Solving the 8-Puzzle Problem Using Different Search Algorithms

Note By: Himanshi Liyanage

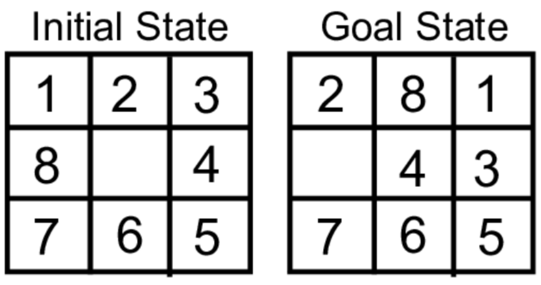
Faculty of Mathematics and Computer Science

University of Łódź, Poland

January 14, 2025

1 Problem Statement

The 8-puzzle problem is a classic AI problem where the objective is to transform an initial state into a goal state using valid moves. The initial and goal states are given below:



Initial State

1	2	3
8		4
7	6	5

Goal State

2	8	1
	4	3
7	6	5

2 Algorithms Used

We solve the 8-puzzle problem using the following algorithms:

- Breadth-First Search (BFS)
- Depth-First Search (DFS)
- Best-First Search
- A* Search

3 Python Implementations

3.1 Breadth-First Search (BFS)

```python
from collections import deque

def bfs(initial_state, goal_state):
    queue = deque([initial_state])
    visited = set()

    while queue:
        state = queue.popleft()
        visited.add(tuple(state))

        if state == goal_state:
            return state

        for neighbor in generate_neighbors(state):
            if tuple(neighbor) not in visited:
                queue.append(neighbor)

    return None

def generate_neighbors(state):
    # Implementation for neighbor generation
    pass
```

3.2 Depth-First Search (DFS)

```python
def dfs(initial_state, goal_state):
    stack = [initial_state]
    visited = set()

    while stack:
        state = stack.pop()
        visited.add(tuple(state))

        if state == goal_state:
            return state

        for neighbor in generate_neighbors(state):
            if tuple(neighbor) not in visited:
                stack.append(neighbor)

    return None
```

3.3 Best-First Search

```
import heapq

def heuristic(state, goal_state):
    # Example heuristic: Misplaced tiles
    return sum(1 for i, j in zip(state, goal_state) if i != j)

def best_first_search(initial_state, goal_state):
    heap = [(heuristic(initial_state, goal_state), initial_state)
       ]
    visited = set()

    while heap:
        _, state = heapq.heappop(heap)
        visited.add(tuple(state))

        if state == goal_state:
            return state

        for neighbor in generate_neighbors(state):
            if tuple(neighbor) not in visited:
                heapq.heappush(heap, (heuristic(neighbor,
                   goal_state), neighbor))

    return None
```

3.4 A* Search

```
def a_star_search(initial_state, goal_state):
    heap = [(0 + heuristic(initial_state, goal_state), 0,
       initial_state)]
    visited = set()

    while heap:
        f, g, state = heapq.heappop(heap)
        visited.add(tuple(state))

        if state == goal_state:
            return state

        for neighbor in generate_neighbors(state):
            if tuple(neighbor) not in visited:
                g_new = g + 1
                f_new = g_new + heuristic(neighbor, goal_state)
                heapq.heappush(heap, (f_new, g_new, neighbor))

    return None
```