Analysis

Planning Search

Version 2

Andreas Mayer

1st March, 2017

Introduction

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

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.

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.

Results - Code Version 1

The following screenshot illustrates the results that were achieved with the first implementation.

	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

Results - Code Version 2

As my first submission was denied, and I got the info that more searches should terminate within the 10 minutes time limit, I profiled my code with **cProfile** and refactored the methods that were causing the biggest effect on the runtime. The main issue was the use of the **PropKB** class and its API. Now the results are much better and the coverage has also increased!

	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	0.03s	17s	2.2m
breadth_first_stree_search	1458			1459			5960			6			0.87s	>1h	-
depth_first_graph_search	21	107	292	22	108	293	84	959	2388	20	105	288	0.01s	0.37s	1.5s
depth_first_limited_search	101	213491		271	1967093		414	1967471		50	50		0.08s	9.4m	>3h
uniform_cost_search	55	4853	18223	57	4855	18225	224	44041	159618	6	9	12	0.05s	59s	8.7m
recursive_best_first_search	4229			4230			17023			6			2.7s	>1h	-
greedy_best_first_graph_search	7	998	5578	9	1000	5580	28	8982	49150	6	21	22	0.004s	9.3s	2.4m
astar_h_1	55	4853	18223	57	4855	18225	224	44041	159618	6	9	12	0.05s	59s	8.7m
astar_h_ignore_preconditions	41	1506	5118	43	1508	5120	170	13820	45650	6	9	12	0.03s	20s	2.1m
astar_h_pg_levelsum	11	86	414	13	88	416	50	841	3818	6	9	12	1s	53s	5.5m

Setup

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
```

Search analysis

The following sub-chapters provide an analysis of the different algorithms used. Every algorithm is described with a table that consolidates the values that were estimated during my evaluations. The numbers in this table state the actual value **x** and in parentheses *a factor* that states the increase compared to the prior problem.

	Problem 1	Problem 2	Problem 3
Expansions	**10**	**100** (10)	**10000** (100)

In this example the value of Problem 2 is 10 times higher than the value of Problem 1, and the value of Problem 3 is 100 times higher than the value of Problem 2.

Algorithms **1 - 7 are non-heuristic** search functions, algorithms **8 - 10 are heuristic** search functions.

1 - breadth_first (BFS)

- An **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.
- After the performance improvements within my code base, the **runtimes** were much better than before and they **are absolutely competitive** when compared to the other algorithms.

	Problem 1	Problem 2	Problem 3
Expansions	**43**	**3346** (77.81)	**14120** (4.22)
Goal Tests	**56**	**4612** (82.36)	**17673** (3.83)
New Nodes	**180**	**30534** (169.63)	**124926** (4.09)
Plan length	**6**	**9**	**12**
Time	**0.03s**	**17s** (567)	**2.2m** (7.76)

2 - breadth_first_stree_search

- It was **only able to solve P1**.
- The runtime was higher for P1 than for most of the other algorithms but still under a second.
- 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 could not be solved within one hour indicates that this assumption might be valid.
- P3 was not evaluated, as the runtime for P2 was already greater than one hour.

	Problem 1	Problem 2	Problem 3
Expansions	1458	-	-
Goal Tests	1459	-	-
New Nodes	5960	-	-
Plan length	6	-	-
Time	0.87s	>1h	Not evaluated

3 - depth_first_graph_search

- 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.
- The runtimes were so small, that the comparison was neglected.

	Problem 1	Problem 2	Problem 3
Expansions	21	107 (5.09)	229 (2.14)
Goal Tests	22	108 (4.91)	293 (2.71)
New Nodes	84	959 (11.42)	2388 (2.49)
Plan length	20	105	288
Time	0.01	0.37 (37)	1.5s (4.05)

4 - depth_first_limited_search

- It was **able to solve P1 and P2**.
- No optimal solution was found.
- Based on this analysis it makes no sense to use this algorithm for the given problems, as the runtime is much higher than other algorithms and it is unable to find a feasible solution.
- The numbers for Expansions, Goal Tests and New Nodes are the highest among all algorithms, especially for P2.

	Problem 1	Problem 2	Problem 3
Expansions	**101**	**213491** (2114)	**-**
Goal Tests	**271**	**1967093** (7259)	**-**
New Nodes	**414**	**1967471** (4752)	**-**
Plan length	**50**	**50**	**-**
Time	**0.08s**	**9.4m** (7050)	**>3h**

5 - uniform_cost_search

- It was **able to solve all problems**.
- It was able to find **optimal plans** for all three problems.
- The Expansions, Goal Tests and New Nodes values were among the highest of all algorithms.
- As other solutions exist that provide results much faster, I wouldn't consider using this method.
- As every step is assumed uniformly (equal) this search method provides no direction and can be understood as a kind of random-walk.

	Problem 1	Problem 2	Problem 3
Expansions	**55**	**4853** ()	**18223** ()
Goal Tests	**57**	**4855** ()	**18225** ()
New Nodes	**224**	**44041** ()	**159618** ()
Plan length	**6**	**9**	**10**
Time	**0.05s**	**59s** (1180)	**8.7m** (8.85)

6 - recursive_best_first_search

- It was **only able to solve P1**.
- An **optimal plan** was found for P1.
- The runtime was the worst for problem P1 and the numbers of Expansions, Goal Tests and New Nodes were already much higher than any other result for 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.

	Problem 1	Problem 2	Problem 3
Expansions	4229	-	-
Goal Tests	4230	-	-
New Nodes	17023	-	-
Plan length	6	-	-
Time	2.7s	>1h	Not evaluated

7 - greedy_best_first_graph_search

- It was **able to solve** all problems.
- **An optimal plan was found only for P1.**
- This algorithm performed best (runtime) for problem P1.

	Problem 1	Problem 2	Problem 3
Expansions	7	998 (143)	5578 (5.59)
Goal Tests	9	1000 (111)	5580 (5.58)
New Nodes	28	8982 (321)	49150 (5.47)
Plan length	6	21	22
Time	0.004s	9.3s (2325)	2.4m (15.48)

8 - a_star_h_1

- It was **able to solve all problems**.
- It was able to find **optimal plans** for all three problems.
- The results of this search are **equal to uniform_cost_search**. The reason for this is that the h1 heuristic function always returns 1. Therefore the search is not directed towards the goal and every possible action is treated equally.

	Problem 1	Problem 2	Problem 3
Expansions	**55**	**4853** ()	**18223** ()
Goal Tests	**57**	**4855** ()	**18225** ()
New Nodes	**224**	**44041** ()	**159618** ()
Plan length	**6**	**9**	**12**
Time	**0.05s**	**59s** (1180)	**8.7m** (8.85)

9 - a_star_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.

	Problem 1	Problem 2	Problem 3
Expansions	**41**	**1506** (37)	**5118** (3.4)
Goal Tests	**43**	**1508** (35)	**5120** (3.4)
New Nodes	**170**	**13820** (81.3)	**45650** (3.3)
Plan length	**6**	**9**	**12**
Time	**0.03s**	**20s** (667)	**2.1m** (6.3)

10 - a_star_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 the worst for P1, compared to the other heuristic searches. This is due to the overhead of the solution. As the problem is an easy one to solve, the creation of the planning graph takes longer than actually solving the problem.
- The runtime for P2 is a little bit better than the runtime of a_star_h_1. It seems that the point was reached where the tradeoff starts to pay off.
- The runtime for P3 is still 2.6 times higher than the runtime of the runtime of astar_h_ignore_preconditions, but it is 1.6 times faster than a_star_h_1.

	Problem 1	Problem 2	Problem 3
Expansions	**11**	**86** (37)	**414** (3.4)
Goal Tests	**13**	**88** (35)	**416** (3.4)
New Nodes	**50**	**841** (81.3)	**3818** (3.3)
Plan length	**6**	**9**	**12**
Time	**1s**	**53s** (53)	**5.5m** (6.23)

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 to all problems within the 10 minutes timeframe. However I had to implement *caching,* an optimization in the *__hash__* functions, and I had to refactor my code 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)