Heuristic Analysis

For this analysis we will focus on the different cargo problems, with the schema provided:

```
Action(Load(c, p, a),
        PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
        PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
        EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
        PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
        EFFECT: ¬ At(p, from) ∧ At(p, to))
```

Cargo problem 1:

A simple optimization transporting problem, needing to move two loads present at two airports, from-to another one. Having an available plane on each origin for each cargo.

```
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))
```

The ideal solution to the problem needs to follow the subsequent path:

1. Load the cargo into a plane, present already at the airport
2. Fly the plane from the present airport to the desired destination one
3. Unload the cargo at the destination airport

An implementation could be:

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

Cargo problem 2:

A simple optimization transporting problem, needing to move tree loads present at three different airports, having each one an available plane on their origin.

```
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))
```

The ideal solution to the problem needs to follow the subsequent path:

1. Load the cargo into a plane, present already at the airport
2. Fly the plane from the present airport to the desired destination one
3. Unload the cargo at the destination airport

An implementation could be:

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

Cargo problem 3:

A less simple optimization transporting problem, needing to move four loads present at four airports, to another one. Having just two planes available for the task.

```
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))
```

The ideal solution to the problem needs to follow the subsequent path:

1. Load the cargo into a plane, present already at the airport
2. Fly the plane from the present airport to the origin one of another cargo with same destination as the one already loaded.
3. Load the second cargo on the plane.
4. Fly to their (the cargos) common airport destination.
5. Unload both cargos at their expected destination.

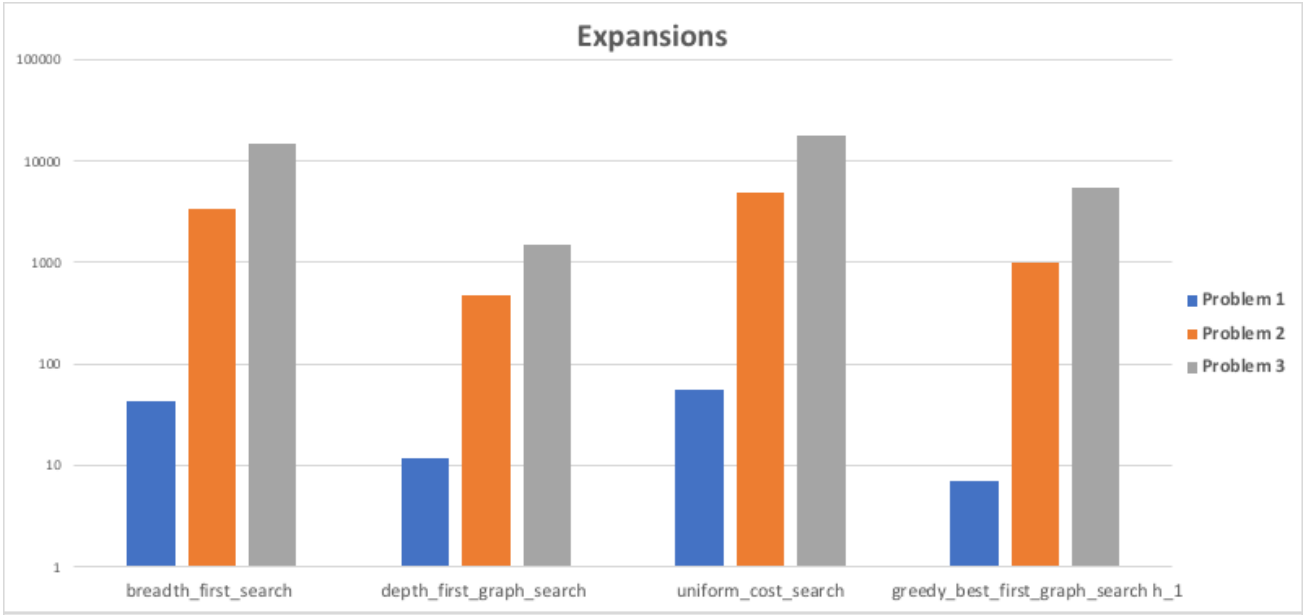
An implementation could be:

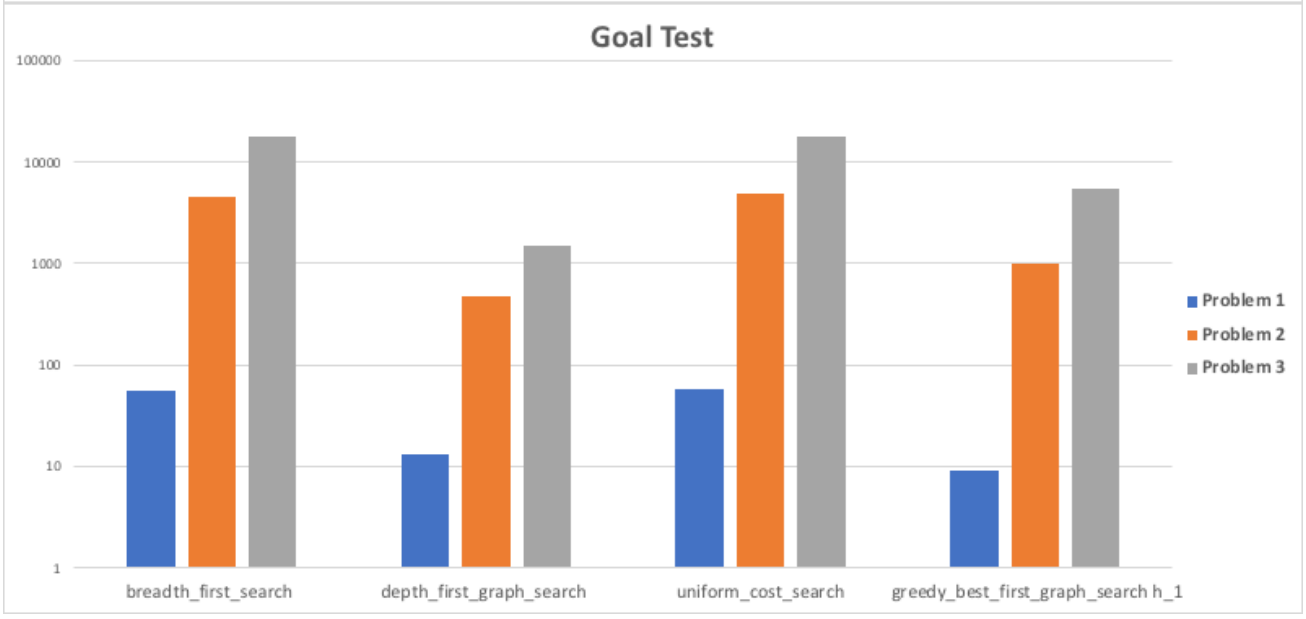
- Load(C1, P1, SFO)
- Load(C2, P2, JFK)
- Fly(P1, SFO, ATL)
- Load(C3, P1, ATL)
- Fly(P2, JFK, ORD)
- Load(C4, P2, ORD)
- Fly(P2, ORD, SFO)
- Fly(P1, ATL, JFK)
- Unload(C1, P1, JFK)
- Unload(C2, P2, SFO)
- Unload(C3, P1, JFK)
- Unload(C4, P2, SFO)

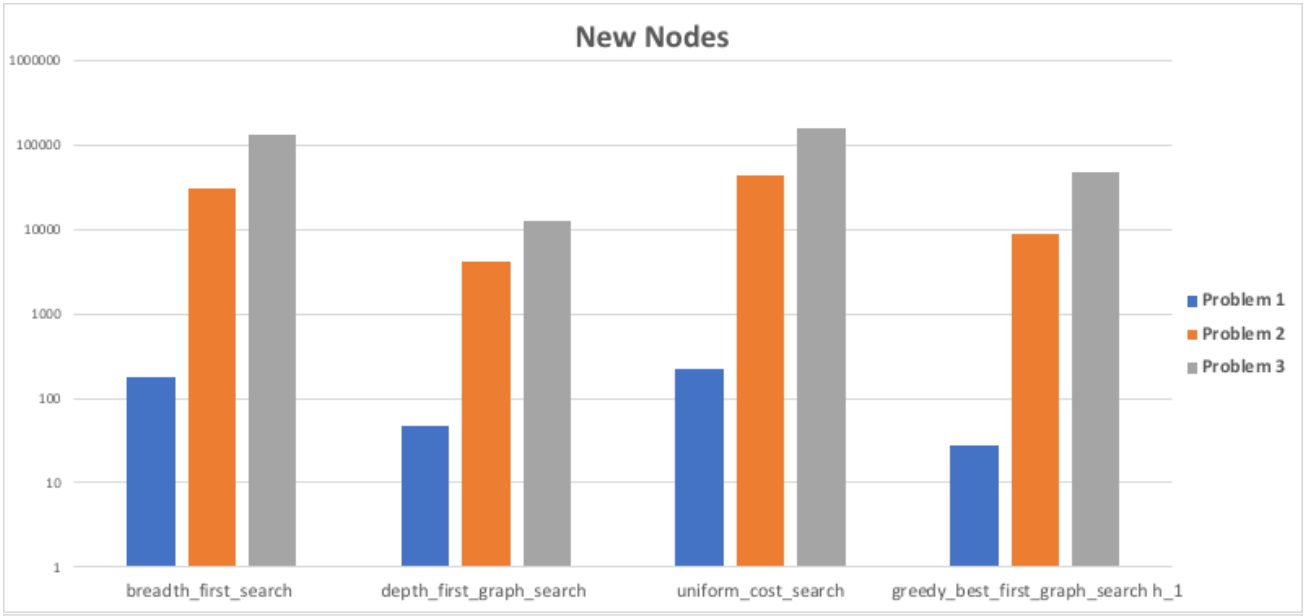
Heuristics comparison

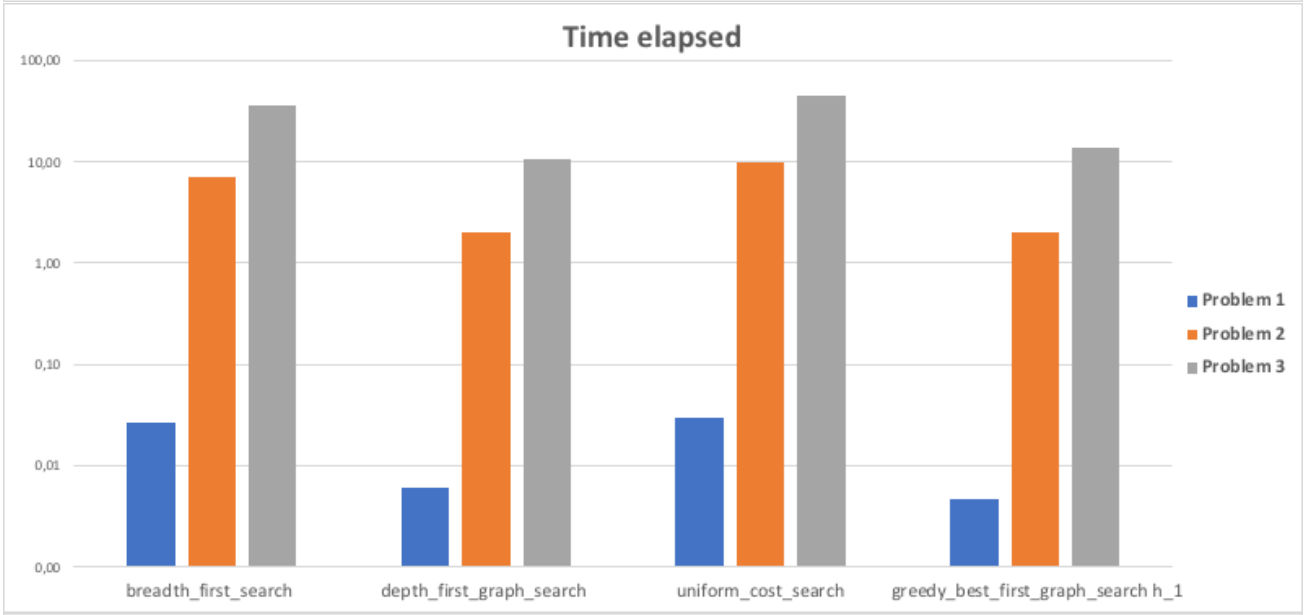
Non-heuristic search

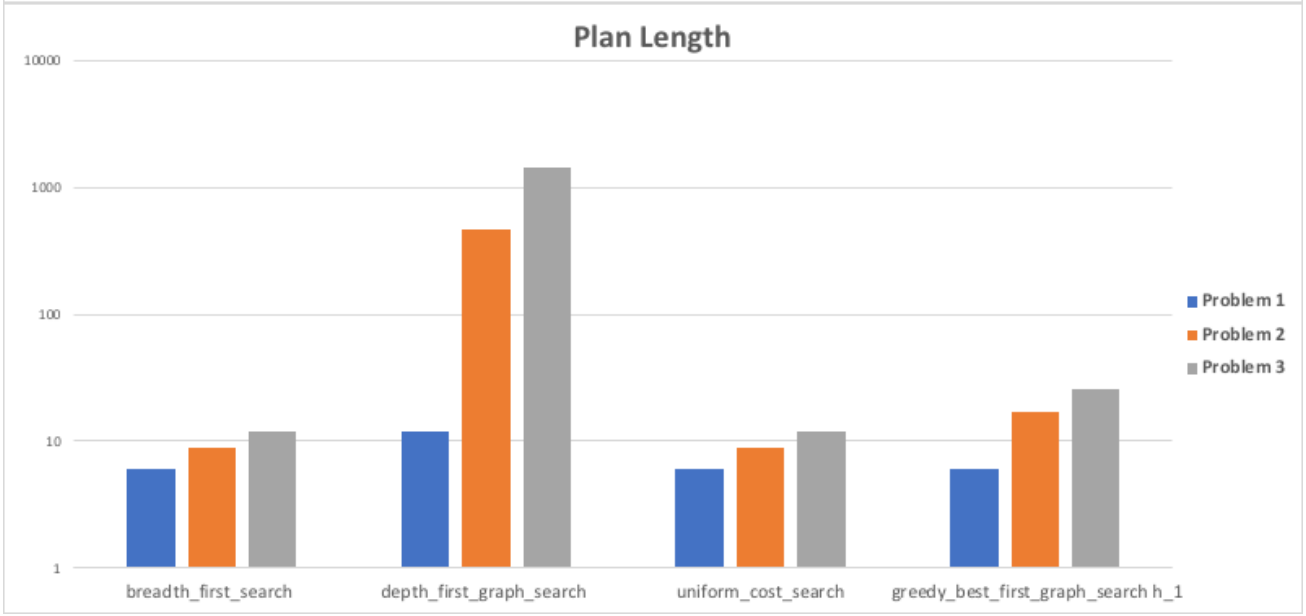
Problem	Uninformed Planning Algorithm	Expansions	Goal Tests	New Nodes	Time elapsed	Plan Lenght
1	breadth_first_search	43	56	180	0,03	6
1	depth_first_graph_search	12	13	48	0,01	12
1	uniform_cost_search	55	57	224	0,03	6
1	greedy_best_first_graph_search h_1	7	9	28	0,00	6
2	breadth_first_search	3343	4609	30509	7,11	9
2	depth_first_graph_search	476	477	4253	2,02	466
2	uniform_cost_search	4853	4855	44041	9,80	9
2	greedy_best_first_graph_search h_1	998	1000	8982	2,01	17
3	breadth_first_search	14663	18098	129631	36,37	12
3	depth_first_graph_search	1511	1512	12611	10,66	1442
3	uniform_cost_search	18151	18153	159038	45,52	12
3	greedy_best_first_graph_search h_1	5398	5400	47665	13,66	26






New Nodes
1000000
100000
10000
1000
100
10
1
breadth_first_search
depth_first_graph_search
uniform_cost_search
greedy_best_first_graph_search h_1
Problem 1
Problem 2
Problem 3


Time elapsed
100,00
10,00
1,00
0,10
0,01
0,00
breadth_first_search
depth_first_graph_search
uniform_cost_search
greedy_best_first_graph_search h_1
Problem 1
Problem 2
Problem 3


Plan Length
10000
1000
100
10
1
breadth_first_search
depth_first_graph_search
uniform_cost_search
greedy_best_first_graph_search h_1
Problem 1
Problem 2
Problem 3

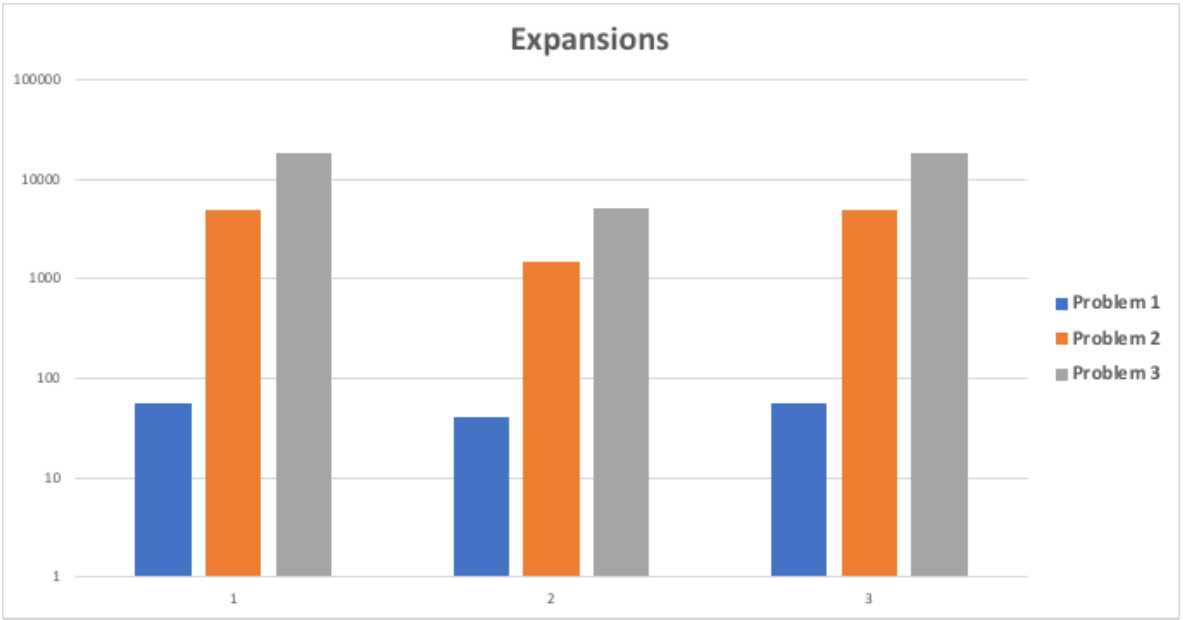
The non-heuristic search algorithms provided with very different behavior for a same problem depending on the exploration technique used.

Suma de Plan Lenght	Algorithm			
Problem	greedy_best_first_graph_search h_1	uniform_cost_search	breadth_first_search	depth_first_graph_search
1				
0,004752065	6			
0,006096217				12
0,026755933			6	
0,029882163		6		
2				
2,010439988	17			
2,020732024				466
7,108746924			9	
9,800904881		9		
3				
10,65915967				1442
13,65849623	26			
36,3701944			12	
45,51639412		12		

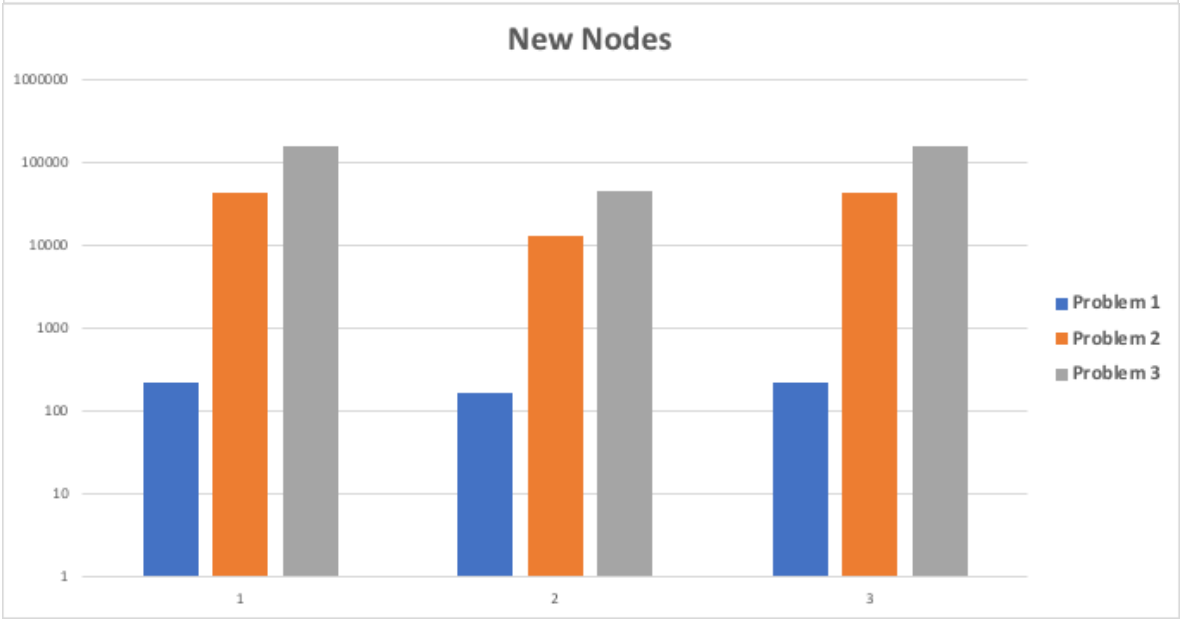
On the above image we see the ordering of the used search algorithms based on just two metrics, the length of the final solution provided, and the amount of time used to reach it.
This two metrics have been brought forward in detriment of others because they are the most salient ones for the final user of this optimization programs. The time needed to obtain an answer and the relation between the length of the solution and the cost/time in the real world are prime for real world operations (considering each action, takes the same cost/time to be performed). Being aware that for large problems the amount of memory used varies largely depending on which is used, i.e. depth first search only occupies the memory need to store the current evaluated solution, as where breath first search saves all current space till the level where the frontier is placed.
Basing us in this two metrics (time elapsed and simplicity of the solution), we find that the algorithm that outperforms the rest is the "***Breath First Search"***.

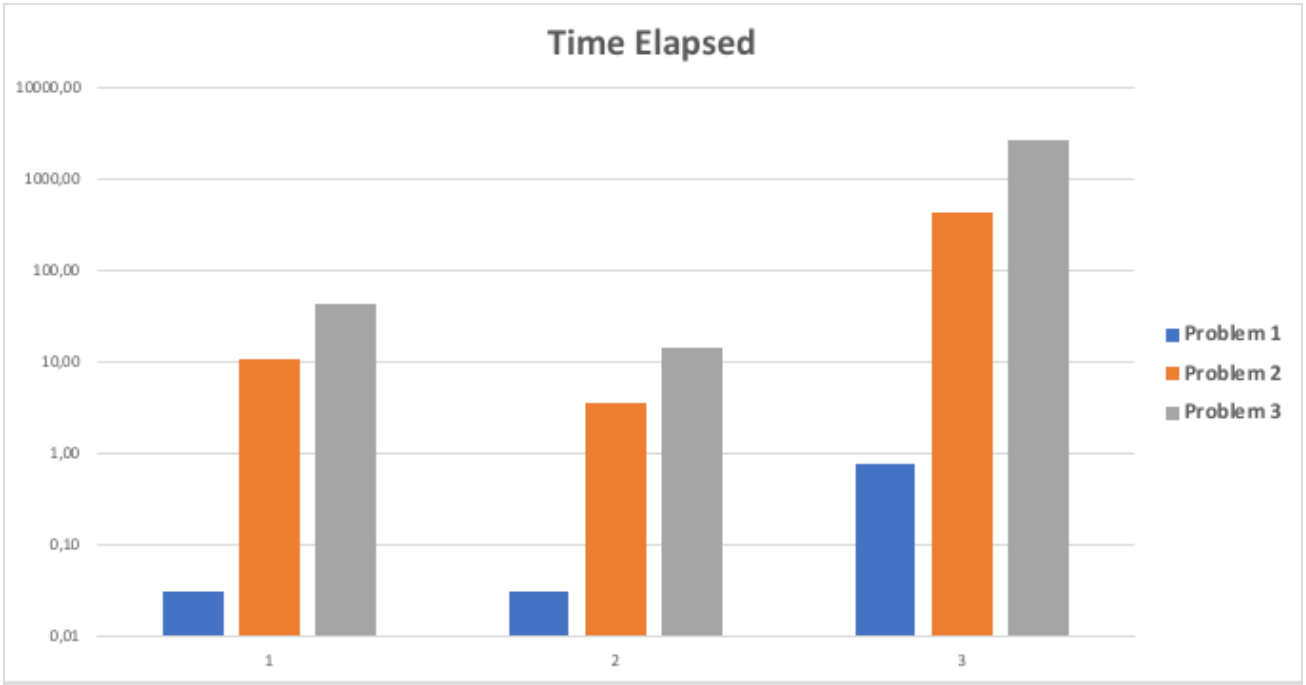
Heuristic search

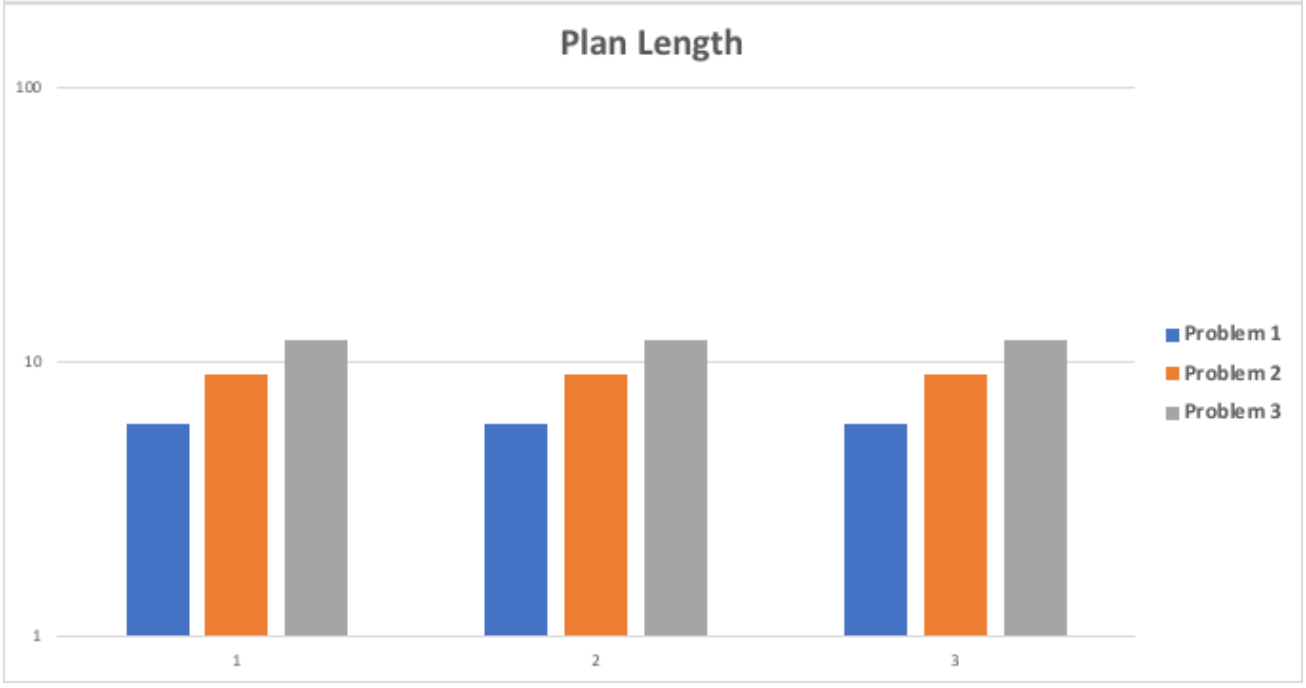
Problem	Uninformed Planning Algorithm	Expansions	Goal Tests	New Nodes	Time elapsed	Plan Lenght
1	astar_search h_1	55	57	224	0,03	6
1	astar_search h_ignore_preconditions	41	43	170	0,03	6
1	astar_search h_pg_levelsum	55	57	224	0,79	6
2	astar_search h_1	4853	4855	44041	10,66	9
2	astar_search h_ignore_preconditions	1450	1452	13303	3,52	9
2	astar_search h_pg_levelsum	4853	4855	44041	440,65	9
3	astar_search h_1	18151	18153	159038	43,76	12
3	astar_search h_ignore_preconditions	5038	5040	44926	14,65	12
3	astar_search h_pg_levelsum	18.151	18153	159038	2.630,93	12


Expansions
100000
10000
1000
100
10
1
1
2
3
Problem 1
Problem 2
Problem 3


Goal Tests
100000
10000
1000
100
10
1
1
2
3
Problem 1
Problem 2
Problem 3


New Nodes
1000000
100000
10000
1000
100
10
1
1
2
3
Problem 1
Problem 2
Problem 3

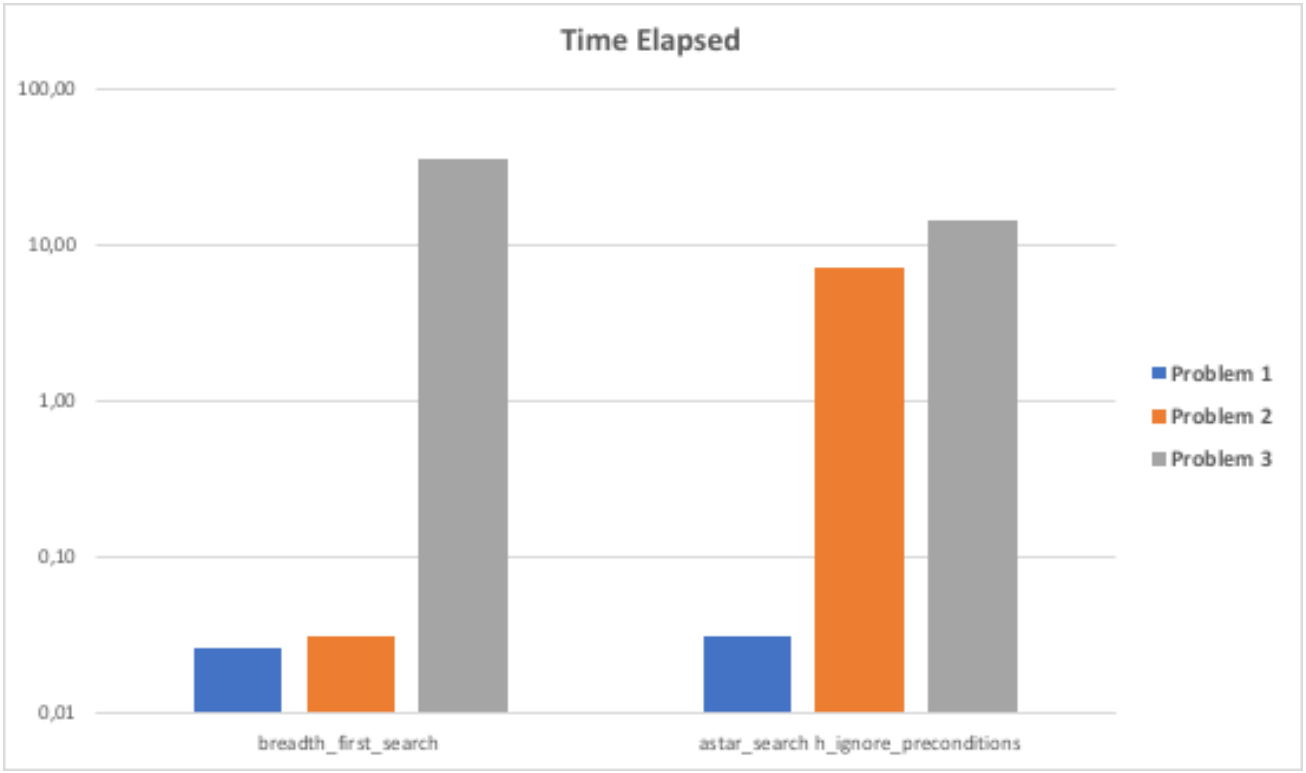




Suma de Expansions	Algorithm		
Problem	astar_search h_1	astar_search h_ignore_preconditions	astar_search h_pg_levelsum
1			
0,030897655		41	
0,031583127	55		
0,788765085			55
2			
3,518898128		1450	
10,65685568	4853		
440,6475632			4853
3			
14,65304		5038	
43,75535947	18151		
2630,931702			18151

On the heuristic searching algorithms approach, we had in the 3 variants, the same solution length for each problem independent of the one employed. Furthermore, this were ***"perfect solutions"***, naming perfect the shortest ones available (taking as granted that each action costs the same and the are no simultaneous actions performed).
In consequence, for this tier of searching algorithms, we focus our comparison on the time elapsed to provide the solution as well as the number of expansions performed (used as a proxy of the memory consumptions requirements), deciding to point the preference more into the speed than the expansion reduction. One can asseverate that both conditions are met with the best results (the lower time and number of expansions), for each of the three problems using the ***"astar_search h_ingore_preconditions"***.

Conclusions



Selecting the best perfect algorithms from each of the categories, we can observe that the behavior of the ***"breath_first_search"*** algorithm works really well when the problem involves just the straightforward solution, i.e. a plane for each cargo already waiting at the origin to take it straight to their expected destination.
In contraposition as the problem gets tougher, the third problem; where we need to circumvallate the initial idea to use a plane for a cargo at each time and in contraposition, start to do more complex transportations paths, we discover a great yield for the ***"astar_search h_ignore_preconditions"***.
Due to the fact that ignoring preconditions makes the path length (metric used for A-start) be optimistic, thus making A-* find a solution. And A-* performs generally better than Breath First search as it **focuses the search space** just in the direction where there is a bigger reduction in the "left distance to reach goal" rather than exploring the whole space, expanding it in all possible directions. Concluding that the ***A-* algorithm ought to perform better for even more complex problems***.