Planning Search: Heuristic Analysis

Artificial Intelligente Nanodegree

Celso Araujo

December 16th, 2017

Introduction

This report is a requirement of the “Implement a Planning Search” project included in Udacity's Artificial Intelligente Nanodegree. The project description, this report and all code, including code for my solution, can be found on my GitHub repository: https://github.com/celsofaf/AIND-Planning.

The aim of the project is:

- To complete two unfinished Python scripts which aim to implement planning search algorithms based on propositional logic;
- To run 5 algorithms on 3 planning search problems and write this very report on their performances.

The planning search problems are instances of the classic Airport Cargo problem: cargoes, planes and airports (the entities of our problem) are provided, and one must move certain cargoes to certain airports using the planes according to certain rules and allowed actions.

In the end, an optimal sequence of actions is provided for each problem.

Algorithms

According to the project instructions, from the 5 algorithms to be tested, 3 of them are to be non-heuristic ones, and 2 of them are to be the A* search with two different heuristics. Two of the non-heuristic algorithms are already chosen on the project instructions, the third one being my choice and the two A*-based algorithms are also previously chosen. So, the algorithms to be tested here are:

- Breadth-first search
- Depth-first graph search
- Uniform cost search (this one is my choice)
- A* with ignore preconditions heuristic
- A* with plangraph levelsum heuristic

The aim of every algorithm is to provide an optimal plan for each problem.

Problem 1

This is a problem with 2 cargoes, 2 planes and 2 airports. The initial state has 4 positive sentences and 8 negative sentences. The goal has 2 sentences. So, by summing it all, I could say the total size of this problem is 20.

The table below shows the results found for this problem by each algorithm:

Method	Time (s)	Node expansions	Goal tests	New nodes	Plan length
Breadth-first search	0.04	43	56	180	6
Depth-first graph search	0.02	21	22	84	20
Uniform cost search	0.05	55	57	224	6
A* ignore preconditions	0.05	41	43	170	6
A* plangraph levelsum	1.41	55	57	224	6

All algorithms ran very quickly on my laptop, with all but one running on a fraction of a second. A* plangraph levelsum took considerably more time than the other ones, but still it ran within 2 seconds, which is close to nothing.

Depth-first graph search yields the highest plan length of 20, with all other algorithms providing plan lenghts of 6.

The “new nodes” column provides a rough estimate of the memory size used by the algorithm: the more new nodes were created, the more memory was used. So, to this regard, uniform cost search and A* plangraph levelsum were the least memory efficient algorithms, with depth-first graph search being the more memory efficient – even if it provided the biggest plan length.

Problem 2

This is a problem with 3 cargoes, 3 planes and 3 airports. The initial state has 6 positive sentences and 21 negative sentences. The goal has 3 sentences. So, by summing it all, I could say the total size of this problem is 39.

The table below shows the results found for this problem by each algorithm:

Method	Time (s)	Node expansions	Goal tests	New nodes	Plan length
Breadth-first search	19.4	3343	4609	30509	9
Depth-first graph search	5.00	624	625	5602	619
Uniform cost search	19.6	4852	4854	44030	9
A* ignore preconditions	6.15	1450	1452	13303	9
A* plangraph levelsum	976	4471	4473	40440	9

Again, all algorithms were rather quick, running in under 20 seconds, except for A* plangraph levelsum, which was more than 50 times slower than the second slower one.

And again, depth-first graph search provided the longest plan, an unbelievable plan of length 619, against length 9 from all the other ones.

Using new nodes as a measure of memory usage, again uniform cost search was the least efficient, with A* plangraph levelsum on a close second place. And, again, depth-first graph search was the most efficient to this regard, by a large margin.

Problem 3

This is a problem with 4 cargoes, 2 planes and 4 airports. The initial state has 6 positive sentences and 26 negative sentences. The goal has 4 sentences. So, by summing it all, I could say the total size of this problem is 46.

The table below shows the results found for this problem by each algorithm:

Method	Time (s)	Node expansions	Goal tests	New nodes	Plan length
Breadth-first search	162	14663	18098	129631	12

Depth-first graph search	2.55	408	409	3364	392
Uniform cost search	77.5	18235	18237	159716	12
A* ignore preconditions	24.3	5040	5042	44944	12
A* plangraph levelsum	5860	12626	12628	111961	12

Here we have a similar situation as we had on the other problems: A* plangraph levelsum algorithm taking much more time than the other ones – but now breadth-first search also starts to dettach a little -; depth-first graph search providing the longest plan by a large margin; and uniform cost search being the least efficient memory-wise, but now with breadth-first search on a close second place, A* plangraph levelsum on third and, by a large margin, depth-first search being the most efficient.

Comments and conclusion

All algorithms were successful on providing a feasible plan to reach the proposed goals for each problem. But it doesn't mean they were equally succesful:

- Depth-first graph search was always the quickest algorithm and the one with the least number of node expansions. So, it is obviously a strong contender: very fast and very memory efficient. But it comes with a big “but”: it always yields the longest plan, and always by a large margin. So, it's not optimal. For example: for Problem 2, all cargoes can be delivered to their right destinations with only 9 actions, if you choose any other algorithm; but, using depth-first search, you will need to take more than 600 actions. No, no, no. I won't chose this one to solve a real-life problem. The cost of the solution may be outrageous!
- A* ignore preconditions always ran on a reasonable time (always raking in 2nd by a fair margin, except perhaps for Problem 1, but it can be considered as a technical tie there) and is also quite memory-efficient, ranking in 2nd on every problem. It also provided an optimal plan, when you check into the plan lengths, for each problem. So, this algorithm would be my choice to solve a real life problem.
- On a fictitious world where I had never met A* ignore preconditions, both breadth-first search and uniform cost search could be viable contenders. Looking over Problem 3, they show an interesting tradeoff between speed and memory: breadth-first search uses less memory, but is slower. Both provided optimal solutions for our plans.
- A* plangraph levelsum can be safely ignored, as it is just too slow for a general pourpose algorithm. Maybe it could be efficient on some specific kind of problem? I don't know.

Optimal plans for each problem

For each problem, the plan provided by the A* ignore preconditions algorithm – the one I chose as “best” – will be shown here.

Problem 1	Problem 2		Problem 3	
Fly(P1, SFO, JFK) Unload(C1, P1, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)	Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Load(C1, P1, SFO) Fly(P1, SFO, JFK)	Unload(C1, P1, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C2, P2, SFO)	Load(C1, P1, SFO) Load(C2, P2, JFK) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P2, JFK, ORD) Load(C4, P2, ORD)	Fly(P2, ORD, SFO) Fly(P1, ATL, JFK) Unload(C4, P2, SFO) Unload(C3, P1, JFK) Unload(C1, P1, JFK) Unload(C2, P2, SFO)