Classical Planning: A Heuristics Analysis

In this project we will investigate the world classical planning, creating plans for an Air Cargo transport system. We define the planning problems in PDDL language and solve the problem by both classical uninformed search algorithms and informed (heuristic) search algorithms with auto generated **domain-independent** heuristics

1. Air Cargo action schema definition

```
    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))
```

2. Problem Definition

The problems are defined by giving the initial and goal states

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))
```

3. Problem Solutions

The problems can be solved with many different plans, but the optimal plan length for each problem is 6, 9 and 12 actions respectively.

Here are (some) of the optimal solutions:

Problem 1

```
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

```
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)
```

Problem 3

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

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

4. Uninformed Search Results

Here are there results summarized for each problem in a table

Problem 1

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
Breadth First Search	YES	6	0.11	43	180	56	NO
Depth First Search	NO	20	0.04	21	84	22	NO
Uniform Cost	YES	6	0.12	55	224	57	NO
Recursive Best First Search	YES	6	9.55	4229	17023	4230	NO
Greedy Best First Graph Search	YES	6	0.016	7	28	9	NO

Problem 2

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
Breadth First Search	YES	9	39	3343	30509	4609	NO
Depth First Search	NO	624	8	624	5602	625	NO
Uniform Cost	YES	9	50	4762	43217	4764	NO
Recursive Best First Search	-	-	-	-	-	-	-
Greedy Best First Graph Search	YES	9	5.9	558	5022	560	NO

Problem 3

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
Breadth First Search	YES	12	236	14663	129631	18098	NO

Depth First Search	NO	392	5.37	408	3364	409	NO
Uniform Cost	YES	12	216	17246	151462	17248	NO
Recursive Best First Search	-	-	-	-	-	-	-
Greedy Best First Graph Search	NO	14	0.47	38	329	40	NO

A quick view of the results shows that **Breadth First Search** and **Uniform Cost Search** always give an optimal solution, at roughly the same times. **Greedy Best First Search** is the next that hits the eye. Its much faster than any other, and the solutions it gives are not much larger than the optimal solutions.

5. Uninformed Search Results

Here are there results summarized for each problem in a table

Problem 1

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
A* with h1 heuristic	YES	6	0.12	55	224	57	YES
A* with Ignore Preconditions heuristic	YES	6	0.13	41	170	43	YES
A* with Levelsum heuristic	YES	6	0.41	11	50	13	YES

Problem 2

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
A* with h1 heuristic	YES	9	50	4792	43217	4764	YES
A* with Ignore Preconditions heuristic	YES	9	22	1450	13303	1452	YES
A* with Levelsum heuristic	YES	9	31	86	841	88	YES

Problem 3

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
A* with h1 heuristic	YES	12	229	17650	154810	17652	YES
A* with Ignore Preconditions heuristic	YES	12	85	4790	42535	4792	YES

A* with Levelsum heuristic	YES	12	131	266	2438	268	YES

Of course the first search (A* with h1 heuristic) is nothing else than Uniform Cost search , since we do not provide any actual function as a heuristic. From the other two heuristics we can see that the **A* search with LevelSum heuristic** does a much better job guiding the search to the correct destination. **A* with ignore Preconditions** searches **x17 times** more nodes than the later to reach the correct destination.

The big disadvantage is that the time required to build the planning graph and calculate the sum is huge. On the other hand, there are many optimizations that can be made in the code given. For example in the template code given, we had the Action class store the preconditions and effects in a list. That results, when we iterate over pairs of actions to find if they are in mutex or not, we have a precondition in hand and search for it in the preconditions or effects of the other action. This action in Lists in Python is O(n) but in Sets its O(1) on average. Since the main bottleneck in the algorithm is the mutex computation, such changes will boost the performance a lot.

Of course the main goal of the planning graph is not to provide the level sum heuristic, but to form the basis of the GRAPHPLAN algorithm.

Conclusion

By comparing the results of the best (Optimal & faster) uninformed search method and the best (Optimal & faster) informed search method in the table below, we understand that as the complexity of the problem becomes higher, the importance of a good heuristic to guide the search is important. For easier problems, breadth first search is on par with the heuristic search, and maybe a little bit faster

Search Strategy	Optimal	Length	Time(s)	Expansion	New Nodes	Goal Tests	Heuristic
A* with Ignore Preconditions P1	YES	6	0.13	41	**170**	43	YES
Breadth First Search P1	YES	6	**0.11**	43	180	56	NO
A* with Ignore Preconditions P2	YES	9	**22**	1450	**13303**	1452	YES
Breadth First Search P2	YES	9	39	3343	30509	4609	NO
A* with Ignore Preconditions P3	YES	12	**85**	4790	**42535**	4792	YES
Breadth First Search P3	YES	12	236	14663	129631	18098	NO