AI Search &Planning: Heuristic Analysis

Krishna Kumar

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))

Implementation:

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

Optimal Plan

6 Steps

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

Uninformed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Breadth First Search	43	56	71	Yes (6)
Breadth First Tree Search	1458	1459	2494	Yes (6)
Depth First Search	12	13	19	No (12)
Depth Limited Search	101	271	183	No (50)
Uniform Cost Search	55	57	90	Yes (6)

Among all the searches that were able to successfully search for an optimal solution, BFS was the fastest with the least number of expansions, lowest number of goal tests and had the lowest running time.

The solution generated by DLS was fast, but the plan was not optimal.

Informed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Recursive-Breadth First Search -Null Heuristic	4229	4230	7009	Yes (6)
Greedy-Best First Search	7	9	12	Yes (6)
A* Search H_1	55	57	90	Yes (6)
A*Search_H_IgnoreP	41	43	67	Yes (6)
A* H_PGLS	11	13	6501	Yes (6)

All informed searches came up with an optimal solution. Greedy BFS was the fastest with the least expansions and goal tests, followed by A*-search with preconditions ignored. Recursive BFS had the worst performance with the largest number of expansions and goal tests.

Discussion:

Problem 1 was the easiest, with shortest running times, having only 2 planes, 2 cargoes, and 2 destinations.

For uninformed search, both BFS and UCS were able to find the optimal plans with almost the same performance, BFS was fast as it benefitted the most from having a smaller state space.

Greedy BFS outperformed all other heuristics in informed search as it had to expand the least in a small search space while being guided by a heuristic estimate.

Between all the A*-searches, ignore_preconditions had lowest running times, and levelsum had the least number of expansions and goal tests and largest runtimes, being slowed down by the construction of new planning graph instances.

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))

Implementation:

```
def air_cargo_p2() -> AirCargoProblem:
    # implement Problem 2 definition
    cargos = ["C1", "C2", "C3"]
    planes = ["P1", "P2", "P3"]
    airports = ["JFK", "SFO", "ATL"]
    #in initial state (cargos, airports, planes)
    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)")
           ]
    #not in initial state (cargos, airports, planes)
    neg = [expr("At(C1, JFK)"), expr("At(C1, ATL)"),
           expr("At(C2, SFO)"), expr("At(C2, ATL)"),
           expr("At(C3, JFK)"), expr("At(C3, SFO)"),
           expr("At(P1, JFK)"), expr("At(P1, ATL)"),
           expr("At(P2, SFO)"), expr("At(P2, ATL)"),
           expr("At(P3, JFK)"), expr("At(P3, SFO)"),
           expr("In(C1, P1)"), expr("In(C1, P2)"), expr("In(C1, P3)"),
           expr("In(C2, P1)"), expr("In(C2, P2)"), expr("In(C2, P3)"),
           expr("In(C3, P1)"), expr("In(C3, P2)"), expr("In(C3, P3)")
           ]
    #full state
    init = FluentState(pos, neg)
    #goal p2
    goal = [expr("At(C1, JFK)"),
            expr("At(C2, SFO)"),
            expr("At(C3, SFO)")
            ]
    return AirCargoProblem(cargos, planes, airports, init, goal)
```

Optimal Plan

P2: 9 Steps

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(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P3, SFO)

Uninformed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Breadth First Search	3343	4609	18321	Yes (9)
Breadth First Tree Search	-	-	-	NA
Depth First Search	624	625	5697	No (619)
Depth Limited Search	-	-	-	NA
Uniform Cost Search	4852	4854	29172	Yes(9)

Both Breadth First Search and Uniform Cost Search find the optimal plans, BFS is the fastest with the least number of expansions and goal tests. Depth First Search has the best time yet again, but it's solution is not optimal.

Informed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Recursive-Breadth First Search -Null Heuristic	-	-	-	-NA
Greedy-Best First Search	990	992	5863	No (15)
A* Search H_1	4852	4854	32470	Yes (9)
A*Search_H_IgnoreP	1450	1452	9717	Yes (9)
A* H_PGLS	86	88	**4184483**	Yes (9)

Timeout on Recursive-Breadth First Search. Greedy Best First had short runtimes but produced a sub-optimal solution. A*-search with ignore_preconditions was the best performing search, with A* levelsum having the least expansions and goal tests, but it also had unacceptably high runtimes.

Discussion:

Problem 2 has 3 planes, 3 cargoes and 3 airports, and therefore it is more difficult to solve without a heuristic, and is the main reason behind the poor performances of our uninformed searches, especially compared to A* with preconditions ignored, which expanded less and therefore found the solution faster.

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))

Implementation:

```
def air_cargo_p3() -> AirCargoProblem:
   # implement Problem 3 definition
   cargos = ["C1", "C2", "C3", "C4"]
   planes = ["P1", "P2"]
   airports = ["JFK", "SFO", "ATL", "ORD"]
   # in initial state (cargos, airports, planes)
   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)")
         ]
   # not in initial state (cargos, airports, planes)
   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, JFK)"), expr("At(C3, SFO)"), expr("At(C3, ORD)"),
         expr("At(C4, JFK)"), expr("At(C4, SFO)"), expr("At(C4, ATL)"),
         expr("At(P1, JFK)"), expr("At(P1, ATL)"), expr("At(P1, ORD)"),
         expr("At(P2, SFO)"), expr("At(P2, ATL)"), expr("At(P2, ORD)"),
         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)")
         ]
   #full state
   init = FluentState(pos, neg)
   #goal p3
   goal = [expr("At(C1, JFK)"),
          expr("At(C3, JFK)"),
          expr("At(C2, SFO)"),
          expr("At(C4, SFO)"),
          ]
   return AirCargoProblem(cargos, planes, airports, init, goal)
```

Optimal Plan

12 Steps

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

Uninformed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Breadth First Search	14663	18098	109092	Yes (12)
Breadth First Tree Search	-	-	-	-NA
Depth First Search	408	409	6136	No (596)
Depth Limited Search	-	-	-	-NA
Uniform Cost Search	18235	18237	149744	Yes(12)

Both Uniform Cost Search and Breadth First Search produce optimal solutions, with Breadth First Search being faster.

Informed Search: Results

Search	Expansions	Goal Tests	Average Time (milliseconds)	Optimality (Length)
Recursive-Breadth First Search -Null Heuristic	-	-	-	-NA
Greedy-Best First Search	5614	5616	46568	No (22)
A* Search H_1	18235	18237	149054	Yes (12)
A*Search_H_IgnoreP	5040	5042	43287	Yes (12)
A* H_PGLS	318	320	11284483	Yes (12)

Timeout on Recursive-BFS and A*-level sum. Best performing search was A*-search with ignore preconditions heuristic.

Discussion:

Problem 3 had 4 cargos, 2 airplanes and 4 airports. It was the most complex out of all the given problems, and therefore had the highest running times.

This is also the first problem where A* really shines, clearly beating all the other searches, with its fast performance.

Conclusions:

- **A* optimality:**
  - When A* finds a path, the path found will always have a cost that 's lower than the estimates of all other possible paths, as long as the heuristic uses estimates that optimistic and admissible.

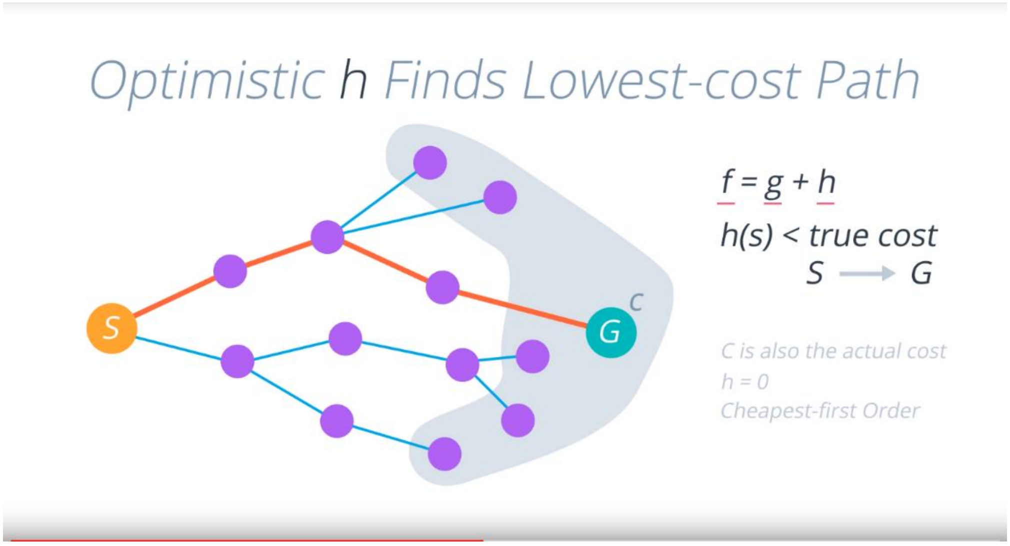

Figure 1: A optimality, Udacity: AI: Planning Search, Adapted from AIMA: Chapter 11 - Russel, Norvig*

- Depth First Search is fastest for proving the existence of a solution, but fails in finding an optimal solution.
- Greedy Best First Search finds a good solution in a smaller time, but doesn't guarantee optimality.
- Using a null heuristic in A* makes it perform like an uninformed search.
- Tree BFS, Recursive BFS and DL-DFS all failed to find a solution, with larger search spaces.
- Informed Search with good heuristics can severely cut down on the expansions, resulting in a better search, especially at larger search spaces.
- Relaxing Heuristics can be a good trade-off for increasing our search speed.
- Python Threading, and wasting time on creating new Planning Graph Instances every time, cripples the search performance even more.