Search strategies and results

I chose following uninformed search strategies:

- Breadth first search (Graph)
- Depth first search (Graph)
- Breadth first search (Tree)
- Depth limited tree search with depth limit = 50 (i.e. the default in search.py)
- Depth limited tree search with depth limit = 12 (changed search.py).

And I ran A* with the following two heurisitcs:

- *h_ignore_preconditions*
- *h_pg_levelsum*

The results are as follows:

- Problem 1

Strategy	Expansions	Goal Tests	New Nodes	Time (secs)	Plan length
Breadth First Graph Search	43	56	180	0.034	6
Depth First Graph Search	21	22	84	0.014	20
Breadth First Tree Search	1458	1459	5960	0.93	6
Depth Limited Tree Search (d=50)	101	271	414	0.085	50
Depth Limited Tree Search (d=12)	63	233	262	0.069	12
A* / h_ignore_preconditions	41	43	170	0.044	6
A* / h_pg_levelsum	11	13	50	0.71	6

- Problem 2

Strategy	Expansions	Goal Tests	New Nodes	Time (secs)	Plan length
Breadth First Graph Search	3343	4609	30509	7.835	9
Depth First Graph Search)	624	625	5602	3.311	619
Breadth First Tree Search	Timeout	Timeout	Timeout	Timeout	Timeout
Depth Limited Tree Search (d=50)	222719	2053741	2054199	848.85	50
Depth Limited Tree Search (d=12)	222681	2053703	205377	882.4	12
A* / h_ignore_preconditions	1450	1452	13303	4.12	9
A* / h_pg_levelsum	86	88	841	62.52	9

- Problem 3

Strategy	Expansions	Goal Tests	New Nodes	Time (secs)	Plan length
Breadth First Search (Graph)	14663	18098	129631	40.521	12
Depth First Search (Graph)	408	409	3364	1.77	392
Breadth First Search (Tree)	Timeout	Timeout	Timeout	Timeout	Timeout

Strategy	Expansions	Goal Tests	New Nodes	Time (secs)	Plan length
Depth Limited Search (d=50)	Timeout	Timeout	Timeout	Timeout	Timeout
Depth Limited Search (d=12)	Timeout	Timeout	Timeout	Timeout	Timeout
A* / h_ignore_preconditions	5040	5042	44944	16.4	12
A* / h_pg_levelsum	325	327	3002	322.1	12

Optimal plans

We know that Breadth first search is going to give us an optimal plan in this case.

- Problem 1:

Load(C1, P1, SFO), Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),

- Problem 2:

Load(C1, P1, SFO), Load(C2, P2, JFK), 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)

- Problem 3:

Load(C1, P1, SFO), Load(C2, P2, JFK), 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)

Analysis

If we assume the same cost for each action (which may not be realistic in the real world air cargo domain), then we know that a Breadth First Search is going to give us an optimal path. We do see this in the solutions for all three problems where Breadth First Search discovers the optimal path.

The search script gives us the total number of *Nodes* that were created during the program run but it does not give us the maximum number of *Nodes* that were alive at the same time - i.e. an indication of the working memory size required by the program. The total number of *Nodes* is still a good proxy for the memory usage by the program. Looking at this number for all the different strategies, we see results that match theoretical expectations.

First, we know that of the two generic categories of searching - *graph search* and *tree search*, in general, because of redundant paths in the state space, we expect a *graph search* to have better (i.e. lesser) memory usage than a Breadth First Tree search. We see this in our results where Breadth First Tree Search times out before discovering a goal node for Problesm 2 and 3 (and thus by extension has a really big memory footprint). For our problems, Depth First Graph search has better memory usage than the Breadth First Graph Search. This however is not guaranteed to always be so and it depends on the topology of the state space, the order in which children node of a node are selected for expansion and where in the tree a goal node is located . Further, in our problems, the Depth First Graph search fails to find the optimal solution and in fact finds severely non-optimal solutions.

Now, there is an interesting memory characteristic when using a Depth First Tree Search. A DFS tree search (with either infinite depth or a finite depth of 50 or 12 as we have) has the best *working memory* usage i.e. the actual live memory usage is better than any *graph search* or a Breadth First Tree Search. Since our script does not capture the *working memory* usage, we cannot highlight this observation. What we do see from our tables above in fact is that a depth limited search may still end up expanding many more nodes than a breadth first search before it stumbles upon a goal node and the goal node need not even be optimal. (If we are lucky, a depth limited search might indeed discover an optimal goal node)

To show the non-optimal nature of a depth limited search, I tweaked the search.py script to be able to run a depth limited search with depth=12. I chose this number since after running the Breadth first search on all three problems we know that depth limit of 12 is deep enough to get to optimal goal node for all three problems. We see that even in this case a depth limited search either times out or expands many more nodes and still discovers a non-optimal solution.

Let us now look at the A* perfomance with the two heuristics we have.

We know that with an admissible heuristic, A* is guaranteed to be optimal. Now, of the two heuristics that we have, we know from the Russell and Norvig text book that *h_pg_level_sum* is not admissible, and *h_ignore_preconditions* is also inadmissible because of the way we have implemented it - An admissible implementation of *h_ignore_preconditions* would, amongst other things, be NP hard since it would involve solving the set cover problem. So this heuristic is usually implemented via a greedy approximation like we have done but that makes it inadmissible.

Surprisingly, even with inadmissible heuristics, A* discovers optimal solutions for all three problems, and further, it also has the best memory usage! This shows that even approxmiate heuristics make a huge difference over uninformed search