Adalaid

Mayumi

Which Heuristics did you use for the A* algorithm? We used a heuristic that tells us the number of misplaced containers in the stacks.

Test your program with a couple of different problems. Increase the size of the problem to test the limits of your program. Make a table comparing how many nodes are searched to find the answer for each problem. For this table, you should compare a number of different problems (at least 3) to avoid a statistical bias. Which of the four algorithms searches the least nodes and which one take the most?

	BFS	DFS	A* Cons	A* Incons
3 (C);(B);(A) (AC);(B);()	50	19	40	42
4 (A,B);(C);(D) (D);(C,B);(A)	26	40	324	244
5 (A,B);(C);(D,E) (B,E);(A);(C,D)	12,372	15	4982	5893

Deep first search searches the less nodes, however, it also didn't find a solution for the last two exercises. Breadth first search shines in the second test, but falls short in both the first and third. I'd say A* is the most efficient one in terms of larger problems.

Why does this happen?

Perhaps it's because of the heuristic. While it points you a certain way, it's only that, an indicator.

Which algorithms are optimal? Why?

By definition, BFS and A* are optimal. This is because of the quantity of nodes visited with each search. Our heuristic played a major role in the A* (of course the admissible one) and although it visited, at times, the most nodes, it always found the most optimal solution.

In your opinion, what are the benefits of simpler algorithms versus more complex ones?

I'd say that simpler algorithms are easier to both implement and understand. They do not require that much time to implement, and are often better when it comes to simpler problems. However, complex algorithms are needed for when the load it too great for the simpler algorithms to bear. However, simpler algorithms will always be the stepping stone needed in the path of learning.