P3 Heuristic Analysis *by Levan Iordanishvili*

Optimal Plans

Problem 1, Length 6

Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

Problem 2, Length: 9

Load(C2, P2, JFK)
Load(C1, P1, SFO)
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)

Problem 3, Length: 12

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)
Unload(C3, P1, JFK)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)

Non-heuristic Search Result Metrics

Problem	Algorithm	Expansions	Goal Tests	New Nodes	Time (s)	Solution Len.
1	breadth_first_search	43	56	180	0.070	6
2	breadth_first_search	3343	4609	30509	28.615	9
3	breadth_first_search	14120	17673	124926	191.916	12
1	breadth_first_tree_search	1458	1459	5960	2.125	6
2	breadth_first_tree_search	N/A	N/A	N/A	10+ mins	N/A
3	breadth_first_tree_search	N/A	N/A	N/A	10+ mins	N/A
1	depth_first_graph_search	12	13	48	0.020	12
2	depth_first_graph_search	582	583	5211	6.734	575
3	depth_first_graph_search	677	678	5608	7.884	660

Metrics show that the `breadth_first_search` is the best non-heuristic function, achieving the optimal solutions with lengths 6, 9, and 12 for problems 1, 2, and 3 respectively. Even though `breadth_first_search` expands more nodes than `depth_first_graph_search` (14,120 to 677 for problem 3) and finds a solution more slowly (192 s to 8 s for problem 3), its solution is far superior (length of 12 to 660 for problem 3).

The key advantage of the `breadth_first_search` is its ability to *always* find the closest/shortest path to the solution as it expands all the nodes at each level of the tree before moving on (Lesson 10: Search Comparison I video). In cases where it's important to find the shortest path, the potentially expensive memory footprint -- accumulated through storing all the visited nodes -- of `breadth_first_search` is necessary.

Considering the problem at hand and the cycles that are present within the graph, `breadth_first_tree_search` is by far the worst search algorithm provided. Its implementation does not retain an "explored" set of nodes, and as a result it revisits nodes over and over, the child node returns to visit its parent again and again. Therefore, the search for even a small problem (2) lasted longer than 10 minutes.

Heuristic Search Result Metrics

Problem	Algorithm	Expansions	Goal Tests	New Nodes	Time (s)	Solution Len.
1	astar_search with h_ignore_preconditions	41	43	170	0.083	6
2	astar_search with h_ignore_preconditions	1450	1452	13303	9.054	9
3	astar_search with h_ignore_preconditions	5040	5042	44944	34.511	12
1	astar_search with h_pg_levelsum	11	13	50	1.359	6
2	astar_search with h_pg_levelsum	86	88	841	121.640	9
3	astar_search with h_pg_levelsum	325	327	3002	659.590	12

Though each search finds the optimal solution with lengths of 6, 9 and 12 for problem 1, 2 and 3 respectively, the metrics for the heuristic searches are more complex to analyze than they were for non-heuristic searches.

A* `h_ignore_preconditions` search is the clear winner in clock time as it beats A* `h_pg_levelsum` (35 s to 660 s for problem 3). `h_pg_levelsum` takes longer to execute due to the fact that for each visit to a state node, the search algorithm builds a Planning Graph, a structure that is crucial for scoring the state. It takes about 300 ms for my implementation to build the Planning Graph. Creators of Graphplan report that "most of the time spent is in graph creation", though "graph creation only takes up a significant portion of Graphplan's running time in the simpler problems, where the total running time is not very large anyway" (Blum & Furst, 1997, p. 8).

Despite the relatively long `h_pg_levelsum` execution time, it's interesting to note that it still showcases the efficiency of the heuristic through the number of nodes it expands. `h_pg_levelsum` beats `h_ignore_preconditions` by an order of magnitude in node expansions (325 to 5,060 for problem 3. Note: `breadth_first_search` expanded 14,120 nodes).

The results suggest that it's important to try out different heuristics for different problems and use the one that works best. For smaller, easier problems, it's likely that `h_ignore_preconditions` will suffice. For the AirCargo problem at hand, `h_ignore_preconditions` is clearly the best of the bunch due to its fast execution

time. However, for larger problems, the node-expansion efficiency of `h_pg_levelsum` may make it the better choice.

References

Blum, A. L., & Furst, M. L. (1997). Fast planning through planning graph analysis. *Artificial Intelligence*, *90*(1-2), 281-300. doi:10.1016/s0004-3702(96)00047-1