Heuristic Analysis

Problem 1

Optimal plan for problem 1

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

Search method comparison for problem 1

	Optimality	Plan Length	Time Elapsed	Expansions	Goal Tests	New Nodes
breadth_first_search	yes	6	0.023	43	56	180
depth_first_graph_search	no	20	0.010	21	22	84
uniform_cost_search	yes	6	0.028	55	57	224
greedy_best_first_graph_search with h_1	yes	6	**0.003**	**7**	**9**	**28**
a* with h_1	yes	6	0.030	55	57	224
a* with h_ignore_preconditions	yes	6	0.033	41	43	170
a* with h_pg_levelsum	yes	6	1.525	11	13	50
a* with h_pg_setlevel	yes	6	1.117	8	10	35

Observation and analysis for problem 1

- *greedy_best_first_graph_search* outperform every other methods in every metrics, and accidentally every nodes poped in this search method lead to the optimal plan(just expand 7 nodes), it is because:
    - This method was implemented by *best_first_graph_search* with *PriorityQueue* where *f=1*: `greedy_best_first_graph_search = best_first_graph_search(problem, lambda node: 1)`
    - With the same priority '1', 'the smaller state' of a node will be pop first, because in Node class:

```
class Node:
    def __lt__(self, node):
        return self.state < node.state
```

  - The state of a node in these problems was like 'FFTTFFTFTFFF', so it intend to pop a state, with more initial 'F', first. Possitive sentenses stand in front of a state, so it intent to expands the nodes, which turn possitive sentense of a state to negative, first. Which leads to the goal in these problems.
  - The order of the expanded nodes is defined by `get_actions` which implemented by `return load_actions() + unload_actions() + fly_actions()`
    This order is similar as the order of a optimal plan 'Load -> Fly -> Unload'. ('Load -> Unload' will rarely happend in a search because it would generate a explored node)
  - Only 2 cargos and 2 planes and 2 airports, so very possible actions in order 'Load -> Fly -> Unload' can leads to the optimal plan.

- *depth_first_graph_search* always expands the deepest node, so this method is not optimal.(Russell-Norvig AIMA 3rd Ed 3.4.3) It use a LIFO stack to implement the frontier, so it pop nodes, in reverse of the actions('Load -> Unload -> Fly'), in this order 'Fly -> Unload -> 'Load'. It's a bit far away from the optimal model, so this method take more steps than *greedy_best_first_graph_search*. But it also faster than many other methods, for it just go deeper to find a solution and it expand less nodes.
- Astar is optimally efficient ofr any given consistent heuristic.(Russell-Norvig AIMA 3rd Ed 3.5.2)
- *astar_with_h_pg_levelsum* and *astar_with_h_pg_setlevel* expand less nodes than other methods which shows they have good heuristic, but too much time they take shows they were calculate too slow for the complexity of planning graph.
- This problem has 12 fluents, so the search space has $2^{12} = 4096$ states, which is not a big number so the time elapsed of every methods in this problem is not significantly different.

Problem 2

Optimal plan for problem 2

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)

Search method comparison for problem 2

	Optimality	Plan Length	Time Elapsed	Expansions	Goal Tests	New Nodes
breadth_first_search	yes	9	**10.284**	3343	4609	30509
depth_first_graph_search	no	619	**2.598**	624	625	5602
uniform_cost_search	yes	9	32.155	4853	4855	44041
greedy_best_first_graph_search with h_1	no	21	5.227	998	1000	8982
a* with h_1	yes	9	32.655	4853	4855	44041
a* with h_ignore_preconditions	yes	9	**10.541**	1506	1508	13820
a* with h_pg_levelsum	yes	9	176.182	**86**	**88**	841
a* with h_pg_setlevel	yes	9	160.780	89	91	**778**

Observation and analysis for problem 2

- *depth_first_graph_search* is the fast solution but not opimal.
- *breadth_first_search* is the fastest solution in optimal methods. It is always a optimal search (Russell-Norvig AIMA 3rd Ed 3.4.1).
- *uniform_cost_search* is also a optimal search (Russell-Norvig AIMA 3rd Ed 3.4.2). But it take more time than *breadth_first_search* because *breadth_first_search* immediately stop the search after expanding the goal node but *uniform_cost_search* do the goal_test when pop the node later.
- *astar_with_h_ignore_preconditions* is the fastest solution in astar methods and not too much slower than *breadth_first_search*
- *astar_with_h_pg_levelsum* and *astar_with_h_pg_setlevel* also perform well in node expansions
- This problem has 27 fluents, so the search space has $2^{27} = 134217728$ states, which is siginificantly larger than problem 1. Some methods like *breadth_first_tree_search* cannot get a solution in reasonable time.

Problem 3

Optimal plan for problem 3

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

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

Search method comparison for problem 3

	Optimality	Plan Length	Time Elapsed	Expansions	Goal Tests	New Nodes
breadth_first_search	yes	12	74.178	14663	18098	129631
depth_first_graph_search	no	392	**1.305**	408	409	3364
uniform_cost_search	yes	12	273.602	18223	18225	159618
greedy_best_first_graph_search with h_1	no	22	71.990	5578	5580	49150
a* with h_1	yes	12	274.343	18223	18225	159618
a* with h_ignore_preconditions	yes	12	**63.538**	5118	5120	45650
a* with h_pg_levelsum	yes	12	1187.516	414	416	3818
a* with h_pg_setlevel	no	13	942.667	**367**	**369**	**3178**

Observation and analysis for problem 3

- *greedy_best_first_graph_search with h_1* works not well as before because the optimal model in this problem is no longer 'Load -> Fly -> Unload'. In this problem the optimal model is 'Load -> Fly -> Load -> Fly -> Unload', so this method lose it's innate advantage.
- *depth_first_graph_search* is the fast solution but not opimal.
- *breadth_first_search* is no longer the fastest solution in optimal methods for more bigger search space. But simple enough so it's not a bad solution.
- *uniform_cost_search* and *astar_with_h_1* has same result in very problems which is not in accident, because implemention of these two methods:
  - `uniform_cost_search = best_first_graph_search(problem, lambda node: node.path_cost)`
  - `astar_with_h_1 = best_first_graph_search(problem, lambda node: node.path_cost + 1)`
- *astar_with_h_ignore_preconditions* is the fastest solution in every methods. It should caused by the simple implemention of the heuristic in this large search space.
- *astar_with_h_pg_levelsum* and *astar_with_h_pg_setlevel* also perform well in node expansions. But the former

give a optimal plan and the latter don't. This is strange for the former can be inadmissible(but works well in practice for problems that are largely decomposable) and the latter is admissible(Russell-Norvig AIMA 3rd Ed 3.10.3). I will debug deeper later for this potential bug.

- This problem has 32 fluents, so the search space has $2^{32} = 4294967296$ states. Every methods take more time and spaces to get a plan.

Conclusions

As shown above, for a optimal plan:

- In comparison of time elapsed, *astar_with_h_ignore_preconditions* is better when search spaces is large and *breadth_first_search* is better when search spaces is small.
- In comparison of node expansions, *astar_with_h_pg_levelsum* is better.