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

The paths for the three runs in discussion for BFS, DFGS and A* searches with heuristics are as follows –

Problem/Search	Path
Problem 1	
BFS	Load(C2, P2, JFK) Load(C1, P1, SFO) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)
DFGS	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)

A* h_ignore_preconditions	Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)
A* h_pg_levelsum	Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C1, P1, JFK) Unload(C2, P2, SFO)
Problem 2	
BFS	Load(C2, P2, JFK) Load(C1, P1, SFO) 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)
DFGS	too long 575 elements - non optimal path
A* h_ignore_preconditions	Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Load(C1, P1, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)

A* h_pg_levelsum	Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Unload(C2, P2, SFO) Unload(C1, P1, JFK)
Problem 3	
BFS	Load(C2, P2, JFK) Load(C1, P1, SFO) 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)
DFGS	too long 596 elements - nonoptimal path
A* h_ignore_preconditions	Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Unload(C4, P2, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)

A* h_pg_levelsum	Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C4, P2, SFO) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)

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.