Project 3: Heuristic Analysis

By Paula Martínez

Introduction

On this project were implemented solutions for classical PDDL problems. The problems were on the Air cargo Domain, with the same action schema but different initial states and goals as follow.

- Air Cargo Action Schema:

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))

- Problem 1 initial state and goal:

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 initial state and goal:

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 initial state and goal:

```
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))
```

Part 1: Planning problems

Uniformed planning searches were applied to the problems. The results are presented below.

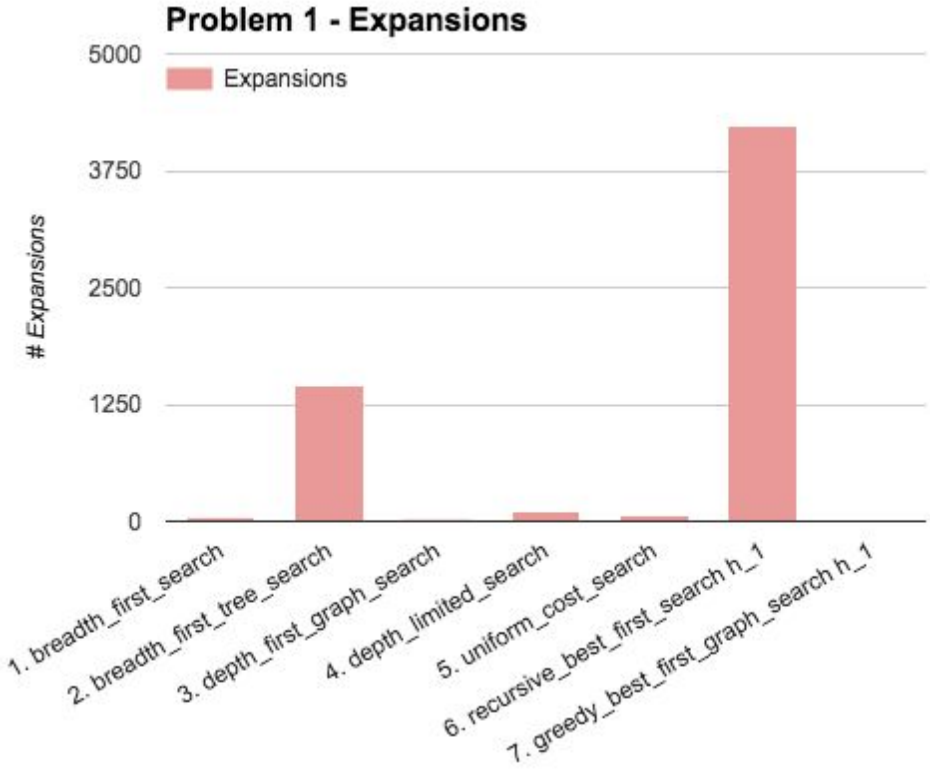
Problem 1

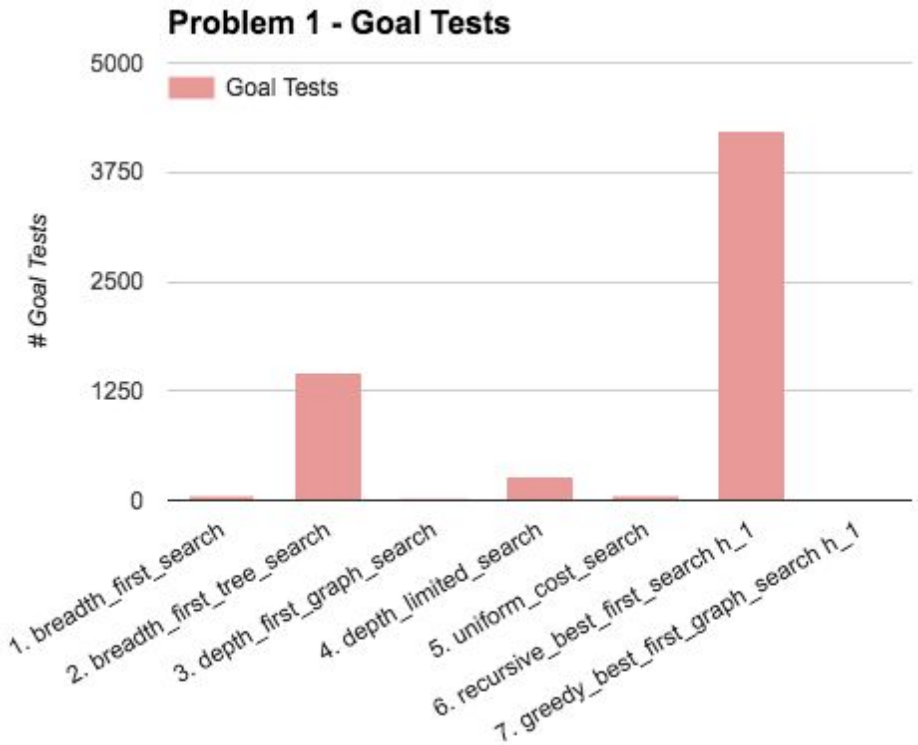
Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
1. breadth_first_search	43	56	180	6	0.056334819
2. breadth_first_tree_search	1458	1459	5960	6	1.273234596
3. depth_first_graph_search	21	22	84	20	0.022060862
4. depth_limited_search	101	271	414	50	0.129283285
5. uniform_cost_search	55	57	224	6	0.054981002
6. recursive_best_first_search h_1	4229	4230	17023	6	3.69996253
7. greedy_best_first_graph_search h_1	7	9	28	6	0.008468471

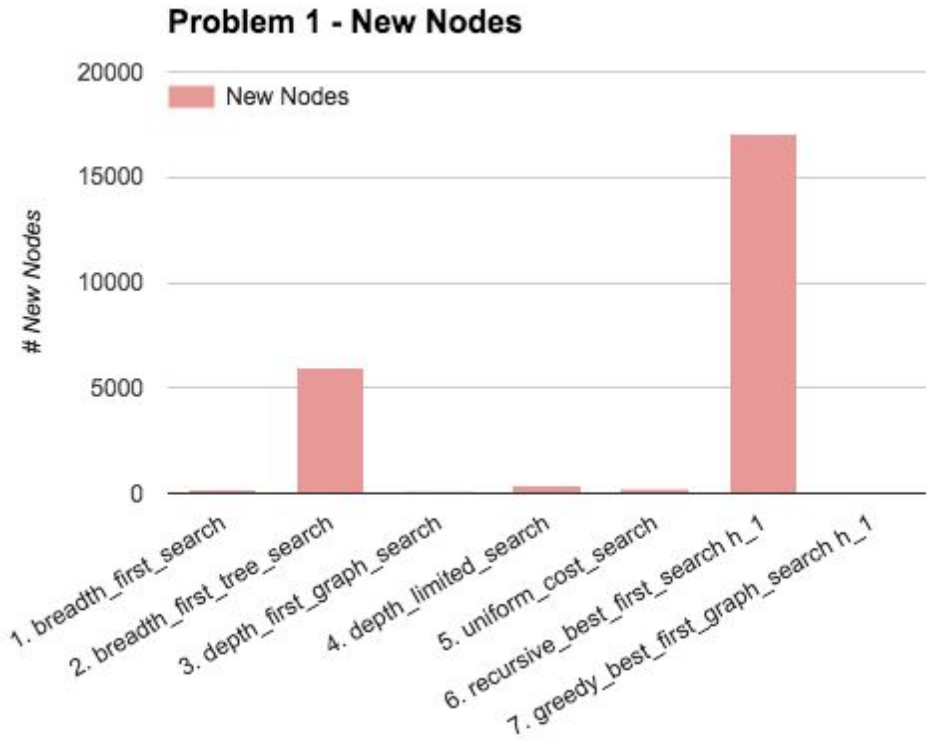
In this case, the algorithm **greedy_best_first_graph_search_h_1** outperformed the others. It expanded 7 nodes, test 9 goals, have a plan length of 6, and was the fastest on compute. The state space for this problem is relatively small, on bigger state spaces this algorithm may not performed well and heuristics will be very helpful.

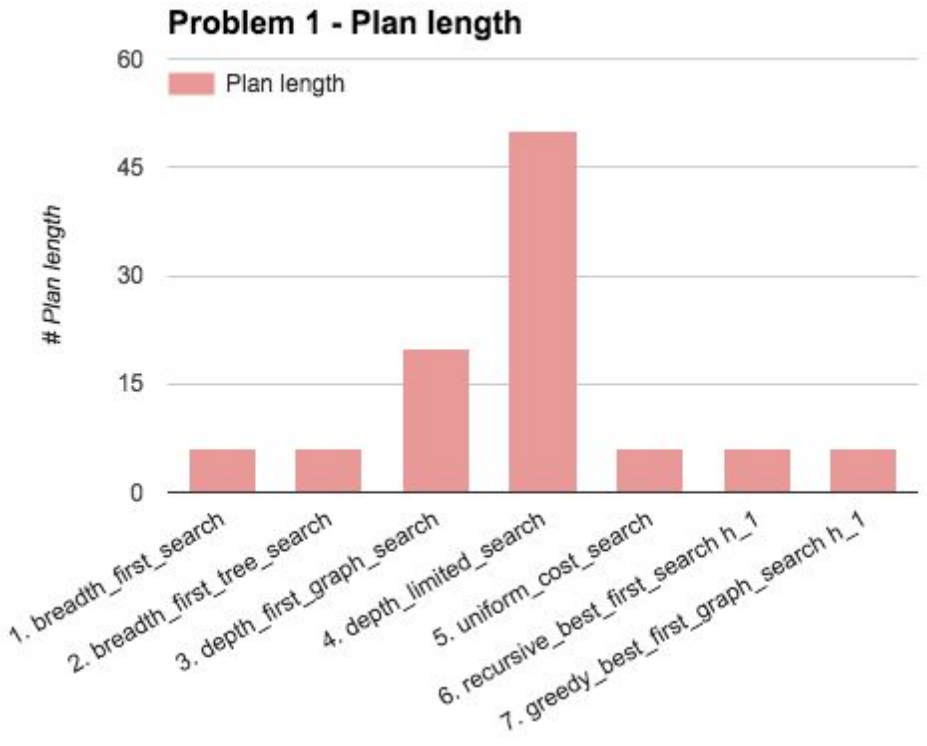
The optimal sequence is:
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)

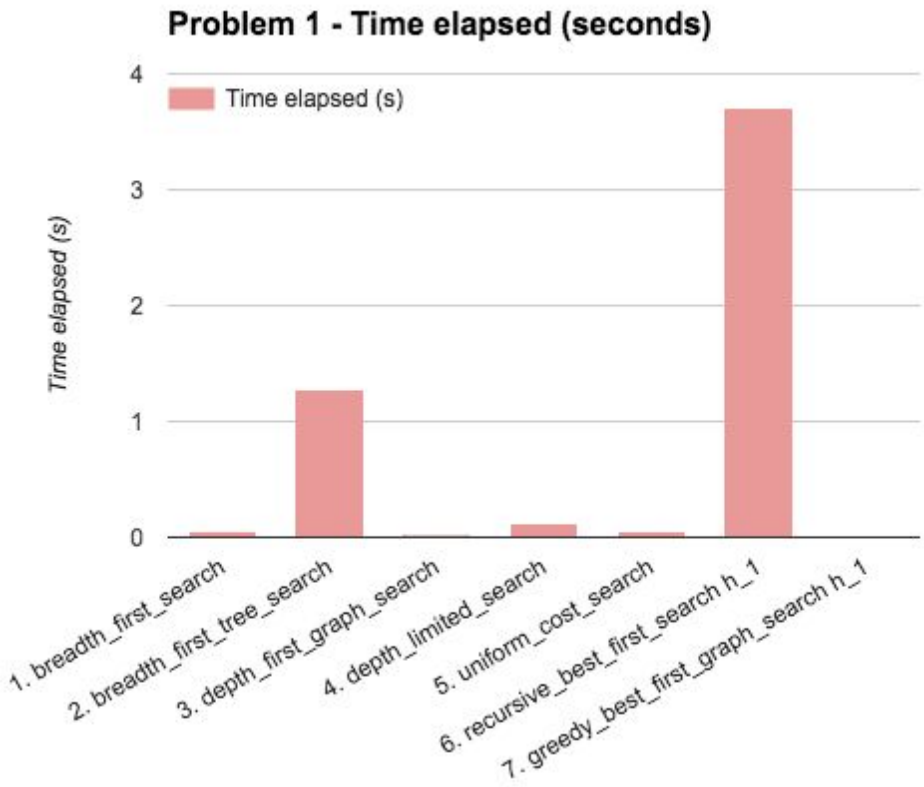
On the following graphs can be compared the results for all search methods applied to problem 1.


Problem 1 - Expansions
Expansions
Expansions
5000
3750
2500
1250
0
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


Problem 1 - Goal Tests
Goal Tests
Goal Tests
5000
3750
2500
1250
0
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


Problem 1 - New Nodes
New Nodes
New Nodes
20000
15000
10000
5000
0
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


Problem 1 - Plan length
Plan length
Plan length
60
45
30
15
0
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



Problem 2

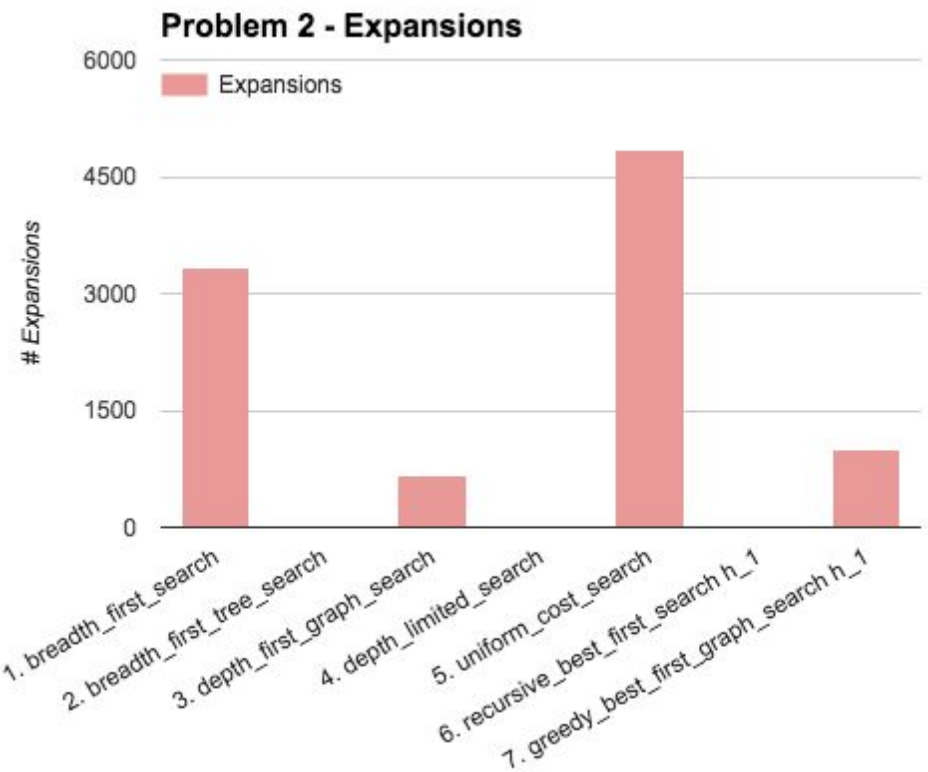
Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
1. breadth_first_search	3346	4612	30534	9	18.73800719
2. breadth_first_tree_search					
3. depth_first_graph_search	661	662	5906	651	5.090673337
4. depth_limited_search					
5. uniform_cost_search	4853	4855	44041	9	16.03009249
6. recursive_best_first_search h_1					
7. greedy_best_first_graph_search h_1	998	1000	8982	21	3.239228494

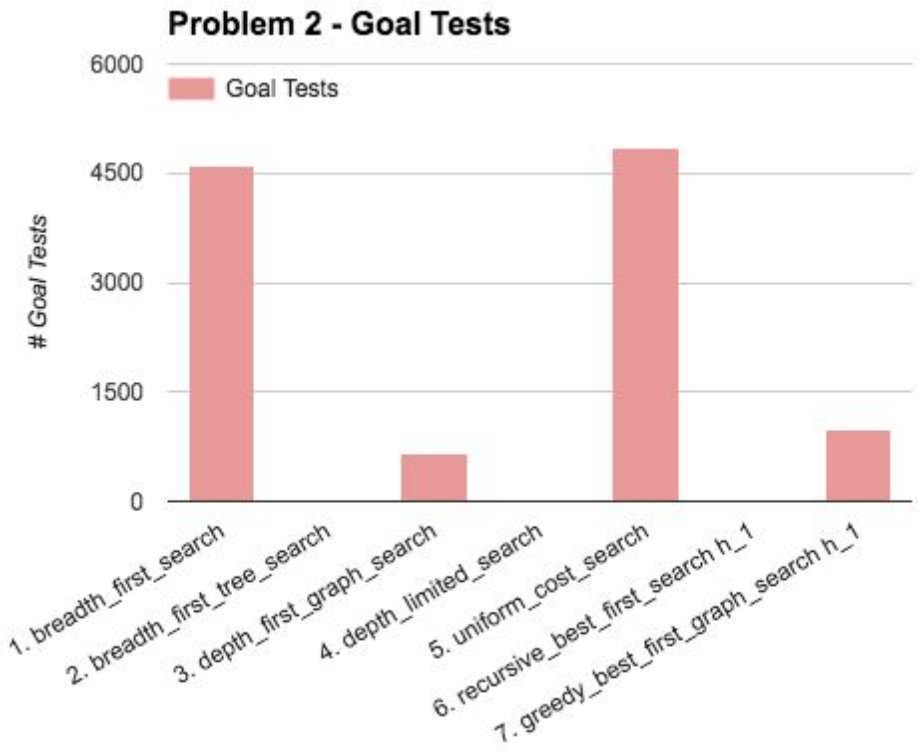
Aclaration: for problems 2 and 3 the following search methods are not presented because it take too long to compute them: breadth_first_tree_search, depth_limited_search and recursive_best_first_search h_1.

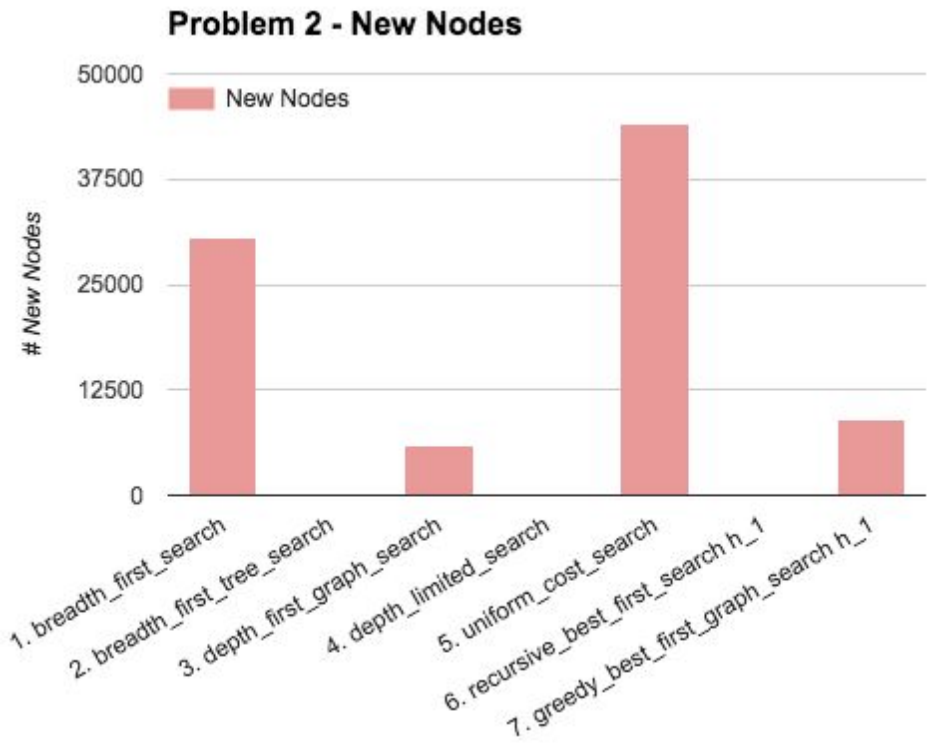
In this case the best algorithm was the **uniform_cost_search** because it found faster than breadth_first_search a plan length of 9. Anyway it has more expansions, goal tests and new nodes that other search methods. In this case is still possible to explore the whole state space, but some heuristics may help improve the computational costs.

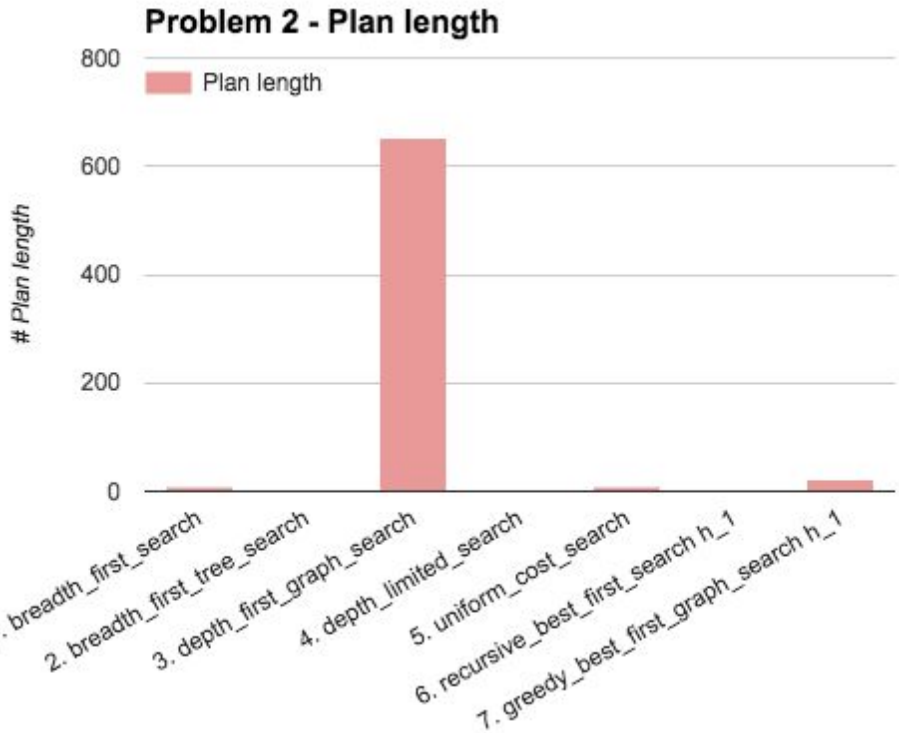
The optimal sequence is:
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

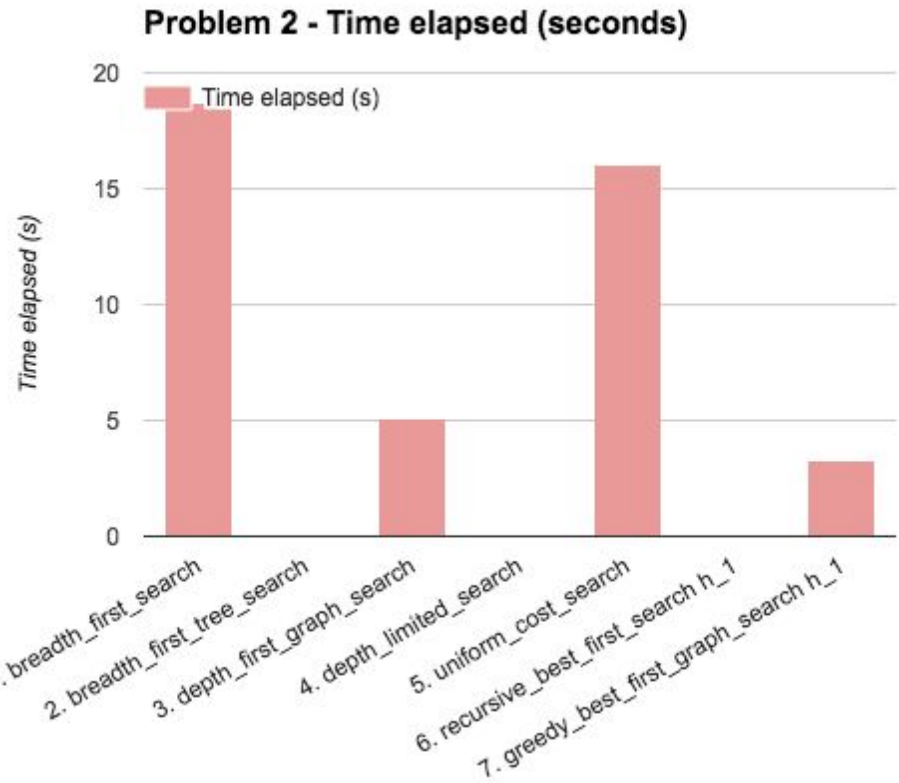
On the following graphs can be compared the results for all search methods applied to problem 2.




Problem 2 - Goal Tests
6000
4500
3000
1500
0
Goal Tests
Goal Tests
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


Problem 2 - New Nodes
50000
37500
25000
12500
0
New Nodes
New Nodes
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


Problem 2 - Plan length
Plan length
Plan length
800
600
400
200
0
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


Problem 2 - Time elapsed (seconds)
Time elapsed (s)
Time elapsed (s)
20
15
10
5
0
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

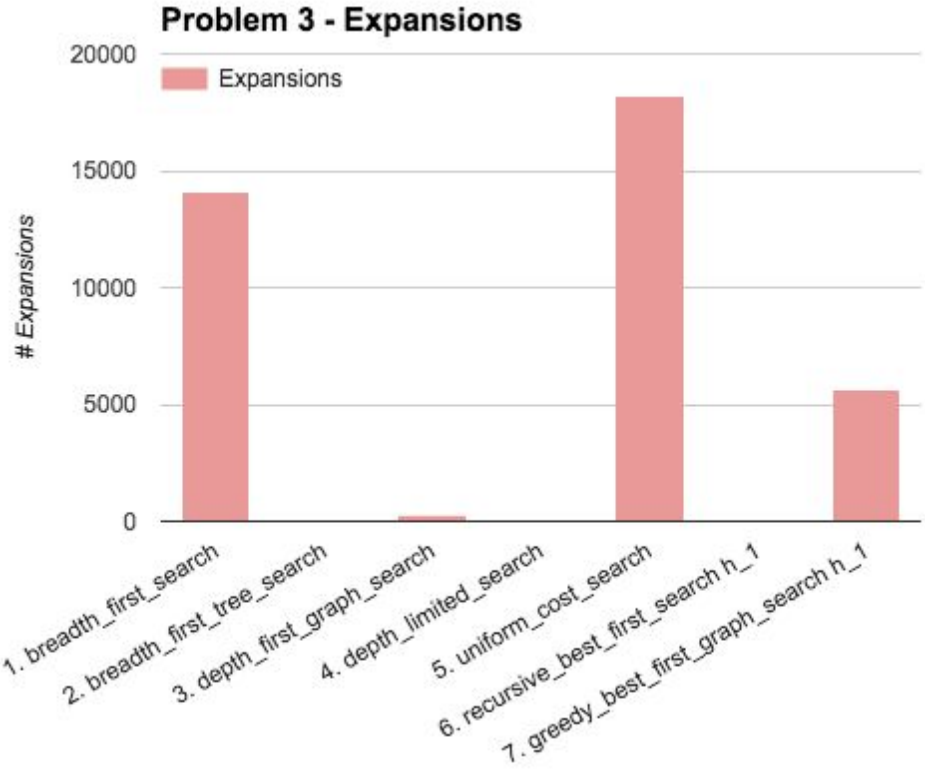
Problem 3

Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
1. breadth_first_search	14120	17673	124926	12	179.9010495
2. breadth_first_tree_search					
3. depth_first_graph_search	292	293	2388	288	1.640951034
4. depth_limited_search					
5. uniform_cost_search	18235	18237	159716	12	81.85872153
6. recursive_best_first_search h_1					
7. greedy_best_first_graph_search h_1	5614	5616	49429	22	28.09980705

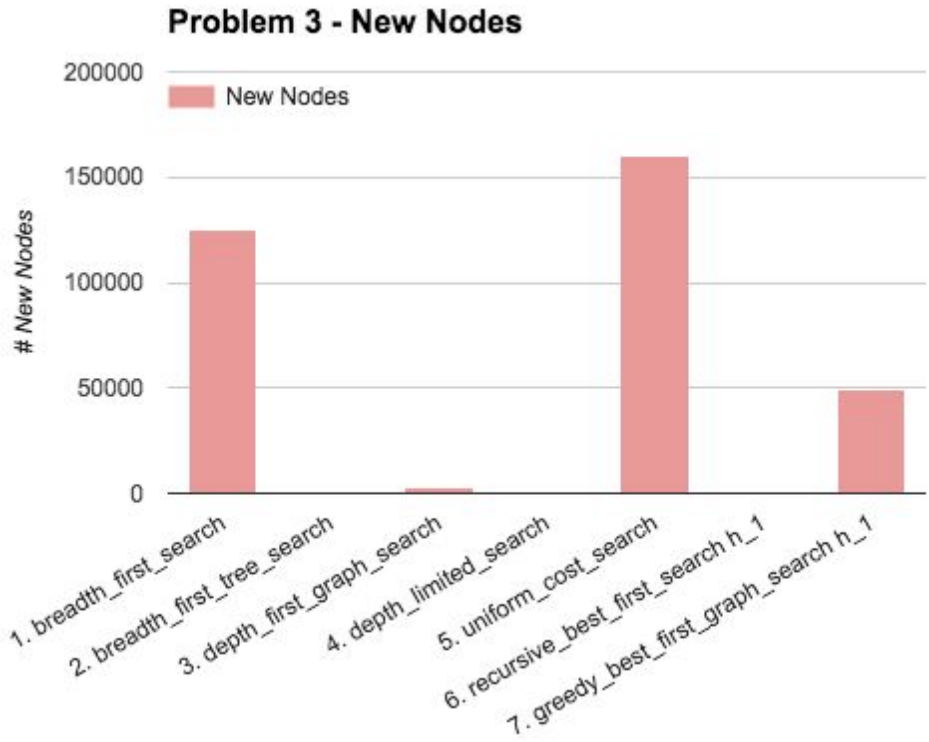
In this case, the optimal path length found was 12 and **uniform_cost_search** found it faster. On this problem the time elapsed is several times bigger than in problems 1 and 2. Therefor an heuristic search might be very helpful to reduce the computational cost in this case, although, as will be presented later in this document, the optimal path length remains on 12.

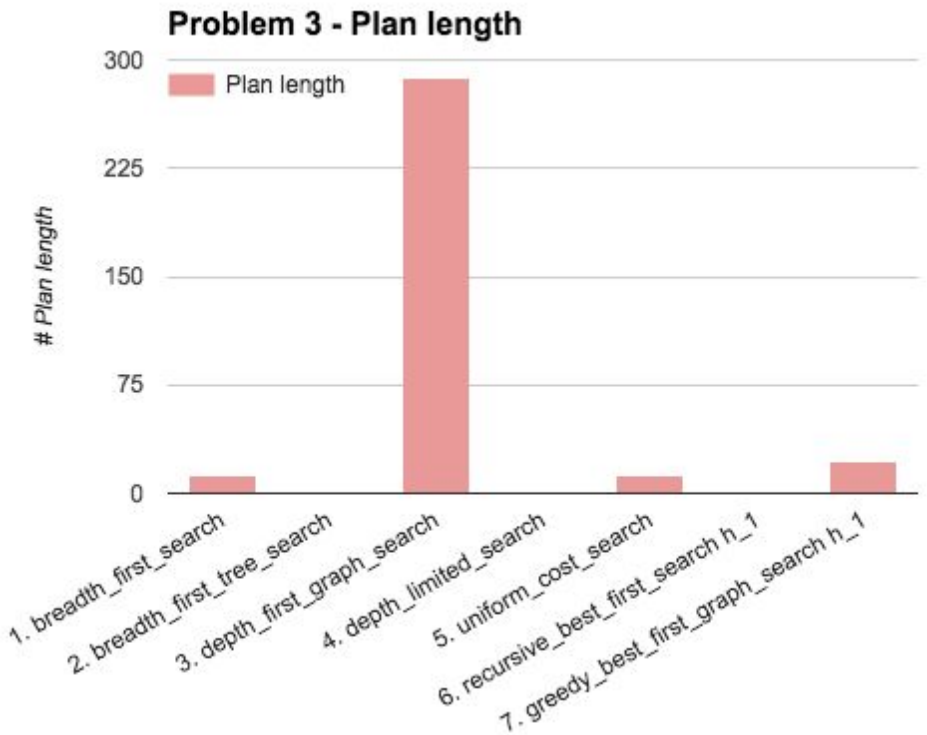
The optimal sequence is:
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Fly(P1, ATL, JFK)
Unload(C4, P2, SFO)
Unload(C3, P1, JFK)
Unload(C2, P2, SFO)
Unload(C1, P1, JFK)

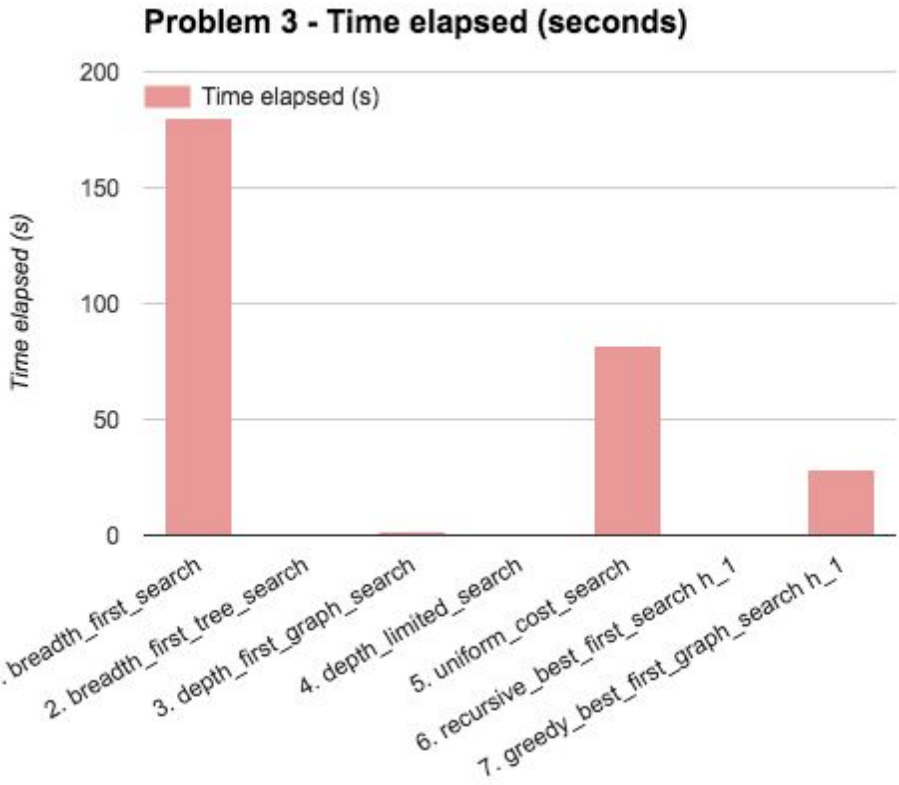
On the following graphs can be compared the results for all search methods applied to problem 2.


Problem 3 - Expansions
Expansions
Expansions
20000
15000
10000
5000
0
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


Problem 3 - Goal Tests
Goal Tests
Goal Tests
20000
15000
10000
5000
0
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


Problem 3 - New Nodes
New Nodes
New Nodes
200000
150000
100000
50000
0
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


Problem 3 - Plan length
Plan length
Plan length
300
225
150
75
0
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



Part 2: Domain-independent heuristics

Three search heuristics were implemented and tested on problems 1, 2 and 3. The results were as follow.

	Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
Problem 1	8. astar_search h_1	55	57	224	6	0.053731685
	9. astar_search h_ignore_preconditions	41	43	170	6	0.065758763
	10. astar_search h_pg_levelsum	11	13	50	6	1.074753858
Problem 2	8. astar_search h_1	4853	4855	44041	9	16.24207455
	9. astar_search h_ignore_preconditions	1450	1452	13303	9	7.464122574
	10. astar_search h_pg_levelsum	86	88	841	9	107.2029992
Problem 3	8. astar_search h_1	18235	18237	159716	12	106.8698558
	9. astar_search h_ignore_preconditions	5040	5042	44944	12	26.47774613
	10. astar_search h_pg_levelsum	315	317	2902	12	471.2125668

It can be observed, that astar_search_h_ignore_preconditions outperformed astar_search_h_pg_levelsum in terms of time on the three problems. The optimal path found

was the same but the computational cost turn out to be expensive with “level-sum” than with “ignore preconditions” heuristics. Although “level-sum” heuristic expand less and tested less goals, the time elapsed for “ignore preconditions” is smaller.

On problem 1, the search method greedy_best_first_graph_search h_1 outperformed the three heuristics. This is because the state space was small, and therefor not too expensive to explore all the space.

On problem 2, although the path length is the same, the search method astar_search h_ignore_preconditions outperformed the non-heuristic one selected previously because it reduces the computational cost significantly as shown in the following table.

Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
5. uniform_cost_search	4853	4855	44041	9	16.03009249
8. astar_search h_1	4853	4855	44041	9	16.24207455
9. astar_search h_ignore_preconditions	1450	1452	13303	9	7.464122574
10. astar_search h_pg_levelsum	86	88	841	9	107.2029992

On problem 3, the need to apply an heuristic search method is more evident, since the state space is bigger than in the two previous problems. And as shown on the following table, the astar_search_h_ignore_preconditions method outperformed the uniform_cost_search in terms of computational cost.

Search method	Expansions	Goal Tests	New Nodes	Plan length	Time elapsed (s)
5. uniform_cost_search	18235	18237	159716	12	81.85872153
8. astar_search h_1	18235	18237	159716	12	106.8698558
9. astar_search h_ignore_preconditions	5040	5042	44944	12	26.47774613
10. astar_search h_pg_levelsum	315	317	2902	12	471.2125668

The heuristic that performed the best on this problems was the “ignore preconditions”. For problems 2 and 3 it found the optimal path with the less computational cost. But on problem 1, a non-heuristic planning search method was better in terms of computational cost. Heuristics are convenient and very useful when the state space becomes big. For small state spaces like on problem 1 a non-heuristic search might be more adequate.