Artificial Intelligence II - Homework 2
8-puzzle

Jonáš Petrovský

Faculty of Business and Economics,
Mendel University in Brno,
Czech Republic
jond@post.cz

November 8, 2015

1 Introduction

The goal of the work is to implement a solution to the "8-puzzle" using two assigned heuristics. There is a board 3x3 with 8 tiles inside (and one blank square). The goal is to rearrange all tiles so they are in order.

2 Solution design

The 8-puzzle can by solved by creating and searching a state space (tree). But another way exists – to use a heuristic. This way, the optimal (or even any) solution may not be found, but the process is much faster. Following two heuristics are considered: h1 – min. wrong placings, h2 – min. Manhattan distance.

A board state is represented by a hash of <tile name> => <tile x,y coordinates>, "moved tile" field and "h_value" field. Tiles are named 1–8, a blank square is named 0 and behaves like a regular tile.

3 Implementation

The program was implemented in Python. A state is saved in a variable with the following structure (example for goal state).

```
self.goal_state = {
        'tiles': {
            1: (1,1), 2: (1,2), 3: (1,3),
            8: (2,1), 0: (2,2), 4: (2,3),
            7: (3,1), 6: (3,2), 5: (3,3)
        },
```

```
            'moved_tile': None,
            'h_value': 0
}
```

All visited states are saved in "solution_path" list (in order). There is a main class `PuzzleWorld` with 3 public methods:

- `solve_given_puzzle(start_state, h_type)`

- `solve_random_puzzle(h_type)`

- `bulk_solve(number_of_puzzles, h_type)`

The main method `solve_given_puzzle` accepts a start state and tries to find a solution using given heuristic.

1. Reset solution path.

2. Insert a start state into solution path.

3. While tiles in current state != tiles in goal state:

   (a) Find possible moves towards zero tile.
   (b) For every move create a new state.
   (c) Check if the states are valid – so far unvisited (not present in solution path). If there are no valid states, a solution cannot be found – a message of failure is displayed and the program ends.
   (d) Calculate heuristic value for every new valid state.
   (e) Choose a state with the lowest heuristic value – insert it into solution path and mark it as current.

If the goal state was reached, a message of success and number of performed moves (length of `solution_path - 1`) are displayed.

The class `PuzzleHeuristics` implements the calculation of heuristics:

1. Hamming distance – number of tiles out of place.

2. Manhattan distance – the sum of minimum number of steps (vertical or horizontal) to move every tile in its goal position.

For both heuristics we choose the minimal value.

4   Results

The program can be run by typing the following command into the operating system's terminal (Python 2.7 has to be installed on the system):
`python run-2.py`
What the program should do must be stated in the run-2.py source code (there are no parameters).

4.1 Hamming distance

5 References

FRANCI, R. Jealous Husbands Crossing the River: A Problem from Alcuin to Tartaglia, *From China to Paris: 2000 Years Transmission of Mathematical Ideas*, edited by Yvonne Dold-Samplonius, Joseph W. Dauben, Menso Folkerts, and Benno van Dalen, Stuttgart: Franz Steiner Verlag, 2002, pp. 289–306. ISBN 3-515-08223-9.