Artificial Intelligence II - Homework 1
Missionaries and cannibals problem

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

October 24, 2015

1 Introduction

The goal of the work is to implement an own solution to the "Missionaries and cannibals problem". You can see the problem description here: `https://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem#The_problem`

2 Solution design

The problem can by solved by creating and searching a state space (tree). A single state has the following form: ((a1,b1,c1), (a2,b2,c2)), where a1, b1, c1 are numbers of missionaries, cannibals and boats on the left bank, and a2, b2, c2 is the same for the right bank.

We need a starting and a target state, operators (actions) and a criterion for finding the cheapest path (here it is a number of river crossings).

We can use Breadth-first search (BFS) or Depth-first search (DFS). BFS finds the optimal solution among all states in the state tree, while DFS finds the first available solution (which doesn't have to be optimal).

3 Implementation

The program was implemented in Python. There is a main class `MissionaryWorld`, which in its constructor accepts number of cannibals (same as missionaries) for choosing whether to solve 2+2, 3+3 or 4+4 problem.

The state tree nodes (states) are saved in `state_tree['nodes']` variable under generated IDs and have a following structure (example for root node):

```
root_node = {'left': self.start_state[0], 'right': self.start_state[1]
 'id': 0,  'parent_id': None, 'level': 0, 'used_op': None}
```

The main method `generate_tree` accepts a search type (DFS or BFS) as its parameter and finds the desired solution.

1. Create a data structure for nodes (queue for BFS and stack for DFS)
2. Insert the root node (starting state)
3. While the data structure is not empty:
   (a) Generate new possible states from the examined node: `_generate_tree_level`
   (b) Add new nodes to the tree. If the new node is the target state, end the program and show the solution.
   (c) Only for DFS – check if the level is not too deep. If it is, do not add any more children (skip to the next node in the stack).
   (d) Add new nodes to the data structure.

A method `_check_new_state(current_state, operator)` is called inside the `_generate_tree_level(current_state)` method and checks for all operators if the generated state is valid – does not violate any conditions: enough people for transport, no more cannibals than missionaries on both banks, the new state is not the same as the starting state, the new state is not the same as the previous state of the current state.

Finally a method `_show_solution(target_node)` reconstructs the path and counts total number of steps. Output format description:
`<level n.> (<node ID>): <state left>, <state right> <- <used operator>`
If no solution is found, the according message is displayed.

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.py <number of cannibals> <search type>`
The first parameter can be 2, 3 or 4 and the second BFS or DFS. Example for a default behaviour (if no arguments are entered): `python run.py 3 BFS`

4.1 3+3

The original problem of 3 missionaries and 3 cannibals can be solved easily. Both BFS and DFS have found the optimal solution.

- Path price (number of river crossings): 11
- BFS – target state was the 26. added state.
- DFS – target state was the 15. added state.

For this problem type is better DFS because of lower number of added states needed (but the difference is not too large).

4.2 2+2

The other problem type was also solved easily (both by BFS and DFS).

- Path price (number of river crossings): 5
- BFS – target state was the 15. added state.
- DFS – target state was the 10. added state.

For this problem type is better DFS because of lower number of added states needed (but the difference is not too large).

If we compare the 2+2 with 3+3 problem, we can see than for BFS the 3+3 is about 73 % (26 vs 15) more complex than 2+2. But the increase has virtually no impact on total performance (with regard to asymptotic computational complexity), so we can say that these problems are similarly difficult to solve.

4.3 4+4

This problem type could not be solved. The program ended after adding 19 states (both for DFS and BFS). After removing the condition that it is not possible to get into the starting state again, the program would run indifinetely.

Edouard Lucas in his Récreations mathématiques proves that for 4 (or 5) couples to cross, the boat must hold 3 people (Franci, 2002). The reason for this is probably the even number of people to cross and too small number of seats in the boat.

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.