Heuristic Analysis

One possible solution for each problem is presented below:

PROBLEM 1

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

PROBLEM 2

```
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

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

To analyze the different search algorithms and heuristics, I plotted the results for Problem 1(I tried to plot them for all three problems but some search algorithms were taking too much time).

The most efficient algorithm and heuristic was clearly **recursive best first search with h1** followed by **breath-first graph search**. When comparing efficiency, what is evident is that the chose heuristic is almost as or more important than choosing the search algorithm. A* with its worst heuristic performed 40 times slower than A* with it's best heuristic! It also appears that greedy algorithms tend to perform better.

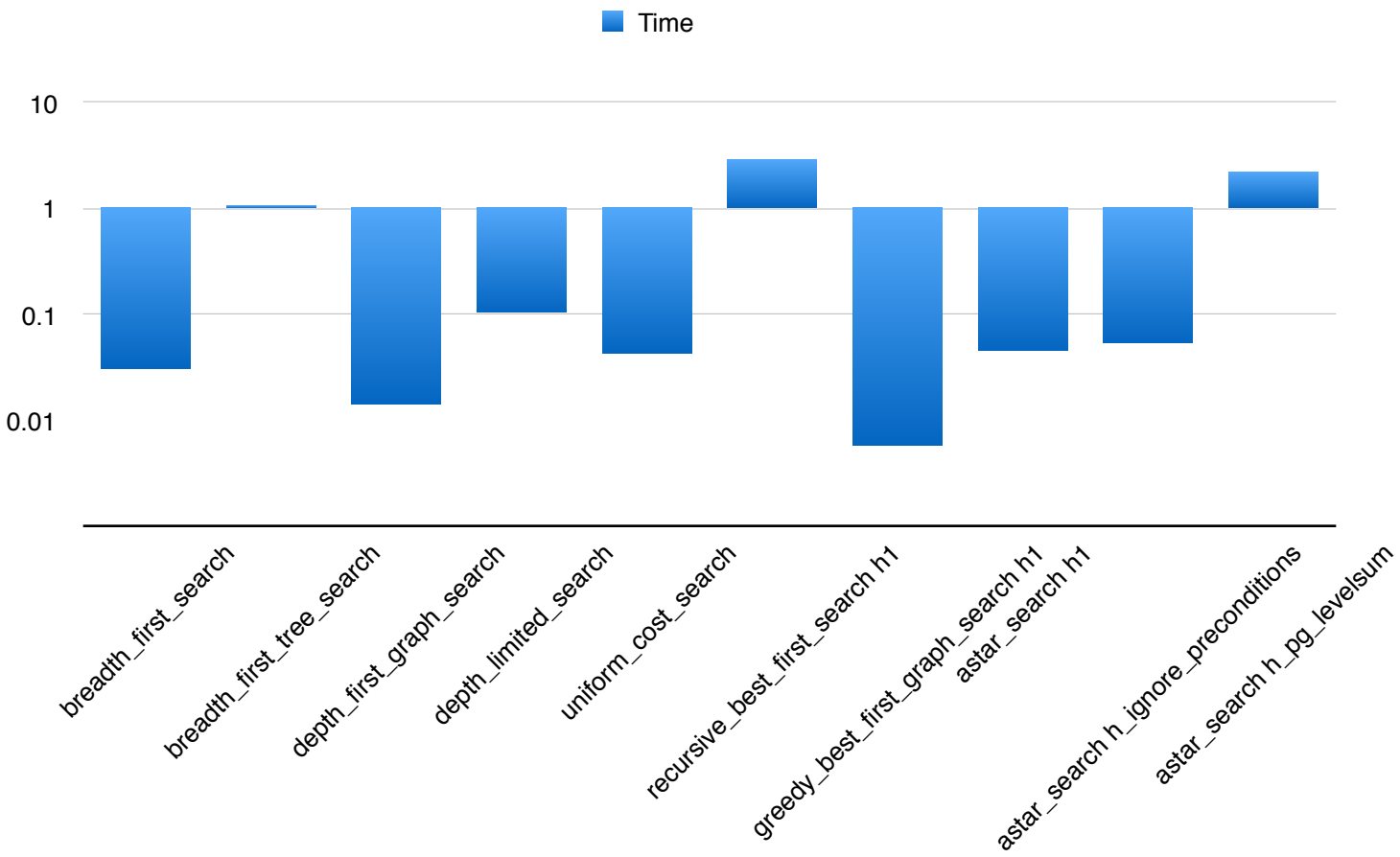
Another apparent thing is that, even though greedy algorithms find a solution faster, the solution found by more holistic algorithms, like A*, produce a better solution to the problem. This is specially true for more complex problems, like in problem 2 and 3.

Problem 1

	Expansions	Goal Tests	New Nodes	Plan Length	Time
breadth_first_search	43	56	180	6	0.03190767398336902
breadth_first_tree_search	1458	1459	5960	6	1.072463056014385
depth_first_graph_search	21	22	84	20	0.014605514996219426
depth_limited_search	101	271	414	50	0.10484412498772144
uniform_cost_search	55	57	224	6	0.04386359499767423
recursive_best_-first_search **h1**	4229	4230	17023	6	2.9926893670053687

	Expansions	Goal Tests	New Nodes	Plan Length	Time
greedy_best_-first_-graph_search **h1**	7	9	28	6	0.005809826019685715 4
astar_search **h1**	55	57	224	6	0.04675007899641059
astar_search **h_ignore_precon-ditions**	41	43	170	6	0.05357753098360263
astar_search **h_pg_levelsum**	11	13	50	6	2.231872236006893

It's easier to compare the times in the graph below, the times are in seconds and plotted in a logarithmic scale.



It is not sufficient to compare just speed to evaluate algorithms, it is also important to evaluate how well they solved the problem. For example, **recursive-best-first-search** solved problem 2 with 21 Actions and problem 3 with 19! While **A* with ignoring preconditions** solved problem 2 in 9 actions and problem 3 in 12.

Problem 2

	Expansions	Goal Tests	New Nodes	Plan Length	Time
depth_first_graph_search	624	625	5602	619	3.959414770
greedy_best_first_graph_search with h_1	998	1000	8982	21	8.710451987019042
astar_search with h_ignore_preconditions	1506	1508	13820	9	16.345200056006433

Problem 3

	Expansions	Goal Tests	New Nodes	Plan Length	Time
depth_first_graph_search	1292	1293	5744	875	3.6512935829814523
greedy_best_first_graph_search with h_1	907	909	5581	19	3.3213371110032313
astar_search with h_ignore_preconditions	2494	2496	19532	12	27.563580977992387

The results in the table show that there is a clear tradeoff between the speed that the algorithm runs and the plan length, or better solution found. The reason for the difference in performance in speed is that greedy algorithms like **depth-first-graph-search** simply return the first solution they find, and don't care about the cost of the solution. When we add a cost factor into greedy, like in greedy-best-first-graph-search with h1, the solutions are significantly more efficient because they are giving some information to the search algorithm about which path will have a preferred cost. Then there are the complete solutions, that will most likely find the best solution in the search space, but they take longer because they compare more options and get information for the heuristic.