Heuristics Analysis

In this project, we have developed a planning search agent for an air cargo transport system to solve the deterministic logistics planning problems. In order to check the performance for both heuristic and non-heuristic search methods, several tests were conducted based on the given problems.

According to the given problems, we can find the optimal plans as below:
Problem 1:

Load(C1, P1) – FLY(P1, SFO, JFK) – Unload(C1, P1)
Load(C2, P2) – FLY(P2, JFK, SFO) – Unload(C2, P2)

Problem 2:

Load(C1, P1) – FLY(P1, SFO, JFK) – Unload(C1, P1)
Load(C2, P2) – FLY(P2, JFK, SFO) – Unload(C2, P2)
Load(C3, P3) – FLY(P3, ATL, SFO) – Unload(C3, P3)

Problem 3:

Load(C1, P1) – FLY(P1, SFO, ATL) – Load(C3, P1) – FLY(P1, ATL, JFK) – Unload(C1, P1) – Unload(C3, P1)
Load(C2, P2) – FLY(P2, JFK, ORD) – Load(C4, P2) – FLY(P1, ORD, SFO) – Unload(C2, P2) – Unload(C4, P2)

Priya: Awesome: Good work! You have identified the optimal no. of steps for each of the 3 problems.

From the optimal plans, we can find for problem 1 we need 6 actions to achieve goal, for problem 2 it's 9 actions and for problem 3 it's 12 actions.

The results of tests are given below:

Problem 1									
	BFS	BFS tree	DFS graph	Depth limited	Uniform cost	Recursive best first	Greedy best first graph	A* h ignore preconditions	A* h pg levelsum
Expansions	43	1458	21	101	55	4229	7	41	11
Goal Tests	56	1459	22	271	57	4230	9	43	13
New Nodes	180	5960	84	414	225	17023	28	170	50
Plan length	6	6	20	50	6	6	6	6	6
Time elapsed	0.0287	0.7585	0.0117	0.0720	0.0302	2.3496	0.0045	0.0309	0.8546

Problem 2							
	BFS	BFS tree	DFS graph	Uniform cost	Greedy best first graph	A* h ignore preconditions	A* h pg levelsum
Expansions	3343	3343	624	4852	990	1450	86
Goal Tests	4609	4609	625	4854	992	1452	88
New Nodes	30509	30509	5602	44030	8910	13303	841
Plan length	9	9	619	9	21	9	9
Time elapsed	11.66	11.67	3.01	9.99	2.04	3.46	133.09

Priya: Very neat comparison of all the different search results on problems 1, 2 and 3.

Problem 3						
	BFS	DFS graph	Uniform cost	Greedy best first graph	A* h ignore preconditions	A* h pg levelsum
Expansions	14663	408	18235	5614	5040	325
Goal Tests	18098	409	18237	4516	5042	327
New Nodes	129631	3364	159716	49429	44944	3002
Plan length	12	392	12	22	12	12
Time elapsed	81.93	1.45	43.89	13.30	13.54	740.35

Priya: Well done! All algorithms have been implemented properly

As we can find, for all three problems, most of the search methods have the plan length equal to the number of optimal actions, but they don't have the shortest time. In the opposite, we can find DFS graph search method's plan length is much bigger than the optimal number but shows us a very good computation time. The reason of this is DFS graph method has the lowest number of expansions, goal tests and new nodes compared with BFS, uniform cost search and greedy best first graph search method.

For the DFS graph search method, it is an instance of graph search algorithm and it uses last in first out queue(LIFOQueue). The mostly visited/generated node will be used for expansion. It will travel directly to the deepest level of the graph and check if the goal is satisfied. If not, the method will travel in another routine from the last parent node. If the goal is reached quickly, there will be less new nodes generated. DFS method is known for its nonoptimal solution (Stuart J. Russell, 2015). The search method will explore the entire subgraph from the current node even the neighbor node is the goal. The advantage of DFS over BFS is its space complexity. In DFS, it only stores a single path from the root to a leaf node, along with the remaining unexpanded sibling nodes for each node on the path (Stuart J. Russell, 2015).

Priya: Suggestion: The link below has an interesting comparison of the BFS and DFS methods on when to chose
one vs the other : http://stackoverflow.com/questions/3332947/when-is-it-practical-to-use-dfs-vs-bfs

In the BFS method, the search agent will check all nodes in the frontier set based on a first in first out queue(FIFOQueue). if goal not reached, it will expand the current nodes, add them to frontier and then go to next node in the queue to do the check. In this way, it will generate enormous new nodes. The time complexity for BFS is O(b**d), and time complexity for DFS is O(b**m). b is the branching factor, d is the depth of the shallowest and m is the maximum depth of the search tree (Stuart J. Russell, 2015). We can find for this the cargo planning problem here, DFS will perform much better.

As for the uniform search and greedy best first graph search, they have the identical algorithm which is the best first graph search method. The best first graph search method is similar to BFS method, except it uses a priority queue instead of FIFOQueue. It will add the node to frontier based on a f score, so the node with the lowest score will have a priority to be visited first in the frontier. As for uniform search and greedy best first graph search, the difference between them is they have different evaluation method for f scores. For the uniform search, it uses the node's path cost and best first graph search uses a heuristic function. As default in the run_search file, it uses a h_1 function, which returns a constant number one. In theory, the best first graph search should give us a good performance but here is not. The reason is when we evaluate the f score, both methods here are not optimized. There is not much difference for path cost between different nodes and constant number is also powerless. I will revisit greedy first graph search later in this report to talk about.

Now let's look at the performance of A star search with two different heuristics functions. A star search is another instance of best first graph search. For the evaluation function, it combines the cost to reach the node and cost to get from the node to the goal. Function h_pg_levelsum is a function to sum the level costs of the individual goals. Function h_ignore_preconditions estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal by ignoring the preconditions required for an action to be executed. Ignoring preconditions is not accurate as level sum(Stuart J. Russell, 2015). But when we look at the code, function of ignore preconditions takes two for loops to count the result. In the function of level sum, it takes three for loops to count the result. With a larger problem, the level sum function could be much slower than function of ignore preconditions. Though A* search with h_pg_levelsum gives us much less expansions and new nodes, it actually uses much more time than A* with h_ignore_preconditions.

Priya: This is an important point. Better heuristics like level sum
may be more efficient in reducing the no. of expansions but costly in terms of taking more time.

Now let's talk about greedy best first graph search and A* search. We know the difference here is the heuristic function. What if we use h_ignore_preconditions in the greedy best first graph search. The test result for problem 3 is shown here:

Problem 3					
	DFS graph	Uniform cost	Greedy best first graph with h_1	Greedy best first graph with h ignore preconditions	A* h ignore preconditions
Expansions	408	18235	5614	49	5040
Goal Tests	409	18237	4516	51	5042
New Nodes	3364	159716	49429	416	44944
Plan length	392	12	22	23	12
Time elapsed	1.45	43.89	13.30	0.1549	13.54

Greedy best first graph with ignore preconditions only has 49 expansions, 416 new nodes, and only uses 0.1549 seconds to finish the problem, which is much faster than A* search. The only problem here is it didn't give us an optimized solution. In A* search, because the path cost is added, when we add new node to the frontier set, the priority of the node will be balanced with the path cost since the number of actions is small. The node for next expansion cannot be guaranteed as the best choice, and then the search routine will not follow a best optimized way, finally result in a larger number of expansions and new nodes.

Overall, for all the tests here, the A* search with ignore preconditions has the best performance. Within all the methods that give us the most optimized plan length, this method used the shortest time. Greedy best first graph with ignore preconditions has the shortest time over all methods, but it didn't give an optimized solution, so it fails. The question now is if we can reduce the time for A* search, that leaves me a further investigation on it.

Priya: I agree with your conclusion