Introduction

This is my analysis of the planning problems listed in README.md. The goal of this project was to successfully implement parts of the classical planning problem, planning graph search, and the `GRAPHPLAN` algorithm. After passing the test cases, I ran the code to find solutions using 7 of the 10 search algorithms: breadth-first search (BFS), depth-first search (DFS), uniform cost search (UCS), greedy best-first graph search (Greedy with h_1), and three A* searches (with h_1, ignore preconditions (IP), and level sum (LS)).

One note on how I ran the algorithms. I used an older computer that I could run the solutions on without worrying about how long it took. The times would be significantly improved if I used a faster computer. For example, the A* with level sum search for problem 3 took 2469.787 seconds. On my faster machine, this took roughly half the amount of time: 1223.57 seconds.

Also, in each table below, the best performance in a metric for the uninformed search algorithms is given in red and the best for the automatic heuristics is given in green. The optimal path given at the end of each section was produced by A* with level sum.

Planning Problem 1

Below is the table of all metrics for the tested searches.

Search	Plan Length	Time(seconds)	Expansions	Goal Tests	New Nodes
BFS	6	0.9246	43	56	180
DFS	20	0.4332	21	22	84
UCS	6	1.1363	55	57	224
Greedy with h_1	6	0.1414	7	9	28
A* with h_1	6	1.1292	55	57	224
A* with IP	6	0.8567	41	43	170
A* with LS	6	3.5986	11	13	50

This first problem is interesting because there are relatively few actions that can be performed in any given state. Therefore, there isn't much variation in the number of nodes expanded (compared to the other problems). The best of the uniformed searches here is breadth-first search. This is the case because there are few actions in each state. We also see that depth-first search is the fastest, but does not find an optimal path (depth-first search will rarely find such a path). We also see that the greedy algorithm finds an optimal path in this case, while it does not in the other problems.

For the automatic heuristics, we can see that A* with level sum performs much better as far as the number of nodes created and expanded. As the complexity increases, this will improve. However, we see that this

search takes longer in terms of computation time. I believe that this extra time comes from traversing the planning graph to calculate the level sum heuristic. There are many different versions of an optimal path, but the one that A* with LS calculated is the following:

Load(C1, P1, SFO)

Fly(P1, SFO, JFK)

Load(C2, P2, JFK)

Fly(P2, JFK, SFO)

Unload(C1, P1, JFK)

Unload(C2, P2, SFO)

Planning Problem 2

Below is the table of all metrics for the tested searches.

Search	Plan Length	Time(seconds)	Expansions	Goal Tests	New Nodes
BFS	9	823.388	3343	4609	30509
DFS	619	153.295	624	625	5602
UCS	9	1228.510	4826	4828	43788
Greedy with h_1	19	157.236	635	637	25696
A* with h_1	9	1232.158	4826	4828	43788
A* with IP	9	384.319	1496	1498	13720
A* with LS	9	370.436	86	88	841

We see once again that depth-first search performs relatively well in terms of the metrics, but fails to find an optimal path. As a matter of fact, the path it finds is nearly 69 times longer than optimal. Also, we see that the greedy algorithm does not find an optimal path, but it is significantly better than depth-first search with comparable metrics. This is consistent with the fact that the greedy algorithm is not guaranteed to be admissible (AIMA page 376).

For the automatic heuristics, we begin to see the power of A* search with level sum. Not only does this search deal with at least 15 times fewer nodes, but it also the fastest of these searches. Below is the optimal path found by A* search with level sum:

Load(C1, P1, SFO)

Fly(P1, SFO, JFK)

Load(C2, P2, JFK)

Fly(P2, JFK, SFO)

Load(C3, P3, ATL)

Fly(P3, ATL, SFO)

Unload(C3, P3, SFO)

Unload(C2, P2, SFO)

Unload(C1, P1, JFK)

Planning Problem 3

Below is the table of all metrics for the tested searches.

Search	Plan Length	Time(seconds)	Expansions	Goal Tests	New Nodes
BFS	12	4102.596	14663	18098	129631
DFS	392	108.051	408	409	3364
UCS	12	5496.508	18221	18223	159612
Greedy with h_1	27	1569.139	5530	5532	48705
A* with h_1	12	5434.953	18221	18223	159612
A* with IP	12	1585.427	5118	5120	45650
A* with LS	12	2469.787	403	405	3708

The results for this problem are comparatively not much different from those of Problem 2. This problem is more difficult so there are more nodes to deal with and the searches take more time, but the comparison between the search methods remain the same. That is, the greedy and depth-first searches do not find an optimal paths, although they perform well in the metrics. Of the automatic heuristics, A* search with level sum is the best performing except for time elapsed: A* search with ignore preconditions is faster. Below is the optimal path found by A* search with level sum.

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(C4, P2, SFO)

Unload(C2, P2, SFO)

Unload(C3, P1, JFK)

Unload(C1, P1, JFK)

Conclusions

Overall, A* search with level sum is the search algorithm seems to be the best choice. It finds an optimal path for all problems and among those searches that find an optimal path, it has a short runtime and deals with the least amount of nodes by far. However, as noted in AIMA, the level sum would not be admissible if there was some dependence between the goals. In that case, we would choose A* search with ignore preconditions. However, if there were dependence between the goals, we would need to change our implementation of this heuristic.

Of the uniformed searches, breadth-first search would be the choice if optimality is required. However, if time is the ultimate consideration, then greedy search with h_1 would be a good choice. It does not find an optimal path, but the speed improvement may be enough to outweigh the difference is path length.