Planning Heuristic Analysis

Optimal Plans

Optimal Plans for Air Cargo Problems 1, 2, and 3, calculated using A* h1, though equivalent results were also provided by other "complete" searches, such as A* searches with other heuristics, breadth-first search, uniform cost search, etc. Note that optimal in the case of these problems is just the minimum number of actions in a plan, since the cost of each action is the same.

Optimal Plans		
Problem 1	Problem 2	Problem 3
Load(C1, P1, SFO)	Load(C1, P1, SFO)	Load(C1, P1, SFO)
Load(C2, P2, JFK)	Load(C2, P2, JFK)	Load(C2, P2, JFK)
Fly(P1, SFO, JFK)	Load(C3, P3, ATL)	Fly(P1, SFO, ATL)
Fly(P2, JFK, SFO)	Fly(P1, SFO, JFK)	Load(C3, P1, ATL)
Unload(C1, P1, JFK)	Fly(P2, JFK, SFO)	Fly(P2, JFK, ORD)
Unload(C2, P2, SFO)	Fly(P3, ATL, SFO)	Load(C4, P2, ORD)
	Unload(C3, P3, SFO)	Fly(P2, ORD, SFO)
	Unload(C2, P2, SFO)	Fly(P1, ATL, JFK)
	Unload(C1, P1, JFK)	Unload(C4, P2, SFO)
		Unload(C3, P1, JFK)
		Unload(C2, P2, SFO)
		Unload(C1, P1, JFK)

Non-Heuristic Searches

Non-heuristic search results for breadth-first search, depth-first graph search, and uniform cost search:

Problem	Search	Plan Length	Seconds	Expansions	Goal Tests	New Nodes
1	breadth_first_search	6	0.021	43	56	180
	depth_first_graph_search	20	0.010	21	22	84
	uniform_cost_search	6	0.025	55	57	224
2	breadth_first_search	9	9.124	3343	4609	30509
	depth_first_graph_search	619	2.309	624	625	5602
	uniform_cost_search	9	8.099	4852	4854	44030
3	breadth_first_search	12	67.586	14663	18098	129631
	depth_first_graph_search	392	1.195	408	409	3364
	uniform_cost_search	12	35.357	18235	18237	159716

For non-heuristic searches, the key thing to note is that the plan length for the depth first graph search is much larger with many redundant steps that does not lead to completing the goal. Even though this search took the least time, the resulting plan is not helpful. Breadth-first search and uniform-cost search both produced optimal plans. As the complexity increased in problem 2 and 3, the uniform cost search took less and less time than breadth-first, even though it also produced larger numbers of expansions, goal tests, and new nodes. One property of the air cargo problem is the uniform cost of each action. The cost is the same for flying between any two airports, so the solution produced by the breadth-first search is equivalent to the solution provided by the uniform cost search. If we introduce the distance between airports as a cost, then the uniform cost search could provide better solutions. The performance of the uniform cost search in terms of the time elapsed and the expansions and goal tests are identical to the results of the A* h1 and A* ignore precondition searches for these three problems. This seems to indicate that for these problems, the cost consideration of the uniform cost search has practically the same effectiveness as the h1 and ignore-precondition H heuristics. The large number of expansions, goals, and new nodes in the breadth-first and uniform-cost searches indicate their exhaustive examination of the search tree to find an optimal solution. It is surprising that the uniform cost search took much less time than the breadth-first search for problem 3 despite a more expansive traversal of the tree (e.g. ~18000 expansions versus ~14000 expansions for breadth-first). This could be an artifact of the AIMA python search implementation.

A* Searches and their H heuristics

Here are the results from the A* searches for the air cargo problems:

Problem	Search	Plan Length	Seconds	Expansions	Goal Tests	New Nodes
1	A* h1	6	0.025	55	57	224
	A* ignore preconditions	6	0.020	41	43	170
	A* level sum	6	0.522	11	13	50
2	A* h1	9	8.125	4852	4854	44030
	A* ignore preconditions	9	2.485	1450	1452	13303
	A* level sum	9	43.871	86	88	841
3	A* h1	12	35.430	18235	18237	159716
	A* ignore preconditions	12	10.062	5040	5042	44944
	A* level sum	12	211.083	325	327	3002

The A* ignore precondition heuristic for these simplified air cargo problems is the minimum number remaining actions to achieve all goals. We assume this minimum number of actions is 1, since for these problems no action can achieve more than one goal. This heuristic helps to focus the search toward goal attainment and reduce the various search efficiency metrics by a factor of 3 relative to the H1 heuristic. Note that the H1 heuristic is essentially no H heuristic, which explains why the uniform cost search (essentially G in A*'s G + H cost function) performs virtually the same as the A* H1 (G + 1).

The main distinction of A* level sum search is the use of a planning graph. The level sum heuristic simply optimizes for how to achieve the most goals in the fewest planning graph levels, which does help to guide the search to minimize the size of the planning graph. However, the main difference with the planning graph is that at each search level, the potential conflicts between various possible nodes are considered via the various mutex computations before node expansion. These mutex checks results in two orders of magnitude less in the number of nodes explored from non-heuristic searches and an order of magnitude less search space relative to A* ignore precondition, but at an order of magnitude more in time cost. While the planning graph reduces node exploration to polynomial, it seems to do so by performing mutex checks on all nodes in the exponential search tree. My implementation of these mutex checks are in python could explain the much longer time cost. There are computation metrics that indicate python is about 30 times slower than other byte-code compiled languages like C#. While pypy3 will help, is there a python interface to a C/C++ planning graph library available (like numpy for linear algebra) that can be used in production?

For these highly simplified air cargo problems and our educational use of python, the A* ignore precondition is the best for producing an optimal solution in the least time. The ignore precondition heuristic's focus on goal attainment reduces the search space by a factor of 3 relative to the non-heuristic searches. While the A* level sum heuristic with planning graph achieve an optimal solution with a much smaller search space, the required mutex computation results in a much higher time cost.