**Project\_3 Analysis**

This document contains analysis according to https://github.com/udacity/AIND-Planning#part-3-written-analysis. It contains examples of optimal solutions for problems 1-3 and comparison for different search techniques.

**1. Optimal plans for problems**

The optimal plans for problems 1,2,3 are:

**1.1 Problem 1**

Load(C2, P2, JFK)

Load(C1, P1, SFO)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Fly(P1, SFO, JFK)

Unload(C1, P1, JFK)

**1.2 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(C3, P3, SFO)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

**1.3 Problem 3**

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)

Unload(C4, P2, SFO)

Unload(C3, P1, JFK)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

**2. Comparison of different search techniques**

Comparison results are shown on the tables

**2.1 Problem 1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Search method** | **Expansions** | **Goal Tests** | **New Nodes** | **Plan Length** | **Time Elapsed** |
| 1. breadth\_first\_search | 43 | 56 | 180 | 6 | 0,09 |
| 2. breadth\_first\_tree\_search | 1458 | 1459 | 5960 | 6 | 2,79 |
| 3. depth\_first\_graph\_search | 12 | 13 | 48 | 12 | 0,02 |
| 4. depth\_limited\_search | 101 | 271 | 414 | 50 | 0,24 |
| 5. uniform\_cost\_search | 55 | 57 | 224 | 6 | 0,12 |
| 6. recursive\_best\_first\_search h\_1 | 4229 | 4230 | 17029 | 6 | 11,27 |
| 7. greedy\_best\_first\_graph\_search h\_1 | 7 | 9 | 28 | 6 | 0,01 |
| 8. astar\_search h\_1 | 55 | 57 | 224 | 6 | 0,09 |
| 9. astar\_search h\_ignore\_preconditions | 41 | 43 | 170 | 6 | 0,12 |
| 10. astar\_search h\_pg\_levelsum | 11 | 13 | 50 | 6 | 3,57 |

**2.2 Problem 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Search method** | **Expansions** | **Goal Tests** | **New Nodes** | **Plan Length** | **Time Elapsed** |
| 1. breadth\_first\_search | 3343 | 4609 | 30509 | 9 | 39,44 |
| 2. breadth\_first\_tree\_search | - | - | - | - | - |
| 3. depth\_first\_graph\_search | 1669 | 1670 | 14863 | 1444 | 37,73 |
| 4. depth\_limited\_search | - | - | - | - | - |
| 5. uniform\_cost\_search | 4853 | 4855 | 44041 | 9 | 128,39 |
| 6. recursive\_best\_first\_search h\_1 | - | - | - | - | - |
| 7. greedy\_best\_first\_graph\_search h\_1 | 998 | 1000 | 8982 | 17 | 21,02 |
| 8. astar\_search h\_1 | 4853 | 4855 | 44041 | 9 | 143,65 |
| 9. astar\_search h\_ignore\_preconditions | 1506 | 1508 | 13820 | 9 | 53,55 |
| 10. astar\_search h\_pg\_levelsum | 86 | 88 | 841 | 9 | 451,68 |

**2.3 Problem 3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Search method** | **Expansions** | **Goal Tests** | **New Nodes** | **Plan Length** | **Time Elapsed** |
| 1. breadth\_first\_search | 14663 | 18098 | 129631 | 12 | 332,46 |
| 2. breadth\_first\_tree\_search | - | - | - | - | - |
| 3. depth\_first\_graph\_search | 592 | 593 | 4927 | 571 | 9,19 |
| 4. depth\_limited\_search | - | - | - | - | - |
| 5. uniform\_cost\_search | 18223 | 18225 | 159618 | 12 | 1258,38 |
| 6. recursive\_best\_first\_search h\_1 | - | - | - | - | - |
| 7. greedy\_best\_first\_graph\_search h\_1 | 5578 | 5580 | 49150 | 22 | 356,17 |
| 8. astar\_search h\_1 | 18223 | 18225 | 159618 | 12 | 1222,06 |
| 9. astar\_search h\_ignore\_preconditions | 5118 | 5120 | 45650 | 12 | 297,94 |
| 10. astar\_search h\_pg\_levelsum | 408 | 410 | 3758 | 12 | 2673,19 |

**3. Conclusions**

The following conclusions can be done:

**3.1 Uniformed search**

Uniformed search works fine for easy problems (as Problem 1). DFS in fact can’t be used for bigger problems. Even if it manages to find an answer, it is usually too far from optimal.

BFS and UCS are acceptable for Problem 2,3, but in terms of expansions they are worse than searches with heuristics. They expectedly found an optimal solution

**3.2 Heuristic search**

h\_1 heuristic works fine only for small problems. In case of harder problems more complex heuristics should be used

Comparing *h\_ignore\_preconditions* and *h\_pg\_levelsum* we can say, that *h\_ignore\_preconditions* works much faster, but *h\_pg\_levelsum* expandes much less nodes.

It is expected result, cause, according to the book *Artificial Intelligence: A Modern Approach* by Norvig and Russell and common sense *h\_pg\_levelsum* is more precise but computationally harder.

Overall, both of them are suits for Problem 2,3 and I expect that *h\_pg\_levelsum* will work fine even for harder problems