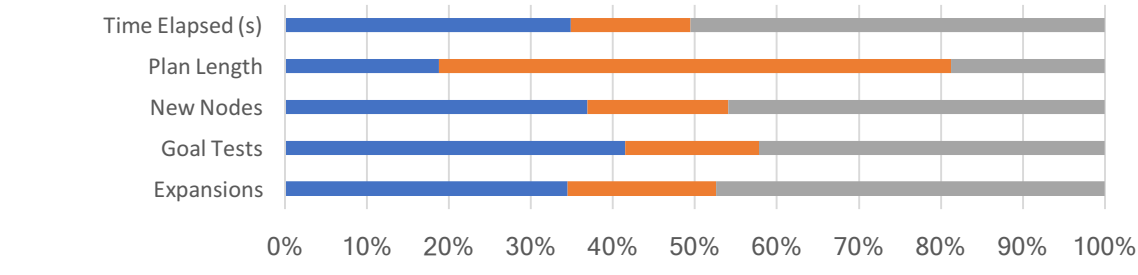
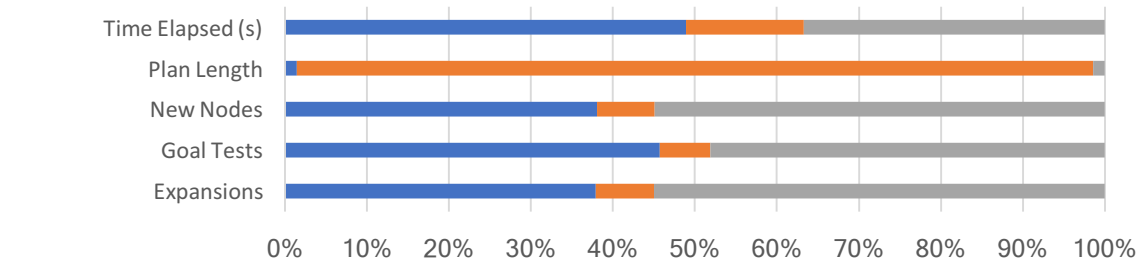
Heuristic Analysis for Planning Searches

Part 1 – Results of Uninformed Planning Searches



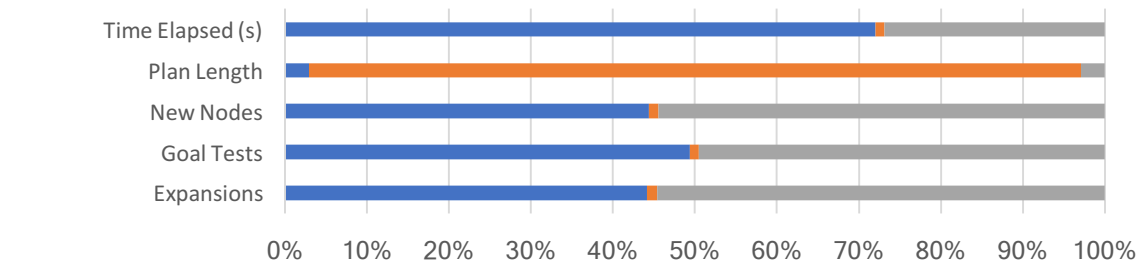
	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
breadth_first_search	40	56	180	6	0.0234
depth_first_graph_search	21	22	84	20	0.0098
uniform_cost_search	55	57	224	6	0.0339

Figure 1 - Comparison of uninformed searches against Problem-1



	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
breadth_first_search	3343	4609	30509	9	10.7725
depth_first_graph_search	624	625	5602	619	3.1698
uniform_cost_search	4853	4855	44041	9	8.0829

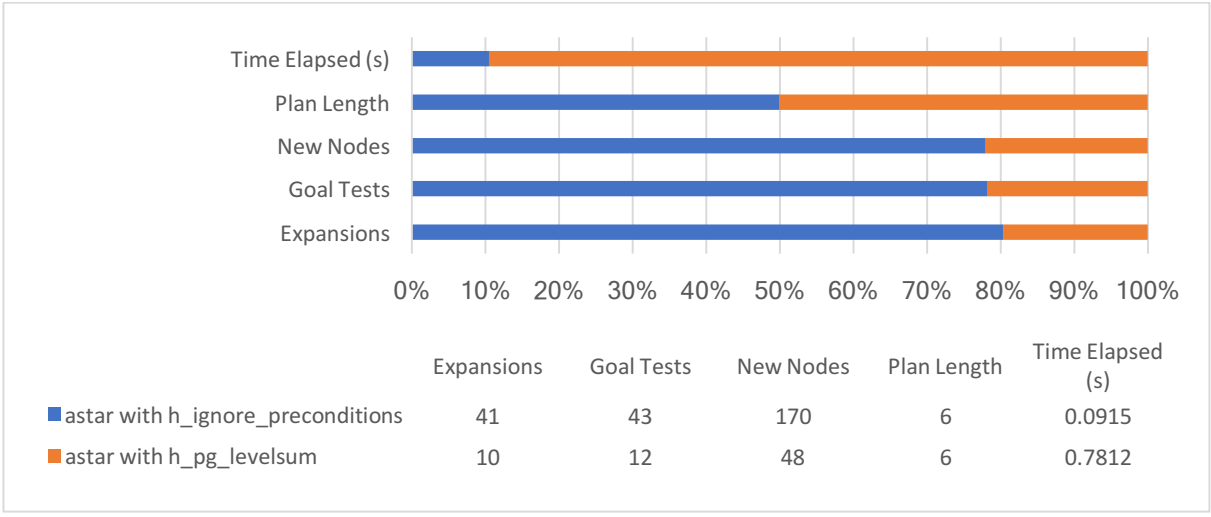
Figure 2 - Comparison of uninformed searches against Problem-2



	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
breadth_first_search	14663	18098	129631	12	90.786
depth_first_graph_search	408	409	3364	392	1.3755
uniform_cost_search	18151	18153	159038	12	33.8472

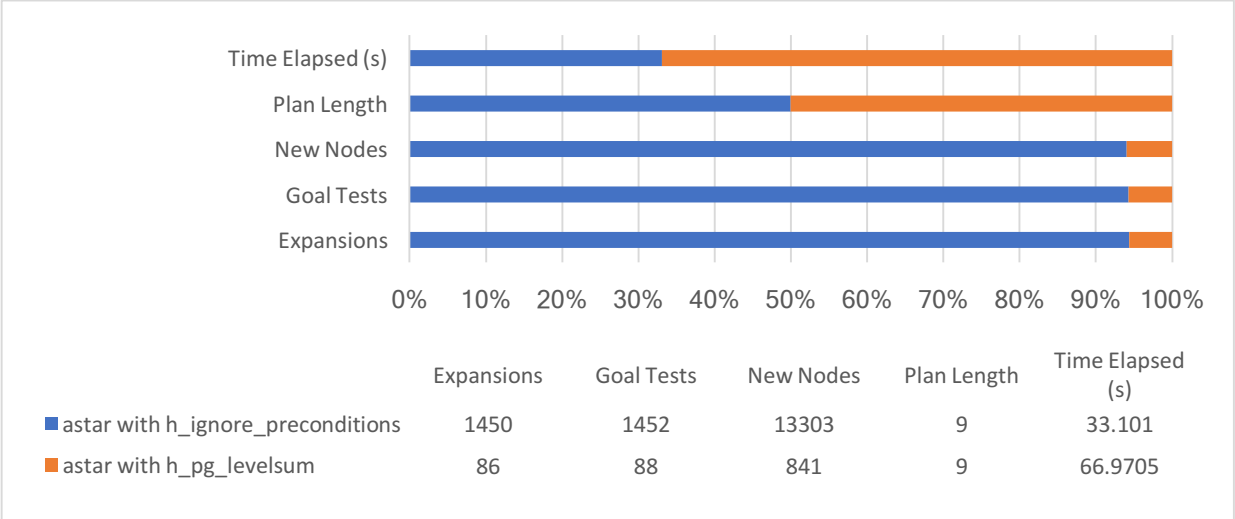
Figure 3 - Comparison of uninformed searches against Problem-3

Part 2 – Results of Searches Using Domain Independent Heuristics



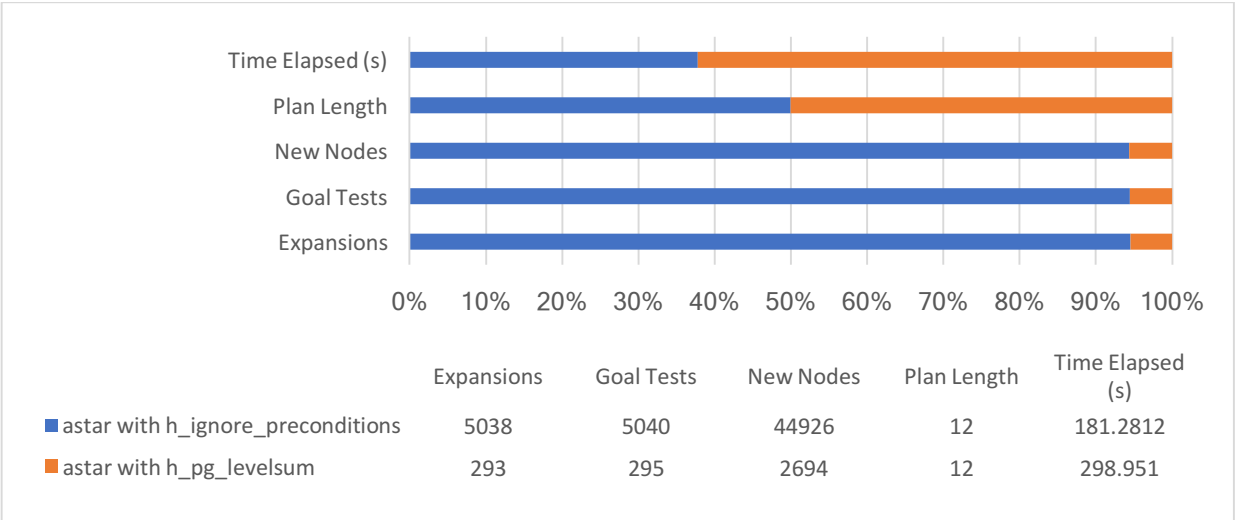
	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
astar with h_ignore_preconditions	41	43	170	6	0.0915
astar with h_pg_levelsum	10	12	48	6	0.7812

Figure 4 - Comparison of domain independent heuristics against Problem-1



	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
astar with h_ignore_preconditions	1450	1452	13303	9	33.101
astar with h_pg_levelsum	86	88	841	9	66.9705

Figure 5 - Comparison of domain independent heuristics against Problem-2



	Expansions	Goal Tests	New Nodes	Plan Length	Time Elapsed (s)
astar with h_ignore_preconditions	5038	5040	44926	12	181.2812
astar with h_pg_levelsum	293	295	2694	12	298.951

Figure 6 - Comparison of domain independent heuristics against Problem-3

Part 3 – Analysis of Planning Search Algorithms

Part 3.1 – Optimal Plans for Air Cargo Problems

#Problem	Optimal Plan
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)
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)
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)

Table 1 - Optimal plans for air cargo problems as found by breadth-first search

Part 3.2 – Analysis of Uninformed Planning Searches

The "air cargo transport problem" that was discussed in lectures was tested with three non-heuristic planning solution searches. Each search, namely breadth-first search, depth-first search and uniform-cost search, was applied to Problem-1, Problem-2 and Problem-3 that were specified in the "Implementing a Planning Search" readme documentation.

Before going into how these search algorithms compare when run against these three problems, it is important to highlight how each problem compares to each other in order to better understand how certain characteristics of each search function scales. Table 2 shows some of the key differences in the three problems used in this task.

#Problem	#Cargo	#Airports	#Planes	#Goal Conditions
1	2	2	2	2
2	3	3	3	3
3	4	4	2	4

Table 2 - Comparison of air cargo problems

Problem-1 is the simplest with only 2 cargos, 2 planes and 2 airports. As we know, all actions yield either an "In" condition, or an "At" condition, where "In" condition is a mapping between cargos and planes and the "At" condition is a mapping between cargos and airports or between planes and airports. This gives us 2 x 2 possible "In" conditions and 4x2 possible "At" conditions, which means we can have a total of 12 different conditions. Since each condition can be either true or false, this will give us $2^{12} = 4096$ different states. The goal state requires two of the conditions to be in a desired Boolean state and it doesn't specify what the other 10 condition should be (i.e. we don't care where each plane is, we just care about where the cargos end up in). This means roughly a quarter of the states ($2^{10} = 1024$) states will match our goal state. Note that this is "roughly" because preconditions and effects of actions will prevent certain combination of conditions from appearing in a single state and this will shrink the state space to a smaller number. That said, this rough calculation is still useful to compare the difficulty of solving each of the problems at a high level. The same calculation for all three problems is shown in Table 3.

#Problem	#In Literals	#At Literals	#States	#Goal States	#Success Rate
1	4	8	~4,000	~1,000	25%
2	9	18	~134,000,000	~17,000,000	12.5%
3	8	24	~4,000,000,000	~268,000,000	6%

Table 3 - Comparison of problem difficulties

When the number of goal states and the success rate is analysed together, it is obvious that each problem is exponentially harder than its predecessor. For example, Problem-3 has roughly one million times more states than Problem-1 and solutions are rarer since success rate is 6% as opposed to 25% in Problem-1.

A summary of how each search performed against these tasks are shown in Figure 1, Figure 2 and Figure 3 in "Part 1". Note that while both breadth first search and uniform cost search guarantee to find the optimal path, they had a slight variation in their performance when we look at number of expansions and the time elapsed. The similarity is not surprising, because the path cost used by the uniform cost search in this example is the number of steps, which will be equal to the depth of the state in the search tree. In other words, since the path cost is simply the depth of the state, uniform cost search will actually perform a breadth first search. However, the order of evaluating states at any depth can differ from breadth first search, since uniform cost search uses a heapq as the underlying collection, while breadth first search uses a FIFO queue, which can explain the slight variations in performance. Also note that uniform-cost search expands all nodes at the goal-state depth to see if there is any node with smaller cost while breadth first search would stop examining nodes as soon as it finds the goal state, which would explain why uniform cost search had more goal tests and more expansions than breadth first search across all three problems.

Another important conclusion is that although depth first search doesn't guarantee optimality, it is considerably faster at reaching a solution than both of the other uninformed search algorithms. The reason is obvious in the number of nodes expanded. Because the actions defined by the schema are reversible (i.e. you can load and unload a cargo, fly the planes back and forth), the algorithm minimises branching. It keeps taking actions that result in different states (guaranteed to be different as the algorithm keeps track of visited states) until it reaches a solution. Since the assumption in this problem is to optimise the cost of delivering cargos to certain locations, the solutions found by depth first search might not look usable, but in a different domain where speed of finding a solution is more important than finding one of the optimal solutions, it could be preferable. Depth first search performed particularly well when the optimal solution was deeper in the tree (e.g. in Problem 2 and Problem 3). This is because breadth first search and uniform cost search get exponentially slower at every new depth they examine, which makes it easy for depth first search to surpass their performance despite finding solutions at depths over 300. Taking the worst case as an example, depth first search found a solution at depth 619 for Problem-2 while the optimal solution was at depth 9. However, depth first search only had to expand 624 nodes to get to the solution, while breadth first search had to expand 3343 nodes to get to the solution at depth 9.

It is important to highlight that this doesn't mean depth first search will always outperform optimal search algorithms in similar problems. The success of the depth first search doesn't only depend on the depth of the optimal solution, but also on how rare the solution is within the state space. To demonstrate this, I have added three more conditions to the goal state of problem 2, which are shown below. The greyed-out conditions are what we had in the original problem, and the conditions in green are what has been added.

$$\text{At(C1, JFK)} \land \text{At(C2, SFO)} \land \text{At(C3, SFO)} \land \text{At(P1, JFK)} \land \text{At(P2, SFO)} \land \text{At(P3, SFO)}$$

This means by adding these 3 new conditions we have effectively reduced the number of goal states from $\sim 2^{24} = 17{,}000{,}000$ to $\sim 2^{21} = 2{,}000{,}000$ which means solutions form only 1.5% of the total state space as opposed to 12.5%. Making the possibility of finding a solution harder had a dramatic performance hit on depth first search as expected and it took 24s to find a solution at depth 447 with 4188 expansions.

The same modified Problem-2 was solved in 11s by breadth first search and the number of nodes expanded stayed at 3343. Briefly, since the optimal solution hasn't changed, the same solution as before was found, making it 3 times faster than depth first search. Briefly, although depth first search looks to be a good choice in the original problems, we can not generalise and should always pay attention to the commonality of the solution within the state space.

Part 3.3 – Analysis of Searches Using Domain Independent Heuristics

We used A* search algorithm with two different heuristics across the same three problems to be able to assess how the heuristics performed and also to be able to compare the performance with uninformed searches analysed in Part 3.2.

It should be noted that as both heuristics are admissible, that is neither of the heuristics over estimate the actual cost of reaching a given state, the solutions are guaranteed to be optimal. The admissibility of the pg_levelsum heuristic is dependent on the goal conditions being independent which holds true for the given goals. And the admissibility of the ignore_preconditions heuristic is guaranteed by definition, since having preconditions entail that the actual cost to reach a state would be equal or more.

The first thing that stands out in the results is that despite the smaller number of expansions and goal tests needed by pg_levelsum heuristic compared to ignore_preconditions heuristics, it consistently run slower across all three problems. This highlights the importance of simplicity in heuristic calculations. Finding smarter heuristics that require considerably less node expansions might sound like a good idea at first, but if this additional intelligence comes at a price of heavier computations, it will likely be less preferable.

That said, one observation that might not be that obvious is that as the problems gets more difficult, the performance of the "smarter" pg_levelsum heuristic improves against ignore_precondition heuristics. This means when the complexity of the problem reaches a certain level, pg_levelsum might be able to outperform pg_levelsum. This estimation is based on the fact that as the solution space gets larger, the importance of evaluating less nodes will increase. This is also evident in the rate of change in the nodes expanded as we progress from Problem-1 to Problem-3. Figure 7 shows how the execution time of the two algorithms compare across the three problems.

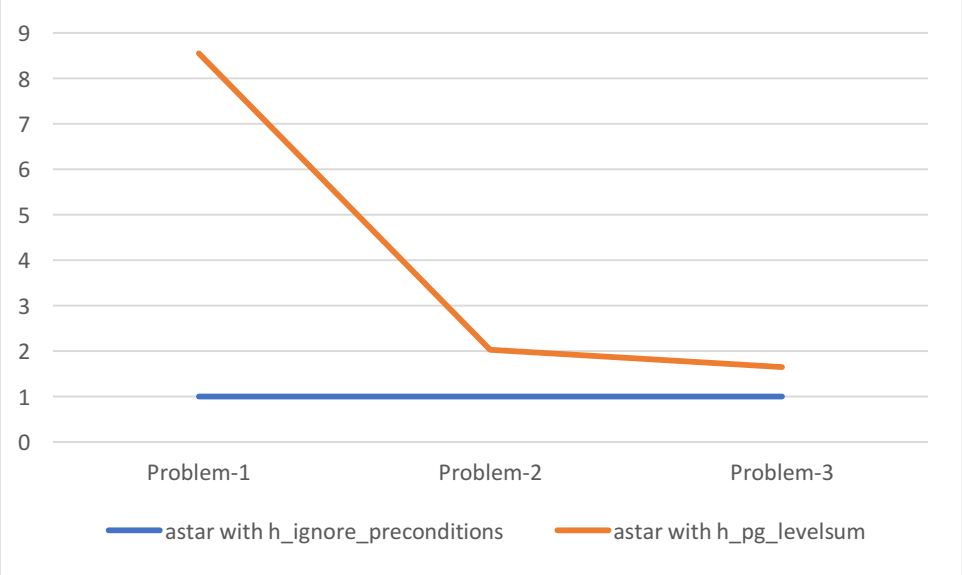


Figure 7 - Relative execution time of heuristics

(execution time of astar_with_h_ignore_preconditions divided by astar_with_pg_levelsum)

Part 3.4 – Analysis of Uninformed Searches vs Searches Using Domain Independent Heuristics

If we limit the problem space to these 3 specific problems, and assuming the definition of success is reaching the optimal solution the fastest, the optimal uninformed searches (i.e. breadth first search and uniform cost search) performed better than the A* search that leveraged admissible heuristics. However, as discussed in Part 3.2, this success can not be generalised to say uninformed searches fit this domain better. The search algorithms that use heuristics would perform much better for more complex problems where the solution

lies deeper in the search space and where the search space is larger. In fact, if we look at how the execution times compare between the A* search that uses pg_levelsum, which in this case is considered smarter than the ignore_preconditions heuristic due to giving more accurate estimations to the actual cost while remaining admissible, and the uninformed breadth first search as we progress from Problem-1 to Problem-3, we can see a pattern of relative improvement as the problem gets more complex.

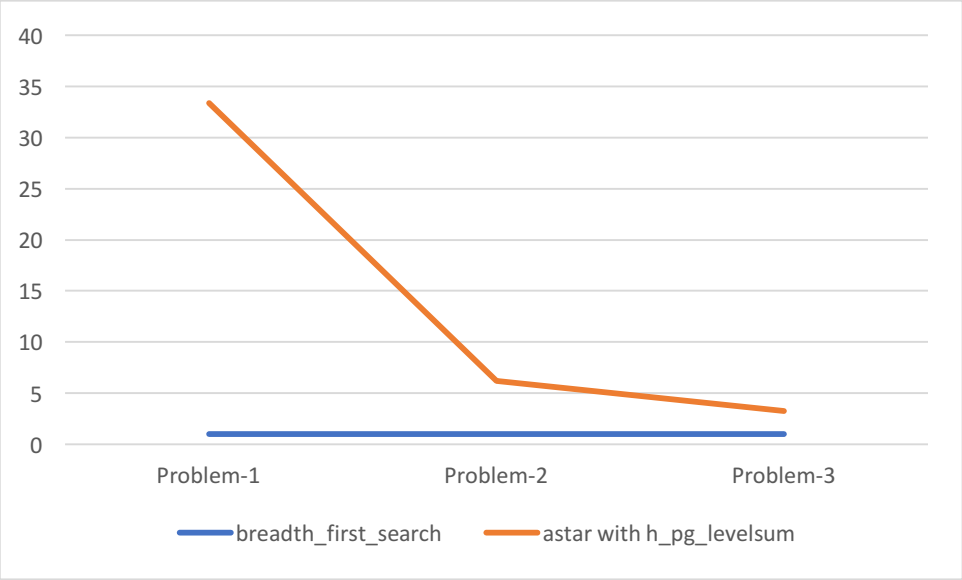


Figure 8 - Relative execution time of informed vs uninformed searches

(execution time of pg_levelsum divided by breadth first search)

To prove that pg_levelsum heuristic performs better in more complex problems, I modified Problem-3. The updated goal state is shown below.

$$At(C1, JFK) \land At(C2, SFO) \land At(C3, JFK) \land At(C4, SFO) \land At(C5, ATL) \land At(P2, ATL)$$

By adding a fifth cargo, the possible actions that can be taken from a given state (branching factor) has increased which will make finding the solution more challenging for breadth first search. The additional two conditions at the goal state also make the solution rarer and puts the solution deeper in the search space. This was done to reduce the chances of reaching a solution when using depth first search. The summary of the problem modifications are shown in Table 4 and the results of the searches are shown in Table 5 and Figure 9. The results highlight how A* with pg_levelsum heuristic outperformed uninformed searches that deceivingly looked as better choices for simpler problems.

#Problem	#In Literals	#At Literals	#States	#Goal States	#Success Rate
3	8	24	~4,000,000,000	~268,000,000	6%
3 (Modified)	10	28	~275,000,000,000	~4,000,000,000	1%

Table 4 - Comparison of the modified version of Problem-3 to the original

Algorithm	Problem-1	Problem-2	Problem-3	Problem-3 (Modified)
breadth_first_search	0.0234	10.7725	90.786	3668
depth_first_graph_search	0.0098	3.1698	1.3755	1565
h_pg_levelsum	0.7812	66.9705	298.951	780

Table 5 – Time elapsed (s) for each search algorithm for modified Problem-3

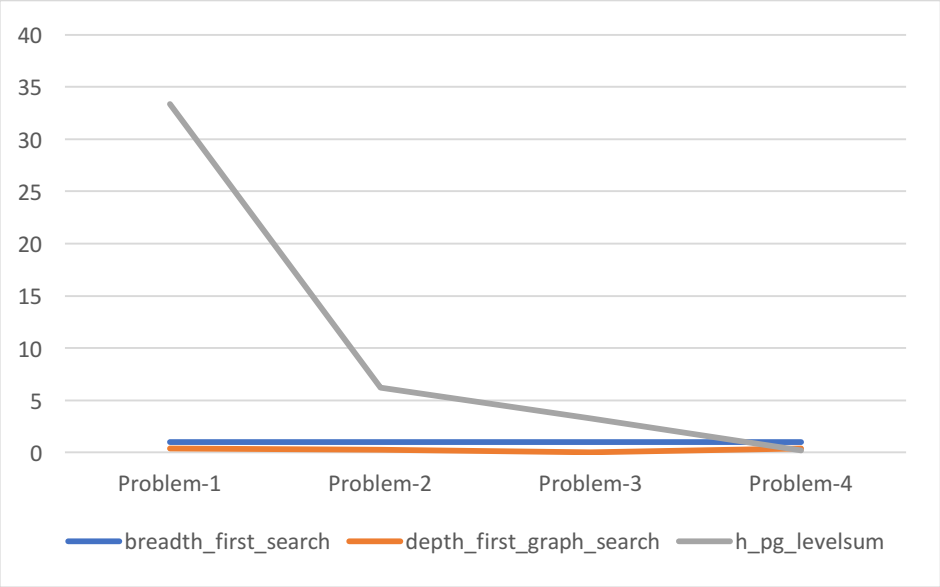


Figure 9 - Performance comparison of search algorithms across all problems including modified Problem-3

(execution time of pg_levelsum and depth first search divided by breadth first search)

In conclusion, this analysis highlighted that for simpler problems where the solution is close to the initial state and where the search space is not too big, breadth first search and uniform cost search performed better than other options. Depth first search is also promising for similar problems where the solution is not too rare in the search space and where the objective is not to reach the optimal solution, but to reach a solution as fast as possible.

For more complex problems however, it was obvious that heuristics had the upper hand. Although A* searches required more time to expand and analyse each node due to requiring heavier computations compared to uninformed searches, they made up for it in complex problems by expanding far less nodes by smartly identifying the states that are more likely to take them to the solution. A similar relationship was found within the two heuristics used in the A* algorithm. Ignore preconditions heuristics was easier to compute, but did not produce the costs as close to the real cost as pg_levelsum heuristics did. This was obvious in the number of nodes expanded as the problems got more complex. Again, when the problem reached a certain complexity, pg_levelsum heuristic easily outperformed ignore preconditions as expected.

As a final note, it should be noted that generalising on what planning search works best on a given domain is difficult and the answer highly depends on how individual problems are setup, e.g. how big the solution space is, how deep the solutions are located at and whether finding the optimal solution is more important than finding a solution fast.