Calibron-12 Write Up

Trishal Muthan

December 15th, 2020

For the Calibron-12 Modeling Challenge assignment, I have decided to declare failure. I felt that this project was considerably more difficult to model and build an AI around compared to the Rush Hour assignment from Unit 1. All the different orientations of the pieces, all the different possible cases, and more made modeling the Calibron-12 Puzzle and building an AI using things like constraint satisfaction much harder than Rush Hour which was a bit simpler to wrap your head around. That being said, I tried to apply my knowledge to the best of my ability for the one hour after reading and planning and here is what I was able to come up with:

1 What I tried to do

My initial goal was to build something similar to the techniques we used in the Sudoku assignment. I wanted to first build a recursive algorithm, then weave in the constraint propagation and forward-looking techniques that we learned during Sudoku 2. I didn't have a complete idea of how to put both the constraint propagation and forward-looking to use or how to implement that for this specific puzzle, but I figured that I would figure it out while I went. First, I tried to implement the first step delineated in the instructions: to check if the sum of the areas of the little rectangles equates to the whole area. This step was relatively straightforward and I'll talk about the specifics more in the next section. The next step was to model the board/puzzle. I did this through a matrix that was the size and height given as command-line arguments. Next, I attempted to put together the recursive part of the algorithm. This ended up being much harder than I thought. I planned to first build it using brute-force and then implementing the actual AI parts afterward. I first sorted the given rectangles in ascending order according to the area. I passed this into the call of the recursive method. The method tried went through and placed every single rectangle in both vertical and horizontal positions at the position that was closest to the top-left corner. Then slowly and surely, for every rectangle that was placed, you would call the recursive method again on the rest of the rectangles on the new board. This method was very similar to our initial method in Sudoku and NQueens: simple backtracking. That was the extent I was able to get to and I'll describe what the code did next.

2 What it actually does

The first part of my code focuses on that first step I mentioned before: to check if the sum of the areas of the little rectangles equates to the whole area. To do this, I simply made a method called "correctlySized" which would take the height, width, and list of rectangles, go through all the rectangles in the list, multiply their dimensions, keep track of the sum of these products, and check at the end if the sum was equal to the product of the initial width and height. Pretty straightforward. After checking for a valid puzzle, the code tries to get the recursive steps to work. I instantiate a board of the height and width given as command-line arguments, initially set to all False values. I also make a variable called position which holds a tuple indicating the current position (the closest to the top-left corner). I then made a copy of the list of rectangles and sorted it according to the area as I mentioned before. Then I passed everything into the recursive function. The recursive function itself, after goal checking, goes through every rectangle in the sorted list, places the rectangle vertically in the current position (if valid), and if so, recursively calls the method again with the updated board, the updated position, and the updated list of sorted rectangles. The same thing is done to the horizontally oriented rectangle. This is pretty much where I ended.

3 Why it doesn't work

I think the main problem was that there was some issue in the logic of the recursion that either caused an infinite loop or just messed up somewhere making it take an excessive amount of time. I wasn't able to figure out exactly what was making the error in the code in the allotted time but that is the general area of the error. Even with this error fleshed out, it would probably still take a lot more time than I would want it to so I would have to implement some AI into it to make it faster.

4 How I would move forward

First, I would have to fix the issue I was facing in the logic of the recursive backtracking portion which I had tried thus far. After that was fixed, I would have to implement the forward looking and constraint propagation (both similar to the Sudoku assignment) so that I would be able to increase efficiency and attain significantly faster run times.