Planning Search Analysis

AIND-Planning Project

Non-heuristic Planning Solution Searches

For the air_cargo_p1, air_cargo_p2 and air_cargo_p3 problems, following algorithms are tested:

- breadth_first_search: Breadth-first search
- depth_first_graph_search: Depth-first search
- depth_limited_search: Depth-first search with the depth limitation of 50

Problem	Algorithms	Expansions	Goal Tests	Plan Length	Time Elapsed
Air Cargo P1	**Breadth-first search**	43	56	**6**	0.040s
	Depth-first search	21	22	20	0.016s
	Depth-first limited	101	271	50	0.109s
Air Cargo P2	**Breadth-first search**	3343	4609	**9**	15.284s
	Depth-first search	624	625	619	3.783s
	Depth-first limited	222719	2053741	50	1148.715s
Air Cargo P3	**Breadth-first search**	14663	18098	**12**	121.492s
	Depth-first search	408	409	392	2.031s
	Depth-first limited	NA	NA	NA	>20min

As shown in the above table, the metrics on number of node expansions required, number of goal tests, plan length of solution (breath-first search always give the optimal plan length), and time elapsed for each search algorithm are shown.

Domain-independent Heuristics Searches

For the same three problems, following heuristics are tested with A* search:

- h_1: A constant value 1, not a real heuristic.
- h_ignore_preconditions: The minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed.
- h_pg_levelsum: The sum of the level costs of the individual goals (admissible if goals independent). This heuristic can be inadmissible, but works well in real practice.

Problem	Heuristics	Expansions	Goal Tests	Plan Length	Time Elapsed
Air Cargo P1	**constant 1**	55	57	**6**	0.045s
	ignore precond	41	43	**6**	0.032s
	level sum	11	13	**6**	0.759s
Air Cargo P2	**constant 1**	4826	4828	**9**	12.972s
	ignore precond	1435	1437	**9**	3.986s
	level sum	86	88	**9**	67.664s
Air Cargo P3	**constant 1**	18221	18223	**12**	57.064s
	ignore precond	5040	5042	**12**	16.484s
	level sum	310	312	**12**	328.905s

As shown in the above table, the metrics on number of node expansions required, number of goal tests, plan length of solution (breath-first search always give the optimal plan length), and time elapsed for each search algorithm are shown.

Optimal Plans

In this section we show the optimal results from A* search using h_pg_levelsum heuristic.

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

Air Cargo P2
Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Unload(C2, P2, SFO) Unload(C1, P1, JFK)

Air Cargo P3
Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C4, P2, SFO) Unload(C2, P2, SFO) Unload(C3, P1, JFK) Unload(C1, P1, JFK)

Comparisons among Search Algorithms and Heuristics

Firstly, for the non-heuristic searches in the first section, we can see that only breath-first search gives the optimal solution. However, when looking at the running speed of the algorithms, depth-first search is much faster — though not giving the optimal solution, it explores much less nodes. The worst one is the depth-limited depth-first search: it usually find solution just at the limit depth, and enumerate lots of nodes — one can consider this behavior as an approximation of expanding the search tree to the limit depth, which is a terrible practice.

Secondly, for the three heuristics used in A* search, we can first see that A* search algorithm alone (with the fake h_1 heuristic) has a very close performance as the breath-first search algorithm — they both outputs optimal solutions, and both explores similar number of nodes. When considering the nodes explored (expanded), h_ignore_preconditions have a smaller number, and h_pg_levelsum has the smallest of all. However, since the calculation cost of complex heuristics are higher, thus their running time does not always decrease when one utilize the heuristics. We can find that though h_pg_levelsum explores magnitudes less nodes than h_1 and h_ignore_preconditions, its running time is the highest. Though h_pg_levelsum is not admissible in our problem, but it works well since the goals are still highly decomposable.

Both the number of expended nodes and the calculation cost on each node matter. Sometimes one should make tradeoff between these two: Better heuristics usually takes more time on each node, but it could help reduce the number of expansion; Vice versa, when expected to explore a large space, the algorithm cannot spend to much time on each node and have to make quick decisions. Roughly speaking, one can take estimate the time by the product of expected node expansion number and the calculation complexity of heuristic function on each node. We can see that though the calculation of h_ignore_preconditions is more complex than h_1, but it is acceptable: E.g. for problem 3, the number of explored nodes decreases from 18221 to 5040, and the running time drops from 57s to 16s — quite similar speed up ratios. Furthermore, despite the time complexity, one should also take the space complexity in mind, especially when the memory usage is highly correlated with the size of expanded nodes. For example, in both breath-first and A* searches, the “frontier” of the searching nodes are kept in memory; In A* search, one may also want to cache some of the heuristic function results. Along this dimension of tradeoff, depth-first search makes much less burden on the space complexity.

So there is no one-for-all best combination of search algorithm and heuristic for problems. For finding the optimal solution in Air Cargo Problem, when memory is not a limitation, A* search with h_ignore_preconditions heuristic is the best choice — it makes a good tradeoff between exploration (number of expansion) and exploitation (calculation cost of heuristic function). When memory usage is a concern, complex heuristics like h_pg_levelsum can dramatically reduce the number of expanded nodes, while compensate for the longer running time.