Heuristic Analysis for Search Problems

Optimal sequence of actions		
Problem 1 (6 steps)	Problem 2 (9 steps)	Problem 3 (12 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)	1. Load(C1, P1, SFO) 2. Load(C2, P2, JFK) 3. Load(C3, P3, ATL) 4. Fly(P2, JFK, SFO) 5. Unload(C2, P2, SFO) 6. Fly(P1, SFO, JFK) 7. Unload(C1, P1, JFK) 8. Fly(P3, ATL, SFO) 9. Unload(C3, P3, SFO)	1. Load(C1, P1, SFO) 2. Load(C2, P2, JFK) 3. Fly(P1, SFO, ATL) 4. Load(C3, P1, ATL) 5. Fly(P2, JFK, ORD) 6. Load(C4, P2, ORD) 7. Fly(P2, ORD, SFO) 8. Fly(P1, ATL, JFK) 9. Unload(C4, P2, SFO) 10. Unload(C3, P1, JFK) 11. Unload(C2, P2, SFO) 12. Unload(C1, P1, JFK)

Based on the results of the uninformed non-heuristic searches listed below, it's clear that breadth-first search always results in the optimal plan however it expands a lot of nodes, performs a lot of goal tests, and takes a long time relatively speaking.

Depth-first search, on the other hand, takes far less time, expands much fewer nodes, and performs much fewer goal tests than breadth-first but it doesn't result in the optimal path. In fact it often results in paths which are far longer than the optimal path.

Uniform cost search does give us the optimal path but it does expand more nodes and perform more goal tests that breadth-first search. However, in the larger problems 2 and 3 it runs faster than breadth-first search (Almost twice as fast in problem 3.)

A* search with the ignore preconditions heuristic expands fewer nodes in the first 2 problems than the level sum heuristic but far more in the last problem indicating that the rate of growth of the expansions metric is far greater for the ignore preconditions heuristic than it is for the level sum heuristic. However, the level sum heuristic was far slower than the ignore preconditions heuristic.

Despite the expansions growing faster I believe the ignore preconditions heuristic was better because the level sum heuristic was orders of magnitude slower and the gap in speed only grew with problem size.

A* Search with the ignore preconditions heuristic seems to be the overall best algorithm even in comparison with the non-heuristic searches because it always achieved the optimal path, generally expanded fewer nodes, and performed much faster than the other methods in most cases.

The reason, I believe, is because by ignoring the preconditions we are simplifying the problem hence considering options faster because we don't have to assess preconditions. This makes the problem simpler but, as the problem grows, A* search using a simpler heuristic will expand more nodes. This idea is discussed in the Sliding Blocks Puzzle (Search lesson: 38 - 40 & Planning lesson: 18).

Results of running various searches on problems 1-3 (Respectively):

Algorithm	Expansions	Goal Tests	Time (s)	Plan Length
Breadth First Search	43	56	0.03296	6
Depth First Graph Search	21	22	0.01375	20
Uniform Cost Search	55	57	0.03468	6
Greedy Best First Graph Search	7	9	0.00434	6
A* Search	55	57	0.03838	6
A* Search (Ignore Preconditions)	41	43	0.04004	6
A* Search (PG Level Sum)	45	47	0.67873	6

Algorithm	Expansions	Goal Tests	Time (s)	Plan Length
Breadth First Search	3343	4609	12.56848	9
Depth First Graph Search	624	625	3.17635	619
Uniform Cost Search	4852	4854	11.14795	9
Greedy Best First Graph Search	990	992	2.23854	15
A* Search	4852	4854	11.10179	9
A* Search (Ignore Preconditions)	1450	1452	3.97970	9
A* Search (PG Level Sum)	1643	1645	125.51126	9

Algorithm	Expansions	Goal Tests	Time (s)	Plan Length
Breadth First Search	14663	18098	92.10465	12
Depth First Graph Search	408	409	1.61931	392
Uniform Cost Search	18234	18236	49.69079	12
Greedy Best First Graph Search	5605	5607	14.90632	22
A* Search	18234	18236	48.52815	12
A* Search (Ignore Preconditions)	5040	5042	15.69292	12
A* Search (PG Level Sum)	2841	2843	512.08773	12

Note: The A* Search using the PG Level Sum heuristic was run using pypy3 JIT compiler. The rest used python3.