**Implement a Planning Search - Heuristics Analysis**

The implementations of the following search algorithms were tested for the three Air Cargo problems provided in this project –

|  |
| --- |
| breadth\_first\_search |
| breadth\_first\_tree\_search |
| depth\_first\_graph\_search |
| depth\_limited\_search |
| uniform\_cost\_search |
| recursive\_best\_first\_search h\_1 |
| greedy\_best\_first\_graph\_search h\_1 |
| astar\_search h\_1 |
| astar\_search h\_ignore\_preconditions |
| astar\_search h\_pg\_levelsum |

Initially, for the three problems –

* the positive, negative and goal literals were created
* load\_actions and unload\_actions were programmatically created for all airport, planes and cargo combinations
* action and results functions were implemented thereafter

The run\_search was executed for all the searches listed above excluding the A\* search with heuristics. The results are provided as a part of the combined search results table below along with the heuristics results analysis combined together.

The implementation of the cargo planning graph functions as per the requirements followed.

The run\_search was executed to validate the A\* searches along with the implemented heuristics.

Search results are in the table below -

* NA Represents the runs that took longer than 10 mins and were aborted with the exception of astar\_search h\_pg\_levelsum for problem 3, which was let run for approx. 30 mins to completion
* Time is in seconds

As we can see from the table below for non-heuristics searches –

* Optimality of path and the time it takes to run are unrelated metrics
* We see DFGS is the fastest of all 3 problems, but returns a non-optimal path for Problem 2 and 3
* BFS is a more computationally expensive, but returns the optimal path

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Problem/Search** | **Plan Length** | **Expansion** | **Goal Tests** | **New Nodes** | **Time Elapsed** |
| ***Problem 1*** |  |  |  |  |  |
| breadth\_first\_search | 6 | 43 | 56 | 180 | 0.033 |
| breadth\_first\_tree\_search | 6 | 1458 | 1459 | 5960 | 0.938 |
| depth\_first\_graph\_search | 12 | 12 | 13 | 48 | 0.008 |
| depth\_limited\_search | 50 | 101 | 271 | 414 | 0.100 |
| uniform\_cost\_search | 6 | 55 | 57 | 224 | 0.041 |
| recursive\_best\_first\_search h\_1 | 6 | 4229 | 4230 | 17029 | 2.937 |
| greedy\_best\_first\_graph\_search h\_1 | 6 | 7 | 9 | 28 | 0.005 |
| astar\_search h\_1 | 6 | 55 | 57 | 224 | 0.043 |
| astar\_search h\_ignore\_preconditions | 6 | 41 | 43 | 170 | 0.063 |
| astar\_search h\_pg\_levelsum | 6 | 11 | 13 | 50 | 2.436 |
|  |  |  |  |  |  |
| ***Problem 2*** |  |  |  |  |  |
| breadth\_first\_search | 9 | 3343 | 4609 | 30509 | 14.564 |
| breadth\_first\_tree\_search | NA | NA | NA | NA | NA |
| depth\_first\_graph\_search | 575 | 582 | 583 | 5211 | 3.256 |
| depth\_limited\_search | NA | NA | NA | NA | NA |
| uniform\_cost\_search | 9 | 4853 | 4855 | 44041 | 45.014 |
| recursive\_best\_first\_search h\_1 | NA | NA | NA | NA | NA |
| greedy\_best\_first\_graph\_search h\_1 | 21 | 998 | 1000 | 8982 | 7.393 |
| astar\_search h\_1 | 9 | 4853 | 4855 | 44041 | 47.979 |
| astar\_search h\_ignore\_preconditions | 9 | 1506 | 1508 | 13820 | 25.044 |
| astar\_search h\_pg\_levelsum | 9 | 86 | 88 | 841 | 272.199 |
|  |  |  |  |  |  |
| ***Problem 3*** |  |  |  |  |  |
| breadth\_first\_search | 12 | 14663 | 18098 | 129631 | 118.926 |
| breadth\_first\_tree\_search | NA | NA | NA | NA | NA |
| depth\_first\_graph\_search | 596 | 627 | 628 | 5176 | 3.844 |
| depth\_limited\_search | NA | NA | NA | NA | NA |
| uniform\_cost\_search | 12 | 18221 | 18223 | 159599 | 451.470 |
| recursive\_best\_first\_search h\_1 | NA | NA | NA | NA | NA |
| greedy\_best\_first\_graph\_search h\_1 | 22 | 5560 | 5562 | 49015 | 129.421 |
| astar\_search h\_1 | 12 | 18221 | 18223 | 159599 | 471.653 |
| astar\_search h\_ignore\_preconditions | 12 | 5118 | 5120 | 45650 | 164.357 |
| astar\_search h\_pg\_levelsum | 12 | 414 | 416 | 3818 | 1867.443 |

As we can see from the table above for heuristics searches –

* astar\_search with h\_ignore\_preconditions is the fastest of all 3 problems and does returns an optimal path
* All the three A\* runs return the optimal path, but the number of nodes created and expanded are very different. The planning graph implementation with the h\_pg\_levelsum touched on fewer nodes based on the graph design of alternating S and A levels, which helped narrow the search (using applicable actions and resultant S literals) v/s a more brute force approach with other non-heuristics searches

*Reasoning behind BFGS v/s DFGS variations in path optimality –*

Breadth first search is complete as long as the branching factor is finite. This means that the algorithm will always return a solution if a solution exists. In a path planning sense, breadth first search will always find the path from the starting position to the goal as long as there is an actual path that can be found. Breadth first search is only optimal if the path cost is the same for each direction. In this case, the path cost would be the direction and optimal means the shortest distance path from the start to the goal. It was the case in the aircargo problem.

Depth first search usually requires a considerably less amount of memory that BFS. This is mainly because DFS does not always expand out every single node at each depth. However, there are some drawbacks to depth first search the BFS will not suffer from. Depth first search is not optimal and it is not complete. The reason it is not optimal is that it returns a path as soon as the goal is reached, however, this path is not always the shortest path but a path generated by the result of going down a long branch. This was also the case in aircargo problem.

In conclusion, this was a fun project to learn how the search tree and graph implementation work in much more detail than before. Also, provided a very good view into the planning graph. In future will look forward to implementing GRAPHPLAN.