Writeup

Implementation

Our project implemented two algorithms, backtracking and arc-consistency, to solve a Sudoku puzzle. A Sudoku puzzle is a 9 by 9 grid with some of the cells filled with numbers and the rest left blank. The goal is to fill in the remaining cells of the puzzle with numbers such that a number (1-9) isn't repeated in that row, column, or smaller 3 by 3 grid.

Design

There are seven classes in our program, each providing a specific purpose. We have one class each for backtracking and arc-consistency algorithms, one to read the sudoku file, one that allows you to import the file paths, one to create constraints for different variables, one to manage variables, domains, and constraints in CSP, and one that inherits constraints from the constraint class. The backtracking class implements the backtracking algorithm to solve the puzzle by trying each number in each empty cell and then backtrack to the cell before the current cell if it reaches the end and/or a conflict occurs. The arc-consistency class implements the ac3 algorithm, which includes methods to load the puzzle, initialize domains and the queue, get constraints, get neighbors, and run the algorithm itself. The concrete constraint class implements specific types of constraints for the Sudoku problem to ensure that all rows, columns, and grids in the Sudoku puzzle have unique values. The abstract class allows you to import the file paths. The constraint class is an abstract base class that represents a general constraint which defines a template for specific constraints that can be implemented for various problems. One particular challenge faced in this class was understanding the algorithm itself, as well as how to turn that understanding into code. The class to read the sudoku file is particularly simple, with a single method to take in a file path and read each line in that file. Then, it maps what it reads from that file into an array stored in the variable puzzle, which is then returned. Figuring out all the necessary functions in the CSP class was a bit hard but the helper code made it a lot easier. This class includes methods to add constraints, check the consistency of an assignment, and determine if an assignment is complete.

Problem Setting

Our project addresses the problem of solving a Sudoku puzzle, a popular yet basic number game in the real world that’s explained briefly in the implementation section. We modeled this problem formally through an array of numbers, where the number 0 or any other non alpha-numeric value

represents the empty spaces in the sudoku grid. This allowed our algorithms to effectively read the grid and solve the problem.

Experimental Setup

For our experiments, we created two text files containing Sudoku grids (for reference look at files sudoku1.txt and sudoku2.txt in the test folder) as inputs. The expected output for both algorithms was a filled Sudoku grid with the correct numbers in place. To measure performance, we measured the time taken to output the solution for each Sudoku problem.

Results

For our results, we observed that the arc consistency algorithm solved the Sudoku problems quickly, most likely due to its ability to eliminate invalid paths early, preventing exploration of unnecessary solutions. We did make edits in the folders after, since we had to ensure it would work for both algorithms, but couldn't test that for correctness due to technical issues, mostly due to pandas. We went through the entire code and determined that if both algorithms work properly, it should return a 2d array where any number zero is replaced with the right value for the sudoku to work.

Contributions

In our group project, Emily wrote the backtracking algorithm, the concrete constraint class, the CSP class, and the constraint class(includes the abstract method), while Divita wrote test files, consistency algorithm, the sudoku reader, and the abstract class for testing the files. Both of us communicated a lot to bounce ideas and get some help from each other as well.