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. It has size of 9! = 362,880 states. Only half of the states are reachable from any given (start) state – are solvable (Latombe, 2011). The number is quite large and for 15-puzzle (16! possible configurations) it's much higher.

But another way of solving the puzzle 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 used: 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 4 public methods:

- `solve_given_puzzle(start_state, h_type)`
- `solve_random_puzzle(h_type)`
- `bulk_solve(number_of_puzzles, h_type)`
- `bulk_solve_compare(number_of_puzzles)`

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

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.
    (f) Return solution path.

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 and 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 (the script has no parameters).

4.1 Hamming distance

5 References

Latombe, J.C. Search problems. *Stanford AI Lab* [online]. 2011 [accessed 8.11.2015]. Available at: `http://ai.stanford.edu/~latombe/cs121/2011/slides/B-search-problems.ppt`