Nicholas Beshouri
AIND project 3 heuristic analysis

1. Non-heuristic search algorithm comparison

I chose breadth-first, depth-first, and uniform cost search as my three non-heuristic search algorithms. Predictably, breadth-first search was slower than depth-first search and expanded more nodes (Table 1), but produced optimal solutions, while depth-first search produced long convoluted ones. Also notice that depth-first search solved problem 3 more than twice as fast as problem 2, underscoring how luck dependent depth-first search can be.

Like breadth-first search, uniform cost search also produced optimal solutions but, because it has to check the cost of each node at a given level, uniform cost search ended up expanding and checking more nodes than breadth-first search (Table 1). Surprisingly, uniform cost search was 3 to 4 times faster than breath-first search, even though uniform cost search checked more nodes. This appears to be an implementation issue. The `FIFOQueue` that the provided `breadth_first_search` method uses to manage its frontier uses a `node in some_list` construction in its `__contains__` method. This isn't very efficient, especially given that node comparisons are expensive, each requiring several `Expr` comparisons. The `PriorityQueue` used by `uniform_cost_search` uses a separate dictionary to keep track of inclusion, making its `__contains__` method much more efficient.

2. Heuristic comparison

I tested 3 different heuristics: `ignore_preconditions`, `level_sum`, and `unsatisfied_goals` (which simply counts the number of unsatisfied goals in a state).

Of these, `level_sum` was the most accurate (Table 2). It found optimal solutions to all three problems after checking only 1/20th of the nodes searched by the other two heuristics. Unfortunately, it was also the most expensive: A* searches using `level_sum` took more than 60 times longer than either of the other two heuristics. Profiling the code, I found that almost all of this extra time is spent constructing `PlanningGraph` objects, which has to be done for each new state the algorithm encounters. While it's probably possible to optimize `PlanningGraph`, I'm skeptical that such an expensive heuristic could ever be worth the effort on a problem this size.

The other two heuristics, `ignore_preconditions` and `unsatisfied_goals`, are less accurate, and so search more nodes, but much cheaper to compute and result in much faster searches. They're also admissible, unlike `level_sum`, and have much simpler implementations. In this particular problem, `unsatisfied_goals` is the better choice, because `ignore_preconditions`, as described in the text, wastes time looking for actions that might cover more than one goal, which don't exist in the air cargo problem.

3. Heuristic search verses non-heuristic comparison

A* search using `unsatisfied_goals` was better than any non-heuristic search. It produced optimal plans for all problems and was usually the fastest. Depth-first search was faster in

problem 1 (0.0038s vs 0.0079s) and in problem 3 (0.9s vs 3.2s seconds), but in both cases found long convoluted plans.

Table 1. Non-heuristic search comparison

		Breadth-first	Depth-first	Uniform cost
Problem 1	**Time**	0.0082	0.0038	0.0098
	Expansions	43	21	55
	Goal Tests	56	22	57
	New Nodes	180	84	224
	Plan Length	6	20	6
Problem 2	**Time**	8.4013	2.6753	2.6109
	Expansions	3343	624	4806
	Goal Tests	4609	625	4808
	New Nodes	30509	5602	43618
	Plan Length	9	619	9
Problem 3	**Time**	79.352	0.9883	11.121
	Expansions	14663	408	18211
	Goal Tests	18098	409	18213
	New Nodes	129631	3364	159520
	Plan Length	12	392	12

Tested on a 2.3 GHz 2012 MacBook Pro using CPython 3.6.

Table 2. A* Heuristic comparison

		ignore_preconditions	level_sum	unsatisfied_goals
Problem 1	**Time**	0.0097	0.7827	0.0079
	Expansions	41	11	41
	Goal Tests	43	13	43
	New Nodes	170	50	170
	Plan Length	6	6	6
Problem 2	**Time**	1.030	58.876	0.8043
	Expansions	1399	74	1399
	Goal Tests	1401	76	1401
	New Nodes	12807	720	12807
	Plan Length	9	9	9
Problem 3	**Time**	4.2229	278.76	3.2127
	Expansions	4746	275	4746
	Goal Tests	4748	277	4748
	New Nodes	42105	2526	42105
	Plan Length	12	12	12

Table 3. Optimal solutions

	Problem 1	**Problem 2**	**Problem 3**
1	Load(C1, P1, SFO)	Load(C1, P1, SFO)	Load(C1, P1, SFO)
2	Load(C2, P2, JFK)	Load(C2, P2, JFK)	Load(C2, P2, JFK)
3	Fly(P1, SFO, JFK)	Load(C3, P3, ATL)	Fly(P1, SFO, ATL)
4	Fly(P2, JFK, SFO)	Fly(P1, SFO, JFK)	Load(C3, P1, ATL)
5	Unload(C1, P1, JFK)	Fly(P2, JFK, SFO)	Fly(P2, JFK, ORD)
6	Unload(C2, P2, SFO)	Fly(P3, ATL, SFO)	Load(C4, P2, ORD)
7		Unload(C1, P1, JFK)	Fly(P1, ATL, JFK)
8		Unload(C2, P2, SFO)	Unload(C1, P1, JFK)
9		Unload(C3, P3, SFO)	Fly(P2, ORD, SFO)
10			Unload(C2, P2, SFO)
11			Unload(C3, P1, JFK)
12			Unload(C4, P2, SFO)

These were produced using uniform cost search.