Planning Project - Udacity Artificial Intelligence NanoDegree
Mark Helms - July, 2017

Introduction

This project analyzes the performance of a few different search algorithms when solving three different problems of escalating difficulty. Each problem deals with the transport of cargo (C) via airplane (P) from an origination airport to a destination airport. Available actions are load, unload and fly. All actions must be performed sequentially (vs in parallel) and for all algorithms, each action has a cost of one (1), although heuristics may vary. A plane may carry unlimited amounts of cargo. First, non-heuristic search methods are examined, then heuristic search methods are examined.

The Problems and Their Optimal Plans

Problem 1:

Initial State: C1 at SFO, C2 at JFK, P1 at SFO, P2 at JFK

Goal: C1 at JFK, C2 at SFO

Optimal Solution: The solution here is fairly obvious: load up each cargo, fly each plane to the opposite airport and unload. The minimum number of steps is therefore 6.

```
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:

Initial State: C1 at SFO, C2 at JFK, C3 at ATL, P1 at SFO, P2 at JFK, P3 at ATL

Goal: C1 at JFK, C2 at SFO, C3 at SFO

Optimal Solution: The solution here is the same as in problem 1, with the addition of 3 steps to load, fly and unload the third cargo. The minimum number of steps is 9.

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Fly(P3, ATL, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P3, SFO)
```

Problem 3:

Initial State: C1 at SFO, C2 at JFK, C3 at ATL, C4 at ORD, P1 at SFO, P2 at JFK

Goal: C1 at JFK, C2 at SFO, C3 at JFK, C4 at SFO

Optimal Solution: There are 4 cargoes and 2 planes. The quickest solution is to load up cargo, fly to the airport where cargo is waiting to go to the same destination as the first loaded cargo, load it up, then fly to the destination airport and unload each cargo. That should result in 12 steps.

```
Load(C1, P1, SFO)
```

Load(C2, P2, JFK)
Fly(P1, SFO, ATL)
Fly(P2, JFK, ORD)
Load(C3, P1, ATL)
Load(C4, P2, ORD)
Fly(P1, ATL, JFK)
Fly(P2, ORD, SFO)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)

Search Methods

Uninformed/non-heuristic search methods attempt to find a path to the goal by examining only the incremental cost of exploring. Informed/heuristic search methods attempt to find a path to the goal by minimizing the estimated distance to the goal in addition to the path cost of exploring.

The non-heuristic search methods used were: breadth-first tree search, depth-first graph search, breadth-first search, depth-limited search, and uniform cost search. Breadth-first tree search is a basic search algorithm that allows for backtracking and other redundant paths. Depth-first graph search is a technique that may not find the optimal path, but avoids backtracking and can complete quickly if there aren't negative effects of actions. Breadth-first search is a modification of breadth-first tree search, whereby revisiting nodes is eliminated and the optimal solution should be supplied in this case, where shallowest is equal to lowest cost. Depth-limited search is a modification of depth-first search whereby the maximum pursued depth is limited before starting down the next longest path. Uniform cost search is an algorithm meant to search in order of cheapest path, which is close to the breadth-first search algorithm in this case.

Heuristic search methods used were: greedy best first graph search, recursive best first search with h1, and a-star search with three different heuristics (h1, ignore preconditions, planning graph level sum). Greedy best first graph search always picks the answer with the lowest heuristic function, in an attempt to approach the goal as quickly as possible. Recursive best first search is similar to A* search, but it searches until the cost exceeds the next cheapest known option until the goal is found, after which it reduces memory usage by updating the parent nodes with the lowest f-value of the children, which may result in the need to re-expand multiple sections of the tree multiple times if it is determined their cost is lower than the next option. A-star, or A*, searches in a way that is similar to uniform cost search, but uses both the sunk path cost as well as an admissible heuristic to determine which path is the cheapest to explore next. The h1 heuristic has a constant value of 1. The "ignore preconditions" heuristic assumes all actions are relevant (allowable) at any state. The "planning graph level sum" heuristic calculates the sum of steps to achieve each sub-goal individually and is not necessarily admissible, but can be used with some success.

Search Results and Metrics

	Method	A* IP	UCS	A*h1	BFS	A*PGLS	GBFGS-h1	DLS	DFGS	BFTS	RBFS-h1
P1	**Length**	6	6	6	6	6	6	50	20	6	6
	Expansions	41	55	55	43	11	7	101	21	1458	4229
	Goal Tests	43	57	57	56	13	9	271	22	1459	4230
	New Nodes	170	224	224	180	50	28	414	84	5960	17023
	Time (ms)	25.21	24.73	24.11	19.56	362.52	3.44	59.06	9.14	593.78	1810.44
P2	**Length**	9	9	9	9	9	19	50	105	x	x
	Expansions	1369	4778	4778	3346	78	774	213491	107	x	x
	Goal Tests	1371	4780	4780	4612	80	776	1967093	108	x	x
	New Nodes	12539	43379	43379	30534	763	6938	1967471	959	x	x
	Time (ms)	2444	6812	6832	8698	28699	1103	557257	199	600000	600000
P3	**Length**	12	12	12	12	13	30	x	288	x	x
	Expansions	4478	17653	17653	14120	229	3998	x	292	x	x
	Goal Tests	4480	17655	17655	17673	231	4000	x	293	x	x
	New Nodes	39564	154877	154877	124926	2081	35002	x	2388	x	x
	Time (ms)	8879	29720	29556	63554	111806	6694	600000	707	600000	600000

Table 1: Search Algorithm Results and Metrics

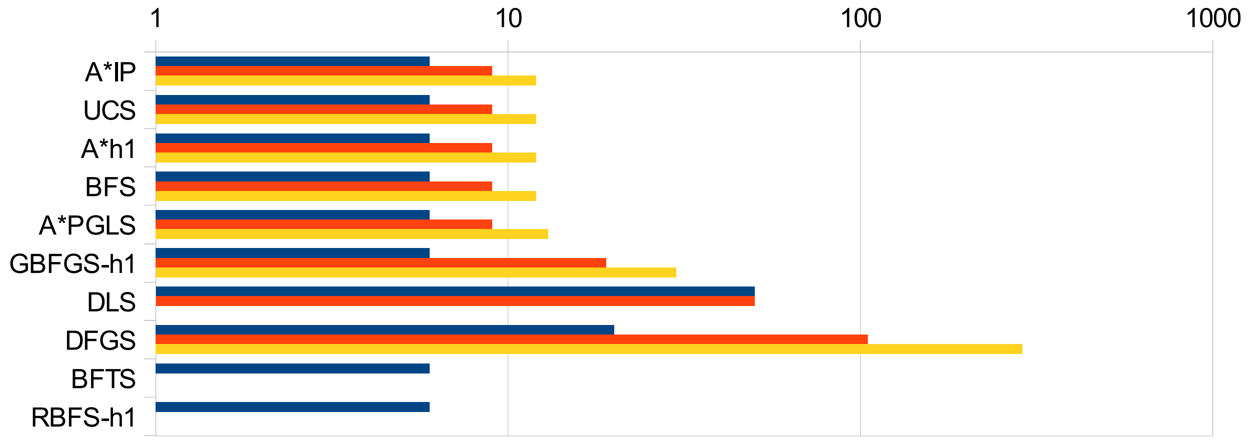

Figure 1: Planning Length for each Search Method for Problems 1, 2 and 3.

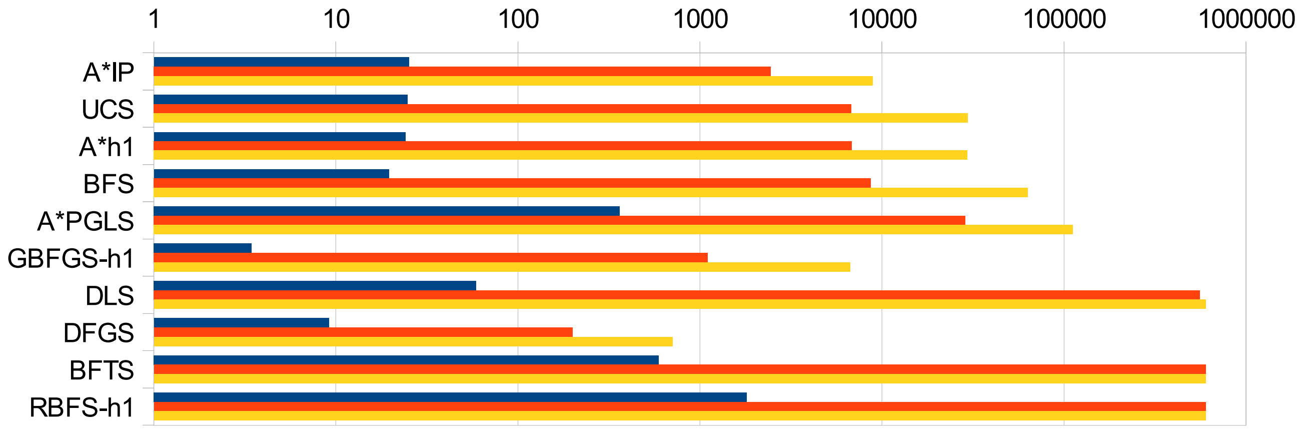

Figure 2: Planning time (ms) for each Search Method for Problems 1, 2 and 3.

Results and Metrics Discussion

For these problems, it's easy to consider the search time as a lower priority than the finding of the optimal path, due to the nature of the task involved. However, because it's also easy to conceive of the algorithms being used for very complex scenarios where the number of cargo, planes or airports expands, the search time is very important to consider as well. That being the case, the results of the algorithms have been sorted first by path length and then by time of execution. The results are shown in Table 1 and in Figures 1 and 2.

Table 1 shows the path length results, search time, and other metrics for each search algorithm and for each problem. Figures 1 and 2, on the next page, plot out the "Length" rows and "Time (ms)" rows. As mentioned, the data has been sorted first by plan length and then by execution time for the second problem.

The algorithm with the lowest overall time while still achieving an optimal solution was the A*-Ignore Preconditions algorithm. This is to be expected, due to the superiority of the "ignore preconditions" heuristic to the overly simple h1 heuristic, which causes wasted searching, and the overly complex "planning graph level sum" heuristic, which results in far less wasted searching but is extremely costly to calculate. This can be confirmed by examining the expansions for each heuristic (how much it searched) and the total execution time.

The second best algorithm was nearly a tie between essentially equivalent algorithms, with perhaps slightly different code causing a small difference in execution time. Those are the Uniform Cost Search and A*-h1 algorithms. These were both successful in finding the optimal solution, although the effect of the poorer heuristic is most evident in the number of expansions and total execution time. The advantage of having a very easy to calculate heuristic was outdone by the wasted expansion nodes.

Surprisingly, the A* (and equivalent) algorithms do not sweep the lead, but are interrupted by the Breadth-First Search algorithm, which is similar to a graph search, but with goal checking at node expansion vs exploration. As stated earlier, the A*-"Planning Graph Level Sum" algorithm has a very costly-to-calculate heuristic, which isn't admissible. This can be seen in the plan length returned for Problem 3, which is 13 vs the optimal value of 12. The breadth-first search, however, is optimal and is fairly quick due to the explored logic eliminating backtracking.

All remaining algorithms produce sub-optimal solutions or are unable to provide any solutions due to inefficiencies. Some run extremely fast, such as the Greedy Best Search First algorithm and the Depth-First Graph Search. Given the choice between the two of these, it appears that the Greedy algorithm produces a significantly shorter path, although after a longer search time for the more complex Problems 2 and 3. Given the poor heuristic "h1," the Greedy algorithm likely spends more time switching back and forth between possibly successful paths, while avoiding routes that move the state further from the desired goal, as the Depth-First search might do.

The depth-limited graph search is an interesting standout in that it produces paths of its maximum search limit (50), but after a very long time. Depth-searches are not guaranteed to be complete (or complete in a given time period), which is evident as the breadth of nodes expands in Problems 2 and 3. No solution is returned for Problem 3, due to a timeout of 10 minutes (note the fastest, optimal solution for Problem 3 was under 9 seconds).

Coming in last are the Breadth-First Tree Search and the Recursive Best-First Search with the "h1" heuristic. Both can produce optimal solutions, but are extremely inefficient. The breadth-first tree search allows for backtracking, which is unhelpful in this situation (or any where the step cost does not decrease with depth). It's able to find a solution for Problem 1, but is far too inefficient to finish in time for Problems 2 and 3. The Recursive Best-First Search is efficient only in memory and suffers from re-expanding visited nodes perhaps many times. This is especially a problem when using the overly simple heuristic of a constant value of 1. The result is a timeout for Problems 2 and 3. As can be seen in Table 1, the number of expansions are far higher than even the backtracking Breadth-First Tree Search for Problem 1.

Summary

In conclusion, the A* algorithm easily performed consistently the best for the more complex problems and nearly equal to the best in a simple problem, when given an admissible heuristic that was somewhat complex, but not too costly to calculate. Some algorithms, such as the Breadth-First Tree Search and Recursive Best First Search-h1 were hopelessly time-inefficient, due to their wasteful backtracking. The best solutions use a great heuristic and keep track of where they've been to find the optimal path the fastest.