Analysis of planning searches

In this project three air cargo problems where given to be solved by different approaches. These problems where defined as follows:

Problem 1:

```
Init(At(C1, SFO) ∧ At(C2, JFK)
    ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2)
    ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

Problem 2:

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
    ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
    ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
    ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

Problem 3:

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
    ∧ At(P1, SFO) ∧ At(P2, JFK)
    ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
    ∧ Plane(P1) ∧ Plane(P2)
    ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Planning Problems

For each of these problems different approaches where used to find a solution for the problem. In the following I will list an optimal solution for each problem and give an overview about the results of running the breadth_first_search, depth_first_graph_search and uniform_cost_search algorithms.

Solution for Problem 1:

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

Solution for Problem 2:

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

Solution for Problem 3:

```
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(C2, P2, SFO)
Unload(C1, P1, JFK)
```

For the first Problem the following results were achieved by using the algorithms above:

P#	**S#**	**Search**	**Expansions**	**Goal Tests**	**New Nodes**	**Plan length**	**Time elapsed**
1	1	breadth_first_search	43	56	180	6	0,03173
1	3	depth_first_graph_search	21	22	84	20	0,018
1	5	uniform_cost_search	55	57	224	6	0,04062

Table 1: Results for problem 1

As the results show breath_first_search does best for problem one finding an optimal solution witch length 6 followed by uniform_cost_search also finding an optimal solution with length 6 and depth_first_graph_search finding a solution which is not optimal with length 20.

From the perspective of needed expansions depth_first_graph_search did best with just 21 expansions. Both other algorithms where close together with 43 for breadth_first_search and 55 for uniform_cost_search. The same shows up when comparing the performed goal tests and the value of new nodes. Deepth_first_graph_search only needed 22 goal tests and 84 new nodes.

Looking at the time needed to find a solution the depth_first_graph_search was the fastest in finding a solution. It takes about half the time the other two needed to find a solution. The solution was not optimal, but it was way faster.

P#	S#	Search	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed
2	1	breadth_first_search	3343	4609	30509	9	16,90364
2	3	depth_first_graph_search	624	625	5602	619	4,24346
2	5	uniform_cost_search	4853	4855	44041	9	14,38106

Table 2: Results for problem 2

For problem 2 the results point out a clearer picture from the observations of problem 1. Again breadth_first_search and uniform_cost_search found an optimal solution with a length of 9. In this case the depth_first_graph_search found a valid solution but with a huge length of 619 with cannot be acceptable from the point of view of the cargo problem.

Depth_first_graph_search with an amount of 624 needed only a few expansions compared to the other searches which needed 3324 (breadth_first_search) and 4853 (uniform_cost_search). As before we see a similar picture when looking at goal tests and new nodes values where depth_first_graph_search takes 625 and 5602 compared to 4609/30509 (breadt_first_search) and 4855/44041 (uniform_cost_search).

For this more complex problem the time needed shows, that depth_first_graph_search was the fastest with 4,2 seconds. Uniform_cost_search took 14,4 and breadth_first_search 16,9 seconds. This time the searches finding an optimal solution needed 3,5 to 4 times the time needed to find a non-optimal solution.

P#	S#	Search	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed
3	1	breadth_first_search	14663	18098	129631	12	125,2553
3	3	depth_first_graph_search	408	409	3364	392	2,02535
3	5	uniform_cost_search	18234	18236	159707	12	65,12683

Table 3: Results for problem 3

For problem 3 breadth_first_search and uniform_cost_search found a solution with the length of 12 which is optimal. Depth_first_graph_search found a solution with the length of 392. It is interesting to see, that problem 3 seems to be more complex than problem 2 but depth_first_graph_search found a solution which has a length of 2/3 of the solution found for problem 2.

Breadth_first_search needed 14663 expansions, 18098 goal tests and 129631 new nodes. Uniform_cost_search had values of 18234 expansions, 18236 goal tests and 159707 new nodes. In comparison to these high values depth_first_graph_search needed 408 expansions, 409 goal tests and 3364 new nodes.

The time needed to find their solutions shows that depth_first_graph_search was very fast with just 2 seconds. Second fastest was uniform_cost_search with 65 seconds and last breadth_first_search with 125 seconds. In this case there is a big difference between the two optimal solutions with a factor of 2.

Planning problems conclusion

The results from the runs for the three problems shown above can be justified with a view on some specifics of the used algorithms. We observed, that the depth first search always was the fastest in finding a solution but had always a non-optimal solution. Solutions of depth search are not always optimal because the algorithm goes down a tree as deep as it can before it moves forward at the same level. So because of this it Is possible to find a solution deep in a tree that is not optimal and terminate with it. This is also the reason, why it terminates faster in my comparison. Depth first needs time of O(b^m) where b is the maximum branching factor and m the maximum state space.

Breadth first search and uniform cost search always found a solution that was optimal. This is a characteristic of both algorithms. For breadth search this is valid if step costs are all identical. From the observations we saw that breadth first search was performing a bit better when looking at expansions and new nodes needed. This is because it needs O(b^d) where b is the maximum branching factor and d the depth of the least-cost solution. Compared to this uniform cost search needs O($b^{1+C^*/\varepsilon}$) where C* are the true cost to the optimal goal and ε is the minimum positive step cost. Our observations between both algorithms is explained by the fact that $d \leq 1 + C^*/\varepsilon$.

Domain-independent heuristics

In this part I will compare the results of the A* search using different heuristics. First heuristic is not a real heuristic at all as it always returns 1 it is A* search without a heuristic. Second used heuristic is ignore preconditions and the last one the lg levelsum heuristic.

P#	S#	Search	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed
1	8	astar_search h_1	55	57	224	6	0,03861
1	9	astar_search h_ignore_preconditions	41	43	170	6	0,03952
1	10	astar_search h_pg_levelsum	11	13	50	6	0,94276

Table 4: A serach results for problem 1*

For the first problem we see that the pg levelsum heuristic was best at expansions, goal test and new nodes but was worst in needed time for finding a solution. In this the ignore preconditions and no heuristic where on a same level but ignore preconditions did better at expansions and new nodes.

P#	S#	Search	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed
2	8	astar_search h_1	4853	4855	44041	9	13,57514
2	9	astar_search h_ignore_preconditions	1450	1452	13303	9	4,26119
2	10	astar_search h_pg_levelsum	86	88	841	9	185,98333

Table 5: A serach results for problem 2*

The second problem again shows the observations from problem one very clearly. No heuristic needs a lot of expansions and new nodes to find a solution. In a more complex scenario it is three times slower than the fastest heuristic. The pg levelsum needs the fewest expansions and new nodes but more than 40 times the time of the fastest one. The ignore preconditions heuristic is very fast in this problem. It is comparable to the depth first search but with a guaranteed optimal solution. Amount of expansions and new nodes is between the others.

P#	**S#**	**Search**	**Expansions**	**Goal Tests**	**New Nodes**	**Plan length**	**Time elapsed**
3	8	astar_search h_1	18234	18236	159707	12	64,1221
3	9	astar_search h_ignore_preconditions	5040	5042	44944	12	19,42567
3	10	astar_search h_pg_levelsum	315	317	2902	12	1263,198

Table 6: A serach results for problem 3*

Problem 3 shows again that no heuristic needs more than 3 times the amount of expansions, goal tests, new nodes and time in comparison to the ignore preconditions heuristic which again is placed in the middle of the heuristics. The pg levelsum heuristic again has very low values in expansions, goal tests and new nodes but takes 1263 seconds to find a solution.

Domain-independent heuristics conclusion

The observations show that a* search does a good job with no real heuristic, but it tends to have a very high need of memory. The ignore preconditions heuristic shows up to be very fast and seems to need acceptable memory sizes. The pg levelsum heuristic should be very good with memory consumption but gets very slow in time consumption if the problem gets more complex.