Implement a Planning Search

Introduction

This project is to finish the code of *my_air_cargo_problem.*py and *my_planning_graph.*py from **https://github.com/udacity/AIND-Planning**. The objective is to use PDDL (Planning Domain Definition Language) to define the problem and then find the solution by applying the various search algorithms. On top of that, different heuristics are implemented in the search to find the optimal solutions. One of the most important heuristics is by using planning graph.

Optimal Solutions

Cargo Problem 1

Below is the optimal solution for cargo problem 1 and it has 6 steps.

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Fly(P1, SFO, JFK)

Fly(P2, JFK, SFO)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

Cargo Problem 2

The optimal solution of cargo problem 2 has 9 steps.

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)

Cargo Problem 3

The optimal solution of problem 3 has 12 steps.

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(C3, P1, JFK)

Unload(C2, P2, SFO)

Unload(C1, P1, JFK)

Analysis of Search Performance

Non-Heuristic Search

Besides breadth-first and depth-first, I picked uniform-cost as the third search algorithm for the performance comparison. Let's look at the statistics on problem 1 first.

P1	breadth_first_search	depth_first_graph_search	uniform_cost_search
Expansions	43	12	55
Goal Tests	56	13	57
New Nodes	180	48	224
Plan Length	6	12	6
Time elapsed	0.03	0.01	0.04

All the three searches are super-fast, which may due to the problem size is too small. Depth-first takes the least time and node expansions but its cost is the highest (plan length). By the nature of depth-first, it's neither complete (though it found the solution in this finite problem and didn't met loops) nor optimal. Breadth-first and uniform-cost provide the optimal solution with cost of 6.

Statistics of problem 2 as below.

P2	breadth_first_search	depth_first_graph_search	uniform_cost_search
Expansions	3343	582	4853
Goal Tests	4609	583	4855
New Nodes	30509	5211	44041
Plan Length	9	575	9
Time elapsed	13.92	3.4	47.26

With the increment of problem search space, the performance varies by orders of magnitudes. Depth-first is still the fastest but a lot more expensive. The uniform-cost takes

longer time than breadth-first. It's not only because it searches more nodes, but also because it's slightly more expensive to calculate the node cost and put the node in the priority queue. We know typically a sorting problem's cost is O(log(n)) which is greater than O(1). Again, only breadth-first and uniform-cost can find the optimal solution. And problem 3 statistics follow the same observation.

P3	breadth_first_search	depth_first_graph_search	uniform_cost_search
Expansions	14663	627	18222
Goal Tests	18098	628	18224
New Nodes	129631	5176	159609
Plan Length	12	596	12
Time elapsed	108.75	3.6	426.59

Heuristic Search

In this project, the A star search has three variations with different heuristic functions. The first one is *h_1* returns 1 all the way. The second one is *h_ignore_preconditions* returns the minimum number of actions to achieve the goal by ignoring preconditions, in other words, the number of unsatisfied states in the goal. The last one is *h_pg_levelsum*, it builds a complete planning graph first and then searches through the state levels to find the first level that exists the individual goal state and sum the level cost for each individual goal state.

All the three heuristics are admissible. The *h_1* and *h_ignore_preconditions* are obviously admissible as we assumed the goals are independent. The actual cost is always equal or greater than them because they always return 1 or ignore preconditions. The *h_pg_levelsum* uses a planning graph to estimate the cost. Our goal is to find a state level that fulfills all the goal states while *h_pg_levelsum* only finds the first state level fulfills each individual goal. Thus, it's admissible. It also provides a better estimation of the cost in terms of search space. See below three tables.

P1	a* h_1	a* h_ignore_precondictions	a* h_pg_levelsum
Expansions	55	55	11
Goal Tests	57	57	13
New Nodes	224	224	50
Plan Length	6	6	6
Time elapsed	0.03	0.04	1.52

P2	a* h_1	a* h_ignore_precondictions	a* h_pg_levelsum
Expansions	4853	4853	86
Goal Tests	4855	4855	88
New Nodes	44041	44041	841

Plan Length	9	9	9
Time elapsed	45.08	43.66	135.68

P3	a* h_1	a* h_ignore_precondictions	a* h_pg_levelsum
Expansions	18222	18222	414
Goal Tests	18224	18224	416
New Nodes	159609	159609	3818
Plan Length	12	12	12
Time elapsed	375.68	373.29	1080.98

The search space of *h_pg_levelsum* is much smaller than the other two as we can see from the number of expansions and new nodes. All the heuristics return the optimal solution. However, the tradeoff is *h_pg_levelsum* needs a lot more time to build the planning graph.

Which One Heuristic/Search Is Better?

Depending on what matters for the problem, it's difficult to say one heuristic is better than the others. *h_1* and *h_ignore_preconditions* may need more memory for the node expansions, but they only take 1/3 of the time of *h_pg_levelsum*. The magnitude of node expansions of *h_pg_levelsum* is two orders smaller than the other two heuristics, a remarkable difference! However, it takes much longer time.

When we consider these problems in real world, probably I won't choose either of the heuristics. The breadth-first search is complete and optimal, most importantly, it's fast!

	Time for best non_heuristic (s)	Time for best heuristic (s)
P1	0.03	0.04
P2	14.22	43.66
P3	108.75	373.29

Other Thoughts Beyond Just Searching

The planning graph algorithm has a lot of code to compute the mutex of actions and states. In this project, it is used to compute the level sum heuristic score, rather than extracting a solution. The mutex information are of no use in this scenario. So, a faster solution may be achieved by ignoring computing mutex. Or extracting a solution from the graph directly instead of using it for heuristic to perform search.