AIND

HEURISTIC ANALYSIS

GIL AKOS // 7 JULY 2017 // PROJECT: IMPLEMENT A PLANNING SEARCH

OPTIMAL PLANS

PROBLEM 1 (From Search 9: A* Ignore Preconditions)

1. Load(C1, P1, SFO)
2. Fly(P1, SFO, JFK)
3. Unload(C1, P1, JFK)
4. Load(C2, P2, JFK)
5. Fly(P2, JFK, SFO)
6. Unload(C2, P2, SFO)

PROBLEM 2 (From Search 9: A* Ignore Preconditions)

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

PROBLEM 3 (From Search 9: A* Ignore Preconditions)

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

NON-HEURISTIC SEARCH RESULTS

Problem	Search	Plan Length	Time Elapsed	Node Expansions	Goal Tests	New Nodes	Search Name
1	1	6	0.02751	43	56	180	breadth_first_search
1	3	12	0.00962	12	13	48	depth_first_graph_search
1	7	6	0.00499	7	9	28	greedy_best_first_graph_search h_1
2	1	9	13.34535	3343	4609	30509	breadth_first_search
2	3	575	3.11002	582	583	5211	depth_first_graph_search
2	7	15	2.45138	990	992	8910	greedy_best_first_graph_search h_1
3	1	12	101.71823	14663	18098	129631	breadth_first_search
3	3	596	3.18308	627	628	5176	depth_first_graph_search
3	7	22	14.47791	5614	5616	49429	greedy_best_first_graph_search h_1

Compare and Contrast:

1. Breadth First Search (Search 1)
2. Depth First Search (Search 3)
3. Greedy Best First Search h_1 (Search 7)

The three non-heuristic search types listed above display significantly differing results. Breadth First Search will always take the longest to compute (as expected) as it is searching each level in the planning graph in its entirety for a goal before moving on to the next level. This ensures that the solution will not skip a shorter path of actions in favor of moving to the next level in the graph. As the table below shows, it finds the shortest (best or tied for best) plan length. Depth First Search acts in the opposite fashion, prioritizing searching deeper in the graph for solution before moving onto the next branch in the graph. This results in fast compute times at the expense of tending to result in longer sequences for the steps in the found plan. Even though a plan is arrived upon quickly, it is not optimal particularly when the problem tested increases in complexity. In Problem 3, Depth First chooses a plan that has 596 steps in contrast to the 12 found by Breadth First. Greedy Best First Graph Search results in solutions in between – it does not always find the shortest plan but it tends towards shorter plan lengths without the long compute times of Breadth First. Greedy Best First would be a good option in application when the complexity of the problem is high but so too is the cost of computing power. For the Cargo Planning Problem the real world cost would be far higher for a longer plan length (fuel, time, etc.) so spending longer planning for ensuring a shorter plan would be optimal i.e. we should use Breadth First.

HEURISTIC SEARCH RESULTS

Problem	Search	Plan Length	Time Elapsed	Node Expansions	Goal Tests	New Nodes	Search Name
1	9	6	0.03921	41	43	170	astar_search h_ignore_preconditions
1	10	6	0.93249	45	47	188	astar_search h_pg_levelsum
2	9	9	4.47435	1450	1452	13303	astar_search h_ignore_preconditions
2	10	9	333.07472	1643	1645	15414	astar_search h_pg_levelsum
3	9	12	17.00113	5040	5042	44944	astar_search h_ignore_preconditions
3	10	12	1250.88531	2841	2843	27040	astar_search h_pg_levelsum

Compare and Contrast:

1. A* Search Ignore Preconditions (Search 9)
2. A* Search Plan Graph Level Sum (Search 10)

The two heuristic search types listed above both find an optimal plan length (as compared to the non-heuristic Breadth First) but display diverging results for compute time and number of node expansions. By ignoring the preconditions with A*, we have a slightly longer compute time as the Greedy Best First (non-heuristic) search but with the added benefit of finding an optimal and shorter plan length. Using Level Sum with A* also arrives at the optimal path but the compute times are longer than without preconditions and even than Breadth First with an order of magnitude difference. It does manage to roughly match or outperform the ignore preconditions A* when it comes to the number of nodes expanded; however, this seems like a marginal benefit in comparison to compute times. The benefits of utilizing A* without preconditions also appear to do be even more valuable as the complexity of the problem increases as it still finds optimal paths in a reasonable amount of time.

*NB: I don't believe my level sum function is optimized for compute time – updating this function could result in better performance for the above tests.

FINAL SEARCH RESULTS

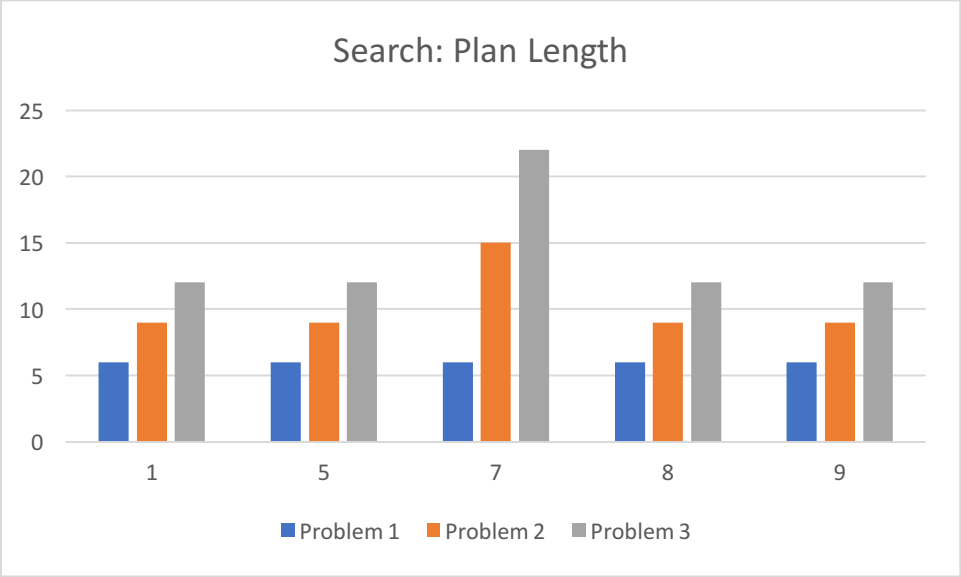
Problem	Search	Plan Length	Time Elapsed	Min/Max PL	Min/Max TE	Normalize Plan Length	Normalized Time Length	Composite	Average Composite
1	1	6	0.02751	6	0.00499	0.0000	0.6580	0.6580	0.8860
1	5	6	0.03713			0.0000	0.9390	0.9390	0.7558
1	7	6	0.00499			0.0000	0.0000	0.0000	0.6667
1	8	6	0.03360			0.0000	0.8362	0.8362	0.7037
1	9	6	0.03921	6	0.03921	0.0000	1.0000	1.0000	**0.4049**
2	1	9	13.34535	9	2.45138	0.0000	1.0000	1.0000	
2	5	9	11.99110			0.0000	0.8757	0.8757	
2	7	15	2.45138			1.0000	0.0000	1.0000	
2	8	9	12.12901			0.0000	0.8883	0.8883	
2	9	9	4.47435	15	13.34534	0.0000	0.1857	0.1857	
3	1	12	101.71823	12	14.47791	0.0000	1.0000	1.0000	
3	5	12	53.98093			0.0000	0.4528	0.4528	
3	7	22	14.47791			1.0000	0.0000	1.0000	
3	8	12	48.21327			0.0000	0.3867	0.3867	
3	9	12	17.00113	22	101.71823	0.0000	0.0289	0.0289	

Compare and Contrast:

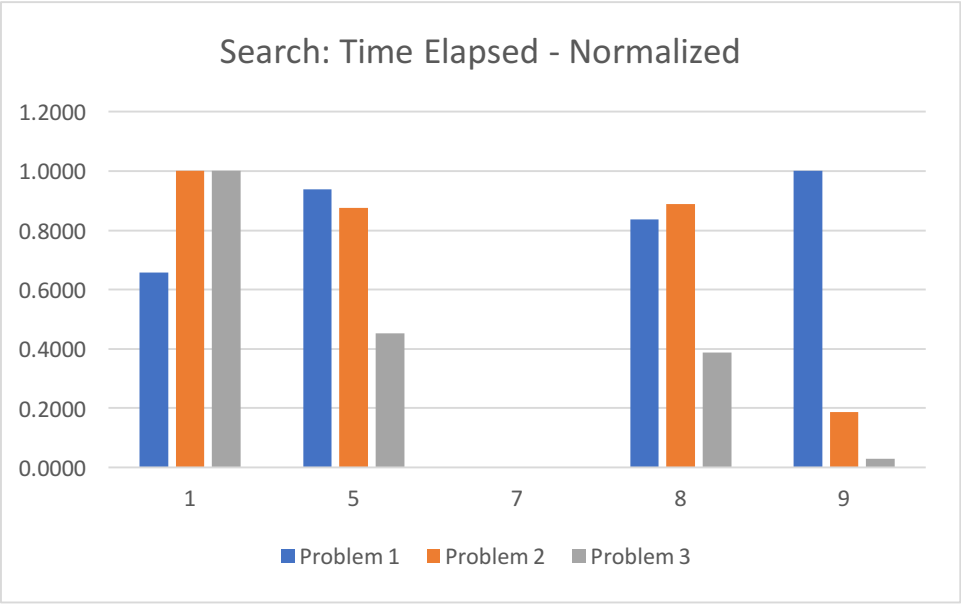
1. Breadth First Search (Search 1)
2. Uniform Search (Search 5)
3. Greedy Best First Search h_1 (Search 7)
4. A* Search h_1 (Search 8)
5. A* Search Ignore Preconditions (Search 9)

For a final comparison of search results, I have included the above list, removing 2, 4, and 6 because of overly long compute times after the first problem and 3 and 10 because they are outliers by order of magnitude in path length and time elapsed respectively. With this set of five options, we can hone in on

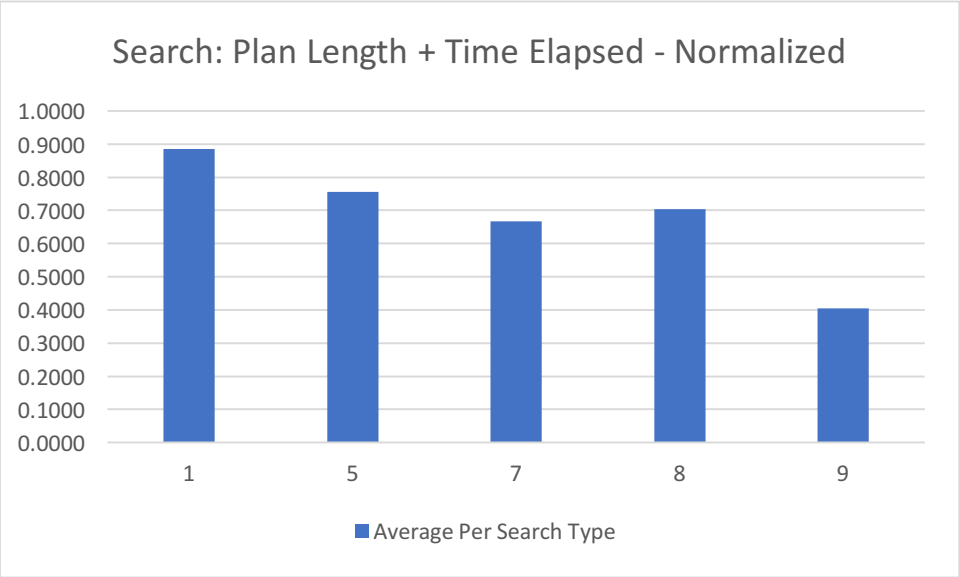
the detailed differences and find an optimal best solution. Since our three problems are composed of cargo planning logistics with high costs for each step in the plan, I will assume that the optimal solution will be measured primarily by the lowest plan length and secondarily by time elapsed. Furthermore, to generalize the results broader problem types (and equivalent metrics for optimization), I have added a normalized evaluation of both plan length and compute time in the above table and below graphs.



For Problem 1 (see table), all five search options find an optimal path with minimal difference in time elapsed. As we increase the complexity of the problem for Problem 2 and Problem 3, the variance in the plan length and time elapsed starts to increase. In the above graph, each search option's results from the three problems are clustered together with color denoting each problem. This would indicate that Search 7 performs the worst with diminishing results as the problem becomes more complex. One might conclude that any of the other four options would suffice.



By comparing the time elapsed for computing a solution when normalized per problem, with the above chart one might conclude the opposite – that Search 7 is the best because it is always the fastest regardless of the problem's complexity. Examining the other options here, one can see trends of performance for each option as well. Search 1 tends to perform worse with more complexity while Search 5 and 8 tend to do somewhat better in comparison to the others as the complexity increases. Most interestingly though, is that Search 9 radically increases its performance on compute time as the problem gets harder.



In order to arrive at a balanced conclusion per search option as well as generalize rules of thumb for implementation, I have calculated a composite performance value. This value is the average per study of the summed normalized plan length (per problem) and time elapsed (per problem). With this value displayed in the above graph and the observations about plan length and time elapsed above, **I conclude that Search 9 (A* Ignoring Preconditions) is the best search option for this study**. In all three problems, it finds the optimal path length and only incurs marginally more compute time.

This study also indicates the following rules of thumb for deciding what time of search algorithm to use.

1. If in doubt, use A* Ignoring Preconditions – because it has reasonable trade offs for path and compute time
2. If the compute time cost is greater than plan length cost, use Greedy Best First Graph Search
3. For simpler problems with shallower planning graphs, use Breadth First to ensure an optimal plan length is found – because it is less complex of a problem, compute time shouldn't matter
4. For more complex problems, use A* Ignoring Preconditions to find a plan – because it does a good job of minimizing compute time

REFERENCES:

Full results, tables, and graphs can be found in searches_results_table.xlsx