Implement a Planning Search

Part 1 - Planning problems

For each of the 3 PDDL problems we ran 8 uninformed planning searches. The number of nodes expanded, the number of goal tests, the number of nodes created, the plan length and the elapsed times can be found in the tables below.

air_cargo_p1	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time (sec)**
breadth_first_search (BFS)	43	56	180	6	0.0297
uniform_cost_search (UCS)	55	57	224	6	0.0386
greedy_best_first_graph_search	7	9	28	6	0.0051
A* - h_1	55	57	224	6	0.0587
depth_first_graph_search	21	22	84	20	0.0142
depth_limited_search	101	271	414	50	0.0994
recursive_best_first_search	4229	4230	17023	6	2.7276
breadth_first_tree_search	1458	1459	5960	6	0.9678

air_cargo_p2	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time**
breadth_first_search (BFS)	3343	4609	3050	9	14.7312
uniform_cost_search (UCS)	4852	4854	44030	9	47.4259
greedy_best_first_graph_search	990	992	8910	17	7.6527
A* - h_1	4852	4854	44030	9	47.0616
depth_first_graph_search	624	625	5602	619	3.5058
depth_limited_search	*too long*				
recursive_best_first_search	*too long*				
breadth_first_tree_search	*too long*				

air_cargo_p3	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time**
breadth_first_search (BFS)	14663	18098	129631	12	104.7813
uniform_cost_search (UCS)	18235	18237	159716	12	566.1909
greedy_best_first_graph_search	5614	5616	4942	22	111.3885
A* - h_1	18235	18237	159716	12	366.7543
depth_first_graph_search	408	409	3364	392	2.0234
depth_limited_search	*too long*				
recursive_best_first_search	*too long*				
breadth_first_tree_search	*too long*				

Searches that return an optimal plan:

Breadth_first_search (BFS), **Uniform_cost_search (UCS)**, **A* - h_1** (which is the same as **UCS** as explained in part 2) all returned an optimal plan (plan of length 6 for problem 1, length 9 for problem 2 and length 12 for problem 3).

In UCS, the number of goal tests equals the number of expanded nodes + 2 because goal test is run twice on the initial node. Otherwise, as a node is always goal-tested and then either expanded or returned, the number of goal tests would equal the number of expanded nodes + 1.

Searches that do not return an optimal plan:

Greedy_best_first_graph_search with h_1 and **Depth_first_graph_search (DFS)** do not return optimal plans.

Searches that took too long to return a plan for problems 2 and 3

Depth_limited_search, **Recursive_best_first_search (RBFS)** and **Breadth_first_tree_search** took too long to return a plan for problems 2 and 3.

- **Depth_limited_search** does not return an optimal plan. The depth limit is set to 50 and, from solving problem 3 using other algorithms, we now know that the optimal plan length is 12. If we had set the depth limit to 12, the algorithm may have returned a plan faster.
- **Recursive_best_first_search (RBFS)** would eventually return an optimal plan because the heuristics h_1 is admissible (h_1 always returns 1 so it never overestimates the number of actions to undertake in order to reach the goal). As it is expanding too many nodes, it did not return any plan in a reasonable time. Its main issue is that it is using too little memory and ends up re-expanding many states many times.
- **Breadth_first_tree_search** would eventually return an optimal plan, just like **breadth_first_search**. However, as it keeps re-expanding the same nodes many times, it was too slow to return a plan.

Part 2 - Domain-independent heuristics:

For each of the 3 PDDL problems we ran two A* searches with domain-independent heuristics:

- **h_ignore_preconditions**: minimum number of actions to undertake from the current state in order to satisfy all the goal conditions by ignoring the preconditions of all the actions.
- **h_pg_levelsum**: creates and uses a planning graph to estimate the number of actions that must be undertaken from the current state in order to satisfy all the goal conditions.

air_cargo_p1	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time (sec)**
A* - h_ignore_preconditions	41	43	170	6	0.0533
A* - h_pg_levelsum	11	13	50	6	1.4456

air_cargo_p2	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time**
A* - h_ignore_preconditions	1506	1508	13820	9	15.6409
A* - h_pg_levelsum	86	88	841	9	165.9616

air_cargo_p3	**Expansions**	**Goal Tests**	**New nodes**	**Plan Length**	**Time**
A* - h_ignore_preconditions	5118	5120	45650	12	94.0072
A* - h_pg_levelsum	408	410	3758	12	1058.9783

Both heuristics are admissible (they will never over-estimate the number of actions needed to reach the goal from the current state), which makes A* optimal. Both algorithms find an optimal plan (length 6 for problem 1, length 9 for problem 2 and length 12 for problem 3).

h_pg_levelsum expands a lot less nodes than **h_ignore_preconditions** (it expands only 7.9% of the number of nodes expanded by **h_ignore_preconditions** for problem 3) but calculating the heuristics takes a lot longer since the planning graph is rebuilt every time. (**h_ignore_preconditions** takes 8.9% of the time of **h_pg_levelsum** for problem 3)

Part 3 - Written Analysis

Optimal plan for Problems 1, 2, and 3

There are several optimal plans for each problem. We gave 1 example of optimal plan for each problem below.

Problem1:

Optimal plan has length 6:

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

Problem2:

Optimal plan has length 9:

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)

Problem3:

Optimal plan has length 12:

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)

All results:

air_cargo_p1		Expansions	Goal Tests	New nodes	Plan Length	Time (sec)
non-heuristic searches	breadth_first_search (BFS)	43	56	180	6	0.0297
	uniform_cost_search (UCS)	55	57	224	6	0.0386
	greedy_best_first_graph_search	7	9	28	6	0.0051
	A* - h_1	55	57	224	6	0.0587
	depth_first_graph_search (DFS)	21	22	84	20	0.0142
	depth_limited_search	101	271	414	50	0.0994
	recursive_best_first_search (RBFS)	4229	4230	17023	6	2.7276
	breadth_first_tree_search	1458	1459	5960	6	0.9678
heuristic searches	A* - h_ignore_preconditions	41	43	170	6	0.0533
	A* - h_pg_levelsum	11	13	50	6	1.4456

air_cargo_p2		Expansions	Goal Tests	New nodes	Plan Length	Time
non-heuristic searches	breadth_first_search (BFS)	3343	4609	3050	9	14.7312
	uniform_cost_search (UCS)	4852	4854	44030	9	47.4259
	greedy_best_first_graph_search	990	992	8910	17	7.6527
	A* - h_1	4852	4854	44030	9	47.0616
	depth_first_graph_search (DFS)	624	625	5602	619	3.5058
	depth_limited_search	*too long*				
	recursive_best_first_search (RBFS)	*too long*				
	breadth_first_tree_search	*too long*				
heuristic searches	A* - h_ignore_preconditions	1506	1508	13820	9	15.6409
	A* - h_pg_levelsum	86	88	841	9	165.9616

air_cargo_p3		Expansions	Goal Tests	New nodes	Plan Length	Time
non-heuristic searches	breadth_first_search (BFS)	14663	18098	129631	12	104.7813
	uniform_cost_search (UCS)	18235	18237	159716	12	566.1909
	greedy_best_first_graph_search	5614	5616	4942	22	111.3885
	A* - h_1	18235	18237	159716	12	366.7543
	depth_first_graph_search (DFS)	408	409	3364	392	2.0234
	depth_limited_search	*too long*				
	recursive_best_first_search (RBFS)	*too long*				
	breadth_first_tree_search	*too long*				
heuristic searches	A* - h_ignore_preconditions	5118	5120	45650	12	94.0072
	A* - h_pg_levelsum	408	410	3758	12	1058.9783

Compare and contrast **non-heuristic search result** metrics (optimality, time elapsed, number of node expansions) for Problems 1,2, and 3.

First, let's compare the searches that return an optimal plan:

- **Uniform_cost_search** and **A*-h_1** are the same: They both use best_first_graph_search, with functions node.path_cost and 1 + node.path_cost respectively. Empirically, they also have the same number of expanded nodes, goal tests and nodes created. Their elapsed times have the same order of magnitude.
- **Breadth_first_search (BFS)** is faster, expands fewer nodes, creates fewer nodes and does fewer goal tests than **Uniform_cost_search (UCS)** for all the problems. This makes sense because BFS and UCS explore nodes in the same order but BFS returns the solution before UCS: BFS does the goal test when the node is created and UCS does the goal test when the node is removed from the

frontier. Therefore, if the solution is of length d, UCS will expand all the nodes at depth d-1 whereas BFS will stop once it has expanded the node at depth d-1 whose child is a goal.

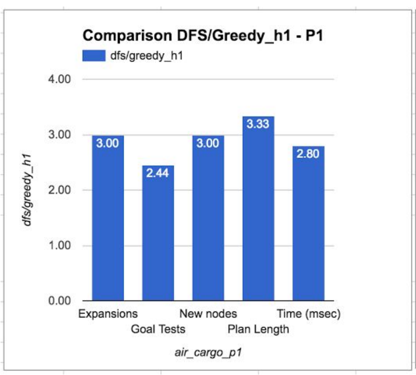
- **Recursive_best_first_search** and **breadth_first_tree_search** are optimal but are too slow because they keep re-expanding the same nodes many times.

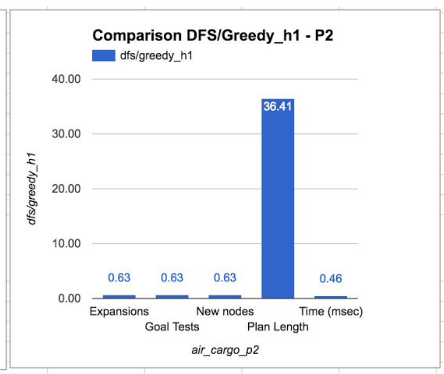
Then, let's compare the searches that do not return an optimal plan:

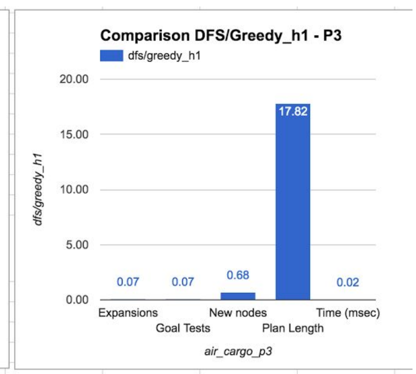
- In **greedy_best_first_graph_search with h_1**, all the nodes, no matter what their depth is, have a cost of 1. Therefore, any node on the frontier can be explored next. The implementation of the PriorityQueue in aimacode.utils.py uses bisect.insort. As the value of the (key,value) pair is always 1, bisect.insort orders the (key,value) pairs by key. Our Nodes are ordered by state (as seen in the __lt__ function defined inside class Node in aimacode.search.py). Therefore, the order in which nodes are expanded does not follow any particular logic.

  I was surprised to see that **greedy_best_first_graph_search** performed best for the first problem. To confirm my intuition that this good result was due to luck, I changed the definition of __lt__ to return True all the time. The plan length for problem 1 went from 6 to 20.

- As seen in the graphs below, **Depth_First_Graph_Search (DFS)** tends to find a solution a lot faster than **greedy_best_first_graph_search with h_1** (at least for problem 2 and 3) but with a much higher plan length.







- **Depth_limited_search** is too slow. As explained in part 2, reducing the value of the depth parameter may have helped.

Compare and contrast heuristic search result metrics using A* with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.

Both h_pg_levelsum and h_ignore_preconditions heuristics are admissible (they will never over-estimate the number of actions needed to reach the goal from the current state), which makes A* optimal. Both algorithms find an optimal plan (length 6 for problem 1, length 9 for problem 2 and length 12 for problem 3).

h_pg_levelsum expands a lot less nodes than h_ignore_preconditions (it expands only 7.9% of the number of nodes expanded by h_ignore_preconditions for problem 3) but calculating the heuristics takes a lot longer since the planning graph is rebuilt every time. (h_ignore_preconditions takes 8.9% of the time of h_pg_levelsum for problem 3)

What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?

The best heuristics used in these problems was **h_ignore_preconditions** because it found an optimal plan a lot faster than **h_pg_levelsum**.

We'll now compare the best heuristics **h_ignore_preconditions** to the best optimal non-heuristics, **breadth_first_search (BFS)**.

- For problem 2, **h_ignore_preconditions** found an optimal plan in the same time as **BFS** (~ 15s). **h_ignore_preconditions** expanded only 31% of the number of nodes expanded by **BFS**.
- For problem 3, **h_ignore_preconditions** found an optimal plan in about the same time as **BFS** (94s v.s. 104s). **h_ignore_preconditions** expanded only 35% of the number of nodes expanded by **BFS,** created only 35% of the number of nodes created by **BFS,** and performed 28% of the number of goal tests done by **BFS.**
- For problem 1, **h_ignore_preconditions** and **BFS** performed similarly on node counts and elapsed time metrics.

For problems 2 and 3, **Depth_first_graph_search (DFS)** was able to find a path in 2 or 3 seconds (v.s. 14s and 104s for **BFS**) but it is far from being optimal (the lengths of the DFS path were 619 and 392 v.s. 9 and 12 for **BFS**). For this problem, it is important to get the optimal path since it is expensive to fly planes.