Heuristic Analysis for Planning Search

Executive Summary:

There are three air cargo planning problems given with the same action schema but different pre-conditions and goals. The main objective of this write-up is to compare the search results of the given three problems using various non-heuristic and heuristic search algorithms. The results are compared and contrasted based on various metrics including memory efficiency and search speed. The best performing search algorithm finding the optimal path for each of the three problems is documented.

Introduction:

The three air cargo problems are as defined below:

Problem 1 initial state and goal:	Problem 2 initial state and goal:	Problem 3 initial state and goal:
Init(At(C1, SFO) ∧At(C2, JFK) ∧At(P1, SFO) ∧At(P2, JFK) ∧Cargo(C1) ∧Cargo(C2) ∧Plane(P1) ∧Plane(P2) ∧Airport(JFK) ∧ Airport(SFO)) **Goal**(At(C1, JFK) ∧At(C2, SFO))	**Init**(At(C1, SFO) ∧At(C2, JFK) ∧ At(C3, ATL) ∧At(P1, SFO) ∧At(P2, JFK) ∧At(P3, ATL) ∧Cargo(C1) ∧Cargo(C2) ∧ Cargo(C3) ∧Plane(P1) ∧Plane(P2) ∧ Plane(P3) ∧Airport(JFK) ∧ Airport(SFO) ∧Airport(ATL)) **Goal**(At(C1, JFK) ∧At(C2, SFO) ∧ At(C3, SFO))	**Init**(At(C1, SFO) ∧At(C2, JFK) ∧ At(C3, ATL) ∧At(C4, ORD) ∧At(P1, SFO) ∧At(P2, JFK) ∧Cargo(C1) ∧Cargo(C2) ∧ Cargo(C3) ∧Cargo(C4) ∧Plane(P1) ∧Plane(P2) ∧Airport(JFK) ∧ Airport(SFO) ∧Airport(ATL) ∧ Airport(ORD)) **Goal**(At(C1, JFK) ∧At(C3, JFK) ∧ At(C2, SFO) ∧At(C4, SFO))

All the above three problems were run using the following non-heuristic and heuristic search algorithms:

Non-heuristic search algorithms tried:

1. *breadth_first_search*
2. *breadth_first_tree_search*
3. *depth_first_graph_search*
4. *depth_limited_search*
5. *uniform_cost_search*

Heuristic search algorithms tried:

1. *recursive_best_first_search_h_1*
2. *greedy_best_first_graph_search_h_1*
3. *astar_search, h_1*
4. *astar_search, 'h_ignore_preconditions'*
5. *astar_search, 'h_pg_levelsum'*

Note: Heuristics wise, h_1 is not a real heuristic (as given in the code descriptions) as it always assigns a constant h value of 1. However it kind of acts like a base case for the search algorithms tried and hence are included.

The runs were manually terminated only when the search time exceeded 60 minutes (3600 seconds). I just wanted to see how long each one would actually take within the above limit (although Udacity accepts a limit of within 10 minutes). The results of the various searches are as given in the below section.

Search Results and Performance Metrics:

The most important metrics in the performance analysis of search algorithms are: time and space. Breadth First Search (BFS) considers neighbouring nodes first before moving on to the next depth. BFS algorithm is complete and hence will always find the optimal solution (if solution is within finite depth). However, it has issues with time and space based on the problem definition.

This is unlike Depth First Search (DFS) algorithm which searches deeper first before back-tracking and searching the shallower ones, if required. Hence, DFS searches may lead to non-optimal solutions, although, most of the times faster than BFS.

Uniform Cost Search (UCS) algorithm always find the cheapest path (if solution within finite cost). Due to this, UCS finds the optimal path but takes longer than BFS due to continued node expansions in an effort to find the least cost path even after finding a path to the goal state.

Best_first_search algorithms are designed to expand on the most promising nodes first. Recursive, greedy and A* are different variations developed from best-first search algorithm. Recursive uses best-first search within linear space but encounters unnecessary node regeneration. Greedy best-first search uses the heuristic to predict the distance to the goal. On the other hand, A* uses distance estimates not just to the goal but also from the starting node. It also takes into account cost/distance considered previously in the search. UCS and most best-first search algorithms are extensions from Dijkstra's algorithm (formulated by Edsger Dijkstra in 1959). A* and variations of it are currently the most popular algorithms for planning and pathfinding. A* with heuristic makes it an informed search algorithm and is guaranteed to find the solution if one exists (as long as the heuristic is admissible). The complexity however is depended on the heuristic chosen for the search. For this report, the main heuristics considered are: (1) ignore_preconditions – which is a relaxed form of the original problem, (2) pg_levelsum – which uses a planning graph representation and finds sum of actions to individual goals at the current state.

Below are the details of the various non-heuristic and heuristic search results for the three air cargo problems. Out of the collected metrics, three main metrics were considered as it affected the results, namely, node expansions (signifying memory/space usage), time taken (execution speed) and optimality (computed based on if the algorithm finds the shortest path or not).

Results for Problem 1:

Sl. No.	Search Algorithm	Problem 1					
		Expansions	Goal Tests	New Nodes	Plan Length	Time Taken (s)	Optimal ?
1	breadth_first_search	43	56	180	6	0.501879641	yes
2	breadth_first_tree_search	1458	1459	5960	6	12.08005637	yes
3	depth_first_graph_search	21	22	84	20	0.269772726	no
4	depth_limited_search	101	271	414	50	1.341845094	no
5	uniform_cost_search	55	57	224	6	0.647226967	yes
6	recursive_best_first_search_h_1	4229	4230	17023	6	32.74724291	yes
7	greedy_best_first_graph_search_h_1	**7**	**9**	**28**	**6**	**0.092918505**	**yes**
8	astar_search, h_1	55	57	224	6	0.623764006	yes
9	astar_search, 'h_ignore_preconditions'	41	43	170	6	0.632447547	yes
10	astar_search, 'h_pg_levelsum'	11	13	50	6	10.78735736	yes

Least Node Expansions Optimal

Least Execution Time Optimal

Problem 1 is a simple problem and hence all the search algorithms (except depth_first_graph_search and depth_limited_search) finds the optimal solution of 6 plan lengths. Considering only the non-heuristic search algorithms, breadth_first_search performs the best over DFS and UCS with 43 node expansions in 0.5 seconds.

Considering all the algorithms given, out of the optimal searches, greedy_best_first_search performs the best with only 7 node expansions and in 0.0929 seconds. Greedy_best_first search considers only the distance to the goal and for less complex problems like Problem 1, this search produces the most direct path to the goal state than all the other search algorithms. Hence, for the first problem, both space and time wise, greedy_best_first_search performs the best.

Optimal Plan for Problem 1:

Search algorithm: Greedy_best_first_search

Action Sequence (length = 6):

Load(C1, P1, SFO)

Load(C2, P2, JFK)

Fly(P1, SFO, JFK)

Fly(P2, JFK, SFO)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

Results for Problem 2:

Sl. No.	Search Algorithm	Problem 2					
		Expansions	Goal Tests	New Nodes	Plan Length	Time Taken (s)	Optimal ?
1	**breadth_first_search**	3343	4609	30509	9	77.81009	yes
2	**breadth_first_tree_search**	-	-	-	-	-	
3	**depth_first_graph_search**	624	625	5602	619	38.67581	no
4	**depth_limited_search**	222719	2053741	2054119	50	3387.7392	no
5	**uniform_cost_search**	4853	4855	44041	9	106.68917	yes
6	**recursive_best_first_search_h_1**	-	-	-	-	-	
7	**greedy_best_first_graph_search_h_1**	998	1000	8982	17	30.493397	no
8	**astar_search, h_1**	4853	4855	44041	9	103.37642	yes
9	**astar_search, 'h_ignore_preconditions'**	**1450**	**1452**	**13303**	**9**	**45.204954**	**yes**
10	**astar_search, 'h_pg_levelsum'**	86	88	841	9	801.51528	yes

Least Node Expansions Optimal

Least Execution Time Optimal

Problem 2 has a bit more complexity than Problem 1. The search results show that depth_first_graph_search and depth_limited_search did not find the optimal solution like in Problem 1. In addition to this, unlike for Problem 1, greedy_best_first_graph_search_h_1 also did not find the optimal solution although it has the lowest execution time. Breadth_first_tree_search and recursive best_first_search_h1 took longer than 1 hour and hence were terminated.

Considering only the non-heuristic search algorithms, breadth_first_search performs the best over UCS with 3343 node expansions in ~77 seconds. As can be seen from the results, UCS keeps expanding to further nodes in search of probable cheaper paths and hence takes longer than BFS. With heuristic as a constant, A* with h_1 performs similar to UCS.

Considering both non-heuristic and heuristic algorithms, out of the optimal solutions, A*_search with 'ignore_preconditions' heuristic performs the best with 45.205 seconds. A* search with 'pg_levelsum' heuristic also finds the optimal solution with the lowest node expansion of only 86 nodes but takes 801.515 seconds to finish execution (almost 9X times more time than A* with ignore_preconditions heuristic). Therefore, I would choose A* with 'ignore-preconditions' heuristic as the selected algorithm for Problem 2.

Optimal Plan for Problem 2:

Search algorithm: A* with 'ignore_preconditions' heuristic

Action Sequence (length = 9):

Load(C1, P1, SFO)

Fly(P1, SFO, JFK)

Unload(C1, P1, JFK)

Load(C2, P2, JFK)

Fly(P2, JFK, SFO)

Unload(C2, P2, SFO)

Load(C3, P3, ATL)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

Results for Problem 3:

Sl. No.	Search Algorithm	Problem 3					
		Expansions	Goal Tests	New Nodes	Plan Length	Time Taken	Optimal ?
1	**breadth_first_search**	14663	18098	129631	12	264.88865	yes
2	**breadth_first_tree_search**	-	-	-	-	-	
3	**depth_first_graph_search**	408	409	3364	392	21.09817	no
4	**depth_limited_search**	-	-	-	-	-	
5	**uniform_cost_search**	18151	18153	159038	12	330.80092	yes
6	**recursive_best_first_search_h_1**	-	-	-	-	-	
7	**greedy_best_first_graph_search_h_1**	5398	5400	47665	26	126.34018	no
8	**astar_search, h_1**	18151	18153	159038	12	313.93659	yes
9	**astar_search, 'h_ignore_preconditions'**	**5038**	**5040**	**44926**	**12**	**131.53534**	**yes**
10	**astar_search, 'h_pg_levelsum'**	314	316	2894	12	3002.5856	yes

Least Node Expansions Optimal

Least Execution Time Optimal

Problem 3 is the most complex of the given three problems. For this problem, breadth_first_tree_search, recursive best_first_search_h1 and depth_limited_search took longer than 1 hour and hence were terminated. BFS performs better than UCS among non-heuristic search algorithms which found the optimal solution of 12 nodes. But it would be interesting to note that with increase in problem complexity (prob1→ prob2 → prob3), the node expansions for A* with 'ignore_preconditions' heuristic does not increase in the same proportion as node expansions required for BFS/UCS (Figure 1).

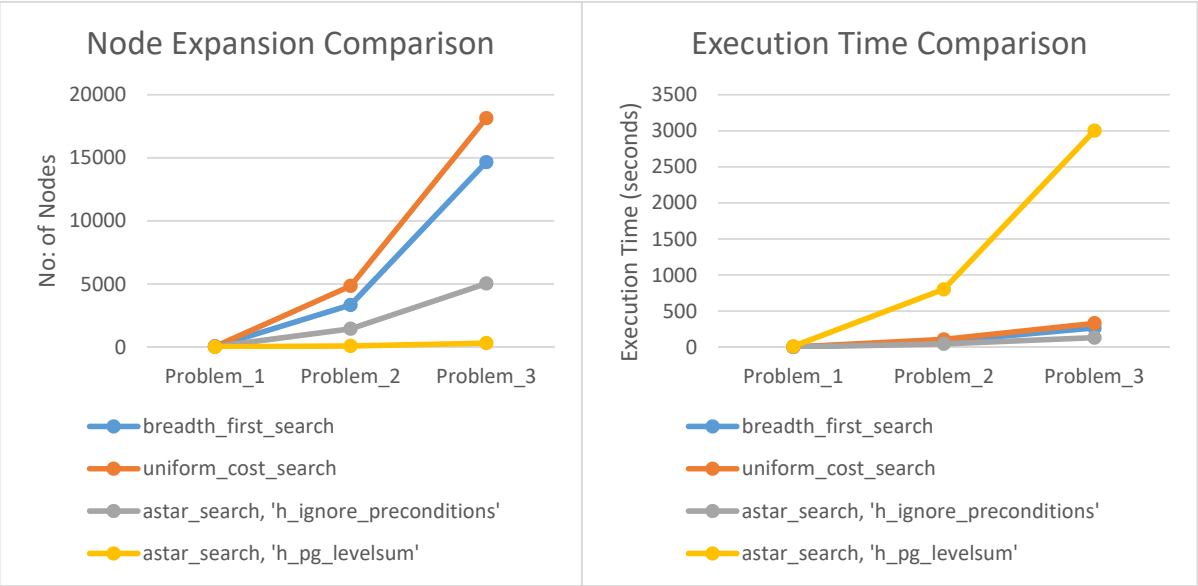

Figure 1: Node Expansion and Execution Time comparison for search algorithms

Out of all the searches which finds optimal solution, A* search with 'ignore_preconditions' heuristic finds the solution in the lowest execution time of 131.535 seconds but expands to 5038 nodes. But, from a different angle, 5038 node expansions is way better than 14663 node expansion for BFS (best optimal choice among non-heuristic searches). Again, A* with 'pg_levelsum' heuristic expands to only 314 nodes but takes around 50 minutes to complete the search (compared to ~2 minutes for A* with 'ignore_preconditions')! Hence, I would again choose A* with 'ignore_preconditions' as the search algorithm for problem 3.

This also brings up the versatility of A* algorithms with heuristics as it could be designed to run for memory efficiency (E.g., here A* with 'pg_levelsum'), speed (E.g., here A* with 'ignore_preconditions') or a balance between both, according to requirements!

Optimal Plan for Problem 3:

Search algorithm: A* with 'ignore_preconditions' heuristic

Action Sequence (length = 12):

Load(C1, P1, SFO)

Fly(P1, SFO, ATL)

Load(C3, P1, ATL)

Fly(P1, ATL, JFK)

Unload(C1, P1, JFK)

Load(C2, P2, JFK)

Fly(P2, JFK, ORD)

Load(C4, P2, ORD)

Fly(P2, ORD, SFO)

Unload(C2, P2, SFO)

Unload(C3, P1, JFK)

Unload(C4, P2, SFO)

Conclusions:

Of the non-heuristic algorithms, BFS and UCS always finds optimal solution. For the heuristic based algorithms, A* with 'h_1', 'ignore_preconditions' and 'pg_levelsum' always finds optimal solutions. Individually, for Problem 1, the best choice is greedy_best_first_search. For problem 2 and problem 3, A* with 'ignore_precondition' heuristic performs the best. Hence, overall for air cargo problems, A* with 'ignore_precondition' works the best as long as the problem complexity is within the limit for A* with 'ignore_precondition' memory requirements for that problem.

Generally, a simple problem (like Problem_1) do not need sophisticated heuristic based search algorithms as a simple uninformed search algorithm like BSF can easily find the solution. However, each algorithm has its advantages and disadvantages. I feel, most algorithms were developed considering one or two specific domains based on problems being worked at and then generalised. Here for problem_1, we find that greedy_best_first search performs the best as problem_1 is very straight-forward (no obstacles) and simple for greedy_best_first to chase after the goal state. But as the problem complexity increases for problems 2 and 3, greedy_best_first fails. With increase in complexity, additional information provided by informed search algorithms seems to be performing better by nudging the search in the correct direction to reach the goal state than un-informed non-heuristic based searches. Experimenting with other heuristics, which would yield a balance between memory efficiency and speed can also be another approach. The main drawback of A* is that it might become out of memory for complex problems. Hence, with increase in problem complexity, we need to design better heuristic so that the search expands the least amount of required nodes. There are also many variations of the standard A* star algorithm developed to further improve its efficiency.

References:

1. Udacity AIND lecture videos on planning and search
2. S Russell, P Norvig, "Artificial Intelligence A Modern Approach", 3rd Edition
3. Wikipedia: Breadth First Search. https://en.wikipedia.org/wiki/Best-first_search
4. Wikipedia: A*. https://en.wikipedia.org/wiki/A*_search_algorithm
5. http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
6. https://en.wikibooks.org/wiki/Artificial_Intelligence/Search/Heuristic_search/Astar_Search

(All websites were last accessed 27th June 2018).