Carlos Conde
Domain-Independent Planner
Artificial Intelligence Nanodegree
September 21, 2017

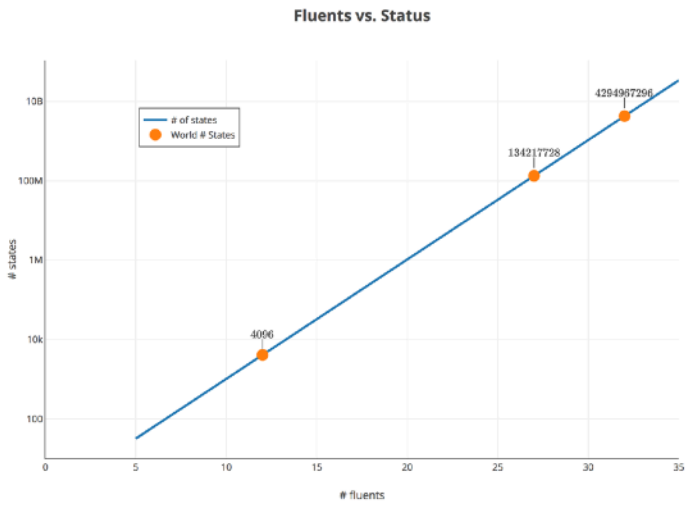
Search Algorithms/Heuristics Analysis

An analysis of the search algorithms and heuristics used as part of the Domain-Independent Planner built is presented in this report. Specifically, the planner is evaluated using the Air Cargo Problem also built as part of the assignment.

The analysis of the different techniques will be done mainly based on the Optimality, Space Complexity and Time Complexity of each of them. For doing so, the results of the following parameters will be collected and presented below: number of expanded nodes, number of goals tested, running time and whether an optimal solution was reached or not.

Environment

All the planning have been run locally on a MacBook Pro. The different scenarios that have been run are based on the Air Cargo Problem and 3 different instances with different number of fluents have been used to be able to compare how the number of total states impacts on the execution of the planners using different search algorithms. In the figure below the relationship between fluents and number of states in the world for each of the problems is shown:



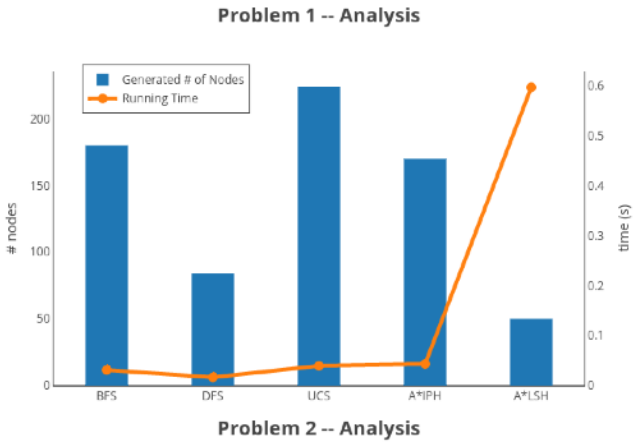
	Number of Fluents	Number of States in the World
P1	12	4096
P2	27	134217728
P3	32	4294967296

Note that the y-axis of the graph is presented in **logarithmic scale** for convenience as the number of states based on the fluents used to represent each problem grows exponentially.

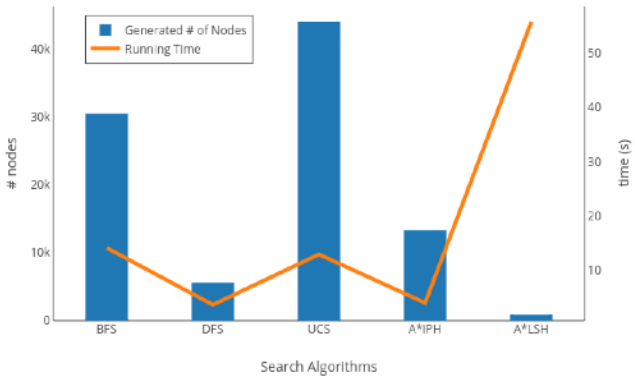
Results

In the analysis (see all the results int he following figures) 4 different search techniques will be used and 2 different heuristic functions. 3 scenarios will focus on comparing the performance of the search algorithms **Breath First Search Graph version (BFS)**, **Uniform Cost Search (UCS)** and **Depth First Search Graph version (DFS)** without using any heuristic function and 2 scenarios will be using **A*** as the search algorithm with **Ignore Preconditions Heuristic (IPH)** and **Level-sum Heuristic (LSH)**.

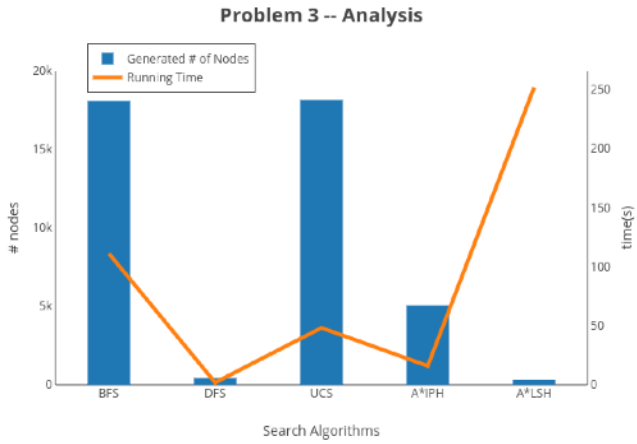
	Number of Expansions	Generated New Nodes	Tested Goals	Running Time (s)	Solution Length
BFS	43	180	56	0.03100	6
DFS	21	84	22	0.01643	20
UCS	55	224	57	0.03940	6
A*IPH	41	170	43	0.04319	6
A*LSH	11	50	13	0.59704	6



	Number of Expansions	Generated New Nodes	Tested Goals	Running Time (s)	Solution Length
BFS	3343	30509	4609	14.15923	9
DFS	624	5602	625	3.68264	619
UCS	4853	44041	4855	12.96310	9
A*IPH	1450	13303	1452	4.01679	9
A*LSH	86	841	88	55.73977	9



	Number of Expansions	Generated New Nodes	Tested Goals	Running Time (s)	Solution Length
BFS	14663	129631	18098	111.14997	12
DFS	408	3364	409	1.75791	392
UCS	18151	159038	18153	48.13875	12
A*IPH	5038	44926	5040	15.86623	12
A*LSH	311	2863	313	251.77981	12



Analysis

Based on the presented results, we proceed to analyze each of the algorithms individually in order to select the best combination of search algorithm and heuristic function to solve the Air Cargo Problem for a range of different number of fluents.

DFS

At a first glance, DFS seems to be providing the best numbers regarding Time Complexity and even Space Complexity. But this is mainly due to the algorithm not providing Optimality. Non optimality of DFS is due to the fact that the search algorithm expands the deepest node in the current frontier. So this algorithm may be visiting non-optimal solutions (deeper in the state tree) before the optimal and return the non-optimal.

This is what happened for the 3 problems, the solution found by DFS is the first solution the algorithm finds, even if its down deep in the state tree. This is why the running time and also number of nodes expanded are very low, as DFS will return as soon as it finds a solution.

BFS and UCS

We analyze both proposals together as their approach is almost the same with the only difference of the cost of each action, being a constant cost int he case of BFS, and having UCS being able to chose next node to expand based on non-constant cost of actions. The Air Cargo problem, as exposed in this exercise, has constant cost value actions. So we will be analyzing the difference between this 2 approaches based on that fact.

The results show that both options perform a similar number of goal tests. Both of them provide similar results when compared to the rest of algorithms used in this study, but the main difference between them is that UCS expands a bigger number of nodes. This is because in order to be able to take the action cost (when this is not the same for all actions) in consideration, UCS applies the goal test when a node is selected for expansion whereas BFS does it as soon as a node is generated and added to the frontier. This will make UCS expand more nodes before choosing a goal, but its needed to avoid choosing a suboptimal path when the action costs are not equal. So in the case of same cost actions, make sense to use BFS instead of UCS.
Why does it take longer to run BFS than UCS???

A* Search

The main idea of A* search is to avoid searching the complete state-space and it does so by searching until a specified depth level in the tree/graph and by applying an heuristic function that estimates how far each of the leaf nodes are from the goal.

In this analysis, we run A* Search in combination with 2 different heuristic functions: Ignore Preconditions Heuristic (IPH) and Level-sum Heuristic (LSH). Based on the results shown in the figures above we can clearly state that LSH, other than providing an optimal solution, it does it by expanding a very low number of nodes of the complete state-world of each Air Cargo problem instance, what makes it the best solution if we are considering Space Complexity.

The main drawback that A* + LSH has is the running-time as it can be seen in the results. The reason for this is that LSH uses a data structure called graphplan as an aid to estimate the costs of getting to the goal for each of the nodes where it's called from. This data structure, although buildable in polynomial time, is built multiple times during the execution of the algorithm.

To provide further insight on this, we have made an analysis of how much time this algorithm spends building the graphplan opposed to the time spent in calculating the level-sum of each leaf node. Results of this sub-analysis can be found in the figure below:

	Total Running Time	Time Spent on PG Creation	%	Time Spent on Computing level-sum
P1	0.5976	0.5550	**0.9288**	0.0331
P2	51.2489	49.6226	**0.9683**	1.3942
P3	244.1563	237.5710	**0.9730**	5.6318

For all of the problems, the A* with LSH spends between 92-97% of the time building the graphplan, and only a 3-8% of the time actually competing the level-sum. This makes as reach the following conclusion. If we could build the graphplan just once at the beginning of each problem instantiation, or at least reduce the number of time that the complete graphplan is created, this approach would not only be the one providing lowest Space Complexity but also one providing really good numbers for Time Complexity.

So under these conditions, and taking a look into the results of using A* + IPH now, we can conclude that IPH, an approach that doesn't make use of an additional data structure, provides a better balance results on Time and Space complexity, giving a very good result on time complexity and expanding a low number of nodes when compared to its non-heuristic based counterparts.

Conclusion

The main conclusion we are able to reach form this study is that among Non-Heuristic solutions we should chose between BFS over UCS (its not as efficient on same cost actions scenarios) and DFS (its not optimal).

Among Heuristic-based solutions we should choose A* with IPH heuristic as its a good balance between number of nodes generated (and thus the required space) and running time. Note that if we are able to generate the Planning Graph just once (instead of doing it every time that the heuristic is called), A* with

LSH would probably the best way forward as it require small amount of memory and the running time would be extremely reduced.