Performance Comparison

The algorithms used to solve the three problems proposed were breadth_first_search, depth_first_graph_search, uniform_cost_search, astar_search h_1, astar_search, h_ignore_preconditions, and astar_search h_pg_levelsum.

Uninformed Planning Algorithms

The tables below presents the results of each one of the three uninformed planning algorithms used to solve the problems.

Problem 1			
Algorithm	**breadth_first_search**	**depth_first_graph_search**	**uniform_cost_search**
Expansions	43	21	55
Goal Tests	56	22	57
New Nodes	180	84	224
Path length	6	20	6
Time [s]	0.057	0.025	0.064

Problem 2			
Algorithm	**breadth_first_search**	**depth_first_graph_search**	**uniform_cost_search**
Expansions	3,343	624	4,852
Goal Tests	4,609	625	4,854
New Nodes	30,509	5,602	44,030
Path length	9	619	9
Time [s]	13.488	5.413	22.891

Problem 3			
Algorithm	**breadth_first_search**	**depth_first_graph_search**	**uniform_cost_search**
Expansions	14,663	408	18,235
Goal Tests	18,098	409	18,237
New Nodes	12,9631	3,364	159,716
Path length	12	392	12
Time [s]	94.569	3.744	109.774

It is possible to observe that, despite being the fastest algorithm to find the solution, the depth_first_graph_search found a path worse than the others did. The depth first search expands each branch of a tree until the deepest node. If this node is not a solution, then the algorithm expands the next branch. Otherwise, if the last node is a solution, even if it is the worse solution, the algorithm stops. One should be careful using the depth first search, because if there is no solution available in one of the branches it searches, it will infinitely expands the tree (it is not an optimal algorithm).

The uniform_cost_search was the most expensive algorithm; it found the same path length of breadth_first_search. The uniform cost search is an optimal algorithm; however, it searches for the cheapest path, so it expands more nodes in order to find the cheapest one, consuming more time and computational resources.

Observing the path length and the time elapsed to reach the goals, we can consider the breadth_first_search as being the best algorithm to solve the problems. This is an optimal algorithm and it searches for the shortest path, it does not take the cost of the path into account.

Automatic Heuristics A*

The tables below presents the results of each one of the three automatic heuristics A* used to solve the three problems.

Problem 1			
	A*_search h_1	**A*_search h_ignore_preconditions**	**A*_search h_pg_levelsum**
Expansions	55	41	11
Goal Tests	57	43	13
New Nodes	224	170	50
Path length	6	6	6
Time [s]	0.057	0.057	1.666

Problem 2			
	A*_search h_1	**A*_search h_ignore_preconditions**	**A*_search h_pg_levelsum**
Expansions	4,852	1,450	86
Goal Tests	4,854	1,452	88
New Nodes	44,030	13,303	841
Path length	9	9	9
Time [s]	16.951	6.209	309.195

Problem 3			
	A*_search h_1	**A*_search h_ignore_preconditions**	**A*_search h_pg_levelsum**
Expansions	18,235	5,040	317
Goal Tests	18,237	5,042	319
New Nodes	159,716	44,944	2,925
Path length	12	12	12
Time [s]	75.115	24.667	1,963.750

It is possible to observe that all the A* search algorithms found the same path length, that is the same as the shortest path found by the uninformed planning algorithms. The search using the heuristics h_1, that is not a true heuristic, has the highest amount of expansions, goal tests and new nodes. The h_pg_levelsum has the lowest amount of expansions, goal tests and new nodes, however it requires a long time to run. We can see that the h_ignore_preconditions is the best solution, because it is faster and it consumes fewer computational resources.

Uninformed Planning Algorithm vs Automatic Heuristics

Comparing the best-uninformed planning algorithm with the best automatic heuristics, we have the results bellow:

Problem 1		
Heuristics	**breadth_first_search**	**A*_search h_ignore_preconditions**
Expansions	43	41
Goal Tests	56	43
New Nodes	180	170
Path length	6	6
Time [s]	0.057	0.057

Problem 2		
Heuristics	**breadth_first_search**	**A*_search h_ignore_preconditions**
Expansions	3,343	1,450
Goal Tests	4,609	1,452
New Nodes	30,509	13,303
Path length	9	9
Time [s]	13.488	6.209

Problem 3		
Heuristics	**breadth_first_search**	**A*_search h_ignore_preconditions**
Expansions	14,663	5,040
Goal Tests	18,098	5,042
New Nodes	129,631	44,944
Path length	12	12
Time [s]	94.569	24.667

We can observe that the A* search is faster than the breadth first. The reason for that is because A* combines two algorithms: greedy search, and uniform cost search. It estimates the distance to the goal and, with this information, it is possible to find the lowest cost path, while expanding the minimum number of nodes possible. From the results, we can state that the more complex the problem, the faster it is to find the solution using A*.

Optimal plans

It was possible to find the following optimal sequence of actions.

Problem 1:

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

Problem 2:

Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)

Problem 3:

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