Heuristic Analysis

Optimal Sequence of Actions:

**Problem 1**

Load (C1, P1, SFO)

Load (C2, P2, JFK)

Fly (P1, SFO, JFK)

Fly (P2, JFK, SFO)

Unload (C1, P1, JFK)

Unload (C2, P2, SFO)

**Problem 2:**

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)

**Problem 3:**

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, P2, SFO)

Unload (C3, P1, JFK)

Unload (C4, P2, SFO)

Analysis

**Uninformed search:**

For all three problems, **breadth\_first\_search** and **uniform\_cost\_search** are only uninformed search strategies that performed optimal plan. On the other hand, depth first search fastest and consumes less memory. For example, in problem 1, it just expands 48 nodes and execution time is just 0.007 sec but the plan is not optimal.

Expansions Goal Tests New Nodes

12 13 **48**

Plan length: 12 Time elapsed in seconds: **0.007828868001524825**

Fly (P1, SFO, JFK)

Fly (P2, JFK, SFO)

Load (C1, P2, SFO)

Fly (P2, SFO, JFK)

Fly (P1, JFK, SFO)

Unload (C1, P2, JFK)

Fly (P2, JFK, SFO)

Fly (P1, SFO, JFK)

Load (C2, P1, JFK)

Fly (P2, SFO, JFK)

Fly (P1, JFK, SFO)

Unload (C2, P1, SFO)

**Heuristics:**

For problem 2 and 3 we were not able to collect data for A\* because it was taking more than 10 minutes. Following are the data points that **astar\_search h\_ignore\_preconditions** is the fastest.

During the lessons we learned that **Breadth First** and **A\*** will always find the goal state no matter where it is in the tree but Depth first search will keep going down and will never get to the path.

As we know that A\* always expands the path with minimum of function (f=g+h) value. In the current project if we look at our functions then **h\_1** always return 1 and **h\_ignore\_preconditions** always returns 0 that’s why these 2 heuristics are faster but for **h\_pg\_levelsum** algorithmstarts evaluating states in all the levels and does not return with in 10 min time

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| astar\_search h\_1 | Problem 1 | 0.032 | 55 | 224 | 57 |
| Problem 2 | 11.29 | 4853 | 44041 | 4855 |
| Problem 3 | 48.38 | 18236 | 156908 | 18238 |
| astar\_search h\_ignore\_preconditions | Problem 1 | 0.030 | 41 | 170 | 43 |
| Problem 2 | 3.92 | 1445 | 13254 | 1447 |
| Problem 3 | 17.11 | 5040 | 44582 | 5042 |
| astar\_search h\_pg\_levelsum | Problem 1 | 1.46 | 58 | 234 | 60 |
| Problem 2 | - | - | - | - |
| Problem 3 | - | - | - | - |

Algorithm Analysis

**DFS:**

As we have learned during the course that DFS is not complete it will start going into depth and will never be able to reach the goal state. But DFS is always

**BFS:**

We also learned that in BFS we always expand the shortest path

Following table shows all the searches and their respective results

**Greedy Best First Search**:

As name implies this algorithm is goal focused and will start expanding path which is closest to the goal. But if there are obstacles along the way greedy best first may or may not find the optimal solution.

**A\*:**

A\*, always expands the path with minimum of function (f=g+h) value. In the current project if we look at our functions then **h\_1** always return 1 and **h\_ignore\_preconditions** always returns 0 that’s why these 2 heuristics are faster but for **h\_pg\_levelsum** algorithmstarts evaluating states in all the levels and does not return with in 10 min time.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Execution Time | Expansion | New Nodes | Goal Test |
| breadth\_first\_search | Problem 1 | 0.02 | 43 | 180 | 43 |
| Problem 2 | 8.02 | 3401 | 31049 | 4672 |
| Problem 3 | 39.55 | 14491 | 126091 | 17947 |
| breadth\_first\_tree\_search | Problem 1 | 0.8 | 1458 | 5960 | 1459 |
| Problem 2 | - | - | - | - |
| Problem 3 | - | - | - | - |
| depth\_first\_graph\_search | Problem 1 | 0.007 | 12 | 48 | 13 |
| Problem 2 | 1.3 | 350 | 3142 | 351 |
| Problem 3 | 68.26 | 9274 | 75518 | 9275 |
| depth\_limited\_search | Problem 1 | 0.07 | 101 | 414 | 271 |
| Problem 2 | - | - | - | - |
| Problem 3 | - | - | - | - |
| uniform\_cost\_search | Problem 1 | 0.034 | 55 | 224 | 57 |
| Problem 2 | 10.64 | 4853 | 44041 | 4855 |
| Problem 3 | 46.52 | 18236 | 156908 | 18238 |
| recursive\_best\_first\_search h\_1 | Problem 1 | 2.39 | 4229 | 17029 | 4230 |
| Problem 2 | - | - | - | - |
| Problem 3 | - | - | - | - |
| greedy\_best\_first\_graph\_search h\_1 | Problem 1 | 0.004 | 7 | 28 | 9 |
| Problem 2 | 2.28 | 970 | 8726 | 972 |
| Problem 3 | 14.88 | 5883 | 50858 | 5885 |
| astar\_search h\_1 | Problem 1 | 0.032 | 55 | 224 | 57 |
| Problem 2 | 11.29 | 4853 | 44041 | 4855 |
| Problem 3 | 48.38 | 18236 | 156908 | 18238 |
| astar\_search h\_ignore\_preconditions | Problem 1 | 0.030 | 41 | 170 | 43 |
| Problem 2 | 3.92 | 1445 | 13254 | 1447 |
| Problem 3 | 17.11 | 5040 | 44582 | 5042 |
| astar\_search h\_pg\_levelsum | Problem 1 | 1.46 | 58 | 234 | 60 |
| Problem 2 | - | - | - | - |
| Problem 3 | - | - | - | - |