**AIND PROJECT 3 Heuristic Analysis**

1. **Optimal plan for problems 1,2, and 3.**

Optimal plans in these problems are very easily obtained in just a little bit of thought processes.

* **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))

**Optimal plan**: Plan length 6

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**

**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)

**Optimal plan**: Plan length 9

Load(C1, P1, SFO) 🡪 Load(C2, P2, JFK) 🡪 Load(C3, P3, ATL) 🡪 Fly(P2, JFK, SFO) 🡪 Unload(C2, P2, SFO) 🡪 Fly(P1, SFO, JFK) 🡪 Unload(C1, P1, JFK) 🡪 Fly(P3, ATL, SFO) 🡪 Unload(C3, P3, SFO) 🡪 Fly(P1, SFO, JFK) 🡪 Unload(C1, P1, JFK)

* **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)

**Optimal plan**: Plan length 12

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)

1. **Search Result**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem 1**  Search method | **Expansions** | **Goal Tests** | **New nodes** | **Plan Length** | **Time(sec)** |
| **Breadth First Search (BFS)** | 43 | 56 | 180 | 6**(optimal)** | 0.1495976 |
| **Breadth First Tree Search** | 1458 | 1459 | 5960 | 6**(optimal)** | 4.7274216 |
| **Depth First Graph Search (DFS)** | 21 | 22 | 84 | 20 | 0.0816313 |
| **Depth Limited Search** | 101 | 271 | 414 | 50 | 0.5334899 |
| **Uniform Cost Search (UCS)** | 55 | 57 | 224 | 6**(optimal)** | 0.1620741 |
| **Recursive Best First Search with h\_1** | 4229 | 4230 | 17023 | 6**(optimal)** | 12.349847 |
| **Greedy Best First Graph Search with h\_1** | 7 | 9 | 28 | 6**(optimal)** | 0.0448996 |
| **A\* with h\_1** | 55 | 57 | 224 | 6**(optimal)** | 0.2425559 |
| **A\* with h\_ignore\_preconditions** | 41 | 43 | 170 | 6**(optimal)** | 0.1120591 |
| **A\* with h\_pg\_levelsum** | 11 | 13 | 50 | 6**(optimal)** | 4.9039768 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem 2**  Search method | **Expansions** | **Goal Tests** | **New nodes** | **Plan Length** | **Time(sec)** |
| **Breadth First Search** | 3343 | 4609 | 30509 | 9**(optimal)** | 39.410010 |
| **Breadth First Tree Search** | **Too Long** | | | | |
| **Depth First Graph Search** | 624 | 625 | 5602 | 619 | 13.316650 |
| **Depth Limited Search** | **Too Long** | | | | |
| **Uniform Cost Search** | 4853 | 4855 | 44041 | 9**(optimal)** | 62.376411 |
| **Recursive Best First Search with h\_1** | **Too Long** | | | | |
| **Greedy Best First Graph Search with h\_1** | 998 | 1000 | 8982 | 21 | 12.927869 |
| **A\* with h\_1** | 4853 | 4855 | 44041 | 9**(optimal)** | 62.637451 |
| **A\* with h\_ignore\_preconditions** | 1450 | 1452 | 13303 | 9**(optimal)** | 11.994419 |
| **A\* with h\_pg\_levelsum** | 86 | 88 | 841 | 9**(optimal)** | 815.23672 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem 3**  Search method | **Expansions** | **Goal Tests** | **New nodes** | **Plan Length** | **Time(sec)** |
| **Breadth First Search** | 14663 | 18098 | 129631 | 12**(optimal)** | 187.65269 |
| **Breadth First Tree Search** | **Too Long** | | | | |
| **Depth First Graph Search** | 408 | 409 | 3364 | 392 | 8.5114794 |
| **Depth Limited Search** | **Too Long** | | | | |
| **Uniform Cost Search** | 18223 | 18225 | 159618 | 12**(optimal)** | 238.68158 |
| **Recursive Best First Search with h\_1** | **Too long** | | | | |
| **Greedy Best First Graph Search with h\_1** | 5578 | 5580 | 49150 | 22 | 67.792445 |
| **A\* with h\_1** | 18223 | 18225 | 159618 | 12**(optimal)** | 239.03638 |
| **A\* with h\_ignore\_preconditions** | 5040 | 5042 | 44944 | 12**(optimal)** | 59.948589 |
| **A\* with h\_pg\_levelsum** | 316 | 318 | 2912 | 12**(optimal)** | 4244.8619 |

Other heuristics except A\* searches are all uninformed non-heuristic search. Also, h\_1 heuristic is not real heuristic that just return 1 for every case(Nevertheless, since it never overestimate the distance, a number of actions to reach the goal state, h\_1 is admissible). For problem 1, the simplest problem, all 10 search methods are able to find the goal in relatively short time. However, Breadth First Tree Search, Depth Limited Search, and Recursive Best First Search with h\_1 in problem 2 and problem 3 couldn’t finish the process in reasonable time. (I waited BFTS in P2 more than 16-hours!)

Among BFS, DFS, and UCS, BFS and UCS guarantee the optimal solution but take a long time. On the contrary, DFS usually gives us some inefficient result, but it is the fastest.

Heuristic searches, A\* with the "ignore preconditions" and "level-sum" also guarantee the optimal solution. Ignore preconditions heuristic performs much better than BFS and UCS in terms of time consuming and expanding nodes. The level-sum heuristic shows the least node expansion, but it takes a very long time because the planning graph is redrawn at every step.

1. **The best search method?**

If each action takes a long time to perform and resources are consumed, A\* with h\_ignore\_preconditions can be evaluated as the best search method since it shows fastest execution speed while ensuring the optimal solution. However, if a problem it to be solved very quickly or if you do not need to consider the time or resource consumption to perform an action, then DFS will find the path in the earliest time, so it can be considered as a priority.