Dalila Benachenhou
AIND Planning Problem

Problem Setting:

We want to investigate (heuristic and non-heuristic) searches for planning the shipment of cargos from starting airport to destination airport. For non-heuristic searches, we have decided on Breath First Search, BFS, Depth First Graph Search, DFGS, and Uniform Cost Search, UCS; for heuristic searches, we have decided on A* search with constant heuristic, A* 1, with preconditions ignored, A*IP, and with Planning Graph Level Sum, A* PG.

To fly a plane we need to load the cargo into the plane. When the plane reach destination, the cargo is unloaded. Hence, to get a cargo from airport A to airport C, where A is different from C, the sequence of actions are as follows:

1. Load cargo into a plane at airport A
2. Fly plane to airport C
3. Unload cargo at Airport C.

One has to assume that preconditions have to be met for a cargo to be flown from a starting airport to a destination airport. All the planning problems are in the appendix.

Three problems where presented to all the searches. The problems go from the simplest with only 2 cargos with dedicated planes, problem 1, the complex with 4 cargos but only 2 planes, problem 3.

Problem 1

In problem 1, we want to move 2 cargos (C1, and C2) from SFO, and JFK, respectively, to JFK, and SFO using 2 planes. Both planes are located at each cargo initial location

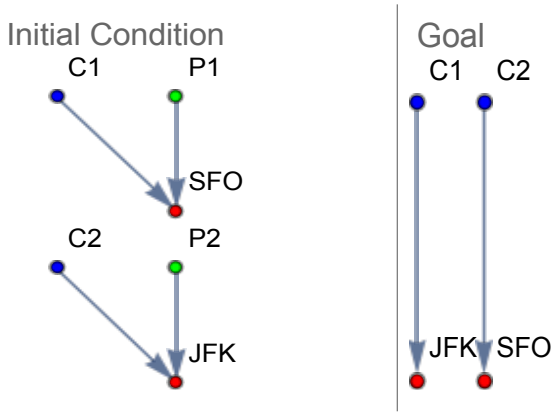

Figure 1. Initial and Goal Conditions

The minimum number of steps to getting from initial condition to goal is also number of cargo * number of actions: 2*3 = 6

Dalila Benachenhou
AIND Planning Problem

Problem 2

In problem2, we want to move 3 cargos using 3 planes located in the same airports as the cargos to 2 distinct airports. The graph in the left shows the initial conditions where C1 and P1 are in SFO, C2 and P2 are in JFK, and C3 and P3 are in ATL. The graph in the right shows the final destination of the cargos: C1 ends up in JFK, and C2 and C3 ends up in SFO.

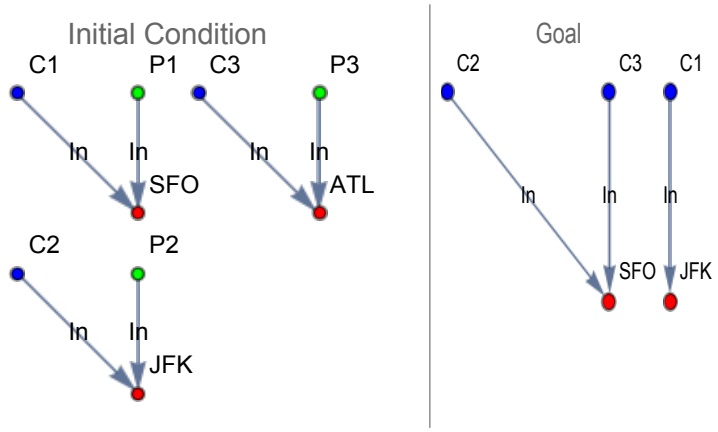

Figure 2. Initial and Goal Conditions

The minimum number of steps to getting from initial condition to goal is also number of cargo * number of actions: 3*3 = 9

Problem 3:

Problem 3 has more cargos to move but uses only 2 planes to move them, and some cargos have the same destination: C1 and C2 destination is JFK, while C2 and C4 destination is SFO.

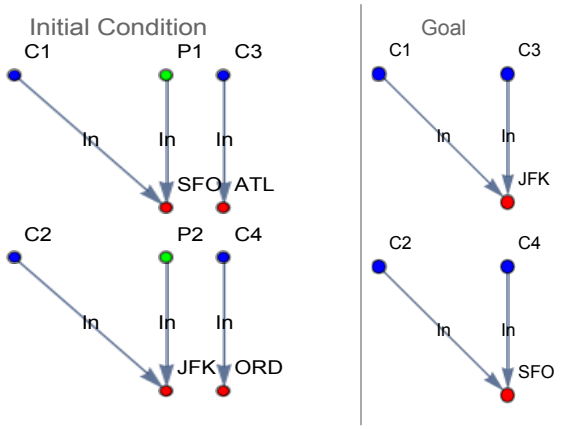

Figure 3. Initial and Goal Conditions

The minimum number of steps to getting from initial condition to the goal is also number of cargo * number of actions: 4*3 = 12
The formula for minimum number of steps from initial condition to the goal will be out constant heuristic in A*H1.

Results

Non Heuristic Searches:

For all three problems, we ran Breadth First Search, BFS, Depth First Graph Search, DFGS, and Uniform Cost Search (UCS). For all these searches we measured the number of expansions, number of goal test, number of new nodes created, the plan length, and time it took to reach the goal.

From Table 1, DFGS produced the least number of expansions, goal tests, number of new nodes, and was faster than both BFS, and UCS for problems 2 and 3—see orange numbers under DFGS. However, it had a big flaw, its plan length was extremely long compared to BFS and UCS in problem 2 and 3. In problem 2, DFGS produced a plan of length of 575 versus the 9 produced by BFS and UCS; in problem 3, DFGS produced a plan of length of 596 versus 12 produced by BFS and UCS. While DFGS was faster than BFS and UCS in problem 2 and problem 3, its outcome was undesirable. This outcome is expected, as DFGS tends to revisits paths, backtrack its path, and hence is suboptimal in graph search as shown in Search lessons 23 to 25 (Search Comparison)

One has to understand, the reason of UCS underperforming BFS in time, has to do with the fact that while BFS will stop when it finds the goal (or destination), UCS will continue looking for other paths that may be shorter than the path already found, and will not stop until it has exhausted all possible paths from an initial point to destination point. While in some cases, taking the first found path may be more desirable, if cost differs given ones path, then UCS is more desirable. Although in this case, both produced the same path length. Hence, going with BFS makes more sense, as it produced less expansions, went over less goal tests, created less new nodes, and took less time than UCS. In short, it was more efficient at its use of memory space, and CPU time.

Dalila Benachenhou
AIND Planning Problem

Table 1. Non-Heuristic Searches Performances

Description	Problems	BFS	DFGS	UCS
Expansions	Problem 1	43	1,458	55
	Problem 2	3,343	582	4,852
	Problem 3	14,663	627	18,223
Goal Tests	Problem 1	56	1,459	57
	Problem 2	4,609	583	4,854
	Problem 3	18,098	628	18,225
New Nodes	Problem 1	180	5,960	224
	Problem 2	30,509	5,211	44,030
	Problem 3	129,631	5,176	159,618
Plan Length	Problem 1	6	6	6
	Problem 2	9	575	9
	Problem 3	12	596	12
Time	Problem 1	0.03	0.97	0.044
	Problem 2	18.92	4.66	56.3
	Problem 3	134.9	4.37	489.96

Heuristic Searches:

We decided solve our three problems using A* Search under various heuristics: constant heuristic, H1, ignore preconditions heuristics, IP, and planning graphs level sum heuristic, PG. As expected, all the heuristics resulted in the same plan length for all three problems. However, IP was the fastest: 0.034 versus (0.52 and 1.48), 15.59 versus (56.74 and 130.53), and 104.13 versus (480.49 and 884.70.) While it was the fastest, PG outperformed H1 and IP in all other performance measurements. It had a less expansions, it went through less tests goals, and created fewer new nodes than H1 and IP. However, when applied to problem 3 it was 8 times slower than IP. This is understandable, PG heuristic has less computational needs, it requires one to seek the location of each goal in the Planning Graph, while IP just requires to see how many of the goals have been reached.

Table 2. A* Heuristic Searches Performances

Descriptions	Problems	H1	IP	PG
Expansions	Problem 1	55	41	11
	Problem 2	4,852	1,506	86
	Problem 3	18,223	5,118	404
Goal Tests	Problem 1	57	43	13
	Problem 2	4,854	1,508	88
	Problem 3	18,225	5,120	406
New Nodes	Problem 1	224	170	50
	Problem 2	44,030	13,820	841
	Problem 3	159,618	45,650	3,718
Plan Length	Problem 1	6	6	6
	Problem 2	9	9	9
	Problem 3	12	12	12
Time	Problem 1	0.53	0.034	1.48
	Problem 2	56.74	15.59	130.53
	Problem 3	480.49	104.13	884.70

Comparing Heuristic and Non Heuristic Searches:

When time and precision are taking into consideration, IP outperformed searches—excluding DFGS. It had the lowest expansions, goal tests, new nodes created, and time lower than BFS. However, when looking at other performance measures, PG outperformed all the searches, including DFSG. H1 performance was equivalent to UCS, which is expected, as H1 is a constant heuristic, and uniform cost search assume a constant heuristic.

Dalila Benachenhou
AIND Planning Problem

We can also compare paths produced by all the searches. One can see, as expected, UCS, and H1 produced same paths in Problem 1 and 2. However, they produced different paths in problem 3. This must be the heuristic that we decided to choose. While H1 used a constant heuristic, the heuristic was not based on equal distance between 2 states. Furthermore, H1and PG produced equivalent paths in Problem 3. Still the memory requirements for H1 were higher than PG. As the purpose of to produced the shortest paths, it matter little what path it is produced, if their cost is equivalent.

Table 3. Problem 1: Paths Found

BFS	UCS	H1	IP	PG
Load(C2, P2, JFK)	Load(C1, P1, SFO)	Load(C1, P1, SFO)	Load(C1, P1, SFO)	Load(C1, P1, SFO)
Load(C1, P1, SFO)	Load(C2, P2, JFK)	Load(C2, P2, JFK)	Fly(P1, SFO, JFK)	Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)	Fly(P1, SFO, JFK)	Fly(P1, SFO, JFK)	Unload(C1, P1, JFK)	Load(C2, P2, JFK)
Unload(C2, P2, SFO)	Fly(P2, JFK, SFO)	Fly(P2, JFK, SFO)	Load(C2, P2, JFK)	Fly(P2, JFK, SFO)
Fly(P1, SFO, JFK)	Unload(C1, P1, JFK)	Unload(C1, P1, JFK)	Fly(P2, JFK, SFO)	Unload(C1, P1, JFK)
Unload(C1, P1, JFK)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)

Table 4. Problem 2: Paths Found

BFS	UCS	H1	IP	PG
Load(C2, P2, JFK)	Load(C1, P1, SFO)	Load(C1, P1, SFO)	Load(C3, P3, ATL)	Load(C1, P1, SFO)
Load(C1, P1, SFO)	Load(C2, P2, JFK)	Load(C2, P2, JFK)	Fly(P3, ATL, SFO)	Fly(P1, SFO, JFK)
Load(C3, P3, ATL)	Load(C3, P3, ATL)	Load(C3, P3, ATL)	Unload(C3, P3, SFO)	Load(C2, P2, JFK)
Fly(P2, JFK, SFO)	Fly(P1, SFO, JFK)	Fly(P1, SFO, JFK)	Load(C1, P1, SFO)	Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)	Fly(P2, JFK, SFO)	Fly(P2, JFK, SFO)	Fly(P1, SFO, JFK)	Load(C3, P3, ATL)
Fly(P1, SFO, JFK)	Fly(P3, ATL, SFO)	Fly(P3, ATL, SFO)	Unload(C1, P1, JFK)	Fly(P3, ATL, SFO)
Unload(C1, P1, JFK)	Unload(C3, P3, SFO)	Unload(C3, P3, SFO)	Load(C2, P2, JFK)	Unload(C3, P3, SFO)
Fly(P3, ATL, SFO)	Unload(C1, P1, JFK)	Unload(C1, P1, JFK)	Fly(P2, JFK, SFO)	Unload(C1, P1, JFK)
Unload(C3, P3, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)

Table 5. Problem 3: Paths Found

BFS	UCS	H1	IP	PG
Load(C2, P2, JFK)	Load(C1, P1, SFO)	Load(C2, P2, JFK)	Load(C2, P2, JFK)	Load(C2, P2, JFK)
Load(C1, P1, SFO)	Load(C2, P2, JFK)	Fly(P2, JFK, ORD)	Fly(P2, JFK, ORD)	Fly(P2, JFK, ORD)
Fly(P2, JFK, ORD)	Fly(P1, SFO, ATL)	Load(C4, P2, ORD)	Load(C4, P2, ORD)	Load(C4, P2, ORD)
Load(C4, P2, ORD)	Load(C3, P1, ATL)	Fly(P2, ORD, SFO)	Fly(P2, ORD, SFO)	Fly(P2, ORD, SFO)
Fly(P1, SFO, ATL)	Fly(P2, JFK, ORD)	Load(C1, P1, SFO)	Unload(C4, P2, SFO)	Load(C1, P1, SFO)
Load(C3, P1, ATL)	Load(C4, P2, ORD)	Fly(P1, SFO, ATL)	Load(C1, P1, SFO)	Fly(P1, SFO, ATL)
Fly(P1, ATL, JFK)	Fly(P2, ORD, SFO)	Load(C3, P1, ATL)	Fly(P1, SFO, ATL)	Load(C3, P1, ATL)
Unload(C1, P1, JFK)	Fly(P1, ATL, JFK)	Fly(P1, ATL, JFK)	Load(C3, P1, ATL)	Fly(P1, ATL, JFK)
Unload(C3, P1, JFK)	Unload(C4, P2, SFO)	Unload(C4, P2, SFO)	Fly(P1, ATL, JFK)	Unload(C4, P2, SFO)
Fly(P2, ORD, SFO)	Unload(C3, P1, JFK)	Unload(C3, P1, JFK)	Unload(C3, P1, JFK)	Unload(C3, P1, JFK)
Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)	Unload(C2, P2, SFO)
Unload(C4, P2, SFO)	Unload(C1, P1, JFK)	Unload(C1, P1, JFK)	Unload(C1, P1, JFK)	Unload(C1, P1, JFK)

Dalila Benachenhou
AIND Planning Problem

Conclusion:

The purpose of this report is to evaluate heuristic and non-heuristic searches for our air cargo-planning problem. We created 3 problems with different degrees of difficulties going from the trivial problem--where one has 2 cargos, 2 planes at the same location as the cargos, but each cargo is at different airplane—to the most complicated one—where we have 4 cargos but only 2 planes, 4 starting airports but only 2 destination airports.

For the non-heuristic searches, we chose Breath First Search, Depth First Graph Search, and Uniform Cost Search. For the heuristic searches, we chose A*H1, A*h_ignore-precondition and A* PG Level Sum. For the non-heuristic, as expected, it came apparent DFS failed at coming with an optimal shortest path, due to its revisiting paths.

All the other searches, except for DFGS, found a shortest path. When cost is not an issue, BSF gave the best performance in time and resulted in the reached one possible short path, but not the optimal path. UCS was able to find the optimal path but required more time; It had to compare all potential paths to the goal. The type of heuristic used in a search had mixed results compared to non heuristics searches. For instance, as expected, H1 performance was equivalent to UCS. On the other hand, IP heuristic outperformed all the non heuristic searches and H1. Introducing a more sophisticated heuristic PG Level Sum in A* search made finding a shortest path less demanding in the computer storage space, but still made finding the optimal shortest path more time consuming.
Unfortunately, what we found here, may not be true when applied to a different problem. If we assume, we will not be moving a large number of cargos, than one should use IP; on the other hand, if we are moving thousands of cargos, and we have space restriction than we should go with A* PG Level Sum.

Dalila Benachenhou
AIND Planning Problem

Appendix:

My 3 Air Cargo Problems:

```
"""
   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))
    """

def air_cargo_p1() -> AirCargoProblem:
    cargos = ['C1', 'C2']
    planes = ['P1', 'P2']
    airports = ['JFK', 'SFO']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, JFK)'),
           expr('At(P1, SFO)'),
           expr('At(P2, JFK)'),
           ]
    neg = [expr('At(C2, SFO)'),
           expr('At(C1, JFK)'),
           expr('At(P1, JFK)'),
           expr('At(P2, SFO)'),
           expr('In(C2, P1)'),
           expr('In(C2, P2)'),
           expr('In(C1, P1)'),
           expr('In(C1, P2)'),
           ]
    init = FluentState(pos, neg)
    goal = [expr('At(C1, JFK)'),
            expr('At(C2, SFO)'),
            ]
    return AirCargoProblem(cargos, planes, airports, init, goal)


def air_cargo_p2() -> AirCargoProblem:
    # TODO implement Problem 2 definition
    """
    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))
    """
    cargos = ['C1', 'C2', 'C3']
    planes = ['P1', 'P2', 'P3']
    airports = ['JFK', 'SFO', 'ATL']
    pos = [expr('At(C1, SFO)'),
           expr('At(C2, JFK)'),
```

```
                expr('At(C3, ATL)'),
                expr('At(P1, SFO)'),
                expr('At(P2, JFK)'),
                expr('At(P3, ATL)'),
                ]
        neg = [expr('At(C2, SFO)'),
                expr('At(C2, ATL)'),
                expr('In(C2, P1)'),
                expr('In(C2, P2)'),
                expr('In(C2, P3)'),
                expr('At(C1, JFK)'),
                expr('At(C1, ATL)'),
                expr('At(C3, JFK)'),
                expr('At(C3, SFO)'),
                expr('In(C1, P1)'),
                expr('In(C1, P2)'),
                expr('In(C1, P3)'),
                expr('In(C3, P1)'),
                expr('In(C3, P2)'),
                expr('In(C3, P3)'),
                expr('At(P1, JFK)'),
                expr('At(P1, ATL)'),
                expr('At(P2, SFO)'),
                expr('At(P2, ATL)'),
                expr('At(P3, JFK)'),
                expr('At(P3, SFO)'),
                ]
        init = FluentState(pos, neg)
        goal = [expr('At(C1, JFK)'),
                 expr('At(C2, SFO)'),
                 expr('At(C3, SFO)'),
                 ]
        return AirCargoProblem(cargos, planes, airports, init, goal)


    def air_cargo_p3() -> AirCargoProblem:
        # TODO implement Problem 3 definition
        """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))"""
        cargos = ['C1', 'C2', 'C3', 'C4']
        planes = ['P1', 'P2']
        airports = ['JFK', 'SFO','ATL','ORD']
        pos = [expr('At(C1, SFO)'),
                expr('At(C2, JFK)'),
                expr('At(C3, ATL)'),
                expr('At(C4, ORD)'),
                expr('At(P1, SFO)'),
                expr('At(P2, JFK)'),
                ]
        neg = [expr('At(C1, JFK)'),
                expr('At(C1, ATL)'),
                expr('At(C1, ORD)'),
                expr('At(C2, SFO)'),
                expr('At(C2, ATL)'),
                expr('At(C2, ORD)'),
                expr('At(C3, SFO)'),
                expr('At(C3, JFK)'),
                expr('At(C3, ORD)'),
                expr('At(C4, SFO)'),
```

```
                expr('At(C4, JFK)'),
                expr('At(C4, ATL)'),
                expr('In(C1, P1)'),
                expr('In(C1, P2)'),
                expr('In(C2, P1)'),
                expr('In(C2, P2)'),
                expr('In(C3, P1)'),
                expr('In(C3, P2)'),
                expr('In(C4, P1)'),
                expr('In(C4, P2)'),
                expr('At(P1, JFK)'),
                expr('At(P1, ATL)'),
                expr('At(P1, ORD)'),
                expr('At(P2, SFO)'),
                expr('At(P2, ATL)'),
                expr('At(P2, ORD)'),
                ]
        init = FluentState(pos, neg)
        #Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
        goal = [expr('At(C1, JFK)'),
                expr('At(C3, JFK)'),
                expr('At(C2, SFO)'),
                expr('At(C4, SFO)'),
                ]
        return AirCargoProblem(cargos, planes, airports, init, goal)
```

Dalila Benachenhou
AIND Planning Problem