Heuristics Analysis: AIND_Planning

John Quinn, Udacity Artificial Intelligence Nano-Degree Candidate

N.B.: Throughout this report I have used (i) terms from the assignment materials and (ii) function names and data items included within my submitted code, my_air_cargo_problems.py and my_planning_graph.py. The former are well explained in the assignment materials, and the latter are sufficiently commented within my submitted code; detailed definitions of either within this report would be repetitive. I have, nonetheless, reiterated some such items and expounded upon others here for the sake of clarity.

The Problems to be Solved:

Within my_air_cargo_problems.py, three air cargo logistics problems needed to be solved. The three problems themselves are:

Problem 1:

Init(At(C1, SFO) ∧ At(C2, JFK)
∧ At(P1, SFO) ∧ At(P2, JFK)
∧ Cargo(C1) ∧ Cargo(C2)
∧ Plane(P1) ∧ Plane(P2)
∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))

Problem 2:

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))

Problem 3:

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
∧ At(P1, SFO) ∧ At(P2, JFK)
∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
∧ Plane(P1) ∧ Plane(P2)
∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))

<u>**Algorithms Considered**</u>:

Uninformed:

For uninformed algorithms, I considered choices 1, 3, and 5. These represent: (a) breadth_first_search, (b) depth_first_graph_search, and (c) uniform_cost_search, respectively. One of the requirements of this assignment was completeness, and these three uninformed algorithms guarantee the finding of a solution if such a solution exists for each of these three problems.

Completeness is true by definition for the first two of these three algorithms. The first, breadth_first_search is of the graph, vis-à-vis tree, variety, as evinced by Lines 181 *et. seq.* in provided search.py.

The third and final uninformed algorithm is also complete for this assignment. The uniform_cost_search algorithm is complete in this specific domain/assignment, inasmuch as (i) the branching factor is finite and (ii) ∀ step_cost ∈ search_space ∃ $\epsilon > 0$ s.t. step_cost ≥ ϵ.

The underlying design of the branching factor at any level for the uniform_cost_search algorithm is, in essence, the combination of the parent nodes and their children nodes. With a finite number of cargos, planes, and airports the $|\{\bigcup(cargo \in cargos, plane \in planes, airport \in airports)\}|$ must be finite, and the branching factor connecting any such level of nodes must also be finite.

The second criterion for completeness for uniform_cost_search similarly can be viscerally explained as follows: (a) the path selected is relegated to the cost of edge to that next node; (b) since there is always some positive cost to evaluating any node cumulative path costs continue to grow; (c) once the cumulative path cost equals or exceeds the step_cost of the next node that next node cannot remain at the front of the priority queue; and (d) at that juncture an infinite loop is obviated. This results in the algorithm diverting from such a futile path and eventually finding a solution.

These three uninformed algorithms are complete. They are relatively efficient as well.

Informed:

For informed algorithms, I have considered choices 8, 9, and 10. These represent: (a) astar_search_h_1, (b) astar_search_h_ignore_preconditions, and (c) astar_search_h-pg_level_sum, respectively. These three are complete and relatively efficient, at least for more complex problems.

Results and Analyses:

Problem 1:

The optimal plan length is 6 (i.e., 6 Actions), and those Actions are:

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

Algorithm	**Optimality**	**Seconds Elapsed**	**# of Node Expansions**
breadth_first_search	Yes	0.02	43
depth_first_graph_search	No	0.01	21
uniform_cost_search	Yes	0.04	55
astar_search_h_1	Yes	0.03	55
astar_search_h_ignore_preconditions	Yes	0.03	41
astar_search_h_pg_level_sum	Yes	2.48	11

Problem 2:

The optimal plan length is 9 (i.e., 9 Actions), and those Actions are:

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)

Algorithm	**Optimality**	**Seconds Elapsed**	**# of Node Expansions**
breadth_first_search	Yes	11.63	3,343
depth_first_graph_search	No	2.84	624
uniform_cost_search	Yes	9.47	4,853
astar_search_h_1	Yes	22.99	4,853
astar_search_h_ignore_preconditions	Yes	3.42	1,450
astar_search_h_pg_level_sum	Yes	249.62	86

Problem 3:

The optimal plan length is 12 (i.e., 12 Actions), and those Actions are:

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(P2, ORD, SFO)
Fly(P1, ATL, JFK)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

Algorithm	**Optimality**	**Seconds Elapsed**	**# of Node Expansions**
breadth_first_search	Yes	83.13	14,663
depth_first_graph_search	No	1.44	408
uniform_cost_search	Yes	41.53	18,223
astar_search_h_1	Yes	41.63	18,223
astar_search_h_ignore_preconditions	Yes	13.34	5,040
astar_search_h_pg_level_sum	Yes	1,506.43	325

<u>Analysis of Results Metrics:</u>

Uninformed/Non-Heuristic Algorithms:

For each of the three problems depth_first_graph_search can immediately be discarded as a viable option. While this algorithm is complete (i.e., it's guaranteed to provide ***an*** answer), it is not optimal (i.e., it's not guaranteed to provide the ***optimal*** solution.) There is little purpose, e.g., in adopting a model that might cause a carrier to ship cargo from LAX to ORD via AKL (Auckland, New Zealand) when an efficient, domestic-only solution exists.

Of the other two uninformed algorithms, breadth_first_search is the better choice for all three problems. Both are optimal, but the breadth_first_search algorithm found the optimal solution in each case/problem with the least node expansions and in at least a comparable amount of time as uniform_cost_search.

Informed/Heuristic Algorithms:

For each of the three problems the h_ignore_preconditions heuristic performed better than the other two. In each case this heuristic required fewer node expansions and less execution time than the h_1 heuristic. As for my reason for suggesting the h_ignore_preconditions heuristic over the h_pg_level_sum heuristic please see the final, italicized section of this part on the next page.

All three heuristics are ostensibly scoring function coupled with the A* algorithm. As such each coupling is both complete and optimal. The h_ignore_preconditions heuristic, however, always executed in the least amount of time, and as problem complexity grew that time efficiency became starker. With Problems 2 and 3, for example, the h_ignore_preconditions heuristic required roughly an order of magnitude less in time than the h_pg_level_sum heuristic and a fraction of the time and space required by the h_1 heuristic.

That the h_ignore_preconditions heuristic performed the best of the three heuristics is not unexpected. It is a rather graphic depiction of the efficiency gained by relaxing and/or removing pre-conditions and/or constraints. Simply looking at the quantum of lines of method check_precond() of class Action within provided search.py (Line 44 *et. seq.*) and recognizing the fact that it iterates over relatively complex objects is evidence enough of how many steps can be avoided. In its place the number of actions is estimated in a simple, direct manner via this seven-line heuristic within my_air_cargo_problems.py

Uninformed vis-à-vis Informed Search:

astar_search_h_ignore_preconditions only outperformed non-heuristic search planning methods, especially breadth_first_search, for more complex problems. It was clearly overkill for Problem 1, which was very straightforward, but performed the best temporally of all optimal algorithms (i.e., viable choices) for Problems 2 and 3.

A common refrain throughout the lecture videos and reading materials – I paraphrase here – was start simple and add intelligence when necessary.   Problem 1 was simple; its solution was direct and obvious.   By Problem 3, however, there were fewer planes than cargos and airports, and all requisite pre-conditions for necessary Actions were not met *a priori* (i.e., at the outset/within the initial state.)   For Problems 2 and 3, more complexity, with its consequent overhead, was warranted.   Problem 1, on the other hand, could and should be handled with a non-heuristic method, in keeping with Occam's razor.

h_ignore_preconditions vis-à-vis h_pg_levelsum heuristics:

This maxim is further evinced in comparing the coupling of the h_ignore_preconditions and h_pg_levelsum heuristics with the A* algorithm.   The former outperformed the latter temporally, as the latter requires the implementation of a PlanningGraph while former requires little more than a simple counter.   That efficiency quickly grew to approximately and order of magnitude.   The h_pg_levelsum heuristic did, however, outperform the h_ignore_preconditions spatially, expanding fewer nodes than the former, but with the relative inexpensiveness of space/memory and speed of execution now considered nearly paramount, I believe speed outweighs time in most situations.   It is for this reason that I suggest the h_ignore_preconditions heuristic over the h_pg_levelsum heuristic for these problems.

<u>Summary and Self-Critique</u>:

While informative and an excellent exercise, the three problems were much simpler than would be the case in production.   This was required, among other reasons, because most students simply don't have the computing power available to them for problems sets such as those managed by Federal Express.

The examples show, nonetheless, the need to add algorithmic complexity only when such intelligence is needed.   For the simpler problems breadth_first_search alone was fine, but for Problems 2 and 3 the A* algorithm with an appropriate heuristic was clearly warranted.   Even then utilizing the *h_ignore_preconditions* heuristic rather than the *h_pg_levelsum* one proved the need to keep the methodology as simple as possible. Otherwise, time complexity grows in orders of magnitude.