Heuristic Analysis

This analysis presents a comparison of different search/planning algorithms when used on 3 planning problems with search spaces of different size. The performance of each problem, algorithm pair is measured using the number of expanded nodes, goal tests, visited states, actions taken and execution time.

Problem 1

This problem is by far the 'easiest' to solve (from computational point of view). The search space consists of 2^{12} states. The following table presents the performance data:

Search Method	Expansions	Goal Tests	New Nodes	Plan Size	Time
breadth_first_search	43	56	180	6	0.031
breadth_first_tree_search	1458	1459	5960	6	0.993
depth_first_graph_search	21	22	84	20	0.014
depth_limited_search	101	271	414	50	0.095
uniform_cost_search	55	57	224	6	0.384
recursive_best_first_search	4229	4230	17023	6	2.960
greedy_best_first_graph_search	7	9	28	6	0.005
A* with h1	55	57	224	6	0.039
A* with h_ignore_preconditions	41	43	170	6	0.044
A* with h_pg_levelsum	11	13	50	6	2.234

One optimal plan consists of the following (6) actions:

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

The best performer is `greedy_best_first_graph_search`. It found optimal plan using far fewer operations. But was it luck?

Problem 2

The search space for this problem is significantly larger, yet still manageable. It consists of 2^{27} states. One key difference from the previous problem is that I wasn't able to produce results for all searches, due to 'waiting too much'. Those that are available are presented in the following table:

Search Method	Expansions	Goal Tests	New Nodes	Plan Size	Time
breadth_first_search	3343	4609	30509	9	11.708
depth_first_graph_search	624	625	5602	619	3.061
uniform_cost_search	4852	4854	44030	9	38.310
greedy_best_first_graph_search	990	992	8910	21	6.166
A* with h1	4852	4854	44030	9	39.245
A* with h_ignore_preconditions	1506	1508	13820	9	12.527
A* with h_pg_levelsum	86	88	841	9	258.185

One optimal plan (9 actions) is the following:

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)

Maybe it was luck? The performance from `greedy_best_first_graph_search` was great but the plan that was found wasn't optimal. Our winner this time is `breadth_first_search`. Note that it did pretty well on `Problem 1` as well.

Problem 3

The state space now grows to 2^{32} states. The time required to run a search is now much larger than before as shown by the following results:

Search Method	Expansions	Goal Tests	New Nodes	Plan Size	Time
breadth_first_search	14663	18098	129631	12	89.299
depth_first_graph_search	408	409	3364	392	1.476
uniform_cost_search	18235	18237	159716	12	329.735
greedy_best_first_graph_search	5614	5616	49429	22	87.676
A* with h1	18235	18237	159716	12	327.593
A* with h_ignore_preconditions	5118	5120	45650	12	74.635
A* with h_pg_levelsum	408	410	3758	12	1713.060

An optimal plan with 12 actions:

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(C1, P1, JFK)
12. Unload(C2, P2, SFO)

Finally, we have a winner that uses a heuristic. In this case it is `A* search with h_ignore_preconditions`. Note that `breadth_first_search` is still performing quite well and obtains an optimal result.

Analysis

`breadth_first_search` expands all nodes at the frontier of the search graph before going deeper. As stated in videos 10,11 in the `Search` section of the lectures, `BFS` always considers the shortest path first. So, it gives optimal plan but it gets slower as the search space grows larger.

`depth_first_graph_search` in contrast of `BFS`, `DFS` goes as deeper as possible before considering other nodes at the frontier. As stated in section `3.4.3` of the `AIMA` book, `DFS` is not optimal. It appears that completely fails on our problems as well.

`uniform_cost_search` or Cheapest-First Search is guaranteed to find the path with the cheapest total cost (Video 16, Search section). This algorithm always expands the node that has the lowest cost. The implementation reveals that it calls `best_first_graph_search`. While this method finds optimal plans it is much slower than `BFS`. While `BFS` stops after finding goal state, `uniform_cost_search` continues its search.

The `A*` search (videos 27-33 in the Search section) is implemented using 3 different heuristic functions. All implementations provide an optimal plan, yet their performance differs by much. `A*` works by expanding the path that has the minimum value of the function `f`, which is defined as a sum of the `g` and `h`:

$$f = g + h$$

$$g(\text{path}) = (\text{path cost})$$

$$h(\text{path}) = h(\text{state}) = \text{estimated distance to goal}$$

Internally, `A*` simply calls `best_first_graph_search`, `uniform_cost_search` did that too.

h1 is the simplest possible heuristic. It always returns 1 for the estimated distance to goal. It is fastest only for problem 1. Not very useful.

h_ignore_preconditions is the best performer. This heuristic estimates the minimum number of actions that must be carried out from the current state in order to satisfy all of the goal conditions by ignoring the preconditions required for an action to be executed. This gives a sweet spot between the too simplistic h1 and too expensive h_pg_levelsum. For small search problems BFS is still better, though.

h_pg_levelsum was the hardest to implement. This heuristic uses a planning graph representation of the problem state space to estimate the sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition. Taking into consideration the preconditions as well, it is simply too expensive and this method is very slow.

One interesting observation is that number of expansions, goal tests and new nodes do not correlate perfectly with the execution time due to the complexity of the heuristic function (see A* with h_ignore_preconditions vs breadth_rst_search in Problem 1 and 2).

Conclusion

The analysis of the different problems did not provide a clear winner that is best in all situations. But why using a heuristic isn't always better? As long as the search space is small(ish), as in Problem 1 and 2, visiting every state is cheap and non heuristic methods are effective. Additionally, they are easier to understand/implement. When that space becomes large, though, this technique proves to be quite inefficient. Now, computing a heuristic has smaller cost compared to visiting every node in the planning graph. Which search algorithm is best? That largely depends on the problem.