Analysis of the problem

Optimal Plans for Each problem

Many different algorithms provide similar optimal planning. Here, the chosen optimal plan is from any of the algorithm that results in the shortest plan length.

- Problem 1, Optimal Planning (length = 6)

problem setting

```
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))
```

an optimal plan

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

- Problem 2, Optimal Planning (length = 9)

problem setting

```
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))
```

an optimal plan

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
```

- Problem 3, Optimal Planning (length = 12)

problem setting

```
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))
```

an optimal plan

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

Non-heuristic search: Compare and Contrast

Three different search methods are tested:

1. breadth first search (BFS)
2. depth first graph-search (DFS)
3. uniform cost search (UCS)

- Problem 1: `air_cargo_p1`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length
BFS	43	56	0.026s	6
DFS	21	22	0.012s	20
UCS	55	57	0.030s	6

- Problem 2: `air_cargo_p2`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length
BFS	3346	4612	15.11s	9
DFS	107	108	0.33s	105
UCS	4853	4855	11.32s	9

- Problem 3: `air_cargo_p3`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length
BFS	14120	17673	121.96s	12
DFS	292	293	1.22s	288
UCS	18223	18225	48.5s	12

Discussion:

It is quite surprising that the DFS algorithm is really fast, although the problem setting markdown file warned that it may take more than 10 minutes to run. I have randomly shuffled the order of inits, pre-conditions, and goals. My intention was to create different order of conditions such that the DFS may stuck at one condition in which it needs to search for a long time. In fact, my random-order strategy didn't bring difficulties to DFS at all. So, in the end, DFS algorithm became the fastest algorithm among all, although it didn't provide an optimal solution. We can see that the solution from DFS is much longer than others.

UCS algorithm should be quite similar to BFS in our current problem setting because the cost of each edge is 1. However, it expands more nodes than BFS. I think it is because of its stopping rule is different from BFS -- there is a subtle difference between when BFS stops and when UCS stops, see the course video from details. Interestingly, UCS is still faster than BFS -- even though the former expands more nodes than the latter.

Heuristic Search: Compare and Contrast

Three Different search methods are tested:

1. astar_search h_1 (A*H1)
2. astar_search h_ignore_preconditions (A*IP)
3. astar_search h_pg_levelsum (A*PG)

- Problem 1: `air_cargo_p1`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length
A*H1	55	57	0.019s	6
A*IP	41	43	0.022s	6
A*PG	11	13	0.385s	6

- Problem 2: `air_cargo_p2`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length
A*H1	4853	4855	6.56s	9
A*IP	1450	1452	2.42s	9
A*PG	86	88	35.48s	9

- Problem 3: `air_cargo_p3`

Search	# Node Expansions	# Goal Tests	Time Elapsed	Plan Length

A*H1	18223	18225	29.60s	12
A*IP	5040	5042	9.46s	12
A*PG	316	318	176.68s	12

Discussion:

From the results, the A-star search with *ignore-preconditions* heuristics performed very well. It provides an optimal solution, searched for less nodes, and computed results really fast.

The *pg_levelsum* heuristics expands the least amount of nodes and performed the least amount of goal tests. That means this algorithm is *intelligent* and only need to explore a small amount of states to come up with an optimal solution. Its speed, however, is not the fastest.

General Discussion

In this project, I have tested six different algorithms to solve the fight-planning problem. Three of them are plain search, and the other three are heuristic search.

It turns out that multiple algorithms have computed an optimal solution for the planning problem -- optimal in the sense that the solution has a minimal plan length. In order to get this optimal solution, different algorithms needs different amount of time and computation.

Heuristic search is helpful to get an optimal solution in a short duration. Among all tested heuristic algorithms, the one that ignores the pre-conditions is the best one used for our current problems. It is better than all Non-heuristic algorithms: it is faster than all other algorithms that can produce the optimal solutions; it provides a better solution than DFS, although DFS is much faster.

Not all heuristic search are good. Although a heuristic search may expand less search nodes or do less goal tests, it may still requires a long time to compute its heuristic function. This is why `A*PG` algorithm is really slow: it takes a long time to calculate the level sum as its evaluation value for each state. Similarly, `A*IP` also spend some time to evaluate its state. In general, to evaluate a search state using the *ignore precondition* heuristic is to solve a *set cover* problem, which itself is a NP-hard problem (Norvig and Russell, 2009). Luckily in our current problem, each action can only reach a single goal, so it is much easier to solve this special *set cover* problem. Thus the `A*IP` is quite fast.

It is interesting that DFS is so fast. But it does not provide an optimal solution.