Planning Search for the Air Cargo Domain

Problem Analysis

Alberto Gaona

November 2017

1 Introduction

We will analyze the metrics of using different search algorithms to find the most efficient solution to the following classical PDDL problems:

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))
```

Using the following action scheme for the Air Cargo Domain:

```
Action(Load(c, p, a),
        PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
        PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
        PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
        EFFECT: ¬ At(p, from) ∧ At(p, to))
```

The result to the problems should be the following paths:

Problem 1:

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

Path length = 6

Problem 2:
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

Path length = 9

Problem 3:
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)
Unload(C4, P2, SFO)
Fly(P1, ATL, JFK)

Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

Path length = 12

2 Non-heuristic planning solution searches

In this first part we used the algorithms of Breadth-First search (BFS), Depth-First graph search (DFS) and Uniform Cost search. The results for each of the problems is:

Problem 1:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
BFS	43	56	180	6	0.9937
DFS	12	13	48	12	0.0301
UCS	55	57	224	6	0.1048

Problem 2:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
BFS	3343	4609	30509	9	35.1976
DFS	582	583	5211	575	7.7024
UCS	4853	4855	44041	9	31.4177

Problem 3:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
BFS	11287	15028	88574	12	183.1604
DFS	832	833	3692	662	4.9232
UCS	15795	15797	122379	12	109.5024

As we can observe in the above tables, when we using Depth-First search our plan length is not correct, it increases considerably which discards immediately that DFS is optimal therefore if it is necessary to select the best in this first part of Non-heuristic planning solution searches it would be between BFS and UCS. The two arrive at the same solution, both are optimal.

BFS explores fewer paths however, although UCS preforms more "work" it is faster for the problems of this analysis, therefore, the best in this context and for this specific problem scenario we will consider use UCS since the time is better than BFS.

Note: I tried to obtain metrics using Depth Limited search (DLS) algorithm however the times is too high obtaining the following partial results:

Metrics of DLS

-	Expansions	Goal tests	New nodes	Plan length	Time (sec)
Problem 1	101	271	414	50	0.25
Problem 2	222719	2053741	2054119	50	2577.48
Problem 3	-	-	-	-	more than 4 h

3 Heuristic planning solution searches

In this second part we will look for the solution to the problems through the search algorithm A star, we will use two different heuristics:

- **ignore preconditions:** The minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed.
- **level sum:** The sum of the level costs of the individual goals (admissible if goals independent)

Note: h_1 is not a true heuristic, only return 1

Problem 1:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
A* h: 1	55	57	224	6	0.1053
A* h: ignore precond	41	43	170	6	0.1035
A* h: level sum	11	13	50	6	2.3245

Problem 2:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
A* h: 1	4853	4855	44041	9	32.8091
A* h: ignore precond	1428	1430	13085	9	11.7245
A* h: level sum	86	88	841	9	474.0205

Problem 3:

Algorithm	Expansions	Goal tests	New nodes	Plan length	Time (sec)
A* h: 1	15795	15797	122379	12	108.69
A* h: ignore precon	3655	3657	29021	12	30.64
A* h: level sum	467	469	3229	12	2240.11

In the above tables we can see that the three different versions of heuristic come to the correct path length of the solution in each of the problems, we can also see that by using the heuristic *level sum* we find in a more direct way the solution, however the time to find it is too big so it is not optimal.

Regarding choosing the best version, we notice that the heuristic *ignore preconditions* is the one that works best so for this second part of heuristic planning solution searches is best to use A start with ignore preconditions heuristic.

4 Conclusion

In part two and part three we observed how the different search algorithms performed, where a non-heuristic search the *Uniform Cost search* works better while if we using heuristics *A star with ignore preconditions heuristic* is the best.

Between these two last algorithms, the definitive one that we must use is *A star with ignore preconditions heuristic*, this because it finds the solution very directly, besides that the time in which it finds it is approximately three times faster (see the following table). Also because the A star algorithm is the combination of the best features of the UCS and BFS algorithms, so it is better in every way.

Algorithm	Problem	Time (sec)
UCS	1	0.1048
A* h: ignore precon	1	0.1035
UCS	2	31.4177
A* h: ignore precon	2	11.7245
UCS	3	109.5024
A* h: ignore precon	3	30.64