Introduction to AI

8-Puzzle Solver

Report

By Tanu Singh

Roll number 202401100300262

Introduction

The 8-puzzle is a sliding puzzle consisting of a 3×3 grid with eight numbered tiles and one empty space. The objective is to move the tiles to reach a goal state by shifting the empty space in different directions. This problem is a fundamental example in artificial intelligence and search algorithms.

Methodology

The solution is implemented using the A* (A-star) algorithm, which is a heuristic-based search technique. The Manhattan Distance heuristic is used to estimate the cost to reach the goal state efficiently. The steps involved in solving the problem are:

1. Representing the puzzle as a 3×3 matrix.
2. Using a priority queue to explore the most promising states first.
3. Expanding the node with the lowest cost (f = g + h).
4. Generating possible moves and tracking the path to the goal.

Code

The following Python code implements the 8-puzzle solver using the A* algorithm with the Manhattan distance heuristic:

```
import heapq

goal_state = (1, 2, 3, 4, 5, 6, 7, 8, 0)    # Goal state of the 8 puzzle

moves = [(-1, 0), (1, 0), (0, -1), (0, 1)]   # Directions for moving the empty space (0)   Up,
Down, Left, Right

def manhattan_distance(state):  # Function to calculate the Manhattan Distance
    distance = 0
    for i in range(9):
        value = state[i]
        if value == 0:
            continue
        target_row, target_col = divmod(value - 1, 3)
        current_row, current_col = divmod(i, 3)
        distance += abs(current_row - target_row) + abs(current_col - target_col)
    return distance

def get_neighbors(state):   # Function to generate the neighbors (valid moves)
    empty_index = state.index(0)
    row, col = divmod(empty_index, 3)
```

```
    neighbors = []

    for dr, dc in moves:
        new_row, new_col = row + dr, col + dc
        if 0 <= new_row < 3 and 0 <= new_col < 3:
            new_empty_index = new_row * 3 + new_col
            new_state = list(state)
            new_state[empty_index], new_state[new_empty_index] =
new_state[new_empty_index], new_state[empty_index]        # Swap the empty space (0)
with the adjacent tile
            neighbors.append(tuple(new_state))

    return neighbors

def a_star_search(start_state):   # A* search algorithm
    open_list = []      # Priority queue for A* (f_score, state, g_score, path)
    heapq.heappush(open_list, (manhattan_distance(start_state), start_state, 0, []))

    visited = set()          # Set to store visited states
    visited.add(start_state)

    while open_list:
        f_score, current_state, g_score, path = heapq.heappop(open_list)

        if current_state == goal_state:           # If goal state is reached, return the solution path
            return path + [current_state]

        for neighbor in get_neighbors(current_state):        # Get neighbors and process them
            if neighbor not in visited:
                visited.add(neighbor)
                new_g_score = g_score + 1
                new_f_score = new_g_score + manhattan_distance(neighbor)
                heapq.heappush(open_list, (new_f_score, neighbor, new_g_score, path +
[current_state]))

    return None  # If no solution is found

start_state = (1, 2, 3, 4, 0, 5, 7, 8, 6)   # Example input: the start state of the puzzle

solution = a_star_search(start_state)

if solution:                    # Printing the solution
    print("Solution found!")
```

```
    for step in solution:
        print(step[:3])
        print(step[3:6])
        print(step[6:])
        print("-" * 9)
else:
    print("No solution found.")
```

Output/Result

```
Solution found!
(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)
---------
```

References/Credits

1. Artificial Intelligence: A Modern Approach by Stuart Russell and Peter Norvig.

2. Online resources on A* Algorithm and 8-puzzle problem.