AIND-Planning project work

Anand Saha <anandsaha@gmail.com>

Objective of this project is to solve a planning problem using various search algorithms. The ‘search graph’ is constructed using PDDL, where initial state, actions, goals are provided using the PDD Language.

Performance of uninformed and informed search algorithms are explored and analysed.

As part of exploring domain independent heuristics, we construct a planning graph to aid in informed search algorithms like A*.

Optimal plans for the three problems are below (Note that some of the steps can be shuffled within themselves without loosing correctness):

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)

Part 1 (Uninformed searches)

Below are my observations and analysis for part 1, where we try search with uninformed search algorithms.

Problem 1	Runtime (sec)	New Nodes	Goal Tests	Path Length	Optimality
breadth_first_search	0.028243611	180	56	6	Yes
breadth_first_tree_search	0.80853993	5960	1459	6	Yes
depth_first_graph_search	0.011853376	84	22	20	No
depth_limited_search	0.081231571	414	271	50	No
uniform_cost_search	0.031430946	224	57	6	Yes
recursive_best_first_search with h_1	2.399802675	17023	4230	6	Yes
greedy_best_first_graph_search with h_1	0.004414796	28	9	6	Yes

Problem 2	Runtime (sec)	New Nodes	Goal Tests	Path Length	Optimality
breadth_first_search	15.15846893	30509	4609	9	Yes
breadth_first_tree_search	Too long, aborted	N/A	N/A	N/A	N/A
depth_first_graph_search	3.819794416	5602	625	619	No
depth_limited_search	974.756212717	2054119	2053741	50	No
uniform_cost_search	12.595941758	44030	4854	9	Yes
recursive_best_first_search with h_1	Too long, aborted	N/A	N/A	N/A	N/A
greedy_best_first_graph_search with h_1	2.579644979	8910	992	21	No

Problem 3	Runtime (sec)	New Nodes	Goal Tests	Path Length	Optimality
breadth_first_search	113.494882378	129631	18098	12	Yes
breadth_first_tree_search	Too long, aborted	N/A	N/A	N/A	N/A
depth_first_graph_search	1.945440141	3364	409	392	No
depth_limited_search	Too long, aborted	N/A	N/A	N/A	N/A
uniform_cost_search	55.434217848	159605	18224	12	Yes
recursive_best_first_search with h_1	Too long, aborted	N/A	N/A	N/A	N/A
greedy_best_first_graph_search with h_1	20.16702115	49000	5562	22	No

Table: Execution stats

Algorithm	Analysis	Verdict
breadth_first_search	BFS is both optimal and complete. However it can be time consuming (compared to other algorithms) if the solution is at considerable depth. In the 3 problems, we see that it is always able to find the shortest path, but in it's endeavour to search the entire breadth at each depth, it's time (and space requirement) to search increases with depth exponentially. In our case, we see the runtime increasing exponentially, and so do node expansion.	Good for small graphs (with moderate depth)
breadth_first_tree_search	This is like BFS, except the algorithm revisits already visited nodes, because it doesn't keep track of the same. As is evident from the results, this is very inefficient even with small graphs.	Never recommended (tree based)
depth_first_graph_search	This is neither complete not optimal algorithm. This takes a branch from it's current node and goes to it's depth. If it's "lucky", it might find the goal in a very short time, but this highly depends on placement of the goal. Hence it is never guaranteed. In our case, the timings are very impressive (1.9 sec for problem 3). But the path length is horrible.	Cannot be relied upon.
depth_limited_search	It seems that this implementation (of the AIND) uses a tree instead of a graph, because we are revisiting the same node again and again. Time and Space requirements are high in this implementation, hence not usable.	Never recommended (tree based)
uniform_cost_search	It is the best algorithm for search algorithms which do not involve using a heuristics. It is complete and optimal (when branching factor is finite) For Problem 2 and 3, Uniform Cost search yields the best timings when optimality is required. It takes a hit on memory though.	**Recommended**
recursive_best_first_search with h_1	Saves space, but takes more time. In our problems 2 and 3, it took more than 15 mins, hence stopped execution.	Taking too long. Not recommended.
greedy_best_first_graph_search with h_1	This approach uses a heuristic function to update priority queue. This is not optimal, though it's timings and memory requirements are impressive, specially when compared to other (luckly) fast algorithm like DFGS.	This can be a good choice if timing is of importance, because the path length if not optimal is very near to optimal.

Table: Analysis

Summary:
- If *optimal path* is mandatory, Uniform Cost Search is the algorithm to use.
- If *performance* is important (at the cost of some optimality), the algorithm of choice is Greedy Best First Graph Search.
- Breadth First Search is also a good option specially when the graph is not too depth (How deep is not too deep?). It guatentees optimality and completeness.

Part 2 (Informed searches)

In part 2, we use A* search with 3 heuristisc. Results are below:

Problem 1	Runtime (sec)	New Nodes	Goal Tests	Path Length	Optimality
astar_search with h_1	0.036711397	224	57	6	Yes
astar_search with h_ignore_preconditions	0.029283288	170	43	6	Yes
astar_search with h_pg_levelsum	1.447519204	50	13	6	Yes
Problem 2	**Runtime (sec)**	**New Nodes**	**Goal Tests**	**Path Length**	**Optimality**
astar_search with h_1	12.557354583	44030	4854	9	Yes
astar_search with h_ignore_preconditions	3.914652865	13303	1452	9	Yes
astar_search with h_pg_levelsum	309.861966446	841	88	9	Yes
Problem 3	**Runtime (sec)**	**New Nodes**	**Goal Tests**	**Path Length**	**Optimality**
astar_search with h_1	57.434610842	159605	18224	12	Yes
astar_search with h_ignore_preconditions	16.332989127	44944	5042	12	Yes
astar_search with h_pg_levelsum	1596.434792112	2934	320	12	Yes

Analysis:

In order to pick the next node to expand, A* chooses the best one based on some criteria. That criteria is derived from two measurements: path cost of the current node + an estimate of the distance from current node to destination. The second part is a heuristic that is provided to the algorithm to use.

In part 2, we try with 3 different heuristics.
- **H_1**: this is a constant measurement irrespective of the current node or destination. When using this, the effective cost of a path is the path cost of current node. It is impressive to see that this performed better than the more sophisticated Plan Graph with Level sum. On the other hand, it has more memory requirements because of more node expansions.
- **H_Ignore_Preconditions**: This simply calculates how many preconditions of the goal is not met by current state. Even though simple, it yields the best performance.
- **H_PG_LevelSum**: This is the most sophisticated of the lot, and the most expensive. So much so that it is not practical to use it. For each heuristic calculation, we create a Plan Graph and calculate the level sum, which accounts for the bad performance. LevelSum however is very memory efficient and takes the least memory of the three. Hence it has it’s usecase when memory is a premium.

H_Ignore_Preconditions provide the best preformance as well as optimality in the case of informed searches.

Overall Analysis - comparison

When comparing the informed and uninformed searches, it is evident that the informed search gives better performance with more complicated graphs.

	Problem #	Best	Second best
Performance	**Problem 1**	greedy_best_first_graph_search with h_1	astar_search with h_ignore_preconditions
	Problem 2	greedy_best_first_graph_search with h_1 (*but not optimal*)	astar_search with h_ignore_preconditions
	Problem 3	astar_search with h_ignore_preconditions	greedy_best_first_graph_search with h_1 (*but not optimal*)
Memory requirements	**Problem 1**	greedy_best_first_graph_search with h_1	astar_search with h_ignore_preconditions
	Problem 2	greedy_best_first_graph_search with h_1 (*but not optimal*)	astar_search with h_ignore_preconditions
	Problem 3	astar_search with h_ignore_preconditions	greedy_best_first_graph_search with h_1 (*but not optimal*)

NOTE: I have deliberately removed astar_search with h_pg_levelsum (very very expensive) and depth_first_graph_search (very very suboptimal)

As is evident from the above table, going by numbers, A* with Ignore Preconditions shine as the state space increases. Moreover, though Greedy BFS has better numbers in Problem 1 and 2, it produced a suboptimal path.

Overall, A* with Ignore Predicions heuristics gave us the best results. The intuition behind it is that it used more information than just the current node's path cost to decide which node to expand next. This proves the usefulness of informed searches. There is even scope of better results than this, by working on a better heuristics.