## Planning heuristic evaluation

By Carsten Isert, July 2017

This heuristic evaluation is part of the Udacity AI Nanodegree project 3 on logic and planning and evaluates the performance of several types of algorithms on 3 air cargo planning problems. The problem definition and tasks can be found at: <https://github.com/udacity/AIND-Planning>

### Results

Table 1 gives an overview of the metrics for the different problems and algorithms. The solutions are color coded. Green in the column for the plan length indicates that the solution is optimal. Yellow indicates that a solution has been found and red indicates that no solution was found within the 10min time limit.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Problem** | **Expansions** | **Goal Tests** | **New Nodes** | **Plan length** | **Time (s)** |
| 1. Breadth first search | 1 | 43 | 56 | 180 | 6 | 0,034 |
| 2 | 3343 | 4609 | 30509 | 9 | 15,21 |
| 3 | 14663 | 18098 | 129631 | 12 | 117 |
| 2. Breadth first tree search | 1 | 1458 | 1459 | 5960 | 6 | 0,95 |
| 2 | not solved in 10min |  |  |  |  |
| 3 | not solved in 10min |  |  |  |  |
| 3. Depth first graph search | 1 | 21 | 22 | 84 | 20 | 0,015 |
| 2 | 624 | 625 | 5602 | 619 | 3,71 |
| 3 | 408 | 409 | 3364 | 392 | 1,93 |
| 4. Depth first limited search | 1 | 101 | 271 | 414 | 50 | 0,092 |
| 2 | not solved in 10min |  |  |  |  |
| 3 | not solved in 10min |  |  |  |  |
| 5. Uniform cost search | 1 | 55 | 57 | 224 | 6 | 0,042 |
| 2 | 4852 | 4854 | 44030 | 9 | 13,6 |
| 3 | 18235 | 18237 | 159716 | 12 | 57,43 |
| 6. Recursive best first search H1 | 1 | 4229 | 4330 | 17023 | 6 | 3,42 |
| 2 | not solved in 10min |  |  |  |  |
| 3 | not solved in 10min |  |  |  |  |
| 7. greedy\_best\_first\_graph\_search h\_1 | 1 | 7 | 9 | 28 | 6 | 0,01 |
| 2 | 990 | 992 | 8910 | 17 | 4,18 |
| 3 | 5614 | 5616 | 49429 | 22 | 28,73 |
| 8. astar\_search h\_1 | 1 | 55 | 57 | 224 | 6 | 0,042 |
| 2 | 4852 | 4854 | 44030 | 9 | 13,07 |
| 3 | 18235 | 18237 | 159716 | 12 | 56,91 |
| 9. astar\_search h\_ignore\_preconditions | 1 | 41 | 43 | 170 | 6 | 0,033 |
| 2 | 1450 | 1452 | 13303 | 9 | 3,82 |
| 3 | 5040 | 5042 | 44944 | 12 | 16,35 |
| 10. astar\_search h\_pg\_levelsum | 1 |  |  |  |  |  |
| 2 |  |  |  |  |  |
| 3 |  |  |  |  |  |

### Discussion of non-heuristic search

The general observation is, that a wide range of algorithms have problems solving even these 3 very simple problems. Those being able to solve the problems take a long time for it. The fastest solution for the non-heuristic searches to find the optimal solution was the uniform cost search with 57,43s. The quickest way to find a solution was the depth first graph search (algorithm 3), but the solutions found by this algorithm were ridiculously long and included many actions that loaded and unloaded cargo several times before making other moves. Clearly, the solutions cannot be used for practical purposes.

As could be expected by the theoretical analysis, breadth first search (algorithm 1) could solve the problem and found the optimal solution for all problems. Compared to uniform cost search (algorithm 5), the number of nodes expanded and the number of goal tests was even lower, but the runtime exceeded the one of uniform cost search. For problem 3 the runtime was more than twice as much. However, this behavior is not consistent over all problem classes, so a hypothesis is, that the memory management of Python makes breadth first slower as it requires much more memory to evaluate all nodes.

### Discussion of heuristic search

Tbd.

### Optimal sequences

Problem 1:

Load(C2, P2, JFK)

Load(C1, P1, SFO)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Fly(P1, SFO, JFK)

Unload(C1, P1, JFK)

Problem 2:

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Load(C3, P3, ATL)

Fly(P1, SFO, JFK)

Fly(P2, JFK, SFO)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

Problem 3:

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Fly(P1, SFO, ATL)

Load(C3, P1, ATL)

Fly(P2, JFK, ORD)

Load(C4, P2, ORD)

Fly(P2, ORD, SFO)

Fly(P1, ATL, JFK)

Unload(C4, P2, SFO)

Unload(C3, P1, JFK)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)