Heuristic Analysis

Uninformed Raw Metrics

Algo	problem	expansions	goal tests	time	plan size
BFS	1	43	56	0.03	6
BF-Tree	1	1458	1459	1.24	6
DF-Graph	1	21	22	0.02	20
DF-Limited	1	101	271	0.02	50
UCS	1	55	57	0.04	6
BFS	2	3343	4609	18.16	9
BF-Tree	2	abort	abort	n/a	n/a
DF-Graph	2	624	625	4.15	619
DF-Limited	2	abort	abort	n/a	n/a
UCS	2	4853	4855	14.7	9
BFS	3	14663	18098	138.1	12
BF-Tree	3	abort	abort	n/a	n/a
DF-Graph	3	408	409	2.21	392
DF-Limited	3	abort	abort	n/a	n/a
UCS	3	18223	18225	59.5	12

Heuristic Raw Metrics

Algo	problem	expansions	goal tests	time	plan size
rec-Best	1	4229	423	2.90	6
greedy-best	1	7	9	0.005	6
a*-h1	1	55	57	0.041	6
a*-ignore-pre	1	41	43	0.107	6
a*-levelsum	1	11	13	0.610	6

Algo	problem	expansions	goal tests	time	plan size
rec-Best	2	n/a	n/a	n/a	n/a
greedy-best	2	998	1000	2.65	21
a*-h1	2	4853	4855	14.32	9
a*-ignore-pre	2	1450	145	85.95	9
a*-levelsum	2	86	88	58.94	9
rec-Best	3	n/a	n/a	n/a	n/a
greedy-best	3	5578	5580	18.0	22
a*-h1	3	18223	18225	59.6	12
a*-ignore-pre	3	5040	5042	640.3	12
a*-levelsum	3	325	327	307.8	12

Optimality Analysis (By Algorithm)

Breadth-first search

BFS does produce an optimal plan, though it can evaluate many irrelevant plans to arrive there. This is absolutely expected because BFS looks through *every* plan of a given number of steps before broadening. Producing the optimal path comes at the expense of an exponentially large search space as the input grows.

Breadth-first tree search

In this case, BFS tree search is strictly worse than BFS. The problem is that BFS maintains an explored set and doesn't bother evaluating states it has already arrived at before. BFS-tree does not care if a state has been seen before, so although it does arrive at the same optimal plan (optimal number of steps) it evaluates *all plans of less than that number of steps, even those with many repeated pointless steps*. For problems 2 and 3 the search ran long enough that I just terminated it.

Depth-First Graph Search

Because DFS does track an 'explored' set, it won't loop infinitely, but it is still possible to find very long plans of non-repeating but also non-relevant intermediate states (like a 619 step plan for problem 2 where a 9 step plan would suffice).

Depth-limited search

Produces an incredibly inefficient plan. This is expected because it has a high cutoff (50) and it allows repeated states. Therefore the first goal state it arrives at which is less than 50 steps it will consider successful, and it will explore the 'deepest' part of that space first. Given how many repeated states are possible in this state space, it's perfectly likely that you'll end up with a plan that is very close to the length of the cutoff, even if there are much shorter plans avaialable (a great example is problem 1 where a 6 step plan is available, and depth-limited search manages to produce a 50-step plan).

Uniform Cost Search

Since our 'cost' in this case is 1 per action, uniform cost should naturally be more expensive than breadth first search to produce the same result (since it spends time calculating the 'cost' which is equivalent to the number of steps, and therefore the same as just using a FIFO queue in order but probably wastes time trying to maintain a priority queue).

Recursive Best First Search

Produced an optimal plan, though searched a lot to find it. For p2 and p3 it never even finished, I eventually aborted.

Greedy Best First Search

Found optimal plan for P1 almost immediately. Since it's also using the h_1 heuristic (not a real heuristic) I think it's mostly coincidental that there was an easy to find solution early in the search space. P1 is fairly small.

For P2, it produced a distinctly unoptimal plan (exactly as expected since the heuristic is returning the same value for any node, it has no way to determine best-ness of a state).

A* (constant heuristic)

h1 as a heuristic means that all nodes score the same, but for P1 it found an optimal plan pretty quick. P2 it also found an optimal path (9 steps), but had to expand a lot of nodes to find it (no good way to prioritize with a constant heuristic). P3, same result, optimal plan but very broad search.

A good algorithm doesn't work well with a crummy heuristic, but on problems this simple it performs pretty quick :).

A* (ignore_preconditions)

This is the first search with a real heuristic. On problem 1 it produced an equally optimal plan to the constant heuristic in 80% of the expansion space, though it took longer in clock time (as it requires computation time to calculate the heuristic value).

For problem 2 it found the same length plan as h1, but only expanded 1450 nodes (much better search space). It seems the tradeoff here is for a very high branching factor, the expensive heuristics will be worth it. For problems with only a few possible paths, the simpler but faster to calculate heuristics would ultimately be better.

A* (level sum)

Took even longer in clock time than ignore_preconditions, so the heuristic is more expensive to calculate, but it trimmed the search space immensely (for p1 it only expanded 11 nodes, preconditions ignoring expanded 41).

For problem 2 it also found an optimal solution (as expected) and took a while to calculate it (58 seconds), but it only expanded 86 nodes. In fact, even though the level sum is a more expensive heuristic to calculate, because it is able to search so much more optimally it still ran in less wall clock time than the ignore-preconditions heuristic on the same problem.

Problem 3 demonstrated the same difference, much lower search space and significantly less clock time.

Optimal Plans:

Problem 1 Optimal Plan

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

Problem 2 Optimal Plan

```
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 Optimal Plan

Conclusion

There is non clear best algorithm or heuristic, it's very dependent on how big/broad the search space is and how expensive the heuristic is to calculate.

For problems that have a small search space, an uniformed search is actually going to perform best in terms of time because it won't spend any cycles calculating heuristics. For an optimal plan, uniform cost search would be a good choice, but if you're more concerned about execution time or memory consumption than optimality of the solution, depth-first search will produce a satisficing plan much more quickly.

For problems with a very large search space, heuristic searches are more or less necessary. Even in this subset of problems, there isn't a clear winner among heuristics for every problem. The ignore_preconditions heuristic is very quick to calculate compared to the levelsum heuristic. For problems that either have an only-moderately sized search space, or that have many individual independent clauses in the goal with few actions that undo those clauses, ignore_preconditions is going to be the clear winner. For problems with a very large state space though (where pruning aggressively at each level is very valuable), levelsum will make up for it's expense by having a much smaller number of nodes to search through for a solution.