Analysis of Search Algorithms

for AIND Planning Project by Oleg Polosin

Optimal Plans for Problems

	Problem 1	**Problem 2**	**Problem 3**
Plan	Load(C2, P2, JFK) Load(C1, P1, SFO) Fly(P2, JFK, SFO) Unload(C2, P2, SFO) Fly(P1, SFO, JFK) Unload(C1, P1, JFK)	Load(C2, P2, JFK) Load(C1, P1, SFO) 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)	Load(C2, P2, JFK) Load(C1, P1, SFO) 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)
Length	**6**	**9**	**12**

These optimal plans were retrieved using breadth-first search.

Comparison of Uninformed Search Algorithms

The following tables show a comparison between breadth-first search (BFS), depth-first search (DFS) and uniform-cost search (UCS) algorithms for three air cargo problems.

Problem 1:

Search Algorithm	**Node Expansions**	**Execution Time**	**Plan Length**
breadth_first_search	43	0.036	6
depth_first_graph_search	12	0.010	12
uniform_cost_search	55	0.042	6

Problem 2:

Search Algorithm	Node Expansions	Execution Time	Plan Length
breadth_first_search	3343	19.287	9
depth_first_graph_search	582	5.845	575
uniform_cost_search	4826	15.863	9

Problem 3:

Search Algorithm	Node Expansions	Execution Time	Plan Length
breadth_first_search	14663	152.112	12
depth_first_graph_search	627	4.919	596
uniform_cost_search	18221	72.114	12

As we can see, DFS is much faster than BFS or UCS, and it expands much less nodes, but its solution is not optimal. It happens because DFS goes to depth, constantly applying first available action to the current state. So it finally finds a combination of actions which leads to the goal, but that combination will almost never be an optimal. BFS and UCS explore every action level by level. It's the reason why they work slower, but always find an optimal solution. Also these algorithms are more space consumable, because they need to store a whole tree level in the frontier.

UCS and BFS algorithms are very similar, they have two main distinctions:

1. UCS uses priority queue instead of FIFO queue.
2. UCS checks the goal once it got the next state from the queue, while BFS checks the goal before adding successors to the queue.

UCS was designed for problems when a cost of actions are different and we need to find a plan with the lowest cost. BFS wouldn't find an optimal solution in that case, because it doesn't care about costs of actions.

For our problem an action cost for UCS is one, so UCS just becomes a BFS, and those distinctions just slow down a search:

1. Priority queue has a time complexity O(log(n)) for dequeuing minimal element, while FIFO queue does it for constant time.
2. UCS will expand more nodes than BFS.

But we see the opposite results in the tables above: UCS works faster than BFS. It happens because the AIMA code has a special realization of priority queue. They additionally use hash table data structure to quickly check that an element is presented in the queue (this functionality is actively used to check if some state is already in the frontier). Realization of FIFO queue has no this feature. After adding a hash table to the code of FIFO queue for constant time lookups, BFS search became much faster: execution time was decreased from 152s to 54s for the 3rd problem, what is faster than UCS, as expected.

Comparison of A* Search Heuristics

The following tables show a comparison between two heuristics of informed A* search algorithm: ignore preconditions heuristic (IPH) and level-sum heuristic (LSH).

Problem 1:

A* Heuristic	Node Expansions	Execution Time	Plan Length
h_ignore_preconditions	41	0.035	6
h_pg_levelsum	11	0.944	6

Problem 2:

A* Heuristic	Node Expansions	Execution Time	Plan Length
h_ignore_preconditions	1435	4.888	9
h_pg_levelsum	86	88.941	9

Problem 3:

A* Heuristic	Node Expansions	Execution Time	Plan Length
h_ignore_preconditions	5040	18.617	12
h_pg_levelsum	310	410.929	12

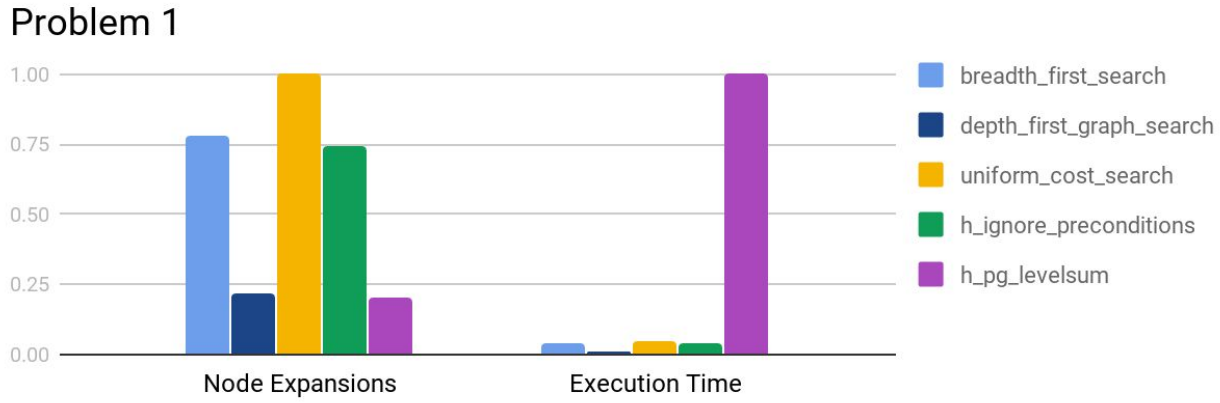
LSH and IPH are both admissible heuristics, they always find an optimal solution.

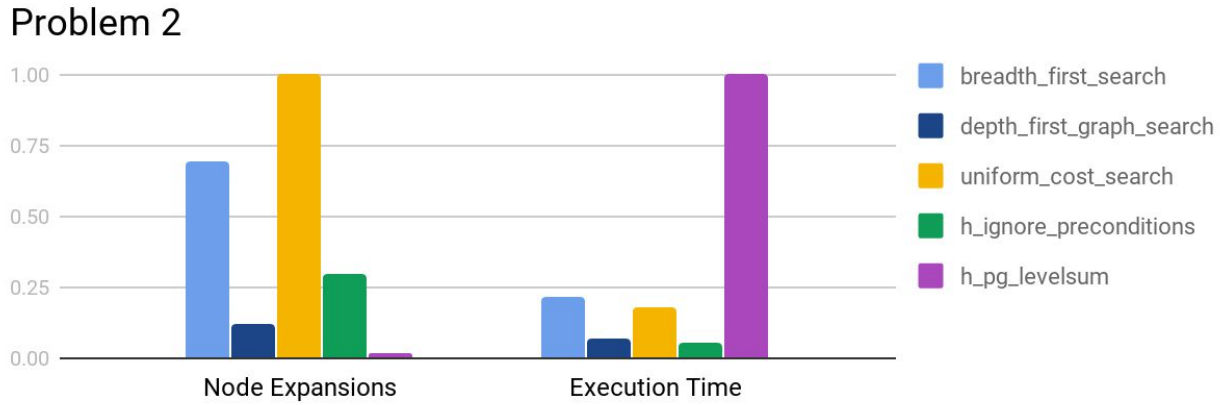
LSH is a very accurate heuristic, it expands much less nodes than IPH and therefore uses less memory. But this heuristic is very time consumable, because it generates a planning graph for each new state.

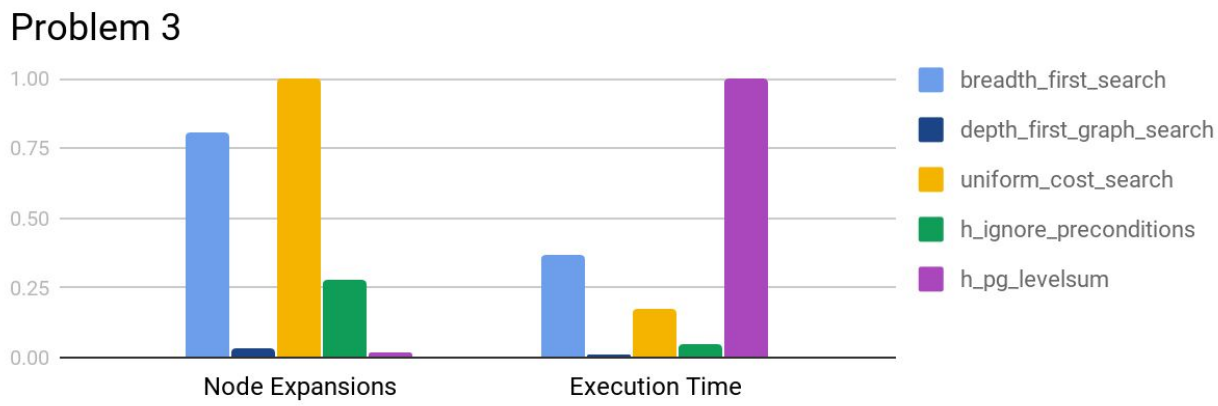
IPH is not so accurate, it uses more memory, but it's much faster. This heuristic relaxes a problem removing all preconditions from the actions. For our particular problems it means that we just need to count a number of unachieved goals for each new state, what is a very fast operation.

Conclusion

The following charts show a comparison between all considered search algorithms. All values were normalized from 0 to 1 to clearly show the differences between algorithms.







A* search algorithm with ignore preconditions heuristic gave us the best results. It outperforms other algorithms in speed and memory. We're not considering DFS algorithm, because it's not an optimal one, and LSH, because it's a very time consumable.

Informed search algorithms like A* help us to find the goal faster using heuristic function. Heuristic function estimates an approximate distance to the goal and therefore the direction of search. Admissible, accurate and cheap heuristic functions can drastically reduce the search space and execution time of the search.