Nick Garvey
Sept 24 2017
AI Nanodegree - Heuristic Analysis

Data:

Problem	Solve Method	Plan Length	Time (s)	Node Expansions
1	breadth_first_search	6	0.16	43
2	breadth_first_search	9	58.46	3343
3	breadth_first_search	12	359.69	14663
1	depth_first_graph_search	20	0.08	21
2	depth_first_graph_search	619	12.05	624
3	depth_first_graph_search	392	8.48	408
1	depth_limited_search	50	0.41	101
2	depth_limited_search	timeout	timeout	timeout
3	depth_limited_search	timeout	timeout	timeout
1	uniform_cost_search	6	0.21	55
2	uniform_cost_search	9	78.7	4603
3	uniform_cost_search	12	352.96	16961
1	greedy_best_first_graph_search with h_1	6	0.27	7
2	greedy_best_first_graph_search with h_1	16	7.91	482
3	greedy_best_first_graph_search with h_1	29	82.51	4107
1	astar_search with h_1	6	0.22	55
2	astar_search with h_1	9	79.4	4603
3	astar_search with h_1	12	355.36	16961
1	astar_search with h_ignore_preconditions	6	8.73	55
2	astar_search with h_ignore_preconditions	timeout	timeout	timeout
3	astar_search with h_ignore_preconditions	timeout	timeout	timeout
1	astar_search with h_pg_levelsum	6	0.6	11
2	astar_search with h_pg_levelsum	9	40.61	74
3	astar_search with h_pg_levelsum	13	165.12	229

Analysis:

A few things stand out in the above data.

First, the failure of depth_limited_search is surprising to me. Looking at the plan for problem 1, it seems it got caught in a loop of loading & unloading P1 at SFO (in fact, it does this 22 times before giving up). I assume this is also the reason why it timed out for problem 2 & 3 - it just got caught in a loop. It isn't obvious why depth_limited_search fell to this, but no other algorithm seemed to.

Second, depth_first_graph_search found a solution for problem 3 extremely quickly. I'd want to see it duplicate this result a few times before making any conclusions here, it might have just gotten lucky.

h_ignore_preconditions stands out as a poor choice. It was very slow for problem 1, and timed out on the others. This is almost certainly due to my implementation. It does a full copy of the problem in order to ignore these preconditions, which must be quite expensive. This is strongly supported by the node expansion count - it is well within acceptable bounds, so it must be the expansion/solver itself with the performance bottleneck.

It is interesting to see all astar methods find optimal methods except for astar_search with h_pg_levelsum. This doesn't suggest a bug in the implementation, the textbook clearly states that this heuristic is not admissible, and therefore may find an unoptimal solution. However the textbook is also correct that the heuristic is good - it found a solution 1 off of optimal more than 2x faster than any of the optimal solutions.

The best algorithm in practice will depend on the cost trade-off between a suboptimal plan and the computational resources used to find the plan. If a suboptimal plan is fine (e.g. real-time route selection in a self driving car), then astar_search with h_pg_levelsum seems to be the best option.

If the plan is for something more expensive, like air cargo, then taking the time to find an optimal solution is obviously worth it, and something like astar_search with h_ignore_preconditions would be a better choice.