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 The code for this project is available at: https://github.com/CarstenIsert/AIND-Planning

Results

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 within 10min timeframe				
	3	not solved within 10min timeframe				
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 within 10min timeframe				
	3	not solved within 10min timeframe				
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 within 10min timeframe				
	3	not solved within 10min timeframe				
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	11	13	50	6	0,56
	2	86	88	841	9	45,1
	3	318	320	2934	12	228,64

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.

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

The general expectation about the heuristic searches was that they would solve the 3 given simple problems easily without much computation requirements. The result table shows, that compared to the non-heuristic search, the heuristic search methods performed much better in terms of being able to find the optimal solutions and the required computation time. So, adding a heuristic generally makes sense.

However, the computation times required to solve the simple problems make it hard to believe, that real world problems can be solved by these algorithms at all. Comparing the A* heuristics for ignore preconditions and the one using the planning graph and computing the

level sum show that a simple heuristic like ignore preconditions can be much faster than constructing a complicated planning graph. For problem 3 the number of nodes created and expanded was by an order of magnitude less for the planning graph, but the computation time was by an order of magnitude longer.

It must be mentioned that the implementation for the planning graph was focusing on mimicking the available instructions and pseudo code rather than runtime optimizations. As there are many nested loops especially for the mutex calculation it is likely that there can be much more efficient implementations for the planning graph which allow working on larger problems.

Conclusion

For the type of problems encountered during the implementation of this exercise a clear recommendation would be to use A* search with the simple ignore preconditions heuristic as this is by far the fastest algorithm to find the correct solution and requires only a simple well known search algorithm. In contrast, the planning graph is a very complex algorithm and structure. To extend this to larger problems a more sophisticated heuristic could be designed that relaxes not so many preconditions.

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)
```