Solving three basic air cargo planning problems using both uninformed and informed search with different heuristics

Felix Brei

April 17, 2017

Abstract

The air cargo problem is one of many classical benchmarks used for planning. Its domain consists of planes, airports and cargo packages which have to be transported to a specific airport. To measure the effectiveness of an approach one considers both the computation time and the amount of state space explored, which is equivalent to the number of nodes expanded in a search tree or graph for example. We will show three very basic uninformed search strategies and discuss them in terms of optimality. We will further use A* search with different heuristics and compare these results with the uninformed algorithms and show that adding information can save a lot of time in the planning domain.

1 Search algorithms used

1.1 Breadth first search

The first algorithm we used is breadth first search. Starting from the initial state it looks at all neighbors of a state first before expanding the state to generate new ones. Since it does not know which actions actually lead us closer to a goal state, it can be called undirected, because some of the actions may lead us farther away from a goal. It is also somewhat systematic because it only increases the area it searches if no goal can be found within its current limitations. Therefore, if the cost of all actions are equal it is guaranteed to find an optimal solution in terms of amount of actions taken.

1.2 Depth first graph search

The second algorithm is called depth first graph search. This algorithm tries to reach out into the search space as deep as possible. It does so by expanding a state node and then choosing an arbitrary (but not random) successor to continue. This method is computationally very inexpensive but often yields plans that are much longer than the optimal plan. This is because DFS always

expands a plan by one more action if it didn't reach a goal, instead of checking alternative actions at the same level.

1.3 Uniform cost search

The third uninformed search algorithm is called Uniform cost search, which is a variation of breadth first search. Instead of optimizing for the number of actions taken, it allows us to assign specific costs to an action. The algorithm always expands a version of the (still incomplete) plan that has the lowest total cost. Since this variant of cost based search assigns equal costs to each action, it does not differ from usual BFS. The difference in performance that we will see later comes from the fact that BFS checks a node's successors for representing a goal state, whereas UCS checks the node itself which results in a delayed recognition of a goal.

2 Heuristics

2.1 Ignore Preconditions

The first heuristic that was used in A* search is called 'Ignore Preconditions'. It tries to estimate the distance to a goal state by solving a relaxed version of the problem, where actions have no preconditions which means that actions can be done even if their preconditions are not fulfilled.

One can easily see that by ignoring preconditions it is possible to achieve every cargo transportation goal in no more than three steps: fly to the destination airport, load the cargo piece (it does not matter if it is actually there) and unload it. Therefore, the 'Ignore Preconditions' heuristic is roughly equivalent to the number of cargos that are not at there target location.

However, since the longest plan to move a cargo piece to its destination is to fly to its current location, pick it up and then fly to the target location, this heuristic still performs well.

2.2 PlanGraph Levelsum

This heuristic utilizes a plan graph to calculate a decent heuristic. Roughly speaking, a plan graph is a representation of states that are possible to reach within a certain distance from a starting point. As these levels of distances are not actually states, but rather sets of possible states, they are often referred to as *belief states*. The Levelsum heuristic takes a planning graph and an initial state and calculates the sum of distances of every part of the goal state from the initial state. For example, if the goal state contains A and B and A is true for the first time in level n and B is true for the first time in level m, than the Levelsum heuristic returns n+m. In other words, if A can be made true with at least n and B can be made true with at least m actions (both starting from the initial state), then A and B can be made true both at the same time with at least n+m actions. If the sub goals are independet, that is, each part of the goal

can be achieved independently of the other parts, than the Levelsum heuristic yields very accurate results.

3 Results

All algorithm were compared in terms of number of nodes expanded, length of the final plan and computation time. The results are shown in table 1 and figure 1 and 2 and discussed in the following sections.

		BFS	**DFS**	**UCS**	**A* IP**	**A* LS**
Problem 1	Nodes expanded	43	12	55	41	11
	Plan length	6	12	6	6	6
	time in seconds	0.0151	0.0042	0.0216	0.0192	1.88
Problem 2	Nodes expanded	3343	582	4852	1506	86
	Plan length	9	582	9	9	9
	time in seconds	10.81	2.68	40.97	11.62	183.86
Problem 3	Nodes expanded	14663	627	18235	5118	408
	Plan length	12	596	12	12	12
	time in seconds	88.54	2.64	425.19	83.34	1509.89

Table 1: Nodes expanded and time consumed for uninformed searches

Problem 1	**Problem 2**	**Problem 3**
Load(C1,P1,SFO)	Load(C1,P1,SFO)	Load(C2, P2, JFK)
Fly(P1,SFO,JFK)	Fly(P1,SFO,JFK)	Fly(P2, JFK, ORD)
Load(C2,P2,JFK)	Load(C2,P2,JFK)	Load(C4, P2, ORD)
Fly(P2,JFK,SFO)	Fly(P2,JFK,SFO)	Fly(P2, ORD, SFO)
Unload(C1,P1,JFK)	Load(C3,P3,ATL)	Unload(C4, P2, SFO)
Unload(C2,P2,SFO)	Fly(P3,ATL,SFO)	Load(C1, P1, SFO)
	Unload(C3,P3,SFO)	Fly(P1, SFO, ATL)
	Unload(C1,P1,JFK)	Load(C3, P1, ATL)
	Unload(C2,P2,SFO)	Fly(P1, ATL, JFK)
		Unload(C3, P1, JFK)
		Unload(C1, P1, JFK)
		Unload(C2, P2, SFO)

Table 2: Optimal plans taken from A* LS

3.1 Uninformed search

Of all uninformed search algorithms that were tested, BFS performs best. It is always able to find the shortest plan because all actions have equal cost. DFS may be better in terms of computation time but the length of the plan is beyond comparison and it is quite obvious that DFS will only find the shortest plan (or

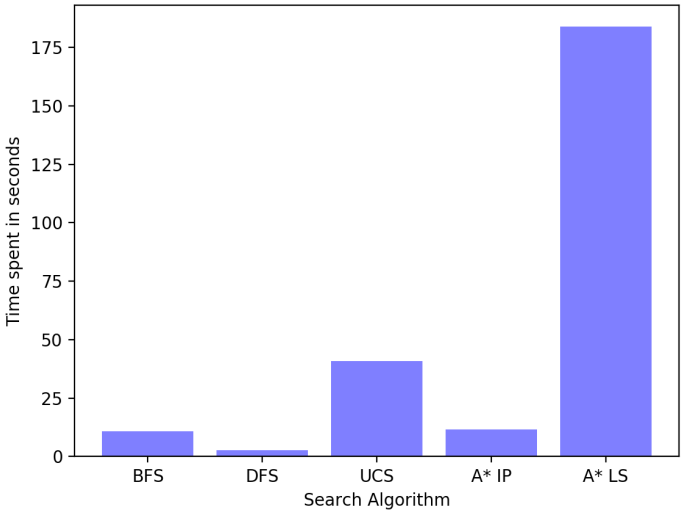

Figure 1: Time spent at solving problem 2

at least something that is close to it) if it gets lucky and hits a goal state. BFS beats UCS too because of the way it was implemented. BFS checks its children before expanding any further, UCS checks the state itself which means that BFS and UCS may find a plan with the same length, but UCS needs to go one level deeper into the search tree to recognize this. These results may vary greatly if the costs of actions are not equal, but in this domain BFS does a great job.

3.2 Informed search

As stated at the very beginning, search algorithms can benefit greatly from added information. Even a simple heuristic like 'Ignore Preconditions' can help guide the search into the right direction, which results in a reduction of expanded nodes by almost two thirds (Problem 3). It is very interesting though, that calculating even this simple heuristic takes up so much time that both BFS and A* IP end up terminating after roughly the same amount of time. This gets even more extreme when comparing BFS to A* LS. The Levelsum heuristic is so powerful that A* LS expands less than 3% of the nodes that BFS expanded (Problem 3). But since the construction of a plan graph takes a lot of time, the comtutation time of A* LS is off the charts and cannot compete with BFS and A* IP.

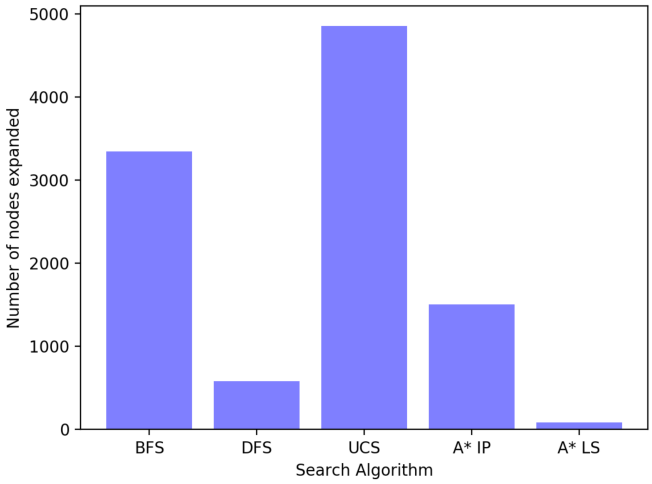

Figure 2: Nodes expanded while solving problem 2

4 Conclusions

Heuristics have proven themselves again to be a very powerful tool in the domain of search and planning. Although small domains can be solved with uninformed methods like BFS and DFS because of their very limited search space, heuristics can help reduce the amount of states that have to be explored and therefore also help reduce memory usage while searching. Simple heuristics are powerful enough to cut down the search area by a great factor while still being computationally easy enough to not raise the total time compared to 'dumb' methods. More complex methods are even better at reducing the search space but computing them takes so much effort that the total time increases by a large factor.

The results clearly show that adding heuristics to search based problems is always a good idea and even classical ones like 'Ignore Preconditions' improve the results. If one wants to use more complex heuristics, esp. those based on plan graphs, one should put a lot of effort into an efficient implementation to make sure that the computation of the heuristic does not become a new bottleneck during search.