Project 3

Heuristic Analysis for Planning Search Algorithm in the Air Cargo transport system

1. Description

The planning search problem for air cargo is defined in classical PDDL. Domain-independent heuristics are defined by implementing relaxed problem heuristics in my_air_cargo_ problems.py and implementing Planning Graph heuristics in my_planning_graph.py

2. Problem 1

The search space consists of 2^12 states, so it's efficient enough to go through the whole state space to find the optimal path. There is no need to use heuristic. In this case, **greedy_best_first_graph_search h_1** is the winner because it uses fewer operations.

Search Method	Command	Expansions	Goal tes	New Nodes	Plan Length	Time Elapsed
breadth_first_search	python run_search.py -p 1 -s 1	43	56	180	6	0.030252
breadth_first_tree_search	python run_search.py -p 1 -s 2	1458	1459	5960	6	1.00058
depth_first_graph_search	python run_search.py -p 1 -s 3	21	22	84	20	0.013589
depth_limited_search	python run_search.py -p 1 -s 4	101	271	414	50	0.090444
uniform_cost_search	python run_search.py -p 1 -s 5	55	57	224	6	0.038332
recursive_best_first_sear ch h_	python run_search.py -p 1 -s 6	4229	4230	17023	6	2.972291
greedy_best_first_graph_ search h_1	python run_search.py -p 1 -s 7	7	9	28	6	0.004935
	Heuristic Search					
A*_search h_1	python run_search.py -p 1 -s 8	55	57	224	6	0.042219
A*_search h_ignore_precondit	python run_search.py -p 1 -s 9	35	37	146	6	0.043446
A*_search h_pg_levelsum	python run_search.py -p 1 -s 10	11	13	50	6	4.107044

3. Problem 2

The search space consists of 2^27 states, so it's efficient enough to go through the whole state space to find the optimal path. There is no need to use heuristic. In this case, **breadth_first_search** is the winner because its plan length is optimal

Air Cargo Problem 2						
	Non-heuristic Search					
Search Method	Command	Expansions	Goal tes	New Nodes	Plan Length	Time Elapsed
breadth_first_search	python run_search.py -p 2 -s 1	3343	4609	30509	9	14.4316
breadth_first_tree_search	python run_search.py -p 2 -s 2	0	0	0	0	0
depth_first_graph_search	python run_search.py -p 2 -s 3	624	625	5602	619	3.544763
depth_limited_search	python run_search.py -p 2 -s 4	0	0	0	0	0
uniform_cost_search	python run_search.py -p 2 -s 5	4628	4630	42032	9	43.81318
recursive_best_first_sear ch h_	python run_search.py -p 2 -s 6	0	0	0	0	0
greedy_best_first_graph_ search h_1	python run_search.py -p 2 -s 7	444	446	3991	21	2.182467
	Heuristic Search					
A*_search h_1	python run_search.py -p 2 -s 8	4796	4798	43357	9	44.88766
A*_search h_ignore_precondit	python run_search.py -p 2 -s 9	1493	1495	13692	9	14.66249
A*_search h_pg_levelsum	python run_search.py -p 2 -s 10	81	83	793	9	1091.122

4. Problem 3

The search space consists of 2^32 states, the heuristic provides an efficient direction of how the search goes forward the goal state without going through the whole state space. In this case, **A*_search h_ignore_precondition** is the winner because it outperforms in terms of time complexity.

Air Cargo Problem 3						
	Non-heuristic Search					
Search Method	Command	Expansions	Goal tes	New Nodes	Plan Length	Time Elapsed
breadth_first_search	python run_search.py -p 3 -s 1	14663	18098	129631	12	199.8826
breadth_first_tree_search	python run_search.py -p 3 -s 2	0	0	0	0	0
depth_first_graph_search	python run_search.py -p 3 -s 3	408	409	3364	392	3.862622
depth_limited_search	python run_search.py -p 3 -s 4	0	0	0	0	0
uniform_cost_search	python run_search.py -p 3 -s 5	18235	18237	159716	12	522.3161
recursive_best_first_sear ch h_	python run_search.py -p 3 -s 6	0	0	0	0	0
greedy_best_first_graph_ search h_1	python run_search.py -p 3 -s 7	5614	5616	49429	22	58.03241
	Heuristic Search					
A*_search h_1	python run_search.py -p 3 -s 8	18235	18237	159716	12	518.0393
A*_search h_ignore_precondit	python run_search.py -p 3 -s 9	5118	5120	45650	12	112.6919
A*_search h_pg_levelsum	python run_search.py -p 3 -s 10	408	410	3758	12	6290.393

5. Optimal solution for Air Cargo problem

Problem	**Air Cargo Problem 1**	**Air Cargo Problem 2**	**Air Cargo Problem 3**
Algorithm	**greedy_best_first_graph _search h_1**	**breadth_first_search**	**A*_search h_ignore_precondi**
Optimal Plan	1.Load(C1, P1, SFO) 2. Load(C2, P2, JFK) 3. Fly(P2, JFK, SFO) 4. Unload(C2, P2, SFO) 5. Fly(P1, SFO, JFK) 6. Unload(C1, P1, JFK)	1. Load(C1, P1, SFO) 2. Load(C2, P2, JFK) 3. Load(C3, P3, ATL) 4. Fly(P2, JFK, SFO) 5. Unload(C2, P2, SFO) \Fly(P1, SFO, JFK) 6. Unload(C1, P1, JFK) 7. Fly(P3, ATL, SFO) 8. Unload(C3, P3, SFO)	1. Load(C2, P2, JFK) 2. Fly(P2, JFK, ORD) 3. Load(C4, P2, ORD) 4. Fly(P2, ORD, SFO) 5. Load(C1, P1, SFO) 6. Fly(P1, SFO, ATL) 7. Load(C3, P1, ATL) 8. Fly(P1, ATL, JFK) 9. Unload(C4, P2, SFO) 10. Unload(C3, P1, JFK) 11. Unload(C1, P1, JFK) 12. Unload(C2, P2, SFO)

6. Analysis

- **Breadth first search (BFS)**: expands all nodes at the frontier of the search graph before going deeper. This search always prefers the shortest path first. The time complexity will become a big disadvantage when search space grows larger. This approach seems successful in problem 1 and 2 when the state space is small.
- **Deep First Search Graph (DFS)** : goes as deeper as possible before considering other nodes at the frontier. **DFS** is not optimal and fails in all 3 problems.
- **Uniform Cost Search** : guarantees to find the path with the lowest total cost. This search finds the optimal path with a slower speed than **BFS.** While **BFS** stops after finding the goal state, Uniform Cost Search keeps searching after the goal state is found.
- **Recursive best first search h1:** does not return an optimal plan, because it expands recursively the same node over again.
- **Greedy best first graph search h1** a does not return an optimal plan even though it reaches the final goal faster.
- **A*_search h1:** always returns 1 for the estimated distance to goal. It is the simplest heuristic and provides the fastest solution only for problem 1.
- **A*_search h_ignore_preconditions** : estimates the minimum number of actions that is carried out from the current state to meet all the goal conditions by re-moving the preconditions for an action to be executed. This search is the best per-former for problem 3.
- **A*_search h_pg_levelsum** : is the planning graph representation of the problem state space to estimate the sum of all actions that are carried out from the current state to meet each individual goal condition. This search is very expensive in computation and the time complexity takes too long.

Bibliography

Russell, S., Norvig, P., & Intelligence, A. (2015). A modern approach. *Artificial Intelligence. Pearson.*