Sudoku Solver Using Linear Programming

Sebastian Gracia, Tong Zhang, Ryan Wolanzky

December 6, 2019

1 Introduction

A Sudoku game is a logic based number placing puzzle. The player is given a 9x9 grid with a total of 81 cells. The objective of the puzzle is to have every cell by non-empty while satisfying all constraints. The constraints are as follows, every cell can only be filled with an integer 1-9, each row and each column can contain exactly one of each integer 1-9, and each 3x3 sub-matrix can only contain exactly one of each integer 1-9. Each puzzle is started with some cells of the grid already filled in such a manner that a well formed puzzle will have one unique solution.[**?**] The main focus of this project was to use the information from the paper and our own prior knowledge to convert this into a linear program.

We also adapted the method used in (put citation here) to solve a sudoku variation called Sudoku X. This variation follows all of the same constraints as normal sudoku, however has an added condition that both of the main diagonals must be filled with the numbers 1-9, with each number used exactly once.

2 Constraints as a linear system

Sudoku problem can be modeled as a sparse linear system of equations[**?**]. We can formulate the Sudoku ruleset as an underdetermined linear system and the objective as an l_0 norm minimization problem. For it is hard to solve an l_0 norm minimization problem in general, [**?**] relax the objective function to a l_1 norm minimization problem.

In this part, we will build a linear programming model and use this model to show the method in [**?**].

2.1 Notations

Notation	**Description**
N	Dimension of a Sudoku puzzle;
n	Number of cells in a Sudoku puzzle, $n = N^2$;
C	Number of clue cells in a Sudoku puzzle;
S_n	The number in each Sudoku puzzle cell, $S_n \in \{1, 2, ..., N\}$;
$I(S_n = k)$	Equals to 1 if $S_n = k$ is true, and 0 otherwise;
i_i	Binary vector for each cell in the Sudoku, $i \in \{1, 2, ..., n\}$;
x	An N^3 size long 0-1 vector;
b	An N^3 size long all-one vector;
A_{fixed}	A $4N^2 \times N^3$ matrix, indicates the constraints corresponding to every Sudoku;
A_{row}	An $N^2 \times N^3$ matrix, indicates the constraints corresponding to Sudoku rows;
A_{col}	An $N^2 \times N^3$ matrix, indicates the constraints corresponding to Sudoku columns;
A_{box}	An $N^2 \times N^3$ matrix, indicates the constraints corresponding to Sudoku boxes;
A_{cell}	An $N^2 \times N^3$ matrix, indicates the constraints corresponding to Sudoku cells;
A_{clue}	A $C \times N^3$ matrix, indicates the constraints corresponding to Sudoku clue cells;
A_{diag}	A $2N \times N^3$ matrix, indicates the constraints corresponding to main diagonals;

2.2 Formulation

$$\min_{x} \|x\|_1 \tag{1}$$

s.t.

$$Ax = \begin{bmatrix} \mathrm{A}_{fixed} \\ \mathrm{A}_{clue} \end{bmatrix} x = \begin{bmatrix} \mathrm{A}_{row} \\ \mathrm{A}_{col} \\ \mathrm{A}_{box} \\ \mathrm{A}_{cell} \\ \mathrm{A}_{clue} \end{bmatrix} x = b \tag{2}$$

$$x = \begin{bmatrix} \mathrm{i}_1 \\ \mathrm{i}_2 \\ \vdots \\ \mathrm{i}_n \end{bmatrix} = \begin{bmatrix} (\mathrm{I}(\mathrm{S}_1 = 1), I(S_1 = 2), \cdots, I(S_1 = N))^T \\ (\mathrm{I}(\mathrm{S}_2 = 1), I(S_2 = 2), \cdots, I(S_2 = N))^T \\ \vdots \\ (\mathrm{I}(\mathrm{S}_n = 1), I(S_n = 2), \cdots, I(S_n = N))^T \end{bmatrix} \tag{3}$$

$$A_{row} = \begin{bmatrix} \mathrm{I}_{N\times N} \mid I_{N\times N} \mid \cdots \mid I_{N\times N} \mid 0_{N\times(N^3-N^2)} \\ 0_{N\times N^2} \mid I_{N\times N} \mid I_{N\times N} \mid \cdots \mid I_{N\times N} \mid 0_{N\times(N^3-2N^2)} \\ 0_{N\times 2N^2} \mid I_{N\times N} \mid I_{N\times N} \mid \cdots \mid I_{N\times N} \mid 0_{N\times(N^3-3N^2)} \\ \vdots \\ 0_{N\times(N^3-N^2)} \mid I_{N\times N} \mid I_{N\times N} \mid \cdots \mid I_{N\times N} \end{bmatrix} \tag{4}$$

$$A_{col} = \begin{bmatrix} \mathrm{I}_{N\times N} \mid 0_{N\times(N^2-N)} \mid I_{N\times N} \mid 0_{N\times(N^2-N)} \mid \cdots \mid I_{N\times N} \mid 0_{N\times(N^2-N)} \\ 0_{N\times N} \mid I_{N\times N} \mid 0_{N\times(N^2-2N)} \mid \cdots \mid 0_{N\times N} \mid I_{N\times N} \mid 0_{N\times(N^2-2N)} \\ 0_{N\times 2N} \mid I_{N\times N} \mid 0_{N\times(N^2-3N)} \mid \cdots \mid 0_{N\times 2N} \mid I_{N\times N} \mid 0_{N\times(N^2-3N)} \\ \vdots \\ 0_{N\times(N^2-N)} \mid I_{N\times N} \mid 0_{N\times(N^2-N)} \mid I_{N\times N} \mid \cdots \mid 0_{N\times(N^2-N)} \mid I_{N\times N} \end{bmatrix} \tag{5}$$

$$J_{N\times N^{\frac{3}{2}}} = \begin{bmatrix} \mathrm{I}_{N\times N} \mid \cdots \mid I_{N\times N} \end{bmatrix} \tag{6}$$

$$A_{box} = \begin{bmatrix} \mathrm{J}_{N\times N^{\frac{3}{2}}} \mid 0_{N\times(N^2-N^{\frac{3}{2}})} \mid \cdots \mid J_{N\times N^{\frac{3}{2}}} \mid 0_{N\times(N^2-N^{\frac{3}{2}})} \mid 0_{N\times(N^3-N^{\frac{3}{2}})} \\ 0_{N\times N^{\frac{3}{2}}} \mid J_{N\times N^{\frac{3}{2}}} \mid 0_{N\times(N^2-2N^{\frac{3}{2}})} \mid \cdots \mid 0_{N\times N^{\frac{3}{2}}} \mid J_{N\times N^{\frac{3}{2}}} \mid 0_{N\times(N^2-2N^{\frac{3}{2}})} \mid 0_{N\times(N^3-N^{\frac{3}{2}})} \\ \vdots \\ 0_{N\times(N^3-N^{\frac{3}{2}})} \mid 0_{N\times(N^2-N^{\frac{3}{2}})} \mid J_{N\times N^{\frac{3}{2}}} \mid \cdots \mid 0_{N\times(N^2-N^{\frac{3}{2}})} \mid J_{N\times N^{\frac{3}{2}}} \end{bmatrix} \tag{7}$$

$$J_{1,N} = \begin{bmatrix} 1\ 1\ 1 \cdots 1 \end{bmatrix}_{1\times N} \tag{8}$$

$$A_{cell} = \begin{bmatrix} \mathrm{J}_{1,N} \mid 0_{1\times(N^3-N)} \\ 0_{1\times N} \mid J_{1,N} \mid 0_{1\times(N^3-2N)} \\ 0_{1\times 2N} \mid J_{1,N} \mid 0_{1\times(N^3-3N)} \\ \vdots \\ 0_{1\times(N^3-N)} \mid J_{1,N} \end{bmatrix} \tag{9}$$

$$A_{clue} = \begin{bmatrix} 0\cdots 0\ 1\ 0\ 0\ 0\ 0\ 0\cdots 0 \\ 0\ 0\ 0\cdots 0\ 0\ 1\ 0\ 0\cdots 0 \\ \vdots \\ 0\ 0\ 0\ 0\ 0\cdots 0\ 1\ 0\cdots 0 \end{bmatrix}_{C\times N^3} \tag{10}$$

$$A_{diag} = \begin{bmatrix} \mathrm{I}_{N\times N} \mid 0_{N\times N^2} \mid I_{N\times N} \mid 0_{N\times N^2} \mid\cdots\mid I_{N\times N} \mid 0_{N\times N^2} \mid\cdots\mid I_{N\times N} \\ 0_{N\times(N^2-N)} \mid I_{N\times N} \mid 0_{N\times(N^2-2N)} \mid 0_{N\times N} \mid\cdots\mid 0_{N\times(N^2-2N)} \mid I_{N\times N} \mid 0_{N\times(N^2-N)} \end{bmatrix} \tag{11}$$

$$Ax = \begin{bmatrix} \mathrm{A}_{fixed} \\ \mathrm{A}_{clue} \\ \mathrm{A}_{diag} \end{bmatrix} x = \begin{bmatrix} \mathrm{A}_{row} \\ \mathrm{A}_{col} \\ \mathrm{A}_{box} \\ \mathrm{A}_{cell} \\ \mathrm{A}_{clue} \\ \mathrm{A}_{diag} \end{bmatrix} x = b \tag{12}$$

Equation (1) to equation (10) are the model to solve normal Sudoku problems and equation (1) and (3) to (12) are used to solve Sudoku X.

Originally, it is proved by [**?**] that the x, which has the minimum number of nonzero elements, is the solution of the Sudoku. As we mentioned above, solving the l_0 norm minimization problem hard in general. Therefore, the objective function (1) is decided to use the method in [**?**], approximating the l_0 norm minimization problem by l_1 norm minimization problem. The approximation method can solve most of the Sudoku problems but not all.

Equation (2) is the main constrain for normal Sudoku linear programing. A_{fixed} is a combination of A_{row}, A_{col}, A_{box} and A_{cell}, which are the same for all Sudoku problems. Then we use (3) to (10) to explain the elements in (2).

x in (3) is an N^3 size long 0-1 vector grouped by every N number, denoting the solution in every cell in the Sudoku cell. This means in every group we have a size N long 0-1 vector and the kth element in anyone group is $I(S = k)$, which equals to 1 if $S = k$ is true, and 0 otherwise. For instance, if the number in the first cell is 4, then the vector for the first cell is $[0\ 0\ 0\ 1\ 0\cdots 0]$ with size N. Combining all the N^2 vectors we will get our x.

Equation (4) is the row constrain. In Sudoku problems, each row should contain exactly one of each integer 1-N. The first N rows in A_{row} are the constrains for the first row in Sudoku. $A_{row_{firstrow}}x$ should equal to one, indicating that in the first row there exist and only exist one number 1. $A_{row_{firstNrow}}x$ equals to a size N long all one vector, indicating that in the first row in Sudoku there exist number 1 to N and each number only appear once. There are N identical matrices in each row and others are 0 matrices.

Equation (5) is the column constrain. In Sudoku problems, each column should also contain exactly one of each integer 1-N. Similarly, the first N rows in A_{col} are the constrains for the first column in Sudoku. $A_{col_{firstcol}}x$ equals to one, means that in the first column number 1 appears and only appears once. There are also N identical matrices in each row and others are 0 matrices.

Equation (7) is the box constrain. Generally saying, each $N^{\frac{1}{2}} \times N^{\frac{1}{2}}$ sub-matrix can only contain exactly one of each integer 1-N. The first N rows in A_{box} are corresponding to the first box at the top left corner in Sudoku. For easier understanding, we give each cell a number. The first cell is cell No.1, the second cell in the first row is cell No.2, the first cell in the second row is cell No.$(N+1)$, etc. The last cell is cell No.(N^2). For the first box, our constrains guarantee the No.1 to 3 cells, No.$(N+1)$ to $(N+3)$ cells and No.$(2N+1)$ to $(2N+3)$ cells exist number

1 to N and each number only appear once. There are $N^{\frac{3}{2}}$ numbers of $J_{N\times N^{\frac{3}{2}}}$ matrices in each row and others are 0 matrices.

We also need to guarantee that each cell should be filled and should be filled by only one number. Therefore, we introduce equation (9) to our constrains. There are N^2 rows in A_{cell} and each row in A_{cell} is corresponding to each cell in Sudoku. i_i is the vector of each cell. This constrain can make sure, in each i_i, there are only one 1 and other $N-1$ elements are all 0.

Equation (10) is the constrain for clues. If there are C clues in the Sudoku, there will be C rows in A_{clue}. If the second clue is 3 in the No.4 cell, the second row of A_{clue} will be all zeros but the $(3N+3)th$ element will be 1.

For the Sudoku X problems, we add A_{diag} to the end so that the main constrain of the Sudoku X problems become to Equation (12) instead of equation (2).

A_{diag} is the constrain for requirements in Sudoku X. Both of the main diagonals must be filled with the numbers 1-9, with each number used exactly once. There is only $2N$ rows in A_{diag} for the 2 diagonals. For the first N rows in A_{diag}, the $N\times N$ identical matrices correspond to the cells on the main diagonal and 0 matrices correspond to other cells. For the second N rows in A_{diag}, only the first and the last 0 matrix is $N\times(N^2-N)$ size. Other 0 matrix are all $N\times(N^2-2N)$ size.

3 User's Guide

The program is fairly intuitive to use. On startup the solver prompts for a Sudoku puzzle to solve. The format that the code requires the input is as a string of numbers with no spaces representing the puzzle to be solved. Either 0 or . can be entered for a cell that is empty in the Sudoku puzzle being entered. The Sudoku string should be input in order as if the Sudoku was a matrix with the upper left hand corner being a_{ii}. It is recommended for 9×9 puzzles that the website QQWing be used to generate Sudoku as a string in the format can be easily generated. The supported Sudoku puzzles sizes are 4×4, 9×9, 16×16, ... etc. the program will prompt the user for a correct length string if an incorrect dimension is inputs. Then the program will ask if the entered Sudoku is a Sudoku X puzzle. Only hit yes if this is applicable. The program should then should solve the Sudoku and create a display for the 9×9 case.

4 Code Description

The bulk of the program is spent generating the constraint matrix corresponding to the Sudoku rules (row, column, box, clues, and any Sudoku variation rules). For any $N\times N$ Sudoku problem the constraints corresponding to the row, column, box, and cell rules remain exactly the same and are generated solely depending on the dimension N. This dimension is automatically obtained from your input, since the input will always have length N^2. The clue constraints are generated iteratively from the input as well. The code loops through all indices of the input such that the value at that index is not 0 or (.), that is, all boxes where a clue is given, and constructs a $1\times N^3$ constraint corresponding to that clue, then appends it to the clue constraint matrix. For the Sudoku X variation, the code also generates a $2N\times N^3$ matrix corresponding to the two additional diagonal constraints and appends it to the rest of the constraints corresponding to the normal Sudoku rules.

After the constraints are generated, the program uses the "linprog" function included in Matlab to minimize the $\|x\|_1$ over $x\in\mathrm{R}^{N^3}$. The solution found, as outlined in (cite) is the solution to the Sudoku problem, but is not easily readable as a solution.

So after the solution is found, the program constructs a $N^2\times N^3$ "translation" matrix , M, as detailed in (13), performs the matrix multiplication Mx, then takes the resulting vector and

converts it into an easily legible $N \times N$ matrix.

$$T = \begin{bmatrix} \mathrm{V} \mid 0_{N\times(N^3-N)} \\ 0_{N\times N} \mid V \mid 0_{N\times(N^3-2N^2)} \\ \vdots \\ 0_{N\times(N^3-2N^2)} \mid V \mid 0_{N\times N} \\ 0_{N\times(N^3-N)} \mid V \end{bmatrix} \tag{13}$$

Where V is given by (14)

$$V_{1\times N} = \begin{bmatrix} 1, 2, 3 \cdots N \end{bmatrix} \tag{14}$$

5 Algorithm Performance

Here is the performance of our algorithm on different types of Sudoku problems:

Difficulty	**Trials**	**Success Rate**
Easy	10	100%
Hard	10	100%
Total	**20**	**100%**

Table 2: Trials for 4x4 Sudoku

Difficulty	**Trials**	**Success Rate**
Easy	15	100%
Medium	15	100%
Hard	15	20%
Total	**45**	**73.33%**

Table 3: Trials for standard 9x9 Sudoku

Difficulty	**Trials**	**Success Rate**
Easy	8	75%
Medium	8	75%
Hard	8	100%
Total	**24**	**83.33%**

Table 4: Trials for Variant 9x9 Sudoku X

16×16 Sudoku problems are also tried. Due to our limitations inputting the 16×16 Sudoku problem, we were only able to run two tests on 16x16. But they were both solved successfully.

6 Conclusion

The Sudoku solver program performed very well handling simple to medium difficulty problems but struggled to solve hard level Sudokus. This could have been due to the fact that in our minimization solver function, the A matrix to be a positive matrix rather than using the absolute value. Given more time the algorithm could have been improved by implementing this change. This would have likely improved the handling of difficult Sudoku puzzle. Also due to particular input methods the dialog boxes could not handle the input of 16x16 Sudoku puzzles. Though

the solver can theoretically handle any N^2 Sudoku, Sudokus larger than 9x9 could not be input through the standard UI because spaces in the string would be necessary which is incompatible with the current input format. Given more time this problem is also something that could be improved if work was to continue on this project.