ASSIGNMENT 8

Report : Simulated Annealing to Solve the 8-Puzzle Problem

Introduction: The 8-puzzle problem is a classic problem in artificial intelligence and computer science, which involves rearranging numbered tiles on a 3x3 grid with one empty space. The objective is to reach a predefined goal state from a given initial state by sliding tiles into the empty space. Simulated Annealing (SA) is a probabilistic optimization algorithm that mimics the process of annealing in metallurgy. It is widely used to find approximate solutions to optimization problems when traditional methods are impractical or too computationally expensive.

Algorithm Overview: Simulated Annealing iteratively improves a solution by randomly selecting new states and accepting them based on a probability distribution. At each iteration, the algorithm evaluates the energy (or cost) of the current state using a heuristic function. If a new state has lower energy, it is always accepted. However, if the new state has higher energy, it may still be accepted with a certain probability, which decreases over time according to a cooling schedule. This probabilistic acceptance allows the algorithm to escape local optima and explore the solution space effectively.

Pseudo-code for Simulated Annealing: Pseudo-code for using Simulated Annealing to solve the 8-puzzle problem with both heuristics h1 (Number of Displaced Tiles) and h2 (Total Manhattan Distance):

```
plaintext

function simulated_annealing(initial_state, T0, alpha, max_iter):
    current_state = initial_state
    current_energy = calculate_energy(current_state)
    best_state = current_state
    best_energy = current_energy
    T = T0

    for iter = 1 to max_iter:
        new_state = generate_neighbor(current_state)
        new_energy = calculate_energy(new_state)
        delta_E = new_energy - current_energy

        if delta_E < 0 or random(0, 1) < exp(-delta_E / T):
            current_state = new_state
            current_energy = new_energy
            if current_energy < best_energy:
                best_state = current_state
                best_energy = current_energy

        T *= alpha

    return best_state

calculate_energy(state):
    # Use heuristic to calculate energy
    return heuristic(state)
```

```
    generate_neighbor(state):
        # Generate a neighboring state
        # by moving a random tile
        # into the empty space
        return new_state

    # Heuristic h1 (Number of Displaced Tiles)
    heuristic_h1(state):
        displaced_tiles = 0
        for each tile in state:
            if tile != goal_position(tile):
                displaced_tiles += 1
        return displaced_tiles

    # Heuristic h2 (Total Manhattan Distance)
    heuristic_h2(state):
        total_distance = 0
        for each tile in state:
            if tile != empty_space:
                goal_pos = goal_position(tile)
                total_distance += manhattan_distance(tile.position,
    goal_pos)
        return total_distance

    # Utility function to calculate Manhattan distance
    manhattan_distance(pos1, pos2):
        return abs(pos1.x - pos2.x) + abs(pos1.y - pos2.y)
```

Explanation:

- The `simulated_annealing` function iteratively improves the solution by generating neighboring states and accepting them based on the acceptance probability.
- The `calculate_energy` function evaluates the energy (or cost) of a state using a chosen heuristic.
- The `generate_neighbor` function generates a neighboring state by moving a random tile into the empty space.
- Two heuristics are defined: `heuristic_h1` calculates the number of displaced tiles, while `heuristic_h2` calculates the total Manhattan distance.
- `manhattan_distance` calculates the Manhattan distance between two positions on the grid.

Heuristic Comparison: Two commonly used heuristics for the 8-puzzle problem are h1 (Number of Displaced Tiles) and h2 (Total Manhattan Distance).

Heuristic h1 (Number of Displaced Tiles):

- **Description**: Counts the number of tiles that are not in their goal position.
- **Advantages**: Simple to implement and computationally inexpensive.
- **Disadvantages**: May not accurately reflect the distance from the goal state, especially for configurations where displaced tiles are close to their goal positions.

Heuristic h2 (Total Manhattan Distance):

- **Description**: Calculates the sum of the Manhattan distances of each tile from its goal position.
- **Advantages**: Takes into account the actual distance each tile needs to move to reach its goal position, often providing better estimates of the cost to reach the goal state.
- **Disadvantages**: Slightly more complex to implement and requires additional computation to calculate the Manhattan distances.

Comparison:

- **Accuracy**: h2 generally provides a more accurate estimate of the distance from the current state to the goal state compared to h1.
- **Computational Cost**: h1 is computationally cheaper since it only involves counting displaced tiles, while h2 requires calculating Manhattan distances.
- **Effectiveness**: h2 tends to guide the search process more effectively towards the goal state by considering the actual distances to be covered, potentially leading to faster convergence to a solution.
- **Robustness**: h2 is more robust across different puzzle configurations as it takes into account the spatial relationships between tiles.

Conclusion: Simulated Annealing, with carefully selected heuristics such as h2 (Total Manhattan Distance), provides a flexible and efficient approach to solving the 8-puzzle problem. While the choice of heuristic may depend on specific problem instances and computational constraints, h2 is generally preferred due to its better accuracy and effectiveness in guiding the search process towards the goal state.

Analysis of Time and Space Complexity:

- **Time Complexity**: The time complexity of Simulated Annealing depends on the number of iterations, the complexity of the energy function, and the generation of neighboring states. In the worst case, it is O(max_iter).
- **Space Complexity**: The space complexity primarily depends on the memory required to store the current and best states, as well as the neighbor states. It is typically O(1) since only a constant amount of memory is used.

Success Rate: The success rate of Simulated Annealing in solving the 8-puzzle problem depends on various factors, including the choice of initial state, the cooling schedule, and the parameters of the SA algorithm. Proper tuning of these parameters can improve the success rate, but SA does not guarantee an optimal solution.
