HEURISTIC ANALYSIS FOR AN AIR CARGO PLANNING

Federico Martini (Software Engineer)

Udacity

***Abstract**: Development and Analysis of a planning search agent to solve deterministic logistic planning problems for an Air Cargo transport system. With progression search, optimal plans for each problem will be computed.*

***Keywords:** planning, artificial intelligence, optimal plans, search agent*

1. Introduction

This document goes through the implementation, computation and results analysis of a planning search agent to solve deterministic planning problems applied to an Air Cardo transport system. For this purpose, three different problems have been considered, and they have been defined using PDDL language. Each problem have the same action schema, but different initial states and goals. The algorithms applied to solve the problems are based both on a non-heuristic and heuristic approach. The results will be compared to understand what's the optimal algorithm in terms of path length, computational time, number of nodes, etc. to be used to solve the three problems. All the algorithms that will take more than 10 minutes to provide a solution, will be discarded.

2. Search Types

The planning search agent has been implemented by taking advantage of different search algorithms leveraging of both non-heuristic and heuristic methods. Below is the list of the search types used to compare the search results in the following sections:

- Breadth First Search (BFS)
- Depth First Graph Search (DFGS)
- Depth Limited Search (DLS)
- Uniform Cost Search (UCS)
- A* Search Heuristic 1 (A* h_1)
- A* Search Ignore Preconditions Heuristic (A* h_IP)
- A* Search Planning Graph Level Sum Heuristic (A* h_LSum)

Where *h_1* is not a true heuristic cause it returns always 1. Instead, as for the other two heuristics adopted in the A* Search, below are the descriptions of how they have been implemented:

- **h_IP**: This heuristic estimates the minimum number of actions that must be carried out from the current state in order to satisfy all the goal conditions by ignoring preconditions required for an action to be executed.
- **h_LSum**: This heuristic uses a planning graph representation of the problem state space to estimate the sum of all actions that must be carried out from the current state in order to satisfy each individual condition

3. Air Cargo Action Schema

The Action schema is the same for each problem we'll try to solve in this document. The description of the Action Schema in PDDL is the following.

```
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))
```

4. Air Cargo transport system problem 1

The first Air Cargo transport system problem is detailed by the following description in PDDL:

```
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))
```

The results of the Search algorithms described at Chapter 2 are shown below:

	Exp.	Goal Tests	New Nodes	Lenght	Time	Optimal
BFS	43	56	180	6	0,0341	Yes
DFGS	12	13	48	12	0,0082	No
DLS	101	271	414	50	0,0921	No
UCS	55	57	224	6	0,0405	Yes
A* Search h_1	55	57	224	6	0,0372	Yes

A* h_IP	41	43	170	6	0,0408	Yes
A* h_Lsum	11	13	50	6	0,9538	Yes

Table 1 Air Cargo problem 1 – Search results

5. Air Cargo transport system problem 2

The second Air Cargo transport system problem is detailed by the following description in PDDL:

```
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))
```

The results of the Search algorithms described at Chapter 2 are shown below:

	Exp.	Goal Tests	New Nodes	Lenght	Time	Optimal
BFS	3401	4672	31049	9	14,8304	Yes
DFGS	350	351	3142	346	1,5914	No
DLS					> 10 min	
UCS	4761	4763	43206	9	13,1435	Yes
A* Search h_1	4761	4763	43206	9	13,2992	Yes
A* h_IP	1450	1452	13303	9	4,9187	Yes
A* h_Lsum	86	88	841	9	180,8819	Yes

Table 2 Air Cargo problem 2 – Search results

6. Air Cargo transport system problem 3

The third Air Cargo transport system problem is detailed by the following description in PDDL:

```
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))
```

The results of the Search algorithms described at Chapter 2 are shown below:

	Exp.	Goal Tests	New Nodes	Lenght	Time	Optimal
BFS	14491	17947	128184	12	118,5426	Yes
DFGS	1948	1949	16253	1878	21,8626	
DLS					> 10 min	
UCS	17783	17785	155920	12	62,6144	Yes
A* Search h_1	17783	17785	155920	12	59,2845	Yes
A* h_IP	5003	5005	44586	12	20,4329	Yes
A* h_Lsum	311	313	2863	12	1000,0314	Yes

Table 3 Air Cargo problem 3 – Search results

7. Results comparison

The results show that a *Breadth First Search* approach always provides an optimal result in a good amount of time, cause it always considers the shortest path before moving on. The *Depth First Graph Search*, is not optimal, even though it's not expensive in terms of memory usage as we can see by reading the amount of expansions, goal tests and new nodes required by this algorithm. The problem of the DFGS algorithm is that it prioritizes the depth instead of the shortest path and doesn't take into account when a node lead to a better solution compared to another. The Depth limited search. The Depth Limited Search doesn't provide an optimal solution and for the Problem 2 and 3 it didn't event find a solution in 10 minutes. The reason behind this behavior is that DLS is able to find a solution only if it's within a certain depth. As for the Uniform Cost Search, it always find an optimal solution, but it requires more memory compared to the other non-heuristic method used during the tests.

As for the heuristic methods, as we can see, the complexity of the algorithm is not justified when the problem to solve is easy (Problem 1 and 2), where the even the non-heuristic methods have given good results. One interesting aspect emerge when comparing the *A* Search h_1* with the UCS. In fact, they give almost the same results for all the three problems. The reason behind such behavior relies on the heuristic h_1, which add just 1 to the cost of the node, so they should be ordered in the same way as in UCS in the priority queue. Further, the *A* Search h_LSum* performed good with easy problem, but it behaved poorly in Problem 3, showing that the *h_LSum* heuristic it's likely too complex to be used to solve hard problems.

4. Conclusions

Even though the non-heuristic methods performed good for Problem 1 and 2, they were outperformed by the *A* Search h_IP* in the third problem. Thus, the best algorithm that seems to fit well for all the problems tackled in this search is the *A* Search h_IP*, that was able to find an optimal solution in a reasonable amount of time and without even requiring too much memory. Below are the optimal sequences of actions found with the Search Agent, for the three Air Cargo problems:

Air Cargo Problem	Search Algorithm	Optimal
1	A* Search h_IP	Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)

		Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)
2	A* Search h_IP	Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)
3	A* Search h_IP	Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Unload(C4, P2, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)