each element should travel to reach the desired goal state
all()	This function checks whether the current state is a duplicate of some other state that has been traversed in the past and returns 1 if it is and 0 if it is not using Python's set form.
misplaced()	This function takes the current and target states as inputs and returns the number of tiles that have been lost. If no tiles are misplaced, the result is nil.

Solving 8 – Puzzle using A* Search Algorithm

Lable(parameter)	This function returns the single integer value of the coordinates after assigning integer values as coordinates to any input passed as a parameter.
eval(Initial_state, goal)	The manhattan heuristic is used to solve the 8-puzzle problem in this function. G(n), h(n), and f(n) are determined in this function, and a priority queue is used to store the node fn and place value as a key-value pair. If a node hasn't been explored yet, we use the merge sort technique to find the lowest cost node and explore it first. To search for duplicates, we use the all() function. To compile, we use Mh_dis(). After identifying the blank tile, we proceed to the next available steps. Then, create new nodes and decide whether or not this new node is the target state. h(n) calculates the amount of the number of steps each tile in the current state would take to achieve the target state using the formula $f(n)= g(n)+h(n)$. The step cost of each step is g(n).
eval_misplaced	Using the Misplaced Tiles heuristic, this function solves the 8-puzzle problem. G(n), h(n), and f(n) are determined in this function, and a priority queue is used to store the node fn and place value as a key-value pair. If a node hasn't been explored yet, we use the merge sort technique to find the lowest cost node and explore it first. To search for duplicates, we use the all() function. To calculate the cost of misplaced tiles, we use the misplaced tiles() function. We find the blank tile, then look for the next available steps, create new nodes, and decide whether or not this new node is the target state. In this case, $f(n)= g(n)+h(n)$ h(n) calculates the number of misplaced tiles in the current state versus the target state. The path cost for each move is g(n).
Deepcopy()	A copy of object is copied in another object. It means that any changes made to a copy of object do not reflect in the original object.

2.3 Source Code

Created on Wed Mar 10 21:53:45 2021

@author: Shimpi Kalse

@author: Devyani Barde

```
import numpy as np
```

```
import time
```

```
from copy import deepcopy
```

Taking input from user as an initial state and handling error for invalid input

```
def inputo():
```

```
    global initial_state
```

```
    initial_state = []
```

```
    print(" Input values from range 0-8 for desired goal state(0 is the blank tile)")
```

```
    for i in range(0,9):
```

```
        x = int(input("Enter values for tiles :"))
```

```
        initial_state.append(x)
```

```
    for ele in initial_state:
```

```
        if ele < 0 or ele > 8:
```

```
            print("Invalid Input")
```

```
            inputo()
```

Taking input from user as desired goal state and handling error for invalid input

```
def goalo():
```

```
    global goal
```

```
    goal = []
```

```
    print(" Input values from range 0-8 for desired goal state(0 is the blank tile)")
```

```
    for i in range(0,9):
```

```
        x = int(input("Enter values for tiles :"))
```

```
        goal.append(x)
```

Solving 8 – Puzzle using A* Search Algorithm

```
for ele in goal:  
    if ele < 0 or ele > 8:  
        print("Invalid Input")  
        goalo()  
# function to take the input of initial states and analyzing the best path to goal state  
def best_path(state):  
    bestsolution = np.array([], int).reshape(-1, 9)  
    count = len(state) - 1  
    while count != -1:  
        bestsolution = np.insert(bestsolution, 0, state[count]['initial_state'], 0)  
        count = (state[count]['parent_node'])  
    return bestsolution.reshape(-1, 3, 3)  
# function to calculate Manhattan distance cost between initial tile(input) and final goal  
def mh_dis(initial_state, goal):  
    u = abs(initial_state // 3 - goal // 3)  
    v = abs(initial_state % 3 - goal % 3)  
    mhcost = u + v  
    return sum(mhcost[1:])  
# function to check the distinctiveness of the iteration(i) state, to check if it was previously traversed  
def all(checkarray):  
    set=[]  
    for i in set:  
        for checkarray in i:  
            return 1  
    else:  
        return 0  
# function to compute the number of misplaced tiles between the given state and goal state  
def misplaced(initial_state,goal):
```

Solving 8 – Puzzle using A* Search Algorithm

```
mscost = np.sum(initial_state != goal) - 1
return mscost if mscost > 0 else 0

# function to determine the coordinates of every state(initial or goal)
def coordinates(initial_state):
    pos = np.array(range(9))
    for w, s in enumerate(initial_state):
        pos[s] = w
    return pos

# code for 8 puzzle evaluation using Manhattan heuristics
def eval(initial_state, goal):
    steps = np.array([('up', [0, 1, 2], -3), ('down', [6, 7, 8], 3), ('left', [0, 3, 6], -1), ('right', [2, 5, 8], 1)])
    dtype = [('move', str, 1), ('position', list), ('head', int)]
    state1 = [('initial_state', list), ('parent_node', int), ('gn', int), ('hn', int)]

    # initializing the variables :parent_node, gn and hn where hn is manhattan distance
    function call
    costg = coordinates(goal)
    parent_node = -1
    gn = 0
    hn = mh_dis(coordinates(initial_state), costg)
    state = np.array([(initial_state, parent_node, gn, hn)], state1)

# using priority queues where position treated as keys and fn as value.
    priorityq = [('position', int), ('fn', int)]
    priority = np.array([(0, hn)], priorityq)
    while 1:
        priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])
        position, fn = priority[0]
        priority = np.delete(priority, 0, 0)
        # sorting priority queue using merge sort
        initial_state, parent_node, gn, hn = state[position]
```

Solving 8 – Puzzle using A* Search Algorithm

```
initial_state = np.array(initial_state)

# code to spot the blank square in input

blank = int(np.where(initial_state == 0)[0])

gn = gn + 1

d = 1

start_time = time.time()

for s in steps:

    d = d + 1

    if blank not in s['position']:

        # generate new state as copy of current

        openstates = deepcopy(initial_state)

        openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank]

        # The distinct function is called to check previous traversal

        if ~(np.all(list(state['initial_state']) == openstates, 1)).any():

            end_time = time.time()

            if ((end_time - start_time) > 1):

                print(" 8 puzzle can not be solved for given set of input ! \n")

                exit

        # calls the mh_dis function to calculate the manhattan distance

        hn = mh_dis(coordinates(openstates), costg)

        # generate and add new state in the list

        q = np.array([(openstates, position, gn, hn)], state1)

        state = np.append(state, q, 0)

        # f(n): addition of the cost to reach the node plus cost to reach goal from current node

        fn = gn + hn

        q = np.array([(len(state) - 1, fn)], priorityq)

        priority = np.append(priority, q, 0)
```

Solving 8 – Puzzle using A* Search Algorithm

```
# Checking if the node in openstates are matching the goal state given by user.

if np.array_equal(openstates, goal):
    print(' 8 puzzle can be solved for given set of input ! \n')
    return state, len(priority)

return state, len(priority)

# code for 8 puzzle evaluation using misplaced tiles in initial_state

def eval_misplaced(initial_state, goal):

    steps = np.array([('up', [0, 1, 2], -3), ('down', [6, 7, 8], 3), ('left', [0, 3, 6], -1), ('right', [2, 5, 8], 1),])

    dtype = [('move', str, 1), ('position', list), ('head', int)]

    state1 = [('initial_state', list), ('parent_node', int), ('gn', int), ('hn', int)]

    costg = coordinates(goal)

    # initializing the parent node, gn and hn, where hn is misplaced function call

    parent_node = -1

    gn = 0

    hn = misplaced(coordinates(initial_state), costg)

    state = np.array([(initial_state, parent_node, gn, hn)], state1)

#using priority queue where position treated as keys and fn as value.

priorityq = [('position', int), ('fn', int)]

priority = np.array([(0, hn)], priorityq)

while 1:

    priority = np.sort(priority, kind='mergesort', order=['fn', 'position'])

    position, fn = priority[0]

    # sort priority queue using merge sort,the first element is picked for exploring.

    priority = np.delete(priority, 0, 0)

    initial_state, parent_node, gn, hn = state[position]

    initial_state = np.array(initial_state)

    # Identify the blank square in input
```

Solving 8 – Puzzle using A* Search Algorithm

```
blank = int(np.where(initial_state == 0)[0])

# Increase cost g(n) by 1

gn = gn + 1

c = 1

start_time = time.time()

for s in steps:

    c = c + 1

    if blank not in s['position']:

        # generate new state as copy of current

        openstates = deepcopy(initial_state)

        openstates[blank], openstates[blank + s['head']] = openstates[blank + s['head']], openstates[blank]

        # The check function is called, if the node has been previously explored or not.

        if ~(np.all(list(state['initial_state']) == openstates, 1)).any():

            end_time = time.time()

            if ((end_time - start_time) > 1):

                print(" 8 puzzle can NOT be solved for given set of input ! \n")

                break

        # calls the Misplaced function to compute the cost

        hn = misplaced(coordinates(openstates), costg)

        # generate and add new state in the list

        q = np.array([(openstates, position, gn, hn)], state1)

        state = np.append(state, q, 0)

        # f(n) is the sum of cost to reach node and the cost to reach fromt he node to the goal state

        fn = gn + hn

        q = np.array([(len(state) - 1, fn)], priorityq)

        priority = np.append(priority, q, 0)

        # Checking if the node in openstates are matching the goal state.

        if np.array_equal(openstates, goal):
```

Solving 8 – Puzzle using A* Search Algorithm

```
        print('8 puzzle can be solved for given set of input ! \n')
        return state, len(priority)

    return state, len(priority)

# -----Taking inputs and calling defined functions as per choice provided -----

# function call to ask for input

inputo()

goalo()

n = int(input("1. Manhattan distance \n2. Misplaced tiles \n"))

if(n!=1 and n!=2):

# handling wrong input from user

    print('please choose correct input, either 1 or 2')

    n = int(input("1. Manhattan distance \n2. Misplaced tiles \n"))

    if(n ==1 ):

        state, visited = eval(initial_state, goal)

        bestpath = best_path(state)

        print(str(bestpath).replace('[', ' ').replace(']', ''))

        tot_steps = len(bestpath) - 1

        print('Steps to reach goal:',tot_steps)

        expanded = len(state) - visited

        print('Total nodes expanded: ',expanded, "\n")

        print('Total generated:', len(state))

    if(n == 2):

        state, visited = eval_misplaced(initial_state, goal)

        bestpath = best_path(state)

        print(str(bestpath).replace('[', ' ').replace(']', ''))

        tot_steps = len(bestpath) - 1

        print('Steps to reach goal:',tot_steps)

        expanded = len(state) - visited
```

Solving 8 – Puzzle using A* Search Algorithm

```
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))
```

2.4 Result Summary

Input 1:

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

1. Manhattan Distance

The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code cell contains the following Python code:

```
tot_steps = len(bestpath) - 1
print('Steps to reach goal:',tot_steps)
expanded = len(state) - visited
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))
```

Below the code cell, there are two sections of user input. The first section is for tiles 1 through 8, with prompts like "Enter values for tiles :1", "Enter values for tiles :2", etc. The second section is for tiles 1 through 8 again, with prompts like "Input values from range 0-8 for desired goal state(0 is the blank tile)", "Enter values for tiles :1", etc.

Solving 8 – Puzzle using A* Search Algorithm

In this screenshot of a Jupyter Notebook, the code cell contains the following text:

```
Enter values for tiles :8  
Enter values for tiles :6  
Enter values for tiles :4  
Enter values for tiles :7  
Enter values for tiles :5  
Enter values for tiles :0  
1. Manhattan distance  
2. Misplaced tiles  
1  
8 puzzle can be solved for given set of input !  
  
1 2 3  
7 4 5  
6 8 0  
  
1 2 3  
7 4 0  
6 8 5  
  
1 2 3  
7 0 4  
6 8 5  
  
1 2 3  
7 8 4  
6 0 5  
  
1 2 3  
7 8 4
```

In this screenshot of a Jupyter Notebook, the code cell contains the following text:

```
1 2 3  
7 8 4  
0 6 5  
  
1 2 3  
0 8 4  
7 6 5  
  
1 2 3  
8 0 4  
7 6 5  
  
1 2 3  
8 6 4  
7 0 5  
  
1 2 3  
8 6 4  
7 5 0  
Steps to reach goal: 8  
Total nodes expanded: 9  
  
Total generated: 19
```

2.Misplaced_tiles

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two code cells in a Google Colab notebook titled "Untitled2.ipynb".

Code Cell 1:

```
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :7
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :8
Enter values for tiles :6
Enter values for tiles :4
Enter values for tiles :7
Enter values for tiles :5
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !

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

Code Cell 2:

```
2
8 puzzle can be solved for given set of input !

1 2 3
7 4 5
6 8 0

1 2 3
7 4 0
6 8 5

1 2 3
7 0 4
6 8 5

1 2 3
7 8 4
6 0 5

1 2 3
0 8 4
7 6 5

1 2 3
8 0 4
```

Solving 8 – Puzzle using A* Search Algorithm

```
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
0 6 5
1 2 3
0 8 4
7 6 5
1 2 3
8 0 4
7 6 5
1 2 3
8 6 4
7 0 5
1 2 3
8 6 4
7 5 0
Steps to reach goal: 8
Total nodes expanded: 23
Total generated: 44
```

Input2:

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

1. Manhattan Distance:

```
File Edit View Insert Runtime Tools Help
+ Code + Text
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :2
Enter values for tiles :8
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :5
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :3
Enter values for tiles :2
Enter values for tiles :1
Enter values for tiles :8
Enter values for tiles :0
Enter values for tiles :4
Enter values for tiles :7
Enter values for tiles :5
Enter values for tiles :6
1. Manhattan distance
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !
```

Solving 8 – Puzzle using A* Search Algorithm

In this screenshot of Google Colab, we see the start of an 8-puzzle solution. The code prompts for tile values and defines Manhattan distance and misplaced tile metrics. It then prints out several initial states of the 8-puzzle board:

```
Enter values for tiles :7  
Enter values for tiles :5  
Enter values for tiles :6  
1. Manhattan distance  
2. Misplaced tiles  
1  
8 puzzle can be solved for given set of input !  
2 8 1  
3 4 6  
7 5 0  
  
2 8 1  
3 4 0  
7 5 6  
  
2 8 1  
3 0 4  
7 5 6  
  
2 0 1  
3 8 4  
7 5 6  
  
0 2 1  
3 8 4  
7 5 6  
  
3 2 1
```

In this screenshot of Google Colab, the 8-puzzle problem has been solved. The code shows the steps taken to reach the goal state, the total number of nodes expanded, and the total number of generated states.

```
2 8 1  
3 4 0  
7 5 6  
  
2 8 1  
3 0 4  
7 5 6  
  
2 0 1  
3 8 4  
7 5 6  
  
0 2 1  
3 8 4  
7 5 6  
  
3 2 1  
0 8 4  
7 5 6  
  
3 2 1  
8 0 4  
7 5 6  
Steps to reach goal: 6  
Total nodes expanded: 6  
Total generated: 13
```

2. Misplaced_tile

Solving 8 – Puzzle using A* Search Algorithm

In this screenshot of Google Colab, the user has defined a function to solve an 8-puzzle. The code prompts the user to enter values for tiles from 0 to 8, defining 0 as the blank tile. It then calculates Manhattan distance and misplaced tiles for each state. Finally, it concludes that an 8-puzzle can be solved for the given input. The input provided is:

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

In this screenshot of Google Colab, the A* search algorithm is running. It shows several intermediate states of the 8-puzzle, including:

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

At the bottom, the algorithm displays performance metrics:

```
Steps to reach goal: 6  
Total nodes expanded: 7  
Total generated: 15
```

Input 3

Initial state:	Goal State:
7 2 4	1 2 3
5 6	4 5 6
8 3 1	7 8

1. Manhattan Distance

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two consecutive screenshots of a Google Colab notebook titled "Untitled2.ipynb".

Screenshot 1: The code cell contains the following pseudocode for the A* search algorithm:

```
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :7
Enter values for tiles :2
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :0
Enter values for tiles :6
Enter values for tiles :8
Enter values for tiles :3
Enter values for tiles :1
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !
```

Screenshot 2: The code cell contains the following output of generated initial states:

```
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !

7 2 4
5 0 6
8 3 1

7 2 4
5 3 6
8 0 1

7 2 4
5 3 6
8 1 0

7 2 4
5 3 0
8 1 6

7 2 4
5 0 3
8 1 6

7 2 4
0 5 3
8 1 6

0 2 4
7 5 3
```

Solving 8 – Puzzle using A* Search Algorithm

```
1 4 5  
7 8 6  
  
0 2 3  
1 4 5  
7 8 6  
  
1 2 3  
0 4 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  
Steps to reach goal: 28  
Total nodes expanded: 282  
Total generated: 452
```

2 Misplaced Tile

```
Input values from range 0-8 for desired goal state(0 is the blank tile)  
Enter values for tiles :7  
Enter values for tiles :2  
Enter values for tiles :4  
Enter values for tiles :5  
Enter values for tiles :0  
Enter values for tiles :6  
Enter values for tiles :8  
Enter values for tiles :3  
Enter values for tiles :1  
Input values from range 0-8 for desired goal state(0 is the blank tile)  
Enter values for tiles :1  
Enter values for tiles :2  
Enter values for tiles :3  
Enter values for tiles :4  
Enter values for tiles :5  
Enter values for tiles :6  
Enter values for tiles :7  
Enter values for tiles :8  
Enter values for tiles :0  
1. Manhattan distance  
2. Misplaced tiles  
2  
8 puzzle can be solved for given set of input !  
  
7 2 4  
5 0 6  
8 3 1
```

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two instances of the Google Colab interface. Both instances have the same title bar and toolbar. The top instance shows code for calculating Manhattan distance and checking for misplaced tiles, followed by several initial state configurations of the 8-puzzle. The bottom instance shows the continuation of the A* search process, including generated states and statistics like steps to goal and total nodes expanded.

```
+ Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !
7 2 4
5 0 6
8 3 1

7 2 4
5 3 6
8 0 1

7 2 4
5 3 6
8 1 0

7 2 4
5 3 0
8 1 6

7 2 4
0 5 3
8 1 6

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

0 2 3
1 4 5
7 8 6

1 2 3
0 4 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
Steps to reach goal: 20
Total nodes expanded: 3812
Total generated: 5886
```

Input 4

Initial state: 132465780

Goal state: 123456780

1. Manhattan Distance

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two sessions of a Google Colab notebook titled "Untitled2.ipynb".

Session 1 (Top):

- Code cell:

```
print("Steps to reach goal:",tot_steps)
expanded = len(state) - visited
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))
```
- Output cell:

```
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :2
Enter values for tiles :4
Enter values for tiles :6
Enter values for tiles :5
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
```
- Session end time: 7:06 PM
Date: 3/12/2021

Session 2 (Bottom):

- Code cell:

```
1. Manhattan distance
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !
```
- Output cell:

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

1 3 2
4 6 0
7 8 5

1 3 2
4 0 6
7 8 5

1 0 2
4 3 6
7 8 5

0 1 2
4 3 6
7 8 5

4 1 2
0 3 6
7 8 5
```
- Session end time: 7:07 PM
Date: 3/12/2021

Solving 8 – Puzzle using A* Search Algorithm

```
0 1 2  
4 5 3  
7 8 6  
  
1 0 2  
4 5 3  
7 8 6  
  
1 2 0  
4 5 3  
7 8 6  
  
1 2 3  
4 5 0  
7 8 6  
  
1 2 3  
4 5 6  
7 8 0  
Steps to reach goal: 18  
Total nodes expanded: 591  
Total generated: 932
```

2. Misplaced Tile

```
print('Total nodes expanded:', expanded, "\n")
print('Total generated:', len(state))

Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :2
Enter values for tiles :4
Enter values for tiles :6
Enter values for tiles :5
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
```

Solving 8 – Puzzle using A* Search Algorithm

The image shows two screenshots of a Google Colab notebook titled "Untitled2.ipynb".

Session 1 (Top):

- Code cell:

```
Enter values for tiles :8
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !

1 3 2
4 6 5
7 8 0

1 3 2
4 6 0
7 8 5

1 3 0
4 6 2
7 8 5

1 0 3
4 6 2
7 8 5

0 1 3
4 6 2
7 8 5

4 1 3
0 6 2
7 8 5
```
- Output cell:

```
7:11 PM 3/12/2021
```

Session 2 (Bottom):

- Code cell:

```
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

Steps to reach goal: 18
Total nodes expanded: 1774

Total generated: 2815
```
- Output cell:

```
7:12 PM 3/12/2021
```

Input 5

Initial state: 013425786

Goal state: 123456780

1. Manhattan Distance

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two consecutive screenshots of a Google Colab notebook titled "Untitled2.ipynb".

Code Snippet:

```
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))

Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :0
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :2
Enter values for tiles :5
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :6
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
1
8 puzzle can be solved for given set of input !

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

Steps to reach goal: 4
Total nodes expanded: 4
```

Execution Results:

The notebook displays the following output:

- Print statements: "Total nodes expanded: ", "Total generated:", and two sets of prompts for entering tile values.
- Comments: "1. Manhattan distance" and "2. Misplaced tiles".
- A message: "1"
- A success message: "8 puzzle can be solved for given set of input !"
- Four initial state configurations:

 - 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

- A final summary:
Steps to reach goal: 4
Total nodes expanded: 4

2 Misplaced Tile

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows two instances of a Google Colab notebook titled "Untitled2.ipynb". The notebook contains Python code for solving an 8-puzzle using the A* search algorithm. The code includes functions for calculating Manhattan distance and misplaced tiles, and a main loop that solves the puzzle for various initial states. The output shows the steps taken to reach the goal state from several different initial configurations.

```
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :0
Enter values for tiles :1
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :2
Enter values for tiles :5
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :6
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !
0 1 3
Type here to search 7:16 PM 3/12/2021
```



```
Enter values for tiles :0
1. Manhattan distance
2. Misplaced tiles
2
8 puzzle can be solved for given set of input !
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
Steps to reach goal: 4
Total nodes expanded: 4
Total generated: 10
Type here to search 7:16 PM 3/12/2021
```

Input6:

Initial state: 124376580

Goal State: 123456780

Solving 8 – Puzzle using A* Search Algorithm

1 Manhattan distance

A screenshot of a Google Colab notebook titled "Untitled2.ipynb". The code cell contains Python code for an A* search algorithm. The output shows a series of prompts for entering tile values from 0 to 8, indicating the algorithm is求解一个8-puzzle问题。下方显示了搜索树的遍历过程，包括节点数和生成节点数的统计。

```
print('Steps to reach goal:',tot_steps)
expanded = len(state) - visited
print('Total nodes expanded: ',expanded, "\n")
print('Total generated:', len(state))

Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :4
Enter values for tiles :3
Enter values for tiles :7
Enter values for tiles :6
Enter values for tiles :5
Enter values for tiles :8
Enter values for tiles :0
Input values from range 0-8 for desired goal state(0 is the blank tile)
Enter values for tiles :1
Enter values for tiles :2
Enter values for tiles :3
Enter values for tiles :4
Enter values for tiles :5
Enter values for tiles :6
Enter values for tiles :7
Enter values for tiles :8
```

A screenshot of a Google Colab notebook titled "Untitled2.ipynb". The code cell contains Python code for an A* search algorithm. The output shows the sequence of states generated during the search process, starting from the initial state [1, 2, 4, 3, 7, 0, 5, 8, 6] and progressing through various configurations until reaching the goal state [2, 3, 4, 1, 0, 7, 5, 8, 6].

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

1 2 4
3 7 0
5 8 6

1 2 4
3 0 7
5 8 6

1 2 4
0 3 7
5 8 6

0 2 4
1 3 7
5 8 6

2 0 4
1 3 7
5 8 6

2 3 4
1 0 7
5 8 6

2 3 4
```

Solving 8 – Puzzle using A* Search Algorithm

```
1 2 3  
4 0 6  
7 5 8  
  
1 2 3  
4 5 6  
7 0 8  
  
1 2 3  
4 5 6  
7 8 0  
Steps to reach goal: 20  
Total nodes expanded: 563  
Total generated: 921
```

2 Misplaced tile:

```
Input values from range 0-8 for desired goal state(0 is the blank tile)  
Enter values for tiles :1  
Enter values for tiles :2  
Enter values for tiles :4  
Enter values for tiles :3  
Enter values for tiles :7  
Enter values for tiles :6  
Enter values for tiles :5  
Enter values for tiles :8  
Enter values for tiles :0  
Input values from range 0-8 for desired goal state(0 is the blank tile)  
Enter values for tiles :1  
Enter values for tiles :2  
Enter values for tiles :3  
Enter values for tiles :4  
Enter values for tiles :5  
Enter values for tiles :6  
Enter values for tiles :7  
Enter values for tiles :8  
Enter values for tiles :0  
1. Manhattan distance  
2. Misplaced tiles  
2  
8 puzzle can be solved for given set of input !  
1 2 4
```

Solving 8 – Puzzle using A* Search Algorithm

The screenshot shows a Google Colab notebook titled "Untitled2.ipynb". The code implements the A* search algorithm to solve an 8-puzzle. The initial state is given as a 3x3 grid:

```

1 2 3
7 4 6
0 5 8

```

The goal state is:

```

1 2 3
0 4 6
7 5 8

```

The algorithm explores several states, including:

```

1 2 3
4 0 6
7 5 8

1 2 3
4 5 6
7 0 8

1 2 3
4 5 6
7 8 0

```

Output from the script:

```

Steps to reach goal: 20
Total nodes expanded: 3558
Total generated: 5553

```

The Colab interface includes tabs for "Code" and "Text", and a sidebar with "Comment", "Share", "Settings", and "S" (Save) buttons. The status bar at the bottom shows RAM usage, disk space, and editing mode.

Summary In table:

Input no		Steps to Reach the goal	Total nodes expanded	Total generated		Steps to Reach the goal	Total nodes expanded	Total generated
1	Initial state	8	9	19		8	23	44
	123745680							
	Goal state							
	123864750							
2	Initial State	6	6	13		6	7	15
	281346750							
	Goal State							
	321804756							
3	Initial State	20	282	452		20	3812	5886
	724506831							
	Goal State							
	123456780							
4	Initial State	18	591	932		18	1774	2815
	132465780							
	Goal State							
	123456780							
5	Initial State	4	4	10		4	4	10
	13425786							
	Goal State							
	123456780							
6	Initial State	20	563	921		20	3558	5553
	124376580							
	Goal State							
	123456780							

CONCLUSION

As per above results, the 8-puzzle problem is solved using A* algorithm which uses heuristics such as manhattan distance and misplaced tile. Also, number of nodes expanded in manhattan distance heuristics are less as compared to misplaced tiles in most of the cases.

REFERENCES

- [1] S. J. R. a. P. Norvig, Artificial Intelligence: A Modern Approach, Alan Apt, 1995.
- [2] "What is the A* algorithm?", [Online]. Available: <https://www.educative.io/edpresso/what-is-the-a-star-algorithm>. [Accessed 12 03 2021].
- [3] "PY4E," [Online]. Available: www.py4e.com. [Accessed 12 03 2021].