Udacity - Artificial Intelligence Nanodegree

Algorithm Heuristic Research by Tomasz Szyborski

Heuristics Analysis - Air Cargo Transport Problem

The algorithms were ran using PyPy3 to decrease runtime with watchdog - theinterruptingcow to check if a single run takes more than 10 minutes. If it does the run is cancelled without displaying any other data than "Search took more than 10 minutes"

Used machine was: MacBook Pro with processor 2.4 GHz Intel Core i5 and RAM 8 GB 1600 MHz DDR3

Problem 1

The first problem has 2 pieces of cargo, 2 airports and 2 planes. Each airport has a single piece of cargo and a plane, and the goal state is that the cargo is swapped between the airports.

Start

- SFO has C1 and P1
- JFK has C2 and P2

Goal

- JFK has C1
- SFO has C2

Solution

The most optimal solution to the problem is in 6 steps:

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)

Algorithm Comparison

For this problem **greedy_best_first_graph_search with h_1** performs the best by far, achieving the fastest speed with the lowest number of node expansions. Second one in terms of speed is **depth_first_graph_search**, however it expands 20 new nodes which is more than 3 times more than most optimal solution. The most pleasant explanation can be found in Chapter 3.5 Informed (heuristic) Search Strategies: "Greedy best-first search tries to expand the node that is closest to the goal, on the grounds that this is likely to lead to a solution quickly. Thus, it evaluates nodes by using just the heuristic function; that is, f(n) = h(n)."

Problem	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Time elapsed (s)
1	breadth_first_search	43	56	180	6	0.24543810996692628
1	breadth_first_tree_search	1458	1459	5960	6	2.4397842009784654
1	depth_first_graph_search	21	22	84	20	0.07720585400238633
1	depth_limited_search	101	271	414	50	0.32029334199614823
1	uniform_cost_search	55	57	224	6	0.19985272595658898
1	recursive_best_first_search with h_1	4229	4230	17023	6	5.831007299013436
1	greedy_best_first_graph_search with h_1	7	9	28	6	0.03033737698569894
1	astar_search with h_1	55	57	224	6	0.19437616399955004
1	astar_search with h_ignore_preconditions	41	43	170	6	0.20747600600589067
1	astar_search with h_pg_levelsum	11	13	50	6	0.9400990300346166

Problem 2

The second problem has 3 pieces of cargo, 3 airports and 3 planes. Each airport has a single piece of cargo and a plane, and the goal state is that the cargo is moved.

Start

- SFO has C1 and P1
- JFK has C2 and P2
- ATL has C3 and P3

Goal

- JFK has C1
- SFO has C2 and C3

Solution

The most optimal solution to the problem is in 9 steps:

1. Load(C1, P1, SFO)
2. Load(C2, P2, JFK)
3. Load(C3, P3, ATL)
4. Fly(P1, SFO, JFK)
5. Fly(P2, JFK, SFO)
6. Fly(P3, ATL, SFO)
7. Unload(C1, P1, JFK)
8. Unload(C2, P2, SFO)
9. Unload(C3, P3, SFO)

Algorithm Comparison

For problem two depth_first_graph_search gives the fastest solution, which is not optimal as it takies 619 steps to most optimal's 9. I find it rather stange, because it is the alogrithm with the highest Plan Length among all ran (that didn't time out).

It has to be seen that from breadth_first_tree_search, depth_limited_search, and recursive_best_first_search with h_1 are not time efficient in resolving this problem.

While astar_search with h_pg_levelsum has expanded and created the least new nodes of all (by far - 720 new nodes vs. 41828 nodes in uniform_cost_search), it also took 8 more seconds to run than the latter.

NOTE: breadth_first_tree_search, depth_limited_search, and ecursive_best_first_search with h_1 reached 10 minute timeout and further execution was halted.

Problem	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Time elapsed (s)
2	breadth_first_search	3343	4609	30509	9	11.035423444001935
2	breadth_first_tree_search	-	-	-	-	timeout
2	depth_first_graph_search	624	625	5602	619	1.927157653030008
2	depth_limited_search	-	-	-	-	timeout
2	uniform_cost_search	4604	4606	41828	9	20.32350147899706
2	recursive_best_first_search with h_1	-	-	-	-	timeout
2	greedy_best_first_graph_search with h_1	455	457	4095	16	2.456716775079258
2	astar_search with h_1	4604	4606	41828	9	24.04864026501309
2	astar_search with h_ignore_preconditions	1398	1400	12806	9	7.387566438992508
2	astar_search with h_pg_levelsum	74	76	720	9	28.388777951011434

Problem 3

The third problem has 4 pieces of cargo, 4 airports and 2 planes. Each airport has a single piece of cargo and two have planes, and the goal state is that all cargo is moved to two airports.

Start

- SFO has C1 and P1
- JFK has C2 and P2
- ATL has C3
- ORD has C4

Goal

- JFK has C1 and C3

- SFO has C2 and C4

Solution

The most optimal solution to the problem is in 12 steps:

1. Load(C1, P1, SFO)
2. Fly(P1, SFO, ATL)
3. Load(C3, P1, ATL)
4. Fly(P1, ATL, JFK)
5. Unload(C1, P1, JFK)
6. Unload(C3, P1, JFK)
7. Load(C2, P2, JFK)
8. Fly(P2, JFK, ORD)
9. Load(C4, P2, ORD)
10. Fly(P2, ORD, SFO)
11. Unload(C2, P2, SFO)
12. Unload(C4, P2, SFO)

Algorithm Comparison

Suprisingly - the fastest algorithm to find a way as depth_first_graph_search, however its number of steps was far from optimal.

The most optimal in terms of Plan Length with the best search time was astar_search with h_ignore_preconditions.

NOTE: breadth_first_tree_search, depth_limited_search, and ecursive_best_first_search with h_1 reached 10 minute timeout and further execution was halted.

Problem	Algorithm	Expansions	Goal Tests	New Nodes	Plan Length	Time elapsed (s)
3	breadth_first_search	14663	18098	129631	12	112.47396870690864
3	breadth_first_tree_search	-	-	-	-	Timeout
3	depth_first_graph_search	408	409	3364	392	2.5078743509948254
3	depth_limited_search	-	-	-	-	Timeout
3	uniform_cost_search	16963	16965	149136	12	134.12502289097756
3	recursive_best_first_search with h_1	-	-	-	-	Timeout
3	greedy_best_first_graph_search with h_1	4007	4009	35104	29	39.53955342900008
3	astar_search with h_1	16963	16965	149136	12	107.2099614610197
3	astar_search with h_ignore_preconditions	4422	4424	39027	12	19.34002199000679
3	astar_search with h_pg_levelsum	229	231	2081	13	61.959192529087886

Conclusions

We can choose "best" algorithm regarding two different aspects - time and space. The optimal choice usually takes a lot of both, but nowadays space is considered cheap while time is most expensive (especially in real-time systems). The perfect candidate for quick search should be A* algorithm, however this research shows that the quickest one is depth_first_graph_search for *ALL* problems but then again it was far from optimal. It should be stated that the quickest AND optimal for ALL problems is astar_search with h_ignore_preconditions algorithm. What has to be noted is that with aforementioned algorithm with ignoring precoditions for problem 2 and 3 expanded less nodes than the others reaching optimal solution.