Heuristics analysis
AIND October2017

KAROLY ALBERT SZABO
szabo.karoly.a@gmail.com

Problems

Definitions

The three problems suggested had different complexity. The first one is easy to solve, as the number of state spaces are limited. The second one have some more complexity, as the number of features grows from 2 per type to 3 per type. The third problems appear already complex at first sight, as we have only 2 airplanes to move 4 cargos from 4 airports.

P1	P2	P3

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))
	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))
		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))

Solutions

I'll show one optimal solution for each problem. For each of them, different algorithm gave different solutions, depending on the road they took to solve it.

P1 - 6 steps	P2 - 9 steps	P3 - 12 steps

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

Unload(C1, P1, JFK)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
	Unload(C3, P3, SFO)
	Load(C2, P2, JFK)
	Fly(P2, JFK, SFO)
	Unload(C2, P2, SFO)
	Load(C1, P1, SFO)
	Fly(P1, SFO, JFK)
	Unload(C1, P1, JFK)
		Load(C4, P2, ORD)
		Fly(P2, ORD, SFO)
		Unload(C4, P2, SFO)
		Load(C1, P1, SFO)
		Fly(P1, SFO, ATL)
		Load(C3, P1, ATL)
		Fly(P1, ATL, JFK)
		Unload(C3, P1, JFK)
		Unload(C2, P2, SFO)
		Unload(C1, P1, JFK)

Uninformed search

I dropped Breadth First Graph search due to its terrible performances.

Table of results

	Problem 1					Problem 2					Problem 3				
Alg	Exp	Goal	Node	Time	Plan	Exp	Goal	Node	Time	Plan	Exp	Goal	Node	Time	Plan
bfs	**43**	**56**	**180**	**0.058**	**6**	**3343**	**4609**	**30509**	**15.67**	**9**	14663	18098	129631	115.32	12
dfgs	21	22	84	0.018	20	624	625	5602	3.939	619	408	409	3364	2.02	392
dls	101	271	414	0.123	50	-	-	-	-	-	-	-	-	-	-
ucs	55	57	224	0.051	6	4852	4854	44030	13.97	9	**18235**	**18237**	**159716**	**60.6**	**12**

Problem 1

Due to the limited state space of the problem, breadth first is making a better job compared even to uniform cost search, although only slightly. This is because the plan is relatively short and the algorithm can expand a number of nodes for each layer, until the sixth one. The chosen algorithm will be **Breadth First Search**

Problem 2

I saw a similar behaviour, breadth first is visiting less nodes compared to uniform cost search, while depth limited search was taking too much time to be kept a valid solution.

The time gap between UCS and BFS starts to widen a bit as the problem complexity grows, still the best uninformed algorithm seems to be **Breadth First Search**

Problem 3

In this problem the number of explored nodes grows and I can see the advantages of uniform cost search, the number of new nodes, expansions and goals checks will still determine breadth first search as the best algorithm, but the execution time is almost half with UCS, so for this problem I'll rather use **Uniform Cost Search.**

The speed of depth first is due to its one way discovery, once it reach the bottom of the search tree with any valid plan, it returns.

Informed Search

I dropped recursive best first search with h1, due to its poor performances already on Problem 1, it got stuck already on problem 2.

Table of results

	Problem 1					Problem 2					Problem 3				
Alg	Exp	Goal	Node	Time	Plan	Exp	Goal	Node	Time	Plan	Exp	Goal	Node	Time	Plan
gbfgs	**7**	**9**	**28**	**0.006**	**6**	990	992	8910	2.789	21	5614	5616	49429	18.4	27
ash1	35	57	224	0.048	6	4852	4854	44030	13.972	9	18235	18237	159716	60.1	12
aship	41	43	170	0.048	6	**1450**	**1452**	**13303**	**5.247**	**9**	**5040**	**5042**	**44944**	**19.66**	**12**
apglev	11	13	50	1.266	6	86	88	841	230.21	9	315	317	2902	1415.2	12

Problem 1

I can see that the **Greedy best First Graph Seach** algorithm got lucky and found an optimal plan by visiting a really little amount of nodes in no time. It can be considered the winner for the first easy problem. It probably had this chance because the problem offered few 'best' choices at each level.

Problem 2

I can discard the fast, but not reliable Greedy algorithm. A* with level sum heuristic emerges as a really interesting way of exploring the tree by expanding far fewer nodes than other algorithms, but the execution time is, on the other hand, not competitive with **A* with Ignore Preconditions heuristic**, that will be my algorithm of choice for this problem, as it offer the best tradeoff between nodes explored and execution time. If I suppose that exploring nodes involves a real cost, I'll switch to the **level sum heuristic.**

Problem 3

By comparing A* with h1 and with Ignore preconditions we can see how a valid heuristic gives advantages to A*, the explored nodes decreases even more if we put an even more intelligent heuristic like level sum. In terms of time, on the other hand, level sum is getting really slow, probably due to the string comparisons involved in each calculation of the heuristic plus the cost of the planning graph construction.

Conclusions

Informed search outperformed the uninformed algorithms both on nodes explored and on time. This is valid only if our goal is to have an optimal plan.

A* with a useful heuristic is the best algorithm for every problem (given the Greedy exception) to have fast and qualitative plans by visiting as few nodes as possible, without having to rely on planning graph.

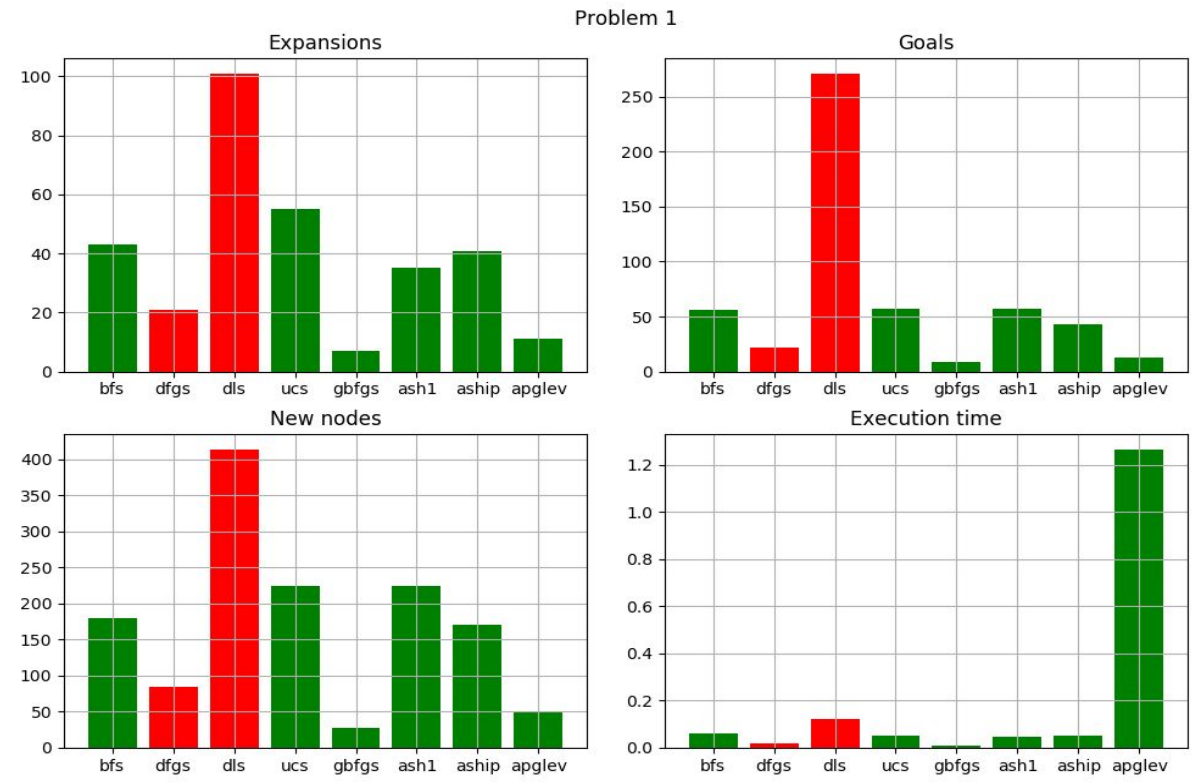
I think my Planning Graph implementation can be improved by replacing strings with hashes, adding memoization and pruning some branches from the search. If I'll be able to do so, probably the execution time will drop to acceptable gap with h_ignore_preconditions.

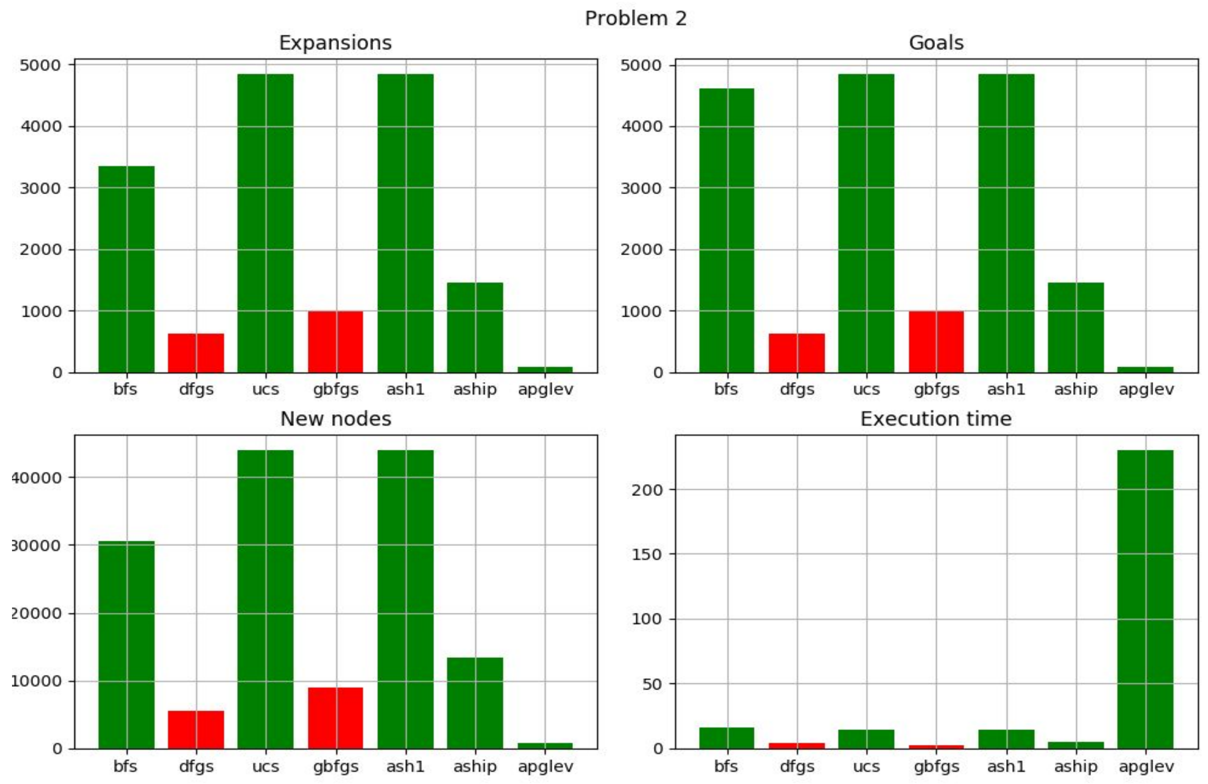
If a sub optimal plan is enough, but time is the most expensive constraint, Depth First Graph Search returns the first valid plan it finds, and is really unlikely that it will be optimal.

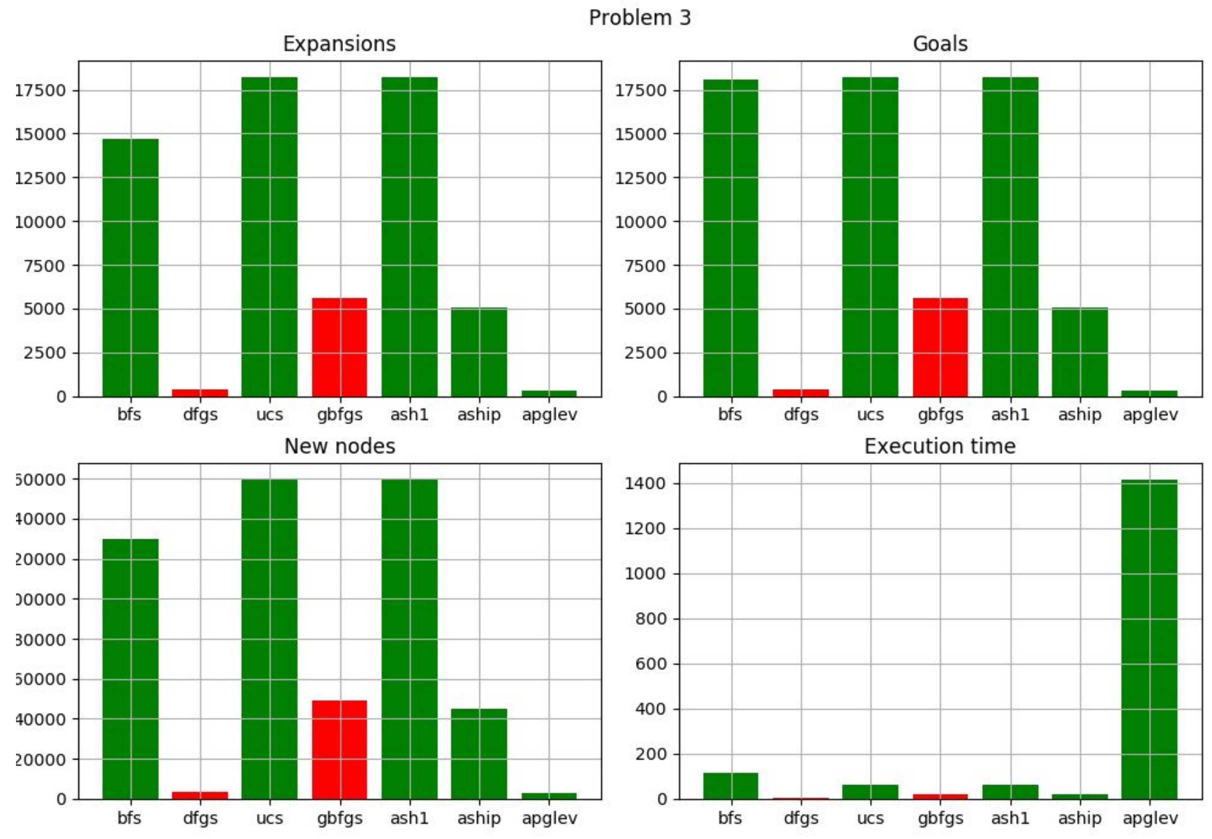
Algorithms shorthands

1. breadth_first_search = afs
2. breadth_first_tree_search = bfts
3. depth_first_graph_search = dfgs
4. depth_limited_search = dls
5. uniform_cost_search = ucs
6. recursive_best_first_search h_1 = rbfs
7. greedy_best_first_graph_search h_1 = gbfgs
8. astar_search h_1 = ash1
9. astar_search h_ignore_preconditions = aship
10. astar_search h_pg_levelsum = apglev

Red bars means non optimal resulting plans, some results were took out from the graph, due to out of scale values (P1: breadth_first_graph_search had 1400+ expansions for instance)




Problem 2
Expansions
Goals
New nodes
Execution time
bfs
dfgs
ucs
gbfgs
ash1
aship
apglev


Problem 3
Expansions
Goals
New nodes
Execution time
bfs
dfgs
ucs
gbfgs
ash1
aship
apglev