Planning Search: Performance & Heuristic Analysis

Rohit Mehra
contactrohitmehra@gmail.com

I. Introduction: Why this analysis?

In this project we implemented deterministic logistics planning problems for an Air Cargo transport system. We were given three problems defined in classical PDDL (Planning Domain Definition Language) and our first goal was to implement these problems in Python. Our next step was to implement relaxed problem heuristic, planning Graph and automatic heuristic (levelsum heuristic, based on planning graph). Earlier we implemented generic graph search algorithms for Problem Solving Agents, these algorithms can also be used by Planning Search Agents, with a change in search space from state space to plan space. This report gives an analysis of these searches and heuristics when applied to the given planning problems.

II. Air Cargo transport system: Action Schema

Below is the Action Schema applicable to the given problems in Air Cargo transport system 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))
```

III. Problems: Defined in classical Planning Domain Definition Language

Given are the problem definitions:

1. 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))
```

2. Problem 2:

```
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))
```

3. Problem 3:

```
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))
```

Following are the optimal solutions of the given problems respectively, as searched by Planning Search agents:

1. Optimal length of plan for this problem is 6, and the plan is as follows:

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

2. Optimal length of plan for this problem is 9, and the plan is as follows:

```
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)
```

3. Optimal length of plan for this problem is 12, and the plan is as follows:

```
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)
```

IV. Search Strategy Type: Uninformed

These type of searches, as the name suggests do not have much information about the states in the search space given to them. The only valuable information they have is to identify the goal state. The general idea for these algorithms is to chose a state to explore/expand, test it for goal state and then continue. The choice of expanding a node is made trivially by the logic of ordering i.e. FIFO, LIFO etc. They do not have the information to decide which non-goal state is the "best option" towards the end goal. Here, we compare the performance of seven such search algorithms when used by Planning agent in the given problems. Comparison is in terms of speed (execution time), memory usage (corresponding to the node expansions by strategy) and optimality (as in the search algorithm found the optimal path or not).[Russell & Norvig]

Note that when the time exceeded 10 mins, performance is not reported.

Search Strategy	Time(s)	Plan Length	Expansions	Optimal	New Nodes
Breadth First Search	0.06	6	43	Yes	180
Breadth First Tree Search	1.06	6	1458	Yes	5960
Depth First Graph Search	0.02	20	21	No	84
Depth Limited Search	0.09	50	101	No	414
Uniform Cost Search	0.04	6	55	Yes	224
Recursive Best First Search (Null Heuristic)	3.10	6	4229	Yes	17023
Greedy Best First Graph Search (Null Heuristic)	0.005	6	7	Yes	28

Table 1: Performance Report of Problem 1

Search Strategy	Time(s)	Plan Length	Expansions	Optimal	New Nodes
Breadth First Search	17.45	9	3343	Yes	30509
Breadth First Tree Search	>10min	-	-	-	-
Depth First Graph Search	5.19	619	624	No	5602
Depth Limited Search	>10min	-	-	-	-
Uniform Cost Search	17.00	9	4852	Yes	44030
Recursive Best First Search (Null Heuristic)	>10min	-	-	-	-
Greedy Best First Graph Search (Null Heuristic)	3.49	15	990	No	8910

Table 2: Performance Report of Problem 2

Search Strategy	Time(s)	Plan Length	Expansions	Optimal	New Nodes
Breadth First Search	135.00	12	14663	Yes	129631
Breadth First Tree Search	>10min	-	-	-	-
Depth First Graph Search	1.96	392	408	No	3364
Depth Limited Search	>10min	-	-	-	-
Uniform Cost Search	62.9	12	18235	Yes	159716
Recursive Best First Search (Null Heuristic)	>10min	-	-	-	-
Greedy Best First Graph Search (Null Heuristic)	20.16	22	5614	No	49429

Table 3: Performance Report of Problem 3

V. Search Strategy Type: Informed

Unlike uniformed search strategies, these strategies use the information about a states "goodness" in terms of achieving the end goal. The goodness measure is used to direct the search in a more promising direction, rather than searching over the whole space uniformly or using some other trivial logic.[Russell & Norvig] This heuristic comes usually from the knowledge of problem domain. We experimented with A* search strategy using null(equivalent to uniform cost search), Ignore Preconditions and Level Sum heuristic.

Search Strategy	Time(s)	Plan Length	Expansions	Optimal	New Nodes
A* (Null Heuristic)	0.062	6	55	Yes	224
A* (Ignore Preconditions Heuristic)	0.26	6	41	Yes	170
A* (Level Sum Heuristic)	20.16	6	11	Yes	50

Table 4: Performance Report of Problem 1

Search Strategy	Time(s)	Plan Length	Expansions	Optimal	New Nodes
A* (Null Heuristic)	14.71	9	4852	Yes	44030
A* (Ignore Preconditions Heuristic)	88.63	9	1450	Yes	13303
A* (Level Sum Heuristic)	68.54	9	86	Yes	841

Table 5: Performance Report of Problem 2

Search Strategy	Time (s)	Plan Length	Expansions	Optimal	New Nodes
A* (Null Heuristic)	60.41	12	18235	Yes	159716
A* (Ignore Preconditions Heuristic)	382.93	12	5040	Yes	44944
A* (Level Sum Heuristic)	371.66	12	325	Yes	3002

Table 6: Performance Report of Problem 3

VI. What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?

Overall best performing strategy is A* Search using Level Sum Heuristic. If we consider problems with small branching factor BFS is both fast and optimal. But as the branching factor increases exponentially, its performance degrades, specially in terms of storage.

Levelsum Heuristic which is based on planning graph, is very accurate and gives the best heuristics with an overhead of computing the graph once. This heuristics is valuable in problems where branching factor is high, the problem is deterministic and decomposable.

It performed better than non-heuristic search planning in bigger problems as it expanded fewer nodes and also very few nodes were added to the graph. It did not perform better in terms of execution time because of the overhead of computing the graph, which is acceptable as the graph is rich in information and can be used for other heuristics and strategies as well.