Udacity AIND program

Planning Algorithm Heuristic Analysis

Andrew Sperazza

*"A beaver relaxing because he efficiently finished building his **optimally-planned** beaver dam"*

Problem Statement

The problem solved in this project is to create an optimal sequence of actions for a cargo transportation airline to transport multiple cargo between multiple airports. The inputs to the planning system consists of planes, airports, and cargo containers. The target of this planning project is to achieve a desired goal from an initial starting set of requirements. There are three problem cases, and the goal in all cases is ultimately to get some cargo from point A to a different destination B in the most optimal number of steps, in the least amount of calculation time, and space complexity. Each case takes in the same type of parameters, but differ in the number of variables that are required to complete the action steps. Ten different searching algorithms were used and contrasted.

Problem Setup

This cargo transportation planning system uses PDDL, Planning Domain Definition Language, to define the requirements that are used as inputs to the search algorithms. The specific problem is defined using an initial state and goal, and the initial states are a conjunction of ground atoms, and the goal is just like a precondition. The problem is solved when we find a sequence of actions in a state that satisfies the desired goal.
The following defines the action schemas with an initial state ground actions, and goals:

Problem 1:

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:

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:

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))

Problem Solutions

The PDDL above, is solved using propositional logic, and thusly does not incorporate the use of variables in state descriptions. Several different search algorithms are utilized, contrasted and compared to identify search algorithms that can obtain accurate optimal solutions, in the fastest computational time, and with the least space complexity. The search algorithms analyzed are:

- breadth first
- breadth first tree
- depth first graph search
- depth limited search
- uniform cost
- recursive best fit
- greedy best first graph
- a star with h_1
- a star h_ignore precond
- a star with h_pg levelsum

The final search algorithm incorporates the use of a planning graph, which uses a polynomial size approximation of the very expensive exponential tree (representing all possible solution paths), this gives the advantage of automatically generating admissible heuristics.

Performance comparison

The following results were obtained by running the 10 search algorithms against the 3 problem domains.

solution number	solution description	time	expansions	goal tests	new nodes	plan length
PROBLEM 1						
1	breadth first	0.114	43	56	180	6
2	breadth first tree	1.29	1458	1459	59060	6
3	depth first graph search	0.01	12	13	48	12
4	depth limited search	0.113	101	271	414	50
5	uniform cost	0.053	55	57	224	6
6	recursive best fit	3.05	4229	4230	17029	6
7	greedy best first graph	0.005	7	9	28	6
8	a star with h_1	0.044	55	57	224	6
9	a star h_ignore precond	0.057	41	43	170	6
10	a star with h_pg levelsum	1.42	11	13	50	6
PROBLEM 2						
1	breadth first	5.27	3346	4612	30534	9
*2	breadth first tree	--	--	--	--	--
3	depth first graph search	0.394	107	108	959	105
4	depth limited search	1062.37	213491	1967093	1967471	50
5	uniform cost	6.291	4852	4854	44030	9
6	recursive best first	36872.28	70664634	70664653	637813875	9
7	greedy best first graph	1.666	990	992	8910	21
8	a star with h_1	6.418	4852	4854	44030	9
9	a star h_ignore precond	1.964	1450	1452	13303	9
10	a star with h_pg levelsum	26.49	86	88	841	9
PROBLEM 3						
1	breadth first	12.81	14633	18098	129631	12
*2	breadth first tree	--	--	--	--	--
3	depth first graph search	1.066	592	593	4927	571
*4	depth limited search	--	--	--	--	--
5	uniform cost	11.257	18236	18238	159726	12
*6	recursive best first	--	--	--	--	--
7	greedy best first graph	7.338	5623	5625	49495	21
8	a star with h_1	12.672	18236	18238	159726	12
9	a star h_ignore precond	12.16	5040	5042	44944	12
10	a star with h_pg levelsum	117.815	318	320	2934	12

*red indicates did not finish, green indicates fastest time among those with optimal sequence of actions

Optimal Search

In search planning terminology, each search algorithm can be classified as either optimal or sub-optimal. If an algorithm is classified as optimal, that means the goal with an optimal solution will be found first, if it is sub-optimal, that means a goal may be found, but it may not be the best one, and a true optimal solution remains undiscovered.

Optimal Sequence of actions

Problem 1 Optimal Sequence of actions

Problem 1 was the smallest dataset, and had the fastest runtime. The optimal sequence of actions 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)

These actions were obtained from the "greedy_best_first_graph_search with h_1". It had only 7 expansions, the fastest time, .005 seconds, and had the minimum number of steps needed to reach the goal (6).

Problem 2 Optimal Sequence of actions

The optimal sequence of actions for problem 2 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(C1, P1, JFK)
UnLoad(C2, P2, SFO)

These actions were obtained from the "a* search with h_ignore preconditions". It had 4852 expansions, 4854 Goal tests, and generated 44,030 new nodes. Comparing the fastest time among those searches that returned the minimum number of steps it had the fastest time of 6.42 seconds.

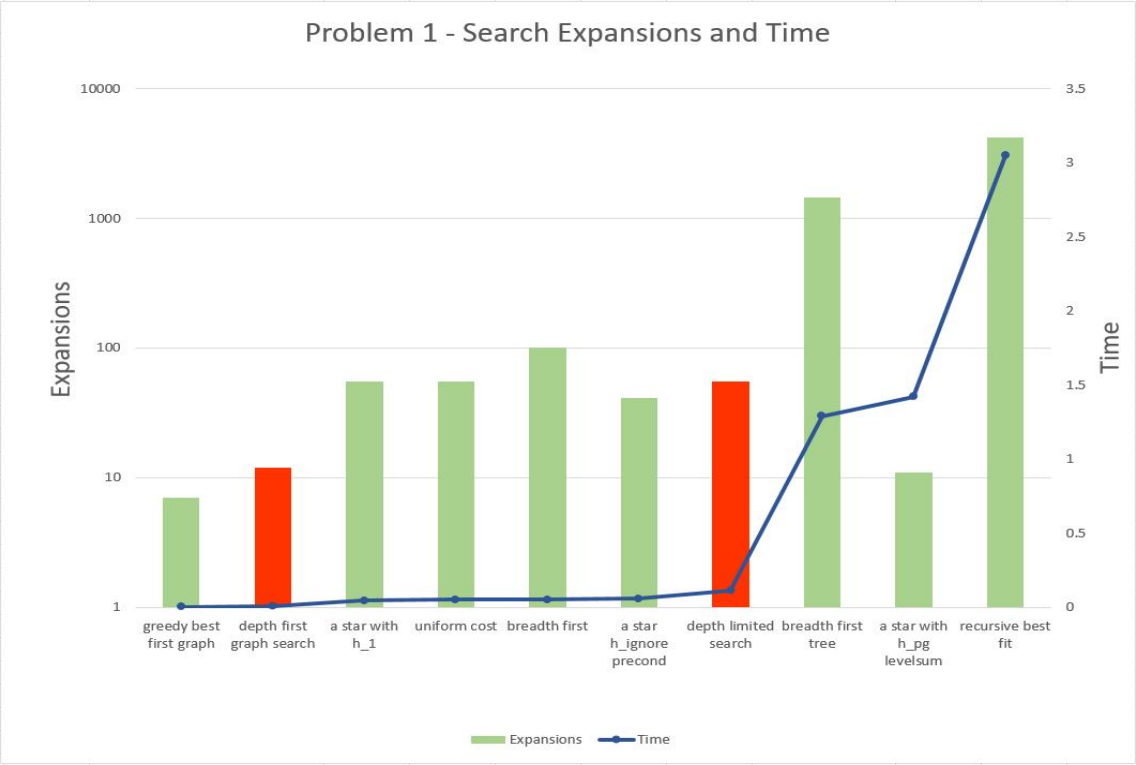
Problem 3 Optimal Sequence of actions

The optimal sequence of actions for problem 3 is:

Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
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(C1, P1, JFK)
UnLoad(C2, P2, SFO)

These actions were obtained from the "a* search with h_ignore preconditions". It had 5040 expansions, 5042 goal tests, and 44944 new nodes. Comparing the fastest time among those searches that returned the minimum number of steps it had the fastest time of 12.15 seconds.

Search Results Comparisons



Non-Heuristic, Uninformed Comparisons Problem 1

Search description	Time Complexity	time	Node expansions	Plan Length
breadth first	O(b^d)	0.114	43	6
breadth first tree	O(b^d)	1.29	1458	6
depth first graph search	O(b^max_d)	0.01	12	12
depth limited search	O(b^ d_lim)	0.113	101	50
uniform cost	O(b^(1+c/eps))	0.053	55	6
recursive best first	O(b^d)	3.05	4229	6

Comparing the depth limited search and depth first graph search, the graph search is complete, because it avoids redundant paths and visited nodes, whereas the tree search is susceptible to those issues. Additionally, depth first limited search can also be incomplete if the limit used is greater than the depth. Depth first graph search performed better time wise, than the tree version because it expanded fewer nodes (12 vs 55) and did not revisit nodes. Neither of the depth searches produces the optimal sequence of actions (12 & 55 respectively, shown in red on above graph).

Comparing breadth first and breadth first tree search shows that, while both searches are complete, the tree version performed time wise, much worse than the breadth first, and breadth first tree expanded many more nodes(1458 vs 43) to arrive at the same plan length.. The solutions for breadth first can be misleading if just looking at Problem 1 independently. Breadth first tree is

exponential complexity and and in problems 2 and 3, that exponential cost is excessive, but for problem 1, it is small enough to complete in reasonable time and space.

Out of all the uninformed searches, 'uniform cost' performed the best in time and space, having expanded the fewest nodes, and providing the lowest plan length. The uniform cost search is like the breadth first search except it expands the lowest cost branch first, using a priority queue to store the search frontier.

Non-Heuristic, Informed Comparisons

Search description	Time Complexity	time	Node expansions	Plan Length
greedy best first graph	O(b^d)	0.005	7	6
a star with h_1	O(b^d)	0.044	55	6
a star h_ignore precond	O(b^max_d)	0.057	41	6
a star with h_pg levelsum	O(b^ d_lev)	1.42	11	6

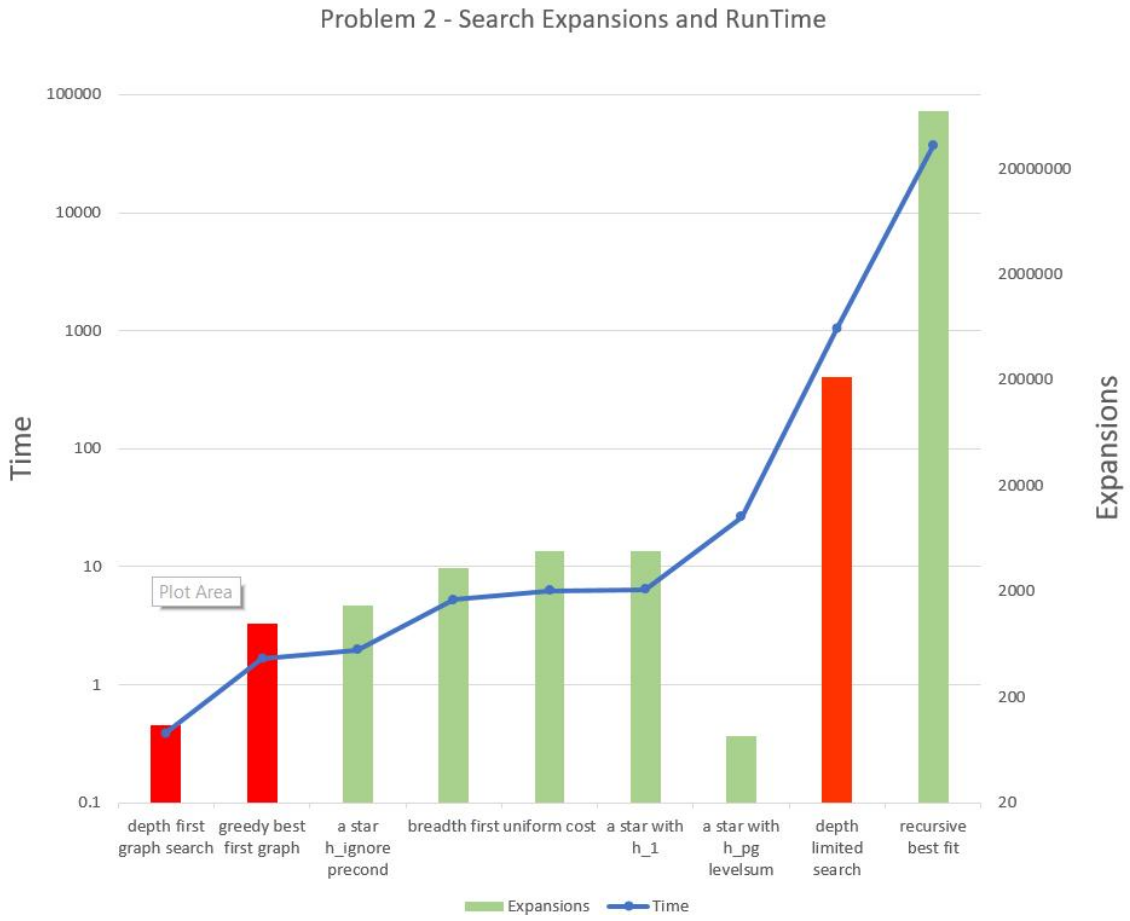
The informed searches all returned the optimal sequence of actions for problem 1(length 6). Greedy best first graph search returned the optimal sequence faster than all the a* searches, with only 7 node expansions. Greedy best first graph search also performed better than all search algorithms, including the uninformed searches, but as you will notice in problems 2 and 3, this is only the case for problem 1. Greedy best first search expands nodes closest to the goal, and in small problems, this is advantageous because a goal test can be positively performed with relative certainty that your final goal is not some local minima.

Comparing A* ignore preconditions with A* levelsum, shows that while both reach the goal, levelsum does this with fewer node expansions, but has a longer time complexity. A* search is optimally efficient, provided the heuristic used is consistent. This means that no other search algorithm that expands all nodes will expand less than A* Search. A* with Levelsum uses a planning graph heuristic, and levelsum utilizes the planning graph levels, expanding until it is 'leveled-off' (2 consecutive levels are identical). Although A* Levelsum expands the fewest nodes (of those all-node expanding searches) it does so at a higher time complexity cost for the planning graph computations, and this makes it's run time longer.

Best Heuristic Problem 1

The overall best heuristic for problem 1 is greedy best first graph search, and this is in part due to the small set of requirements in problem 1. If all the problems searched were guaranteed to have equal or fewer search criteria, it would be a fine choice, however as the problem sets get bigger, such as in Problems 2 and 3, greedy best first graph search does not hold up.

Non-Heuristic, Uninformed Comparisons Problem 2



Uninformed Comparisons Problem 2

PROBLEM 2						
1	breadth first	5.27	3346	4612	30534	9
*2	breadth first tree	--	--	--	--	--
3	depth first graph search	0.394	107	108	959	105
4	depth limited search	1062.37	213491	1967093	1967471	50
5	uniform cost	6.291	4852	4854	44030	9
6	recursive best first	36872.28	70664634	70664653	637813875	9

Breadth first tree search did not finish, this is not entirely unexpected as breadth first expands all nodes at every node, with a time and space complexity of O(b^d), and for example, problem 2 has 36 branches and at depth 6, gives a complexity calculation of 2.2E9. The solution requires more depth to solve than 6.

Contrasting breadth first with depth first graph search, depth first graph returned a goal in the quickest amount of time. This is because it powered its way down the tree (again, depth graph does not have repeats), and returned the first goal solution it finds, this returned a goal of 105, clearly much worse than the optimal sequence of actions in other searches of 9. Depth limited

search suffered the fate of revisiting nodes, and had a larger node expansion value of 1,967,471 and a processing time of 1,062 seconds, making it both inaccurate and slow.

Breadth first graph search returned the optimal action sequence in the fastest amount of time, so considering all uninformed searches for problem 2, it is the winner, with uniform cost search coming close (5.27 sec vs 6.291 sec). breadth first graph search's advantage is that it avoids revisits and expands using a FIFO queue. It is an optimal search strategy, and with this dataset size, it performs the best (out of uninformed searches)

Informed Comparisons Problem 2

7	greedy best first graph	1.666	990	992	8910	21
8	a star with h_1	6.418	4852	4854	44030	9
9	a star h_ignore precond	1.964	1450	1452	13303	9
10	a star with h_pg levelsum	26.49	86	88	841	9

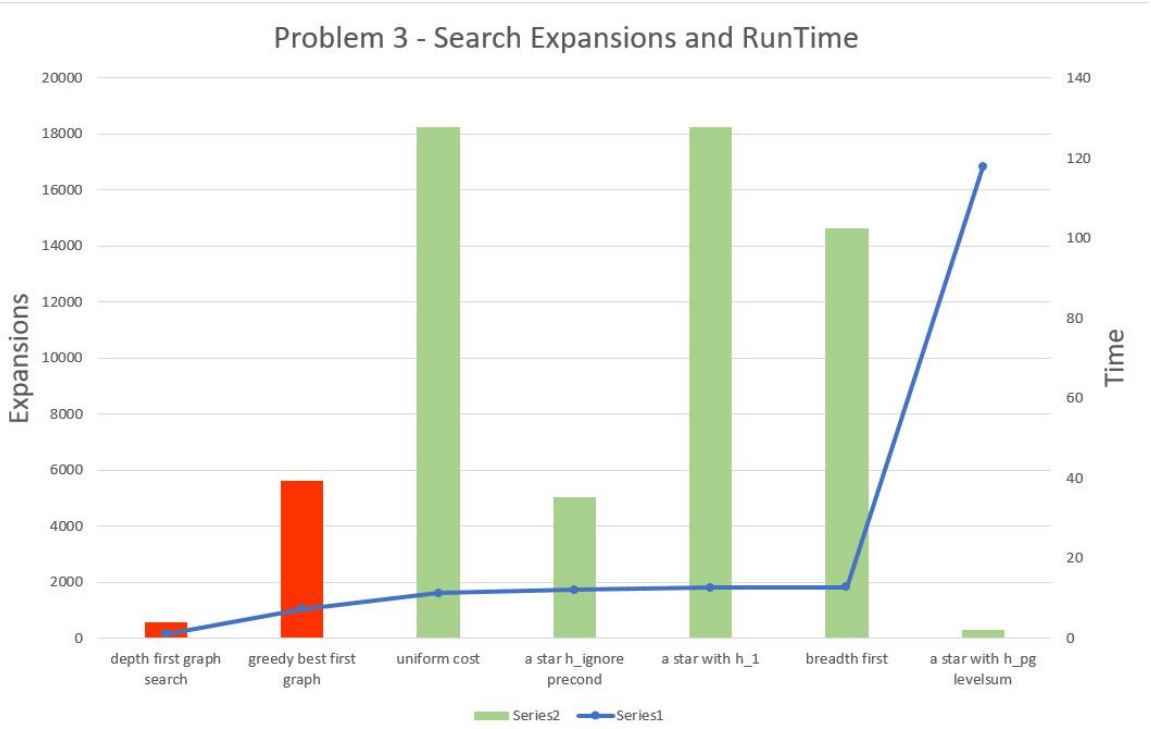
Greedy best first graph search had a suboptimal action sequence, but it's computational performance was the best at only 1.66 secs. Greedy best first search expands nodes closest to the goal, and this can lead to a local minimum condition, where the global minimum was actually in an entirely different sequence of branches. If the length of steps to the goal is large, it can face multiple goals, and each one looks like a possible goal, not a possible optimal goal.

Comparing the A* searches, A* with ignored preconditions performed best out of both the informed and uninformed searches. This is primarily because ignoring the preconditions will drastically reduce the problems data/node requirements. A* levelsum returned an optimal sequence of actions, but performed computationally poor. Even though the number of expansions was a low 86, the constraint computations for the planning graph are computationally expensive. A* using a planning graph and GRAPHPLAN may perform better, but was not used in this project.

Best Heuristic Problem 2

Considering both informed and uninformed searches for problem 2, A* ignore preconditions, faired the best. It returned the optimal sequence of actions, in the fastest time (1.96 secs). This only holds true for this air cargo problem, problems in other domains may not return optimal paths if you ignore preconditions.

Non-Heuristic, Uninformed Comparisons Problem 3



Uninformed Comparisons Problem 3

PROBLEM 3						
1	breadth first	12.81	14633	18098	129631	12
*2	breadth first tree	--	--	--	--	--
3	depth first graph search	1.066	592	593	4927	571
*4	depth limited search	--	--	--	--	--
5	uniform cost	11.257	18236	18238	159726	12
*6	recursive best first	--	--	--	--	--

Problem 3 had a large branching factor (the b, in b^d), making the breadth first tree, depth limited, and recursive best fit, either not finish in 2 days or run out of memory. Breadth first suffered from too many branches, and depth limited search suffered from both too many branches as well as node re-visitation. Recursive best first has space complexity that is linear in the depth of the deepest optimal solution, but suffers from the same problem as depth first tree, re-visitation, leading to exponential time complexity. Thus, none of these finished.

Depth first graph search finished quickly at 1.06 seconds, but as with previous problems gave a very poor sequence of actions (512 verses 12 optimal). Contrasting that to breadth first graph, breadth first gave an optimal sequence of actions, in a decent time (12.81 secs). This better result of breadth first graph is again due to FIFO queue and non-re-visitation of nodes.

Uniform cost search performed the best out of the uninformed searches, performing only slightly better than breadth first graph search(11.26 secs vs 12.81 secs).

Informed Comparisons Problem 3

7	greedy best first graph	7.338	5623	5625	49495	21
8	a star with h_1	12.672	18236	18238	159726	12
9	a star h_ignore precond	12.16	5040	5042	44944	12
10	a star with h_pg levelsum	117.815	318	320	2934	12

Again, 'greedy best first graph' search had a suboptimal action sequence, but it's computational performance was the best at only 7.34 secs. Similar to problem 2 above, greedy best first search expands nodes closest to the goal, and this can lead to a local minimum condition, where the global minimum was in an entirely different sequence of branches.

All the A* searches, (A* with h1 – A* ignore precond – A* levelsum), returned an optimal sequence of actions. And similar to problems above, the number of expanded nodes for A* levelsum was very low (318 vs 18236), and again computing the constraints for the planning graph proved to be very expensive computationally.

Best Heuristic Problem 3

Like problem 2 above, both informed and uninformed searches for problem 2, A* ignore preconditions, faired the best. It returned the optimal sequence of actions, in the fastest time (12.16 secs).

Overall Best Heuristic vs Non-Heuristic

Each problem had the following winning search algorithms for optimal action sequences:

Problem 1 - greedy best first graph search
Problem 2 - A* ignore preconditions
Problem 3 - A* ignore preconditions

Since there is a discrepancy for greedy best first graph search in problem 1, **the recommendation for the most optimal search is to use A* ignore preconditions**. The reason is that it performs the best for larger problem sets, and performs good enough for the small problem set. In using the same search algorithm, you avoid the injecting complexity into the program architecture (having to check for problem length). Additionally, for Problem 1 A* with ignored preconditions performed very well < .04 secs to calculate a solution.