LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

IA Planning TP 1

Halim Djerroud

revision 1.0

TP 1: Introduction to Automated Planning with the 8-puzzle

Objectives

- Discover the concept of automated planning.
- Manipulate basic concepts: state, action, goal, plan.
- Understand the difference between BFS and DFS.

Problem Presentation: The 8-puzzle

A 3×3 grid with 8 numbered tiles and one empty cell (noted as b). An **action** moves an adjacent tile into the empty cell. A **plan** is a sequence of actions leading from the initial state to the goal state.

Exercise 1: Define the States

1. Represent a puzzle state as a 3×3 array.
2. Write an initial state and a simple goal state (for example, with two tiles swapped).
3. How many different states are possible? How many are reachable?

Exercise 2: Define the Actions

1. Describe in your own words an action "move a tile".
2. Give all possible actions from a state with b at the center, at an edge, and in a corner.
3. Explain the adjacency constraint.

Exercise 3: Search for a Plan by Hand (advanced version)

Initial state:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 8 & 7 & b \end{bmatrix}$$

Goal:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & b \end{bmatrix}$$

1. Find a short plan (sequence of actions) to solve this problem.
2. Verify each intermediate state.
3. Are there multiple possible plans?
4. Compare what BFS and DFS would produce.

Exercise 4: BFS vs DFS (concepts)

1. Explain the difference between BFS and DFS.
2. Which one guarantees the shortest plan?
3. Compare memory, time, and completeness.

Exercise 5: Heuristics (bonus)

1. Define the misplaced tiles heuristic.
2. Define the Manhattan distance.
3. Which is more informative?

Exercise 6: State Representation in Python

1. Represent an 8-puzzle state as a list of lists in Python.
2. Write a function `display(state)` that prints the 3×3 grid nicely.
3. Test this function with the initial state and goal state.

Exercise 7: Successor Generation

1. Write a function `successors(state)` that returns the list of states obtained by moving a tile into the empty cell.
2. Test the function on the initial state.

Exercise 8: BFS and DFS in Python

1. Write a function `bfs(initial_state, goal_state)` that returns a plan (sequence of states).
2. Write a function `dfs(initial_state, goal_state, max_depth)` that returns a plan if found.
3. Compare the results on a small example.

Exercise 9: Heuristics (bonus)

1. Write a function `h_misplaced(state, goal)` = number of misplaced tiles.
2. Write a function `h_manhattan(state, goal)` = sum of Manhattan distances.
3. Test both heuristics on the initial state.

Exercise 10: Implement A*

1. Implement the A* algorithm in Python with a priority queue (`heapq`).
2. Use as heuristic: (a) misplaced tiles, (b) Manhattan distance.
3. Compare the length of plans found with BFS.

Exercise 11: Experimental Comparison

1. Measure execution time and number of nodes explored for BFS, DFS, and A*.
2. Create a comparative table for different initial states.
3. Conclude on the advantages of each algorithm.

Exercise 12: Plan Visualization

1. Write a function `display_plan(plan)` that displays each state in the found plan, with a delay (e.g., 0.5s).
2. Test with a plan found by BFS.

Exercise 13: Extension to the 15-puzzle

1. Adapt the functions for a 4×4 grid with 15 tiles and 1 empty cell.
2. Test BFS (on small shuffles) and A* (with Manhattan heuristic).

Exercise 14: Random State Generation

1. Write a function that generates a random initial state by applying n moves from the goal state.
2. Use this function to automatically test your algorithms.