Jack Burdick
6Oct17

Optimal plan

Note: `Breadth first tree`, `depth limited search` and `recursive best first search h1` were unable to run on my computer in a reasonable amount of time. I did not attempt to download PyPy.

All 'optimal plans' were displayed using `astar_search with h_ignore_preconditions`. All charts are shown using a log scale.

Problem 1

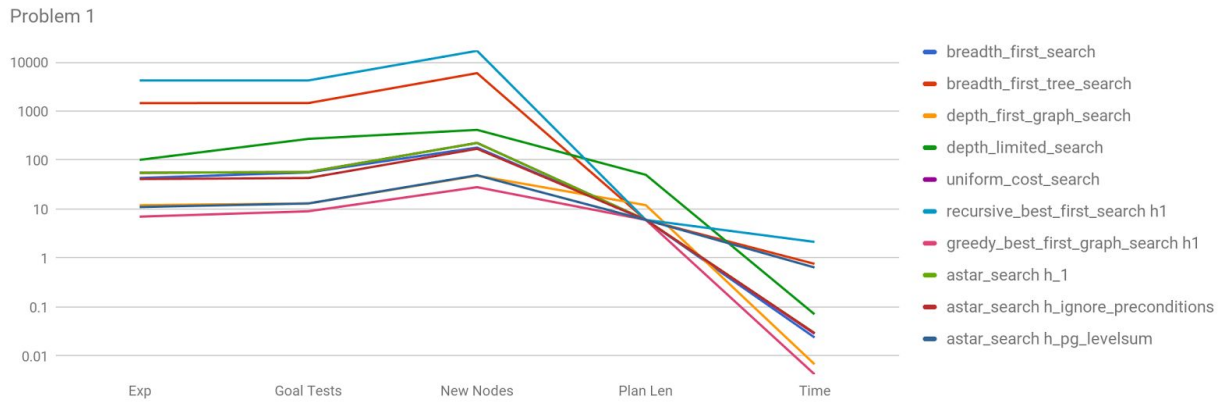
Optimal Plan

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)`

Results

	Problem	P1				
	Algorithm	Exp	Goal Tests	New Nodes	Plan Len	Time
1	breadth_first_search	43	56	180	6	0.024
2	breadth_first_tree_search	1458	1459	5960	6	0.759
3	depth_first_graph_search	12	13	48	12	0.007
4	depth_limited_search	101	271	414	50	0.071
5	uniform_cost_search	55	57	224	6	0.028
6	recursive_best_first_search h1	4229	4230	17029	6	2.134
7	greedy_best_first_graph_search h1	7	9	28	6	0.004
8	astar_search h_1	55	57	224	6	0.029
9	astar_search h_ignore_preconditions	41	43	170	6	0.029
10	astar_search h_pg_levelsum	11	13	49	6	0.641

Jack Burdick
6Oct17



When comparing non-heuristic search results metrics for problem 1, breadth_first_search, uniform_cost_search both performed "best" in terms of creating an optimal plan in a short amount of time. The breadth_first_search created only 180 new nodes while the uniform_cost_search created 224 new nodes. This is relatively insignificant for this problem size, but may be a problem at scale. When comparing heuristic search results using A* with "ignore preconditions" and "level-sum" we obtain the same optimal plan length. However, when using "ignore preconditions" we see over a 20x speed increase. However, this seems to come at the cost of more nodes being created 170 to 49, respectively.

Problem 2

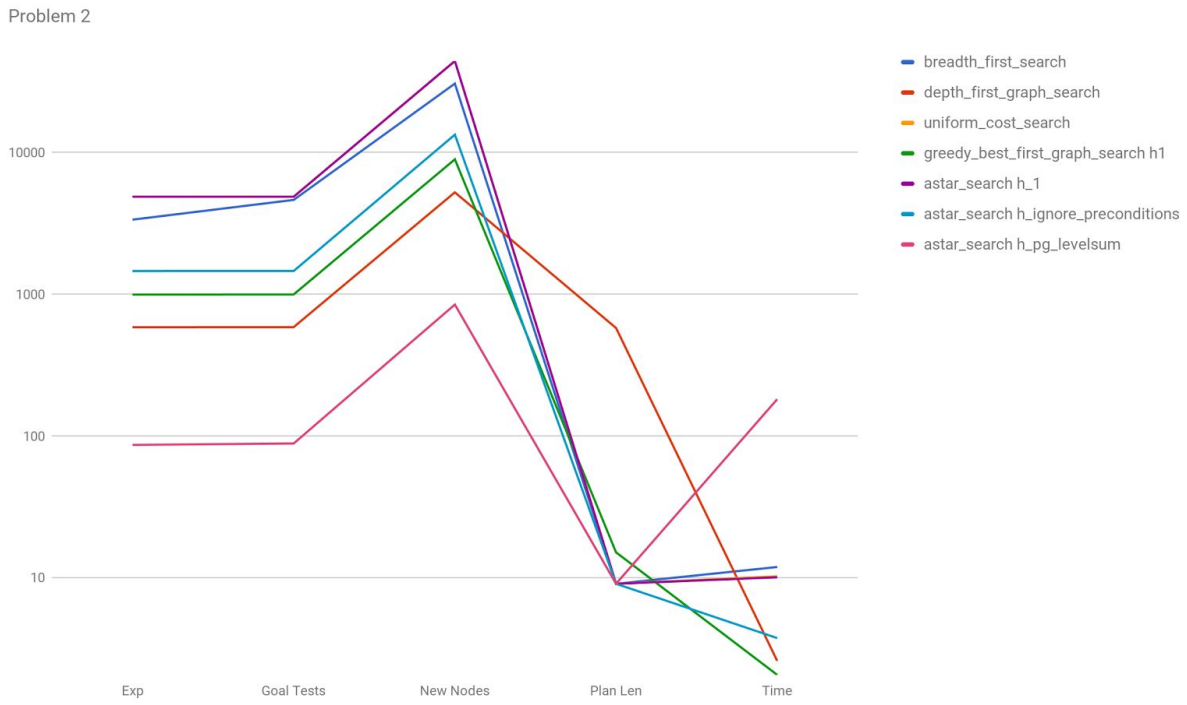
Optimal Plan

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)`

Jack Burdick
6Oct17

Results

	Problem	P2				
	Algorithm	Exp	Goal Tests	New Nodes	Plan Len	Time
1	breadth_first_search	3343	4609	30509	9	11.822
2	breadth_first_tree_search					
3	depth_first_graph_search	582	583	5211	575	2.571
4	depth_limited_search					
5	uniform_cost_search	4852	4854	44030	9	10.165
6	recursive_best_first_search h1					
7	greedy_best_first_graph_search h1	990	992	8910	15	2.048
8	astar_search h_1	4852	4854	44030	9	10.001
9	astar_search h_ignore_preconditions	1450	1452	13303	9	3.723
10	astar_search h_pg_levelsum	86	88	841	9	180.763



Of the three non-heuristic methods that were executed, the `breadth_first_search` and `uniform_cost_search` returned the best optimal paths with lengths of 9. However, these two strategies ran in 11.0 and 10.2 seconds respectively. The `depth_first_graph_search` ran in only 2.5 seconds, but the plan length was 575. Though the `uniform_cost_search` outperformed the `breadth_firs_search` in terms of time, the number of new nodes produced was roughly 1.5 larger. Choosing a "best" method would therefore depend on the system constraints. When comparing heuristic search results using A* with "ignore preconditions" and "level-sum" we

again obtain the same optimal plan length. However, when using "ignore preconditions" we see over a 48x speed increase. Again, this seems to come at the cost of more nodes being created 13303 to 841, respectively -- a roughly 16x difference.

Problem 3

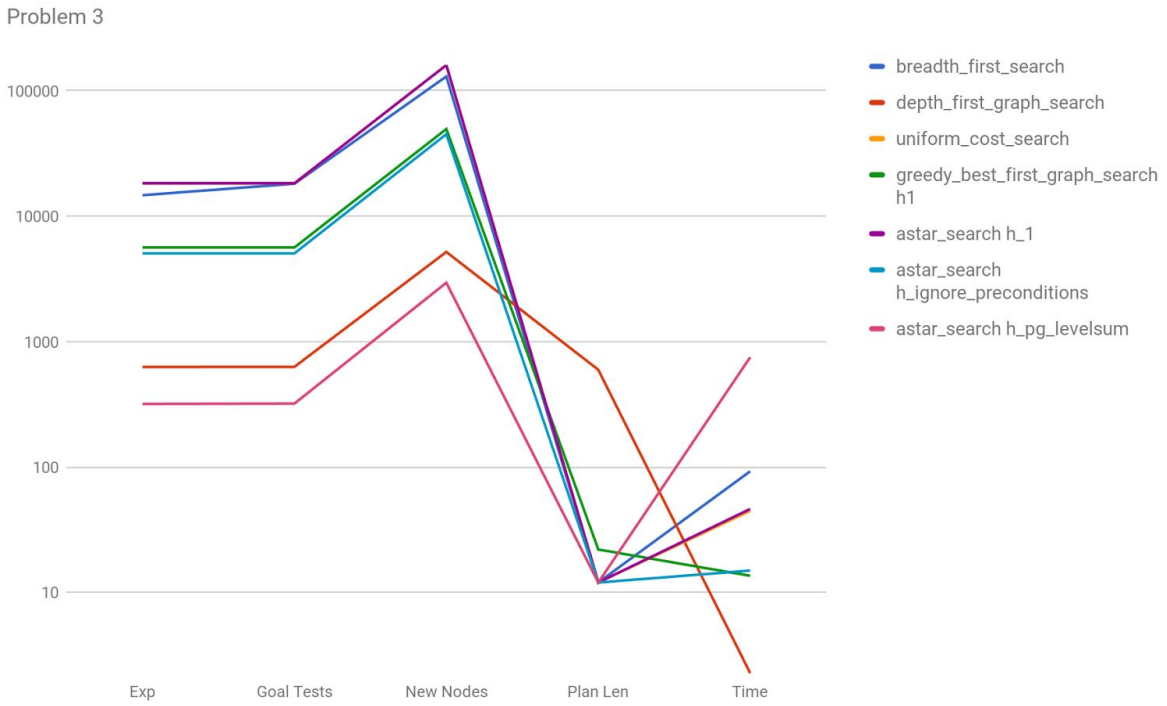
Optimal Plan

```
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)
```

Results

	Problem	P3				
	Algorithm	Exp	Goal Tests	New Nodes	Plan Len	Time
1	breadth_first_search	14663	18098	129631	12	92.337
2	breadth_first_tree_search					
3	depth_first_graph_search	627	628	5176	596	2.273
4	depth_limited_search					
5	uniform_cost_search	18235	18237	159716	12	44.725
6	recursive_best_first_search h1					
7	greedy_best_first_graph_search h1	5614	5616	49429	22	13.574
8	astar_search h_1	18235	18237	159716	12	46.296
9	astar_search h_ignore_preconditions	5040	5042	44944	12	14.941
10	astar_search h_pg_levelsum	318	320	2942	12	749.437

Jack Burdick
6Oct17



Of the three non-heuristic methods that were executed, the `breadth_first_search` and `uniform_cost_search` returned the best optimal paths with lengths of 12. These two strategies ran in 92.337 and 44.725 seconds respectively a roughly 2x difference. The `depth_first_graph_search` ran in only 2.273 seconds, but the plan length was 596. The `uniform_cost_search` outperformed the `breadth_first_search` in terms of time (by ~2x) and the number of new nodes produced was roughly 1.23 larger. Choosing a "best" method would therefore depend on the system constraints but in this case I would say the time vs resource trade-off favors the `uniform_cost_search` for the 2x speed increase.. When comparing heuristic search results using A* with "ignore preconditions" and "level-sum" we again obtain the same optimal plan length. However, when using "ignore preconditions" we see over a 50x speed increase. Again, this seems to come at the cost of more nodes being created 44944 to 2942, respectively -- a roughly 15x difference.

Final Notes

The best heuristic is difficult to determine in that defining "best" depends on your system constraints. The best plan is your first priority and time is your second priority I would say the "ignore_preconditions" is the best bet. However, if you are more concerned with resources than time you may consider "level sum" to be the best heuristic. Overall, the heuristic methods seem better for this problem -- in that they produce the same optimal results in a shorter amount of time and scale better -- they handle the more complex cases in a faster amount of time. Additionally, they did not (in these cases) make any major errors as a result of using the

heuristic. But again, implementing one of these methods would depend on a specific use case and could vary from domain to domain and system to system. It would be interesting to try more problems and see how these results continue to relate to one another - since our three problem space is rather limited. Also, in testing more problems, it would be interesting to see where using a heuristic negatively affects the optimal path score; what (edge cases?) would cause a non-heuristic to produce a more optimal path (at the cost of more time). That would be really interesting. Maybe there would be a way to detect these certain cases (some pattern could be detected?) and we could choose the planning method at run time?