# Analysis

The result metrics for all 10 search methods are in the table below:

| Air Cargo Problem 1 | | | | | | |
|---|---|---|---|---|---|---|
| Search algorithm | Time (sec) | Plan length | Expansions | Goal tests | New nodes | |
| breadth_first_search | 0.2 | 6 | 42 | 57 | 176 | optimal |
| breadth_first_tree_search | 0.54 | 6 | 1300 | 1301 | 5304 | optimal |
| depth_first_graph_search | 0.005 | 9 | 10 | 11 | 40 | |
| depth_limited_search | 0.02 | 50 | 52 | 64 | 208 | |
| uniform_cost_search | 0.2 | 6 | 55 | 57 | 224 | optimal |
| recursive_best_first_search | 1.67 | 6 | 4022 | 4023 | 16164 | optimal |
| greedy_best_first_graph_search | 0.003 | 6 | 7 | 9 | 28 | optimal |
| A* search with h_1 | 0.024 | 6 | 55 | 57 | 224 | optimal |
| A* search with h_ignore_preconditions | 0.024 | 6 | 41 | 43 | 170 | optimal |
| A* search with h_pg_levelsum | 0.29 | 6 | 11 | 13 | 50 | optimal |
| Air Cargo Problem 2 | | | | | | |
| Search algorithm | Time (sec) | Plan length | Expansions | Goal tests | New nodes | |
| breadth_first_search | 8.7 | 9 | 3345 | 4613 | 30526 | optimal |
| breadth_first_tree_search | | | | | | |
| depth_first_graph_search | 0.94 | 324 | 329 | 330 | 2964 | |
| depth_limited_search | 9.48 | 50 | 3674 | 33918 | 34291 | |
| uniform_cost_search | 7.5 | 9 | 4780 | 4782 | 43381 | optimal |
| recursive_best_first_search | | | | | | |
| greedy_best_first_graph_search | 0.9 | 21 | 598 | 600 | 5386 | |
| A* search with h_1 | 7.5 | 9 | 4780 | 4782 | 43381 | optimal |
| A* search with h_ignore_preconditions | 2.72 | 9 | 1450 | 1452 | 13303 | optimal |
| A* search with h_pg_levelsum | 23.55 | 9 | 86 | 88 | 841 | optimal |
| Air Cargo Problem 3 | | | | | | |
| Search algorithm | Time (sec) | Plan length | Expansions | Goal tests | New nodes | |
| breadth_first_search | 67.6 | 12 | 14523 | 17985 | 127417 | optimal |
| breadth_first_tree_search | | | | | | |
| depth_first_graph_search | 24.07 | 2043 | 5650 | 5651 | 45753 | |
| depth_limited_search | | | | | | |
| uniform_cost_search | 32.82 | 12 | 17884 | 17884 | 155325 | optimal |
| recursive_best_first_search | | | | | | |
| greedy_best_first_graph_search | 6.2 | 21 | 3373 | 3375 | 28947 | |
| A* search with h_1 | 33.6 | 12 | 17882 | 17884 | 155325 | optimal |
| A* search with h_ignore_preconditions | 10.8 | 12 | 5034 | 5036 | 44711 | optimal |
| A* search with h_pg_levelsum | 135.84 | 12 | 315 | 317 | 2903 | optimal |

Optimal plans for **problem 1** were generated by 8 out of 10 methods (please note, not all of them always generate optimal plans). They are different only by the orders of actions. This one was generated by breadth_first_search:

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

Unload(C2, P2, SFO)

5 out of 10 methods returned optimal plans for **problem 2**. The optimal plan generated by A* search with h_ignore_precondition looks as follows:

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)

5 out of 10 methods returned optimal plans for **problem 3**. A* search with h_pg_levelsum returned the following optimal plan:

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

Unload(C3, P1, JFK), Unload(C4, P2, SFO)

Breadth-first search always returns optimal plans and for all three problems it is one of the best search methods in terms of speed and memory. Depth-first search is very fast and has small memory footprint but it produces ridiculously long plans. While for the first problem it might be acceptable (9 steps vs 6 steps in the optimal plan), for the second and third problems the plans are simply useless. Moreover, the lengths of the plans generated by depth-first search are very sensitive to the order of actions in the problem description. The order in my code is (load_actions, fly_actions, unload_actions) and the length of the plan returned by the depth-first search for the third problem is 2043. When I changed the order to the original one: (load_actions, unload_actions, fly_actions) I got a plan with "only" 195 steps. For the second problem, it was 324 vs 1444 respectively.

Another example of the uninformed search algorithms that always produce optimal results is the uniform-cost search algorithm. In our case, for all three problems, the uniform-cost search is, in fact, the breadth-first search because all steps costs are equal to 0. The uniform-cost result metrics are slightly different from the metrics of the breadth-first search because the uniform-cost search uses a priority queue for the frontier vs. FIFO in the breadth-first search.

The other two uninformed search algorithms, breadth-first tree search and depth-limited search, are not very practical for this kind of planning problems. The breadth-first tree search generates an enormous number of duplicated nodes and it never ended for me on the second and third problems. On the first problem, it created 5304 new states vs. 176 states created by the breadth-first search. The depth-limited search never ended on the third problem but returned 50-step plans for the first and second problems (50 was the depth limit). It seems more reasonable to use iterative depth search when we cannot estimate the length of optimal plans

Both informed A* algorithms, A* with h_ignore_preconditions and A* with h_pg_levelsum, performed very well on all the problems. Both algorithms returned optimal plans. h_ignore_preconditions was faster than h_pg_levelsum on all 3 problems (0.024 vs 0.29, 2.72 vs 23.55 and 10.8 vs 135.84 respectively) but h_pg_levelsum evaluated significantly less nodes (approximately 3.5, 16 and 15.5 times less states for first, second and third problems respectively). The main overhead of the h_pg_levelsum algorithm is the planning graph generation. In my implementation, I reduced the generation time by caching positive and negative preconditions and effects that are used in the "A" node mutex functions. We can improve performance even more by ignoring mutexes altogether. Mutexes are used in the GRAPHPLAN algorithm but they are not used in h_pg_levelsum. When I disabled update_a_mutex and update_s_mutex in the create_graph method, it took h_pg_levelsum only about 40 sec. instead of 135 sec. on the third problem. Another reason why h_ignore_preconditions is faster than h_pg_levelsum is because in our problems the goals are satisfied by different actions so we can use a very simple and fast implementation (see the code). But in general case (and if we want to get an accurate heuristic), it's an instance of the set-cover problem which is NP-hard and its greedy polynomial version is not admissible.

In general, the informed search algorithms are superior to the uninformed ones. In our case, the problems are not very complex and so we could use the breadth-first search for all of them. But for harder problems, with hundreds of fluents and actions, the breadth-first search would simply run out of space. h_ignore_precondition and (especially) h_pg_levelsum expand far less nodes so they should be much more suitable for harder problems.