Classical Planning project

In this project, a classical search agent is implemented to planning task. In this project, the planning problems are variations on an Air Cargo logistic problem-- if you have certain number of cargo and a set amount of planes, how do you get them from place to place?

Problem set:

The following lists the problems set

```
Note:
C1, C2 etc are the cargo,
P1, P2 etc are the planes
the rest, e.g. SFO are airports
```

problem1.

```
precondition: [ At(C1, SFO) AND At(C2, JFK)AND At(P1, SFO) AND At(P2, JFK) AND
At(P3, ATL)]
goal: [At(C1, JFK) AND At(C2, SFO)]
```

problem2.

```
precondition: [ At(C1, SFO) AND At(C2, JFK)  AND At(C3, ATL)  AND At(P1, SFO) AND
At(P2, JFK)]
goal: [At(C1, JFK) AND At(C2, SFO) AND At(C3, SFO)]
```

problem3.

```
precondition: [ At(C1, SFO) AND At(C2, JFK) AND At(C3, ATL) AND At(C4, ORD) AND
At(P1, SFO) AND At(P2, JFK)]
goal: [At(C1, JFK) AND At(C2, SFO) AND At(C3, JFK) AND At(C4, SFO)]
```

problem4.

```
precondition: [At(C1, SFO) AND At(C2, JFK) AND At(C3, ATL) AND At(C4, ORD) AND
At(C5, ORD) AND At(P1, SFO) AND At(P2, JFK)]
goal: [At(C1, JFK) AND At(C2, SFO) AND At(C3, JFK) AND At(C4, SFO) AND At(C5, JFK)]
```

Experiments

11 different search algorithms are compared: 3 uninformed search methods (breadth first, depth first, and uniform cost searc), and 8 with heuristics (A star and greedy best first graph search with the heuristics of unmet goals, maxlevel, levelsum and setlevel), are tested against 4 different

planning problems. The planning problems are of increasing complexity, and the searches are done through a planning graph. All experiments are run using pypy rather than normal python to speed up calculation speeds.

Code implementation

The code for this project is at this github repo Most of this is 'boilerplate' code from Udacity, and my implementation is only in the planning graph section

An interesting todo is to implement my own planning / search code ▯

Measuring algorithm performance

As per the AIMA book, the performance of search algorithms are measured in terms of:

> Completeness: Is the algorithm guaranteed to find a solution when there is one?
>
> Optimality: Does the strategy find the optimal solution (it has the lowest path cost among all solutions)
>
> Time complexity: How long does it take to find a solution?
>
> Space complexity: How much memory is needed to perform the search?

In the experiment set, we can think of space complexity as the number of nodes expanded, time complexity as the time taken to run the algorithm on a problem, and as we know the path plan for each of the strategies, we also know which algorithms are optimal.

Experiment results

The complete experiment output can be found here: and is also shwon in the table below.

	problem	algo	action	nodes_expanded	goal_test	new_nodes	plan_length	time_to_run	time_run_pypy
0	1	breadth_first_search	20	43	56	178	6	0.006867	0.032478
1	1	depth_first_graph_search	20	21	22	84	20	0.005211	0.007989
2	1	uniform_cost_search	20	60	62	240	6	0.013549	0.019776
3	1	greedy_best_first_graph_search with h_unmet_goals	20	7	9	29	6	0.001590	0.003040
4	1	greedy_best_first_graph_search with h_pg_levelsum	20	6	8	28	6	0.643192	0.668640
5	1	greedy_best_first_graph_search with h_pg_maxlevel	20	6	8	24	6	0.375530	0.187120
6	1	greedy_best_first_graph_search with h_pg_setlevel	20	6	8	28	6	0.627370	0.514928
7	1	astar_search with h_unmet_goals	20	50	52	206	6	0.009225	0.015860
8	1	astar_search with h_pg_levelsum	20	28	30	122	6	1.589226	0.267637
9	1	astar_search with h_pg_maxlevel	20	43	45	180	6	1.448989	0.209042
10	1	astar_search with h_pg_setlevel	20	33	35	138	6	1.558533	0.409566

	problem	algo	action	nodes_expanded	goal_test	new_nodes	plan_length	time_to_run	time_run_pypy
11	2	breadth_first_search	72	3343	4609	30503	9	2.293861	0.335592
12	2	depth_first_graph_search	72	624	625	5602	619	3.429965	0.541581
13	2	uniform_cost_search	72	5154	5156	46618	9	3.829271	0.724422
14	2	greedy_best_first_graph_search with h_unmet_goals	72	17	19	170	9	0.021665	0.024978
15	2	greedy_best_first_graph_search with h_pg_levelsum	72	9	11	86	9	15.307723	0.721489
16	2	greedy_best_first_graph_search with h_pg_maxlevel	72	27	29	249	9	23.846908	1.121079
17	2	greedy_best_first_graph_search with h_pg_setlevel	72	9	11	84	9	18.951996	1.206637
18	2	astar_search with h_unmet_goals	72	2467	2469	22522	9	2.476773	0.569513
19	2	astar_search with h_pg_levelsum	72	357	359	3426	9	410.255449	15.562978
20	2	astar_search with h_pg_maxlevel	72	2887	2889	26594	9	2136.789477	85.678390
21	2	astar_search with h_pg_setlevel	72	1037	1039	9605	9	1632.333792	90.568004
22	3	breadth_first_search	88	14663	18098	129625	12	11.422025	0.817486
23	3	depth_first_graph_search	88	408	409	3364	392	1.173182	0.179323
24	3	uniform_cost_search	88	18510	18512	161936	12	15.097682	1.362900
25	3	greedy_best_first_graph_search with h_unmet_goals	88	25	27	230	15	0.040227	0.007026
26	3	greedy_best_first_graph_search with h_pg_levelsum	88	14	16	126	14	29.806606	1.349766
27	3	greedy_best_first_graph_search with h_pg_maxlevel	88	21	23	195	13	32.883521	1.508719
28	3	greedy_best_first_graph_search with h_pg_setlevel	88	35	37	345	17	100.439219	5.700636
29	3	astar_search with h_unmet_goals	88	7388	7390	6511	12	10.053471	0.930212
30	3	astar_search with h_pg_levelsum	88	369	371	3403	12	589.944548	23.097150
31	3	astar_search with h_pg_maxlevel	88	9580	9582	86312	12	NaN	446.624085
32	3	astar_search with h_pg_setlevel	88	3423	3425	31596	12	NaN	485.887562
33	4	breadth_first_search	104	99736	114953	944130	14	115.053001	6.027091
34	4	depth_first_graph_search	104	25174	25175	22849	24132	NaN	1341.340737
35	4	uniform_cost_search	104	113339	113341	1066413	14	74.787319	13.715020
36	4	greedy_best_first_graph_search with h_unmet_goals	104	29	31	280	18	NaN	0.016012
37	4	greedy_best_first_graph_search with h_pg_levelsum	104	17	19	165	17	NaN	2.443730
38	4	greedy_best_first_graph_search with h_pg_maxlevel	104	56	58	580	17	NaN	6.415941
39	4	greedy_best_first_graph_search with h_pg_setlevel	104	107	109	1164	23	NaN	26.430064
40	4	astar_search with h_unmet_goals	104	34330	34332	328509	14	NaN	4.468241
41	4	astar_search with h_pg_levelsum	104	1208	1210	12210	15	NaN	154.348373
42	4	astar_search with h_pg_maxlevel	104	62077	62079	599376	14	NaN	4474.719203
43	4	astar_search with h_pg_setlevel	104	22606	22608	224229	14	NaN	4942.232729

Measuring space complexity: Nodes expanded vs actions (i.e. problem complexity) and algorithm used

The following slice of the results table shows how the number nodes expanded increases as the problem space increases. As expected, the number of nodes increased as the problem space increased. However, the greedy best first search expanded the smallest number of nodes and more importantly, as the number of actions increased, the nodes expanded increases sublinearly, i.e. the search space doesn't 'explode' like most of the other algorithms tested.

We can also see that the informed searches does better than all 3 of the uniformed search algorithms, especially as the problem becomes more complex. With the exception of the depth first search, all the uniformed search methods expanded a much larger number of nodes than the informed searches. This is becuase the uniformed search methods have no guidance on where the goal state is, and therefore need to explore more space in order to find the goal.

Among uninformed search methods, depth first search is the most efficient, with the least number of nodes expanded. Depth first search expands the deepest node first as opposed to *all* the nodes in a layer as per breadth first/uniform cost search, and therefore has a much smaller branching factor.

For the search algorithms with heuristics, we can see that the greedy best first search does much better than a* , and that the level sum heuristic is the best in guiding the agent towards the final goal as both algorithms when using this heuristic expands the least number of nodes. Since the greedy algorithm does not take into account the cost of the path it takes to reach a particular node, it does not need to consider previous nodes so it's memory requirements are smaller.

```
pivot = pd.pivot_table(df, 'nodes_expanded', 'algo', 'action')
pivot
```

action	20	72	88	104
algo				
astar_search with h_pg_levelsum	28	357	369	1208
astar_search with h_pg_maxlevel	43	2887	9580	62077
astar_search with h_pg_setlevel	33	1037	3423	22606
astar_search with h_unmet_goals	50	2467	7388	34330
breadth_first_search	43	3343	14663	99736
depth_first_graph_search	21	624	408	25174
greedy_best_first_graph_search with h_pg_levelsum	6	9	14	17
greedy_best_first_graph_search with h_pg_maxlevel	6	27	21	56
greedy_best_first_graph_search with h_pg_setlevel	6	9	35	107
greedy_best_first_graph_search with h_unmet_goals	7	17	25	29
uniform_cost_search	60	5154	18510	113339

```
# using an alternative pivot for plotting
pivot2 = pd.pivot_table(df, 'nodes_expanded', 'action', 'algo')
ax = pivot2.plot(title="Nodes expanded vs problem size for different planning
search algorithms", figsize=(12,8))
ax.set_xlabel("actions")
ax.set_ylabel("nodes expanded")
```

Measuring time complexity: Time required vs problem size and algorithm used

The following table and plot shows how the different algorithms perform in terms of time. For uniformed searches, the breadth first search takes the least time, as opposed to the depth first search, which takes much longer, especially as the problem size increases. This is because in the depth first search, if it goes down the 'wrong' path it will have to backtrack, and a bigger problem means it will spend a lot of time backtracking.

For the informed search algorithms, we can see that the heuristic chosen has a large impact on the time, e.g. using the unmet goals heuristic is about 1000 times faster than the set level heuristic! This is likely because the more complex heuristics take longer to compute, especially if the implementation is not efficient. However, if we compare the heuristics against the node expanded, we can see that the fastest heuristics also ended up with a lot more nodes expanded.

```
pivot3 = pd.pivot_table(df, 'time_run_pypy', 'algo', 'action')
pivot3
```

action	20	72	88	104
algo				
astar_search with h_pg_levelsum	0.267637	15.562978	23.097150	154.348373
astar_search with h_pg_maxlevel	0.209042	85.678390	446.624085	4474.719203
astar_search with h_pg_setlevel	0.409566	90.568004	485.887562	4942.232729
astar_search with h_unmet_goals	0.015860	0.569513	0.930212	4.468241
breadth_first_search	0.032478	0.335592	0.817486	6.027091
depth_first_graph_search	0.007989	0.541581	0.179323	1341.340737
greedy_best_first_graph_search with h_pg_levelsum	0.668640	0.721489	1.349766	2.443730
greedy_best_first_graph_search with h_pg_maxlevel	0.187120	1.121079	1.508719	6.415941
greedy_best_first_graph_search with h_pg_setlevel	0.514928	1.206637	5.700636	26.430064
greedy_best_first_graph_search with h_unmet_goals	0.003040	0.024978	0.007026	0.016012
uniform_cost_search	0.019776	0.724422	1.362900	13.715020

```
pivot4 = pd.pivot_table(df, 'time_run_pypy', 'action', 'algo')
ax = pivot4.plot(title="Time required vs problem size for different planning search
algorithms", figsize=(12,8))
ax.set_xlabel("actions")
ax.set_ylabel("time needed (s)")
```

Optimality-- which algorithms can achieve the optimal (shortest plan length) solution?

For the uniformed search algorithms, both breadth first and uniform cost searches are optimal. For the informed searches, both A star and greedy search are optimal for smaller problems, but as the problem size increases, A star gives the optimal solution for all the heuristics except for the level sum heuristic whereas the greedy algorithm slowly diverges away from the optimal solution-- likely because as the problem size increases, the cost of the previous path has a bigger and bigger impact on the final cost and as the previous cost is ignored in greedy search, this means it doesn't find the most optimal solution.

```
pivot5 = pd.pivot_table(df, 'plan_length', 'algo', 'action')
pivot5
```

action	20	72	88	104
algo				
astar_search with h_pg_levelsum	6	9	12	15
astar_search with h_pg_maxlevel	6	9	12	14
astar_search with h_pg_setlevel	6	9	12	14
astar_search with h_unmet_goals	6	9	12	14
breadth_first_search	6	9	12	14
depth_first_graph_search	20	619	392	24132
greedy_best_first_graph_search with h_pg_levelsum	6	9	14	17
greedy_best_first_graph_search with h_pg_maxlevel	6	9	13	17
greedy_best_first_graph_search with h_pg_setlevel	6	9	17	23
greedy_best_first_graph_search with h_unmet_goals	6	9	15	18
uniform_cost_search	6	9	12	14

Applying search algorithms to various scenarios:

> **Which algorithm or algorithms would be most appropriate for planning in a very restricted domain (i.e., one that has only a few actions) and needs to operate in real time?**

For real time, we want an algorithm that runs very fast, as well as giving a short plan length. With the exception of depth first search, all of the other algorithms give an optimal solution. Since the greedy search algorithm gives the shortest times among them as well as the least number of nodes expanded (and hence uses less memory, it is probably a good choice in this scenario.

> **Which algorithm or algorithms would be most appropriate for planning in very large domains (e.g., planning delivery routes for all UPS drivers in the U.S. on a given day)**

We would need an algorithm that runs fairly fast even on large problem size (otherwise the day will be over before the planning calculation finishes), and does not expand too many nodes(otherwise the planning computation will run out of memory). A good choice is the greedy search algorithm-- even if it doesn't give the most optimal solution, the solution isn't too far off optimal, and it runs quicker and expands less nodes as problem size increases as compared to the others.

> **Which algorithm or algorithms would be most appropriate for planning problems where it is important to find only optimal plans?**

From the experiments, we can see that A star, breadth first and uniform cost search all return the optimal plan, even as the problem size increases. If optimality is the only consideration, all of them are appropiate choices. If we need to optimise for memory and/or time as well, then we should use A star (with unmet goal heuristic) since this algorithm + heuristic combination gives the least increase in both memory and time as problem size increases.

Conclusion

From this set of experiments, we can see that informed search algorithms with the right heuristics can perform much better than uninformed search algorithms in terms of both speed and memory requirements, and still return optimal plans. from the experiments, the unmet goals heuristic is the most optimal for time, and the level sum heuristic is the most optimal for memory. Depending on the scenario complexity and computation requirements, we can tune the heuristic to suit.