A Non-Random, Programmable Solution to Sudoku

Joshua Herrera

EE 3223: C++ and Data Structures

**Introduction**

The following paper describes the concepts and program design used to solve the standard, 9x9 version of Sudoku. There are many well-known methods to solving a Sudoku board programmably, such as backtracking and dancing-links. However, these methods involve a guess and check approach which, by definition, forces a program to do “extra” work to find a solution, where extra is taken to be any guess that isn’t included in the final solution. It is the goal of this program to do away with any guess work involved in Sudoku solving, meaning there will be no unnecessary backtracking or state saves involved in solving the board.

**Concepts**

Before approaching the implementation of code for this Sudoku solver, basic groundwork for the concepts used should be established. All methods used for solving are based on a concept the programmer has called “Extrapolating”. Extrapolation takes advantage of Sudoku’s rule that each row, column, and block (hereinafter referred to as units) may have only one of each integer (1-9) contained within them. In the process of extrapolation, the original base board (hereinafter referred to as the base layer) containing integers is processed into 9 distinct layers (hereinafter referred to as sub-layers), where a layer is any 9x9 Sudoku-like board. These 9 layers are identified with integers 1 through 9 where each layer contains information about its integer identifier. The contents of the sub-layers are 1s, 0s, and blanks. 1s correspond to cells which are occupied in the base layer by the integer identifier of the sub-layer. 0s represent cells which cannot be filled in the base layer by the sub-layer’s identifier. Blanks represent cells which can be filled by the sub-layer’s identifier, but have not been determined definitively yet.

In the initial extrapolation of each integer to its sub-layer, cells containing the sub-layer’s identifying integer are filled with 1s. Any units containing a 1 are then filled with 0s. Additionally, any cells that are already populated with other integers in the base layer are also filled with 0s. Figure 1 illustrates this initial extrapolation with Layer 9. Ideally, this extrapolation results in individual cells within a layer that are the only open cell in their units, meaning the sub-layer’s identifying integer must fill that spot in the base board. You can see this with the red 1 in figure 1. After filling in this 1, more 0s are placed into the sub-layer’s units. Additionally, other sub-layers also receive a 0 where the new integer was placed. Simple Sudoku boards can be solved after several iterations using this method.

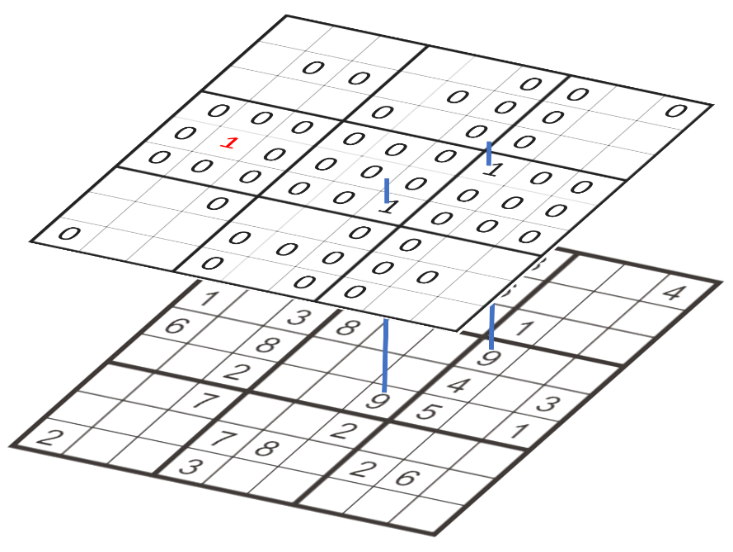


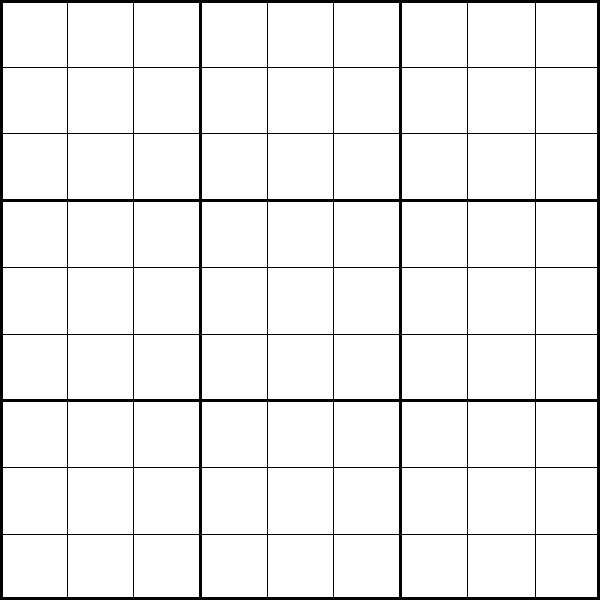
Figure : Extrapolation of Layer 9

However, harder boards require additional methods to find a solution. One of these methods takes advantage of the fact that a block must contain at least one of each integer. Specifically, if there exists a block in a sub-layer which has 2 or 3 cells blank with those cells consolidated into a single row or column, every other item in that row or column must not be a 1, as that would remove all blanks from the block rendering the board incomplete.

The last method implemented in the code is the most complicated. Instead of looking for a single cell within a single sub-layer that can be filled, it searches the units of two sub-layers for a unit that is identical between the two layers and has only two items missing. If found, because the two identical units in these two sub-layers must each have a 1 in one of the empty cells, those cells can be zeroed out in other sub-layers (effectively reserving the cells for the two sub-layers). However, because you still cannot determine which sub-layer has a 1 in each of the cells, you cannot place any 1s and must rely on cells that become singled out because of zeroing out in the other sub-layers.

**Implementation**

The first part of code implementation lays out a basic structure for interacting with the different layers. As such, there are 4 basic classes: board, layers, units, and items. The board class contains 10 layers; the base layer and 9 sub-layers. Each layer then contains 3 arrays of pointers to units, one array each for rows, columns, and blocks. Units then contain pointers to items, which have been referred to as cells up to this point. Heavy utilization of pointers was made to ensure the ability to modify data without having to duplicate operations due to the separation of units, as well as to ensure continuity in layer composition. Unit items were linked early in the initial extrapolation of the board. Traversing through a unit’s items is done as shown in figure 2. Rows are left to right from 0 to 8, columns are top to bottom from 0 to 8, and blocks are left to right and top to bottom from 0 to 8.



0

0

2

3

4

5

6

7

8

9

1

2

3

4

5

6

7

8

1

2

3

4

5

6

7

8

Figure : Unit Traversal

The most difficult part of implementation was turning the concepts detailed above into programmable algorithms. “Blank” cells were filled with -1s in the program to facilitate arithmetic done to calculate unit composition. Because of the non-guessing, structured methods used to achieve a solution, error checking was not necessary as a separate part of the program. By definition, the methods only allowed for a correct filling of the board.

**Conclusion**

As ambitious as the program is, it is still unable to find the solution to some of the hardest sudoku puzzles. These puzzles would require more methods than the 3 implemented in this program to converge to a solution without use of a brute force solution.

Some improvements that could be made in the programs current state include removing redundant check and fill operations. As it stands, the program does not make any effort to fill only when necessary as it keeps no record of what units and sub-layers have been completed; so, there are many unnecessary passes over completed data.

With more time, and perhaps a team to help develop more solving methods, this program will provide solutions to all sudoku puzzles without brute-forcing or guessing for a solution.

Github link:

https://github.com/sirebellum/Cpp\_Spring2017/tree/master/Final%20Project