Heuristic Analysis of Planning Search

Grant Bartel

The following contains a high level analysis of the planning search project involving planning search algorithms targeted at the air cargo domain.

Planning Problems

Below includes an analysis of the methods and metrics for non-heuristic planning solution search for three separate problems.

Problem 1

The first problem can be stated as

Init(At(C1, SFO) ∧ At(C2, JFK)
∧ At(P1, SFO) ∧ At(P2, JFK)
∧ Cargo(C1) ∧ Cargo(C2)
∧ Plane(P1) ∧ Plane(P2)
∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))

Given this problem, several search methods were evaluated, which can be seen in the table below.

	Problem 1				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
breadth_first_search	43	56	180	6	0.030267972
breadth_first_tree_search	1458	1459	5960	6	0.917715797
depth_first_graph_search	21	22	84	20	0.014316913
depth_limited_search	101	271	414	50	0.098581277
uniform_cost_search	55	57	224	6	0.039396604
recursive_best_first_search h_1	4229	4230	17023	6	3.021898546
greedy_best_first_graph_search h_1	24	26	98	8	0.017293352

As you can see, breadth first search provided the shortest planning length at the fastest time. While depth first graph search and greedy best first graph search both had better metrics in regards to expansions, goal tests, and new nodes, they fell short in plan length. However, if time constraints were important, greedy best first graph search might provide a sufficient plan length given it's twice as fast as breadth first search. Also, since we're looking for a single answer, we're not so concerned about space efficiency, and we want the shortest path in the tree, breadth first search is a better choice than depth first search. [1]

Below is an optimal sequence of actions based on the chosen search method.

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

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

Problem 2

The second problem can be stated as

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))

Given this problem, several search methods were evaluated, which can be seen in the table below. Cells with '-' took prohibitively long times to compute.

	Problem 2				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
breadth_first_search	3343	4609	30509	9	8.013116604
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	624	625	5602	619	3.49803946
depth_limited_search	-	-	-	-	-
uniform_cost_search	4853	4855	44041	9	12.514103058
recursive_best_first_search h_1	-	-	-	-	-
greedy_best_first_graph_search h_1	35	37	299	15	0.07981865

Again, similar to problem 1, breadth first search is superior to both depth first graph search and greedy best first graph search in terms of plan length. However, it's far slower, especially when compared to greedy best first graph search. As a matter of fact, the time is so short for greedy best first graph search that users of the algorithm may not care if it doesn't provide the most optimal path length, even if it's off by as much as it is in this example. Greedy best first graph search is actually the best algorithm not including plan length. Also, since we're looking for a single answer, we're not so concerned about space efficiency, and we want the shortest path in the tree, breadth first search is a better choice than depth first search. [1]

Below is an optimal sequence of actions based on the chosen search method.

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)

Problem 3

The third problem can be stated as

Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
∧ At(P1, SFO) ∧ At(P2, JFK)
∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
∧ Plane(P1) ∧ Plane(P2)
∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))

Given this problem, several search methods were evaluated, which can be seen in the table below. Cells with '-' took prohibitively long times to compute.

	Problem 3				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
breadth_first_search	14663	18098	129631	12	41.522112056
breadth_first_tree_search	-	-	-	-	-
depth_first_graph_search	408	409	3364	392	1.729347786
depth_limited_search	-	-	-	-	-
uniform_cost_search	18233	18235	159697	12	65.716441044
recursive_best_first_search h_1	-	-	-	-	-
greedy_best_first_graph_search h_1	4562	4564	40149	26	12.440870803

Just like problems 1 and 2, breadth first search outperformed both depth first graph search and greedy best first graph search in terms of plan length. This time though, depth first graph search beat both algorithms in all other metrics as opposed to greedy best first graph search in problem 2. However, in this case the path length is far too great and would probably not be a good option to the user of the algorithm anyway. Again, greedy best first graph search provides a reasonable path length with an almost four times speedup to breadth first search. Also, since we're looking for a single answer, we're not so concerned about space efficiency, and we want the shortest path in the tree, breadth first search is a better choice than depth first search. [1]

Below is an optimal sequence of actions based on the chosen search method.

Heuristic Analysis of Planning Search

Grant Bartel

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)

Domain-Independent Heuristics

Below includes an analysis of the methods and metrics for domain-independent heuristics for three separate problems.

Problem 1

For the same problem 1 stated previously, several search methods were evaluated, which can be seen in the table below.

	Problem 1				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
astar_search h_1	55	57	224	6	0.038506096
astar_search h_ignore_preconditions	41	43	170	6	0.042161522
astar_search h_pg_levelsum	15	17	66	6	3.578879543

You can see from the table above that the A* search with ignore preconditions did better than level sum by almost two orders of magnitude in terms of elapsed time while providing the same plan length. Due to these characteristics, the ignore preconditions heuristic would be the heuristic of choice for this problem. Also, the ignore preconditions heuristic uses slightly more memory but is much faster (i.e., worse space but much better time complexity) than the level sum heuristic, so it's the preferred method between these two. [2]

Below is an optimal sequence of actions based on the chosen search method.

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

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

Problem 2

For the same problem 2 stated previously, several search methods were evaluated, which can be seen in the table below.

	Problem 2				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
astar_search h_1	4853	4855	44041	9	11.411780392
astar_search h_ignore_preconditions	1428	1430	13085	9	4.770276319
astar_search h_pg_levelsum	114	116	1120	9	668.69557821

Same as for problem 1, the A* search with ignore preconditions outperformed the heuristic level sum, but this time by over two orders of magnitude in terms of elapsed time. Again, the ignore preconditions heuristic would be the heuristic of choice for this problem. Also, the ignore preconditions heuristic uses more memory but is much faster (i.e., worse space but much better time complexity) than the level sum heuristic, so it's the preferred method between these two. [2]

Below is an optimal sequence of actions based on the chosen search method.

Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)

Problem 3

For the same problem 3 stated previously, several search methods were evaluated, which can be seen in the table below. Cells with '-' took prohibitively long times to compute.

Heuristic Analysis of Planning Search

Grant Bartel

	Problem 3				
Search Method	**Expansions**	**Goal tests**	**New nodes**	**Plan length**	**Elapsed time**
astar_search h_1	18233	18235	159697	12	50.053781853
astar_search h_ignore_preconditions	4859	4861	43129	12	18.210239649
astar_search h_pg_levelsum	-	-	-	-	-

There's no real surprise at this point that the ignore preconditions heuristic when used with A* search performs the best. In this case, the A* search with the level sum heuristic didn't even finish within a reasonable time, so there's no other heuristic to consider but the ignore preconditions heuristic. Also, the ignore preconditions heuristic is much faster (i.e., worse space but much better time complexity) than the level sum heuristic, so it's the preferred method between these two. [2]

Below is an optimal sequence of actions based on the chosen search method.

Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C4, P2, SFO)
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

Conclusion

It appears that breadth first search for the original planning problems provides the best plan path length at the fastest rate. The only other search algorithm to approach it in overall performance is the uniform search, but still falls short in terms of elapsed time.

For the domain-independent heuristics, A* search with the ignore preconditions heuristic performs the best in every single case minus a very close second place versus the h_1 heuristic for problem 1.

[1] https://brilliant.org/wiki/breadth-first-search-bfs/

[2] http://www.cs.cmu.edu/~arielpro/15780/lec/780-16.pdf