Khayyam Zubair

AIND Project 3: Planning

Heuristic Analysis

**Air Cargo Planning Problems Heuristic Analysis**

This document compares the performance metrics of different search algorithms for each of the three assigned problems. Each problem was tested with all of the 10 algorithms provided in the run\_search.py script. However, results were only recorded if the algorithm executed in less than 10 minutes.

For each problem, the results are gathered into a table. The metrics of the optimal solution are highlighted in green. The steps in the optimal solution follow the table. When considering two algorithms that return the same solution length, the one with a lower execution time is preferred and considered optimal. A discussion follows the results.

**Problem 1:**

**Algorithm Metrics:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Expansions** | **Goal Tests** | **New Nodes** | **Length** | **Time** |
| **Breadth first search** | 43 | 56 | 180 | 6 | 0.042298 |
| **Breadth first tree search** | 1458 | 1459 | 5960 | 6 | 1.084749 |
| **Depth first graph search** | 21 | 22 | 84 | 20 | 0.020059 |
| **Depth limited search** | 101 | 271 | 414 | 50 | 0.102758 |
| **Uniform cost search** | 55 | 57 | 224 | 6 | 0.045066 |
| **Recursive best first search h 1** | 4229 | 4230 | 17023 | 6 | 2.978399 |
| **Greedy best first graph search h 1** | 7 | 9 | 28 | 6 | 0.006528 |
| **A Star search h 1** | 55 | 57 | 224 | 6 | 0.045077 |
| **A Star search (ignore preconditions)** | 41 | 43 | 170 | 6 | 0.048253 |
| **A Star search ( PG Level sum)** | 11 | 13 | 50 | 6 | 0.673085 |

**Optimal Search Algorithm:** Greedy best first

**Optimal Heuristic:** H1

**Optimal Non-Heuristic Algorithm:** Breadth First Search

**Quickest Algorithm** : Greedy best first

**Optimal Solution:**

Load (C1, P1, SFO)

Load (C2, P2, JFK)

Fly (P1, SFO, JFK)

Fly (P2, JFK, SFO)

Load (C1, P1, JFK)

Load (C2, P2, SFO)

**Problem 2:**

**Algorithm Metrics:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Expansions** | **Goal Tests** | **New Nodes** | **Length** | **Time** |
| **Breadth first search** | 3343 | 4609 | 30509 | 9 | 9.088975 |
| **Breadth first tree search** |  |  |  |  |  |
| **Depth first graph search** | 624 | 625 | 5602 | 619 | 3.393032 |
| **Depth limited search** |  |  |  |  |  |
| **Uniform cost search** | 4780 | 4782 | 414 | 50 | 0.102758 |
| **Recursive best first search h 1** |  |  |  |  |  |
| **Greedy best first graph search h 1** | 598 | 600 | 5382 | 17 | 1.358197 |
| **A Star search h 1** | 4780 | 4782 | 43381 | 9 | 13.12281 |
| **A Star search (ignore preconditions)** | 1450 | 1452 | 13303 | 9 | 10.60692 |
| **A Star search ( PG Level sum)** | 86 | 88 | 841 | 9 | 88.04122 |

**Optimal Search Algorithm:**  Breadth First Search

**Optimal Heuristic:** Ignore Preconditions w/ A\* Search

**Optimal Non-Heuristic Algorithm:** Breadth First Search

**Quickest Algorithm:** Uniform Cost Search

**Optimal Solution:**

Load (C1, P1, SFO)

Load (C2, P2, JFK)

Load (C3, P3, ATL)

Fly (P2, JFK, SFO)

Load (C2, P2, SFO)

Fly (P1, SFO, JFK)

Load (C1, P1, JFK)

Fly (P3, ATL, SFO)

Load (C3, P3, SFO)

**Problem 3:**

**Algorithm Metrics:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Algorithm** | **Expansions** | **Goal Tests** | **New Nodes** | **Length** | **Time** |
| **Breadth first search** | 14663 | 18098 | 129631 | 12 | 85.53610 |
| **Breadth first tree search** |  |  |  |  |  |
| **Depth first graph search** | 408 | 409 | 3364 | 392 | 1.993782 |
| **Depth limited search** |  |  |  |  |  |
| **Uniform cost search** | 17882 | 17884 | 156769 | 12 | 153.4364 |
| **Recursive best first search h 1** |  |  |  |  |  |
| **Greedy best first graph search h 1** | 4498 | 4500 | 39970 | 26 | 18.47378 |
| **A Star search h 1** | 17882 | 17884 | 156769 | 12 | 118.4967 |
| **A Star search (ignore preconditions)** | 5034 | 5036 | 44886 | 12 | 37.59552 |
| **A Star search ( PG Level sum)** |  |  |  |  |  |

**Optimal Solution:**

Load (C1, P1, SFO)

Load (C2, P2, JFK)

Fly (P1, SFO, ATL)

Load (C3, P1, ATL)

Fly (P2, JFK, ORD)

Load (C4, P2, ORD)

Fly (P2, ORD, SFO)

Fly (P1, ATL, JFK)

Load (C1, P1, JFK)

Load (C2, P2, SFO)

Load (C3, P1, JFK)

Load (C4, P2, SFO)

**Optimal Search Algorithm:**  A\* Search

**Optimal Heuristic:** Ignore Preconditions

**Optimal Non-Heuristic Algorithm:** Breadth First Search

**Quickest Algorithm:** Depth First Graph Search

The results indicate that the algorithms perform differently on each problem. However, the sets do have some consensus. They clearly indicate that Breadth First Search is the optimal algorithm when comparing non-heuristic algorithm. BFS was able to find the optimal solution for each of the problems. However, it was pricier than DFS and took much longer to run in each of the problems. It was defeated by heuristic based algorithms in Problems 1 and 3, which found the optimal solution in a quicker time.

The A\* search algorithm found the optimal solution for each of the problems. The Ignore Precondition heuristic yielded the quickest speeds in problems 2 and 3. However, it was marginally slower than the h1 heuristic in problem 1

These results align with expectations. As discussed in *Stuart Russell, Peter Norvig: Artificial Intelligence*, *New Jersey, 2003*. Depth first search always yielded in a lower search time than BFS. However, BFS always found the optimal path. This is because BFS finds the solution by looking at the root node and then going level by level looking at all the nodes. It therefore catches the earliest level at which a solution node appears, i.e. the shortest path. However, generally algorithms that used a heuristic performed much better. They did this by evaluating each state and assigning it a score based on how close it was to the goal state. This score is the heuristic, which is calculated by considering a relaxed version of the problem. This too was reflected in the results. Heuristic algorithms with A\* search found the shortest path each time. However, the second problem seems to be an exception. A\* search was slower than BFS