Report Planning search Project

Overall the following algorithm are implemented:

1. breadth_first_search
2. breadth_first_tree_search
3. depth_first_graph_search
4. depth_limited_search
5. uniform_cost_search
6. recursive_best_first_search h_1
7. greedy_best_first_graph_search h_1
8. astar_search h_1
9. astar_search h_ignore_preconditions
10. astar_search h_pg_levelsum

1. Introduction: solution to the problems

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

The solutions are not exact as the order may differs than the ones above. Here only the length of the paths are optimized. In real conditions, for problem one, it is obvious that you would rather perform the load, fly and unload in parallel rather than waiting for plan 1 to arrive to start loading cargo 2 on plane 2. Therefore, in the following solution of length 6, 9 and 12 for problems 1, 2, 3 respectively are said optimal.

2. non heuristic planning methods

In this part, we test the performance of the non-heuristic algorithm algorithms implemented to solve 3 planning problems going. For the sake of completeness, all raw results of the testing

are provided in the annex in form of tables. We restrict this part to the first 4 algorithms which are generic search algorithm the do not make use of potential additional information from the graph. All of them are either breath first or depth first algorithms.

Table 1: time and node expended for 1,2,3 and 4

time(sec)				Expension node			
search/prob	1	2	3	search/prob	1	2	3
1	0.037	16.18	137.5	1	43	3401	14491
2	1.04	>1000	>>1000	2	1458		
3	0.0087	1.78	60.1	3	12	350	3491
4	0.10	1372.0	>>1000	4	101	254020	

In terms of ability to find a solution, depth first search is better because by property is goes to a solution and returns it. Problem is that this solution is likely not to be optimal.
The other interesting information here is that by only Breadth first search and breadth first search graph return optimal actually return.
In term of nodes expended, the size of the problem quickly becomes not manageable for these algorithms.
The fact that these algorithms are quickly inefficient in terms of space of time is not surprising. These search algorithms have exponential complexity in time. Usually these are $O(b^x)$ where x can be the depth, the branching factor of the parameter of the limited search depending on the algorithms.

Then we can enrich this analysis by including 3 more algorithms: uniform cost search, recursive best first search h1 and greedy best first search h1.
Table 2: time and node expended for 5,6 and 7

time(sec)				expension			
search/prob	1	2	3	search/prob2	1	2	3
5	0.044	13.5	56	5	55	4761	17783
6	3.12	>1000	>>1000	6	4229		
7	0.005	1.51	13	7	7	550	4031

All these algorithms return optimal solutions (when they finish) except for the greedy search for problem 3 where the number of steps required is 22 instead of 12.
Overall in term of time of finding a solution, greedy search is a good alternative but if the optimal solution matters, then the uniform search does a great job at finding solution under a minute and making sure this solution is optimal if the branching factor is finite and cost are positive.

3. A* search

This last part is about the 3 different A* search algorithm implemented: A* search h1, A* search removing precondition, A* search with heuristic Levelsum.
All A* rely on the optimization of g(n) + h(n) where g(n) is the cost of the path realized and h(n) an heuristic of the cost of the cheapest path to a goal states. h_1 means that the heuristic used is the constant 1 this is equivalent to the uniform search algorithm. The second

heuristic will relax the precondition and determine the minimum number of actions to satisfy the goals. Finally the last one represents the problem as a planning graph and then determine the number of actions to achieve the goals.

Table 3:Time and nodes expended for 8, 9 and 10

time(sec)				expension			
search/prob	1	2	3	search/prob2	13	24	35
8	0.45	13.5	60	8	55	4761	17783
9	0.45	4.88	20	9	41	1450	5003
10	2.1	196		10	11	86	

The 3 variants are optimal by construction. However, the time results vary greatly. The 2 first have very simple heuristic and therefore are executed quickly. Actually, the results also show that the algorithm 9 has a better heuristic than algorithm 8 because it expends less nodes and is quicker. The fact that it expends less nodes tells us that the heuristic predicts better the value of the node. It takes longer to compute per node but the overall result is a better efficiency. The last algorithm has the most complex heuristic with multiple for-loop embedded. This leads to a very good heuristic (ie low number of nodes searched, good predictive power of the value of a node) but the time to evaluate the heuristic function is too long.

For P1, the ratio time/expansion give us an approximation of the time of one heuristic function evaluation. The ratio explodes for the algorithm 10 because the heuristic does a lot of embedded loops. For the 2 others, the ratio decreases between P1 and P2. This is most likely due to the fact that fixed computational time are a lower proportional time than for P1.

Table 4: Ratio Time/ number of nodes expanded.

search/prob	1	2
8	0,0082	0,0028
9	0,0110	0,0034
10	0,1909	2,2791

The first and second A* searches are also more effective (or equal) than the 7 searches above. They are optimal, this was not the case of the greedy search. Algo 9 is also, as mentioned, above better than uniform search. This makes it the best algorithm.

ANNEX

expansion

search/prob	1	2	3
1	43	3401	14491
2	1458		
3	12	350	3491
4	101	254020	
5	55	4761	17783
6	4229		
7	7	550	4031
8	55	4761	17783
9	41	1450	5003
10	11	86	

goal

search/prob	1	2	3
1	56	4671	17947
2	1459		
3	13	351	3492
4	271	2344879	
5	57	4763	17785
6	4230		
7	9	552	4033
8	57	4763	17785
9	43	1452	5005
10	13	88	

new nodes

search/prob	1	2	3
1	180	31049	128184
2	5960		
3	48	3142	29322
4	414	2345254	
5	227	43206	155920
6	17029		
7	28	4950	35794
8	224	43206	155920
9	170	13303	44586
10	50	841	

time(sec)

search/prob	1	2	3
1	0.037	16.18	137
2	1.04	>1000	>>1000
3	0.0087	1.78	60
4	0.10	1372	>>1000
5	0.044	13.5	56
6	3.12	>1000	>>1000
7	0.005	1.51	13
8	0.45	13.5	60
9	0.45	4.88	20
10	1.01	196	

length in step of the solution

search/prob	1	2	3
1	6	9	12
2	6		
3	12	366	3335
4	50	50	
5	6	9	12
6	6		
7	6	9	22
8	6	9	12
9	6	9	12
10	6	9	

1. breadth_first_search
2. breadth_first_tree_search
3. depth_first_graph_search
4. depth_limited_search
5. uniform_cost_search
6. recursive_best_first_search h_1
7. greedy_best_first_graph_search h_1
8. astar_search h_1
9. astar_search h_ignore_preconditions
10. astar_search h_pg_levelsum