Analysis

Planning Search

Andreas Mayer

20th February, 2017

Introduction

I have done several tests with all the provided algorithms. The results are summarized in the following table.

	Expansions			Goal Tests			New Nodes			Plan length			Time		
Algorithm	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
breadth_first_search	43	3346	14120	56	4612	17673	180	30534	124926	6	9	12	<1s	1.8m	8.6m
breadth_first_stree_search	1458			1459			5960			6			7s	>10m	>10m
depth_first_graph_search	21	107	292	22	108	293	84	959	2388	20	105	288	<1s	<3s	<8s
depth_first_limited_search	101			271			414			50			<1s	>10m	>10m
uniform_cost_search	55	4853		57	4855		224	44041		6	9		<1s	2.8m	>10m
recursive_best_first_search	4229			4230			17023			6			22.6s	>10m	>10m
greedy_best_first_graph_search	7	998		9	1000		28	8982		6	21		<1s	31.2s	>10m
astar_h_1	55	4853		57	4855		224	44041		6	9		<1s	2.8m	>10m
astar_h_ignore_preconditions	41	1506	5118	43	1508	5120	170	13820	45650	6	9	12	<1s	52s	4.25m
astar_h_pg_levelsum	11	86	414	13	88	416	50	841	3818	6	9	12	<1s	50s	5.2m

The **grey fields** indicate that no result was achieved, due to a runtime of more than 10 minutes.

The **green fields** indicate that this is the best value that was achieved compared to all other algorithms. If two values are very close to each other (or equal) it is possible that more than one fields are green in a column.

The **red fields** indicate that this is the worst value that was achieved compared to all other algorithms. If two values are very close to each other (or equal) it is possible that more than one fields are red in a column.

To control the runtime I used the following command (Mac OS X):

```
$ gtimeout 10m python run_search.py [options]
```

To get this command, one has to install coreutils via brew:

```
$ brew install coreutils
```

For all my experiments I use a MacBook Pro (Early 2011) with a 2,3 GHz Intel Core i5 and 16 GB 1333 MHz DDR3 RAM. It is therefore possible that solutions could have been found within 10 minutes for problems where I was not able to find a solution, when a faster computer is used. For the analysis my results are however meaningful enough.

Non-heuristic search analysis

Expansions			Goal Tests			New Nodes			Plan length			Time		
P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3	P1	P2	P3
43	3346	14120	56	4612	17673	180	30534	124926	6	9	12	<1s	1.8m	8.6m
1458			1459			5960			6			7s	>10m	>10m
21	107	292	22	108	293	84	959	2388	20	105	288	<1s	<3s	<8s
101			271			414			50			<1s	>10m	>10m
55	4853		57	4855		224	44041		6	9		<1s	2.8m	>10m
4229			4230			17023			6			22.6s	>10m	>10m
7	998		9	1000		28	8982		6	21		<1s	31.2s	>10m

The following table provides an analysis of the numbers in the screenshot. Row 1 correlates to the first row that holds values, Row 2 to the second, etc.

Row	Search Algorithm	Summary
1	breadth_first	- The **optimal plan** was always found. - BFS was able to find a solution to all three problems within the time-limit of 10 minutes. - It has high numbers for Expansions, Goal Tests and New Nodes. - BFS was the only non-heuristic method that found optimal plans for all three problems within the given time-limit of 10 minutes.
2	breadth_first_stree	- It was **only able to solve P1**. - The runtime was significantly higher for P1 than for most of the other algorithms. - As the Expansions, Goal Tests and New Nodes values are much higher than in most of the other algorithms, I assume that this behavior can be extrapolated. When the problem gets more complex, the numbers would rise significantly as well. This is also true for the runtime. The fact that P2 and P3 could not be solved within 10 minutes indicates that this assumption might be valid.
3	depth_first_graph	- It was able to find solutions for all three problems. - **No optimal plan could be found** for any of the problems. - The Expansions, Goal Tests and New Nodes were small compared to the other algorithms. - The **runtime is the best** among all algorithms used. - When an evaluation should be performed to check if a solution exists, but the optimal solution is not needed, this algorithm would be my choice due to its superior runtime.

4	depth_first_limited	- It was **only able to solve P1**. - **No optimal solution could be found**. - Based on this analysis it makes no sense to use this algorithm for the given problems.
5	uniform_cost_search	- It was **able to solve** problems **P1 and P2**. - Problem **P3 could not be solved** within the specified 10 minutes. - It was able to find **optimal plans** for any problem. - The Expansions, Goal Tests and New Nodes values were among the highest of all algorithms.
6	recursive_best_first	- It was **only able to solve P1**. - An **optimal plan** was found for P1. - The runtime was the worst for problem P1. - No solutions were found or problems P2 and P3. - As the runtime is a few orders of magnitude above other possible algorithms, it makes no sense to use this algorithm for the given problems.
7	greedy_best_first_graph	- It was **able to solve** problems **P1 and P2**. - Problem **P3 could not be solved** within the specified 10 minutes. - **An optimal plan was found for P1.** - **No optimal plan was found for P2**. - Overall this algorithm performed best for problem P1 when Expansions, Goal Tests and New Nodes are taken into consideration..

Heuristic search analysis

uniform_cost_search	55	4853		57	4855		224	44041		6	9		<1s	2.8m	>10m
recursive_best_first_search	4229			4230			17023			6			22.6s	>10m	>10m
greedy_best_first_graph_search	7	998		9	1000		28	8982		6	21		<1s	31.2s	>10m
astar_h_1	55	4853		57	4855		224	44041		6	9		<1s	2.8m	>10m
astar_h_ignore_preconditions	41	1506	5118	43	1508	5120	170	13820	45650	6	9	12	<1s	52s	4.25m
astar_h_pg_levelsum	11	86	414	13	88	416	50	841	3818	6	9	12	<1s	50s	5.2m

Row	Search Algorithm	Summary
8	a_star_h_1	- The **optimal plan** was found for problems P1 and P2. - **P3 could not be solved**. - The results of this search are **equal to uniform_cost_search**, which is illustrated by the orange thick border in the table. This is clear as the h1 heuristic function always returns 1. Therefore the search is not directed towards the goal and every possible action is treated equally.
9	astar_h_ignore_preconditions	- The **optimal plan** was found for all problems. - As expected the search is now more directed and the number of Expansions, Goal Tests and New Nodes is significantly smaller compared to a_star_h_1.
10	astar_h_pg_levelsum	- The **optimal plan** was found for all problems. - The number of Expansions, Goal Tests and New Nodes is significantly smaller compared to astar_h_ignore_preconditions. - The runtime is a comparable to a_star_h_ignore_preconditions for problems P1 and P2. - For problem P3 the runtime is ~1 minute longer compared to a_star_h_ignore_preconditions.

Conclusion

When the goal is just to find a possible path, but not an optimal one, I would use **depth_first_graph search** as it is by far the fastest method for all of the given problem sets P1, P2 and P3.

When the goal is to find optimal solutions, the heuristic search approaches perform better than the non-heuristic ones as the heuristics direct the search into the *correct* direction. In terms of the overall performance, which also includes the number of Extensions, Goal Tests and New Nodes the **astar_h_pg_levelsum** algorithm performed best. It was able to find optimal solutions within the 10 minutes timeframe. However I had to implement *caching* and an optimization in the *__hash__* functions to achieve this runtimes.

Even though **astar_h_pg_levelsum** has the best overall stats, in this setting I would chose **astar_h_ignore_preconditions** as my methodology of choice for the following reasons:

- The implementation is super simple compared to the additional implementation done for astar_h_pg_levelsum.
- The performance is comparable to astar_h_pg_levelsum and even faster in the case of problem P3. So the heuristic seems to be good enough for solving the given problems.
- The algorithm was able to find paths of optimal length for all three problems.

Appendix A - Optimal plans

The following plans are optimal solutions for the three problem sets.

Optimal Plan - P1	Optimal Plan - P2	Optimal Plan - P3
Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Unload(C1, P1, JFK) Unload(C2, P2, SFO)	Load(C1, P1, SFO) Fly(P1, SFO, JFK) Load(C2, P2, JFK) Fly(P2, JFK, SFO) Load(C3, P3, ATL) Fly(P3, ATL, SFO) Unload(C3, P3, SFO) Unload(C2, P2, SFO) Unload(C1, P1, JFK)	Load(C2, P2, JFK) Fly(P2, JFK, ORD) Load(C4, P2, ORD) Fly(P2, ORD, SFO) Load(C1, P1, SFO) Fly(P1, SFO, ATL) Load(C3, P1, ATL) Fly(P1, ATL, JFK) Unload(C4, P2, SFO) Unload(C3, P1, JFK) Unload(C2, P2, SFO) Unload(C1, P1, JFK)