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, which has a size of 9! = 362,880 states. Only half of the states are reachable from any given (start) state – are solvable (Latombe, 2011). Even so, the number is quite large and for 15-puzzle (16! possible states) it is 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 (Hamming distance), 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 a 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 following 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)`
- `generate_start_state()`

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

1. Reset a solution path.
2. Insert a start state into the 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 created states are valid – so far unvisited (not present in the 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 the 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. Then the solution path is returned to the calling method.

Class **PuzzleHeuristics** implements the calculation of heuristics:

1. Hamming distance – number of tiles out of the correct 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.

Class **PuzzleStatistics** is used for performing and evaluating simulations. Method `bulk_solve` solves given number of randomly generated puzzles by using given heuristic. Method `bulk_solve_and_compare` does the same but every puzzle is solved by both heuristics and data about solutions are being stored and later processed and displayed.

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 Results

The method `bulk_solve_and_compare` was used for experiments. Table 1 shows results of simulations for 1,000 and 100,000 random puzzles (start states).

Notes: h1 = Hamming distance, h2 = Manhattan distance, efficiency = average number of moves needed to find a solution, min. moves = global minimal number of moves for finding any solution.

Table 1: Results of simulations for many random start states

N. of puzzles	1,000	100,000
h1 solved [%]	0.9	0.91
h2 solved [%]	9.9	10.89
h1 efficiency	80	91
h2 efficiency	68	62
h1 min. moves	18	2
h2 min. moves	10	2

It must be noted that results for different simulations of 1,000 puzzles vary quite a lot. But a simulation for 100,000 puzzles is sufficient to determine how is each heuristic successful in solving the 8-puzzle problem.

We can see that h2 is much better (about 10x) than h1 in terms of percentage of puzzles solved (11 vs 1 %). H2 average efficiency is also better than h1, but the difference is not so huge (62 vs. 91 – about 33 %).

Regarding global min. number of moves can be said that both heuristic can solve some puzzles in a very short time. On the basis of performed 1,000 simulations can be seen that h2 sometimes has the number lower but other times it is the same. But it seems that h1 has the number never lower.

In conclusion, even h2 did not manage to solve more than about 10 % of given start states (puzzles, board configurations). Because precisely 50 % of start states have a solution, it is not a very good result. But the algorithm will be much faster than human in solving selected puzzles or human trying to solve all the puzzles. Other algorithms (like A*) or heuristics could give us a better result, but these are not the topic of this work.

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`