Performance Analysis

Introduction

- Progression planning problems can be solved with graph searches such as breadth-first, depth-first, and A*, where the nodes of the graph are "states" and edges are "actions".
- A "state" is the logical conjunction of all boolean ground "fluents", or state variables, that are possible for the problem using Propositional Logic.
- The planning graph is somewhat complex, but is useful in planning because it is a polynomial-size approximation of the exponential tree that represents all possible paths.
- The planning graph can be used to provide automated admissible heuristics for any domain. It can also be used as the first step in implementing GRAPHPLAN.
- The following short report compares the performance of several search algorithms applied to the developed planning problems presented in the project.

Identifying Optimal Sequence of Actions

Problem Description

Problem 1		Problem 2		Problem 3
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))

Solution

Problem 1	Problem 2	Problem 3
Load(C1, P1, SFO) **Load**(C2, P2, JFK) **Fly**(P2, JFK, SFO) **Unload**(C2, P2, SFO) **Fly**(P1, SFO, JFK) **Unload**(C1, P1, JFK)	**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)	**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)

Results

For every air cargo problem, several search algorithms were tested. The following table summarizes the findings. N.B. The yellow shadow indicates the utilized search method took more than 20 minutes. Is in the same folder that contains the submitted code.

Air Cargo Problem 1					
Search Method	**Plan Length**	**Expansions**	**Goal Tests**	**New Nodes**	**Time (sec)**
Breadth First Search	6	43	56	180	0.032
Breadth first Tree search	6	1458	1459	5960	0.988
Depth First graph search	20	21	22	84	0.015
depth limited search	50	101	271	414	0.093
uniform cost search	6	55	57	224	0.040
recursive best first search h1	6	4229	4230	17023	2.924
greedy_best_first graph_search h1	6	7	9	28	0.006
A* search h1	6	55	57	224	0.042
A* search h ignore preconditions	6	41	43	170	0.040
A* search h pg level sum	6	11	13	50	1.108

Air Cargo Problem 2					
Search Method	Plan Length	Expansions	Goal Tests	New Nodes	Time (sec)
Breadth First Search	9	3343	4609	30509	14.585
Breadth first Tree search					
Depth First graph search	619	624	625	5602	3.789
depth limited search	50	222719	2053741	2054119	918.599
uniform cost search	9	4853	4855	44041	13.128
recursive best first search h1					
greedy_best_first graph_search h1	21	998	1000	8982	2.649
A* search h1	9	4853	4855	44041	12.864
A* search h ignore preconditions	9	1450	1452	13303	4.755
A* search h pg level sum	9	86	88	841	187.923

Air Cargo Problem 3					
Search Method	Plan Length	Expansions	Goal Tests	New Nodes	Time (sec)
Breadth First Search	12	14663	18098	129631	107.672
Breadth first Tree search					
Depth First graph search	392	408	409	3364	1.907
depth limited search					
uniform cost search	12	18235	18237	159716	58.372
recursive best first search h1					
greedy_best_first graph_search h1	22	5614	5616	49429	16.832
A* search h1	12	18235	18237	159716	57.536
A* search h ignore preconditions	12	5040	5042	44944	17.686
A* search h pg level sum	12	325	327	3002	934.416

Performance Analysis and Discussion

To start with[1],

- **Expansions** refer to the number of times the frontier was expanded. It increases by 1 every time we call the *PlanningProblem's actions function*.
- **Goal tests** refer to how many times nodes were tested against the goal condition. It increases by 1 every time we call the *PlanningProblem's goal_test function*
- **New nodes** are the number of nodes added to the tree or graph during the search. It increases by 1 every time we call the *PlanningProblem's result function*

For **Air Cargo Problem 1** and because the problem size is relatively small (2 Cargos, 2 planes and 2 airplanes), all search methods could produce the optimal path in a reasonable time. The length of the path was longer (20 and 50) in both depth first and depth limited search respectively. The reason for that is well understood, given the way these searches operate. The algorithm start from a node and searches as far as possible along each branch. Simply, the goal may not be at a certain depth in the left most branch, but rather at a shallow one of the right most one! Moreover, the shortest plan may not be longer than $2^3 - 1 = 7$. The fact that is number is way higher in both depth search mechanisms means that some states were visited more than once!

The recursive best search approach was the slowest among all search methodologies, while the greedy approach was the fastest. Usually recursion in python is a very expensive process as it utilizes the stack and allocates new frame each time a function is called. Things become much worse when the depth of recursion increases as the problem size increases. On the other hand, the greedy approach required the least number of expansions, goal tests, and time and seemed to work smoothly towards the goal for this problem without either plateauing or getting stuck at a local point.

Among the **non-heuristic search methods,** breadth first tree search demanded the maximum number of node expansion, goal tests and new nodes and hence execution time. Again, this may be due to the its internal implementation that utilizes recursion. When compared with depth first graph search, the later performs 66 times faster. For the rest algorithms in this category, breadth first, depth first and uniform cost searches, the performance are almost comparable.

For the **heuristic search category,** the execution time for both A* with h1 and ignore precondition heuristics was very close. It was the worst for level sum heuristic. This discrepancy becomes more prominent in Problem 2 and 3. The current implementation is roughly $O(1)$, $O(n)$, $O(n^3)$ for h1, h1-ignore preconditions and h1-level sum, respectively. More to be said when discussing the other problems.

[1] Thanks to the useful post in the Udacity Forum and the associated responses, especially (Ravin_Kumar & chris_lapollo) https://discussions.udacity.com/t/making-sure-i-understand-expansion-goal-test-and-new-nodes/232906

If I were to choose an algorithm for this small problem, I would go with depth first graph search from the first camp and greedy approach from the other, as they clearly achieve the best execution time.

For **Air Cargo Problem 2,** was somehow larger (3 cargos, 3 airports, 3 planes). The depth first graph search enjoys the fastest execution time among all other search methodologies while A* with ignoring precondition and greedy approach somehow comparable and comes second in performance. Again, the greedy seemed to perform smoothly without being stuck in locals. The performance of uniform cost search and A* with h1 is as expected comparable because of the utilized constant heuristic. The depth limited search expanded excessive number of nodes, tested many goals and hence required the longest execution time (almost 300 times slower than the fastest, greedy approach). Clearly the wrong branch was expanded each time!

Among the **non-heuristic search methods,** depth first search produced the longest plan in a relatively fast time. Compared to breath first search and uniform cost search it required the lest number of nodes instantiation and goal tests.

For the **heuristic search category,** the execution time for both A* with level-sum heuristic was the slowest even though it required the least number of node expansions and goal tests. The is well understood because of the cubic complexity.

If I were to choose an algorithm for this problem, I would go again with depth first graph search from the first camp and greedy approach from the other, as they clearly achieve the best execution time.

For **Air Cargo Problem 3,** was largest to be tested (4 cargos, 4 airports, 2 planes). The greedy approach enjoys the fastest execution time among all other search methodologies while A* with ignoring precondition and depth first graph search were somehow comparable and comes second in performance. Again, the performance of uniform cost search and A* with h1 is as expected comparable because of the utilized constant heuristic.

Among the **non-heuristic search methods,** depth first search produced the longest plan in the fastest time. Compared to breath first search and uniform cost search it required the lest number of nodes instantiation and goal tests.

For the **heuristic search category,** the execution time for both A* with level-sum heuristic was the slowest even though it required the least number of node expansions and goal tests. The is again well understood because of the $O(n^3)$ complexity. The best performance in this category came to greedy and A* with preconditions as they required the second least number of node expansion and goal tests.

If I were to choose an algorithm for this problem, I would go again with depth first graph search from the first camp and greedy approach from the other, as they clearly achieve the best execution time.

Concluding Remarks and Learned Lessons

- We cannot generally combine the analysis of deterministic and heuristic approaches. Nevertheless, I did that at the beginning of each problem to get the general behavior of how algorithms from those different camps compares given several other metrics. I then presented a separate analysis for every approach.
- Generally, depth limited search was not clearly effective, as we do not know apriori the desired depth where the goal is found.
- We cannot usually rely on using depth first search even though it provided the fastest performance. The search may run the risk of not terminating if one branch kept expanding and no goal was found. Therefore, it should be used with caution after considering the nature of the problem.
- Similarly, for heuristic approaches, the greedy algorithm was somehow lucky not to stuck in locals. Again, it should be used with caution after considering the problem of interest. A* with a suitable defined heuristic is more reliable for wider range of problems.
- There should be a balance between choosing an excellent heuristic and its implementation complexity. While level-sum demanded the least number of goal checks and node extensions, its performance was hindered by the $O(n^3)$ compleixity.