AIND Planning Project: Heuristic Analysis

Stefan Heidekrüger

August 14, 2017

In this analysis, we will compare and contrast different search algorithms and their performance on the project problems.

Optimal Plans for the Problems

The optimal plan-lengths for Problems 1,2 and 3 are given by 6, 9 and 12, respectively. The (non-unique!) optimal plans for each problem are detailed in Table 1.

Table 1: Optimal Plans found for each of the problems.

Problem	P1	P2	P3
Optimal Plan	Load(C1, P1, SFO) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)	Load(C1, P1, SFO) Load(C2, P2, JFK) Load(C3, P3, ATL) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK) Fly(P3, ATL, SFO) Unload(C3, P3, SFO)	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(P1, ATL, JFK) Unload(C1, P1, JFK) Fly(P2, ORD, SFO) Unload(C2, P2, SFO) Unload(C3, P1, JFK) Unload(C4, P2, SFO)
Length	6	9	12

Comparison of non-heuristic search algorithms

In this section we will compare the performance of the uninformed algorithms used, in particular Breadth First Search (BFS), Depth First Search (DFS) and Uniform Cost Search (UCS). In our analysis, we do not consider the performance of Breadth First Tree Search and Depth Limited Search, as these timed out on our software for Problems P2 and P3 (i.e. runtime over 10 minutes).

By design, BFS and UCS are guaranteed to always find an optimal solution to the problem (as long as a solution exists and the algorithm terminates, compare Russell and Norvig (2009, Ch. 3.4)). In fact, each step in our plan is associated with a single cost unit, thus BFS and UCS are virtually equivalent in our case, except that UCS has a more "generous" policy towards extending the frontier. This fact explains that in practice, BFS expanded around 15% - 30% fewer nodes and performed somewhat faster than UCS (40ms vs 50ms for P1, 11.41 vs 14.59s for P2, 51.68 vs 63.25 seconds for P3).[1]

While depth first search, on the other hand has no guarantee to find an optimal solution, it was much faster than the other algorithms at the task of finding *any* solutions. In fact, it found an optimal solution to P3 in 2.36 seconds, i.e. less than 5% of the runtime of BFS. Albeit the solutions found by DFS may be

[1] For our experiments, we used a faster implementation of the FIFO-Queue in BFS than the one provided, compare Gearhard (2017).

very suboptimal, checking weather a solution exists may sometimes be advantageous to waiting longer for the optimal solution. For detailed comparison of the result, see Tables 2, 3 and 4.

Comparison of A^* search heuristics

For A^*-search, we considered three different heuristics:

1. The **pseudo-heuristic "H1"**, which returns constant cost. As such A^*-search with H1 is equivalent to UCS and (almost) equivalent to BFS. As we would expect, we see the exact same number of Node Expansions as in UCS and very similar time-performance.

2. The **"ignore preconditions"** heuristic considers the action-distance to the goal state by finding a (posssibly inadmissable) path of actions that result in the desired goal, but where actions can be performed even when their preconditions are not fulfilled. As a valid heuristic, it's guaranteed to find an optimal solution (provided existence) and in hour experiments lead to an almost *fourfold* performance increase over H1/UCS—both terms of node expansions (P2: 1310 vs 4604, P3: 4443 vs 16963) and in time (P2: 4 vs 14, P3: 18 vs 64 seconds).

3. The **PG-Levelsum heuristic** on the other hand, considers each goal literal individually. The minimum level in the planning graph where a single goal literal can be fulfilled is known as its *level cost*. The PG-Levelsum heuristic then returns the sum of the individual level costs. Note that the level sum heuristic is technically inadmissible (Russell and Norvig, 2009, p. 382), as it disregards any possible *positive* coupling effects - in this case the possibility to load multiple parcels onto a plane and fly a route that's shorter in total than considering each parcel separately. In fact, while this heuristic found optimal solutions to P1 and P2, its found solution to P3 in our implementation was indeed suboptimal, using 13 instead of 12 planning steps. (Analysis of this path revealed, that a plane indeed went to its final destination to drop off a package, then circled back to another airport to pick up another cargo and returning to its final destination - instead of picking up that package "along the way".) However, it can be said that this heuristic resulted in significantly less node expansions than the other heuristics considered here, in fact another 95% reduction over "ignore preconditions". However, these fewer node expansions did not result in time-performance speedups as constructing and searching the planning graph multiple times for each goal is quite computationally expensive. In fact, we observed on the order of $20X$ the time required as compared to the "ignore preconditions" heuristic.

For detailed results, the reader is again referred to Tables 2, 3 and 4. In conclusion we observe that for this problem class, the "ignore preconditions" heuristic shows the best performance: A considerable speedup vs nonheuristic methods such as BFS/UCS while still yielding optimal solutions. While not the case here, the PG-Levelsum heuristic may be promising, however, for problems where individual node expansions may be very expensive or prohibitive.

References

C. Gearhard. Uniform cost search faster than breadth first? `https://discussions.udacity.com/t/uniform-cost-search-` *Udacity Community Forums*, 2017.

S. J. Russell and P. Norvig. Artificial intelligence: a modern approach (3rd edition), 2009.

Table 2: Performance on Problem 1.

		Expansions	Goal Tests	New Nodes	Time (s)	Path Length
1	breadth_first_search	43	56	180	0.04	**6**
2	breadth_first_tree_search	1458	1459	5960	1.27	**6**
3	depth_first_graph_search	21	22	84	0.02	20
4	depth_limited_search	101	271	414	0.12	50
5	uniform_cost_search	55	57	224	0.05	**6**
6	recursive_best_first_search, h1	442	4230	17023	3.62	**6**
7	greedy_best_first_graph_search, h1	**7**	**9**	**28**	**0.01**	**6**
8	astar_search, h_1	55	57	224	0.04	**6**
9	astar_search, h_ignore_preconds	41	43	170	0.03	**6**
10	astar_search, h_pg_levelsum	11	13	50	1.00	**6**

Table 3: Performance on Problem 2

		Expansions	Goal Tests	New Nodes	Time (s)	Path Length
1	breadth_first_search	3343	4609	30509	11.41	**9**
2	breadth_first_tree_search	–	–	–	timed out	–
3	depth_first_graph_search	624	625	5602	4.64	619
4	depth_limited_search	–	–	–	timed out	–
5	uniform_cost_search	4604	4606	41828	14.59	9
6	recursive_best_first_search, h1	–	–	–	timed out	–
7	greedy_best_first_graph_search, h1	454	456	4087	**1.37**	19
8	astar_search, h1	4604	4606	41828	13.92	**9**
9	astar_search, h_ignore_preconds	1310	1312	11979	4.02	**9**
10	astar_search, h_pg_levelsum	**74**	**76**	**720**	81.99	**9**

Table 4: Performance on Problem 3

		Expansions	Goal Tests	New Nodes	Time (s)	Path Length
1	breadth_first_search	14663	18098	129631	51.68	**12**
2	breadth_first_tree_search	–	–	–	timed out	–
3	depth_first_graph_search	408	409	3364	**2.36**	392
4	depth_limited_search	–	–	–	timed out	–
5	uniform_cost_search	16963	16965	149136	63.25	**12**
6	recursive_best_first_search, h1	–	–	–	timed out	–
7	greedy_best_first_graph_search, h1	3773	3775	33127	14.97	30
8	astar_search, h1	16963	16965	149136	63.61	**12**
9	astar_search, h_ignore_preconds	4443	4445	39217	17.77	**12**
10	astar_search, h_pg_levelsum	**228**	**230**	**2072**	323.14	13