An improved strategy for solving Sudoku by sparse optimization methods

Yuchao Tang¹, Zhenggang Wu², Chuanxi Zhu¹
1. Department of Mathematics, Nanchang University, Nanchang 330031, P.R. China
2. School of Information Engineering, Nanchang University, Nanchang 330031, P.R. China

Abstract: This paper is devoted to the popular Sudoku problem. We proposed several strategies for solving Sudoku puzzles based on the sparse optimization technique. Further, we defined a new difficulty level for Sudoku puzzles. The efficiency of the method is verified via Sudoku puzzles data-set, and the numerical results showed that the accurate recovery rate can be enhanced from 84%+ to 99%+ by the L1 sparse optimization method.

Keywords: Sudoku; Sparse optimization; L1 norm; Linear programming.

1. Introduction

Sudoku is a world-wide popular number game. The most classical Sudoku is played on a 9 × 9 grid which is broken down into nine 3 × 3 blocks that do not overlap. A Sudoku puzzle usually comes with a partially filled grid. The objective is to fill the 9 × 9 grid with digits 1 to 9, so that each column, each row, and each of the nine 3 × 3 sub-grids contains the digits 1 through 9 only once. There must be a unique solution to a standard Sudoku puzzle. Figure 1 shows an example of a Sudoku puzzle and its solution.

![Figure 1. A standard Sudoku puzzle (left) and its solution (right).](image)

Sudoku is an interesting problem. It attracts a lot of attention from mathematician and computer scientists. There are a number of methods for solving Sudoku have been proposed, such as recursive backtracking[1], Simulated annealing[2], Integer programming[3], Sinkhorn balancing algorithms[4], Sparse optimization method[5,6] and Alternating projection method[7] etc. These methods have their each merit as well as their limitations. In particular, Bartlett et al.[3]
represented the Sudoku problem as a binary integer linear programming problem and solved it by using MATLAB built-in “bintprog” function. But this approach is time consuming especially for solving some difficult Sudoku puzzles. Babu et al.[5] firstly solved the Sudoku puzzles based on the sparse optimization method, which showed very promising in dealing with Sudoku puzzles. In brief, they introduced that how to transform the Sudoku puzzles into a linear system. Since the linear system is an under-determined, there exists infinite solution. They proved that the sparsest solution of this linear equation is the solution of Sudoku. Further, they suggested using $\ell_1$-norm minimization and weighted $\ell_1$-norm minimization models to approximate the sparsest solution. They tested both models on many Sudoku examples of varying levels of difficulty. Most of them could be solved, but it also failed on some Sudoku puzzles. In 2014, McGuire et al.[8] proved that there is no Sudoku puzzles with less 17 numbers already filled in. This answered a long-term open problem in Sudoku making that what is the smallest number of clues that a Sudoku puzzle can have. So we applied the sparse optimization methods to a Sudoku data-set with each one has 17 known clues, which have 49,151 numbers. The sparse optimization method achieved nearly eighty-fourth percent precision. This motivates us that whether there exists a way which can improve the sparse optimization methods to get a higher precision than before. We develop some restart techniques for solving Sudoku puzzles. The idea is to delete repeat numbers in the Sudoku solution and then solve it once again. Our method is tested numerically on Sudoku puzzles data-set with various levels of difficulty from easy to very hard.

The paper is organized as follows. In the next section, we address the problem of encoding Sudoku puzzles into an under-determined linear system, and subsequently solving them using $\ell_1$-norm and weighted $\ell_1$-norm sparse optimization methods. In section 3, we propose an improved strategy for solving Sudoku puzzles based on the sparse optimization methods. These strategies could be regarded as restart techniques. Numerical experiments are presented to show the efficiency of our proposed methods in section 4. Finally, we give some conclusions.

2. Sparse optimization models and methods for solving Sudoku

In this section, we review that the Sudoku puzzles can be made as a linear system. Then, we propose two sparse models for solving it. Finally, two kinds of linear programming methods are suggested to solve the corresponding sparse optimization model.

2.1 Sudoku puzzles represented as a linear system

Babu et al. [5] firstly introduced the way that how to transform a Sudoku puzzle into a linear system. An implicit approach was proposed by [3]. For the sake of completeness, we present the detail as follows. To code Sudoku puzzles as a linear system, we need binary variables (i.e., 0/1) to code the integer numbers 1 to 9.

| Integer numbers | Binary vector | Integer numbers | Binary vector | Integer numbers | Binary vector |
|-----------------|---------------|----------------|---------------|----------------|---------------|
| 1               | (1,0,0,0,0,0,0) | 4              | (0,0,0,1,0,0,0) | 7              | (0,0,0,0,0,0,1) |
| 2               | (0,1,0,0,0,0,0) | 5              | (0,0,0,0,1,0,0) | 8              | (0,0,0,0,0,0,1) |
| 3               | (0,0,1,0,0,0,0) | 6              | (0,0,0,0,0,1,0) | 9              | (0,0,0,0,0,0,0) |

Each entry in the $9 \times 9$ Sudoku puzzles associated with a nine dimensional variables in the Table 1. Then, it has 729 variables. Here, we denote $x_{729\times1}$ be a solution of Sudoku puzzles, then, it must
satisfy the Sudoku constraints. For example, the first row should comprise all the numbers 1,...,9, which can be expressed as 
\[
\begin{pmatrix}
I_{9 \times 9} & I_{9 \times 9} & \cdots & I_{9 \times 9}
\end{pmatrix} \begin{pmatrix} 0_{9 \times 648} \end{pmatrix} = 1_{g \times 1},
\]
where \( I_{9 \times 9} \) denotes the \( 9 \times 9 \) identity matrix, \( 0_{9 \times 648} \) denotes a matrix of size \( 9 \times 648 \) with all elements equal to zero and \( 1_{g \times 1} \) denotes a column vector with elements equal to one. Similarly, the constraint that the first column should contain all digits 1,...,9 can be expressed as below,
\[
\begin{pmatrix}
I_{9 \times 9} & 0_{9 \times 72} & I_{9 \times 9} & 0_{9 \times 72} & \cdots & I_{9 \times 9} & 0_{9 \times 72}
\end{pmatrix} \begin{pmatrix} 0_{9 \times 648} \end{pmatrix} = 1_{g \times 1}.
\]
The constraint that the \( 3 \times 3 \) box in the top-left corner should contain all digits 1,...,9 can be expressed as follows,
\[
\begin{pmatrix}
I_{9 \times 9} & I_{9 \times 9} & 0_{9 \times 54} & I_{9 \times 9} & I_{9 \times 9} & 0_{9 \times 54} & I_{9 \times 9} & I_{9 \times 9} & 0_{9 \times 54} & 0_{9 \times 54}
\end{pmatrix} \begin{pmatrix} 0_{9 \times 648} \end{pmatrix} = 1_{g \times 1}.
\]
The constraint that the first cell should be filled can be expressed as,
\[
\begin{pmatrix}
11 \cdots 1 & 00 \cdots 0
\end{pmatrix} \begin{pmatrix} 0_{9 \times 648} \end{pmatrix} = 1.
\]
Finally, the clues can also be expressed as linear equality constraint. For example in Figure 1, the clue in the second row and the eighth column that clue takes the value 2 can be expressed as,
\[
\begin{pmatrix}
00 \cdots 0 & 010000000 & 00 \cdots 0
\end{pmatrix} \begin{pmatrix} 0_{9 \times 648} \end{pmatrix} = 1.
\]
By combining all these constraints, the linear equality constraints on \( x \) can be expressed in the generic form as,
\[
Ax = \begin{bmatrix} A_{\text{row}} \\ A_{\text{col}} \\ A_{\text{box}} \\ A_{\text{cell}} \\ A_{\text{clue}} \end{bmatrix} x = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{bmatrix},
\]
where \( A_{\text{row}}, A_{\text{col}}, A_{\text{box}}, A_{\text{cell}} \) and \( A_{\text{clue}} \) denote the matrices associated with the different constraints of Sudoku puzzles, respectively. For the \( 9 \times 9 \) Sudoku puzzles, the size of \( A \) is \( (324 + N_{C}) \times 729 \), where \( N_{C} \) denotes the number of clues. For example the Sudoku in Figure 1, the size of \( A \) is \( 341 \times 729 \) and hence the linear system of equations (1) is under-determined and has infinite solutions. However, not every solution of (1) is a valid solution of the Sudoku puzzle. Babu et al.[5] proved if the Sudoku puzzle has a unique solution, then the sparsest solution of (1) is the solution for the Sudoku puzzle.

2.2 Sparse optimization models

In order to find the sparsest solution of linear equation (1), it leads to the \( \ell_0 \) minimization problem,
\[
(P_0) \min \| x \|_0 \\
\text{s.t. } Ax = b,
\]
where \( \| x \|_0 \) represents the number of nonzero elements in \( x \). In fact, the \( \ell_0 \) minimization problem is a basic problem of Compressive sensing. Since the \( \ell_0 \) minimization problem is a NP-hard and non-convex problem, a popular approach is to replace \( \ell_0 \) by \( \ell_1 \)-norm. Then the \( \ell_0 \) minimization problem becomes the \( \ell_1 \)-norm minimization problem as follows,
\[
(P_1) \min \| x \|_1 \\
\text{s.t. } Ax = b,
\]
where $\|x\|_1 = \sum_{i=1}^n |x_i|$. The $\ell_1$-norm minimization problem is convex and hence it has a unique solution. To enhance the sparsity of solution searched by (P_1), Candes et al.[9] proposed a weighted $\ell_1$-norm minimization problem,

$$(WP_1) \min \|Wx\|_1 \quad \text{s.t.} \ Ax = b,$$

where $W = \text{diag}(w_1, w_2, \cdots, w_n)$ is a diagonal matrix with $w_k = \frac{1}{|x_k| + 1 + \epsilon}$, $0 < \epsilon < 1$, $k = 1, 2, \cdots, n$ and $i$ is the iteration numbers. The weighted matrix $W$ is obtained by solving the problem $(P_1)$ from the previous iteration. Therefore the $(WP_1)$ could be seen as solving a series of $\ell_1$-norm optimization problems $(P_1)$.

### 2.3 Sparse Optimization methods

The $\ell_1$-norm optimization problem $(P_1)$ is equivalent to a linear programming problem, which can be efficiently solved by many well known softwares, such as MATLAB, Lingo etc. We show that the $(P_1)$ can be solved respectively by the following two linear programming methods. Firstly,

$$(LP1) \min 1^T \begin{bmatrix} \hat{x} \\ \bar{x} \end{bmatrix} \quad \text{s.t.} \ [A \ -A] \begin{bmatrix} \hat{x} \\ \bar{x} \end{bmatrix} = b$$

where $\hat{x}_{729 \times 1}, \bar{x}_{729 \times 1}$, and $C = \{x | x \geq 0\}$. The solution of $(P_1)$ is obtained by letting $x = \hat{x}_{729 \times 1} - \bar{x}_{729 \times 1}$. If the solution of $(P_1)$ is nonnegative, then we have the second linear programming problem,

$$(LP2) \min 1^T x \quad \text{s.t.} \ Ax = b, \ x \in C,$$

where the constraint $C$ is the same as $(LP1)$.

Next, we show that the problem $(WP_1)$ could be solved via the $(LP1)$ and $(LP2)$, respectively.

| (Weighted LP1) Solving the weighted $\ell_1$-norm minimization problem $(WP_1)$ by $(LP1)$ |
|---|
| Input: $L = 10, \hat{x} = 0, \bar{x} = 0, x_{ori} = \hat{x} - \bar{x}, \text{tol} = 1 \times 10^{-10};$ |
| For $i = 1:L$ |
| $W = \text{diag} \left( \frac{1}{|x_{ori}| + \epsilon} \right)$; |
| Based on the method $(LP1)$, $x_{new} = \hat{x} - \bar{x}$ is obtained by solving the following linear programming problem. |
| $\min W \begin{bmatrix} \hat{x} \\ \bar{x} \end{bmatrix}, \text{s.t.} \ [A \ -A] \begin{bmatrix} \hat{x} \\ \bar{x} \end{bmatrix} = b, \ \begin{bmatrix} \hat{x} \\ \bar{x} \end{bmatrix} \in C;$ |
| if $\|x_{new} - x_{ori}\| < \text{tol}$ |
| break; |
| else |
| $x_{ori} = x_{new}$; |
| End |
| End |
Output: \( x_{\text{new}} \)

(Weighted LP2) Solving the weighted \( \ell_1 \)-norm minimization problem (WP1) by (LP2)

Input: \( L = 10, x_{\text{ori}} = 0, \text{tol} = 1 \times 10^{-10} \);
For \( i = 1:L \)
\[
W = \text{diag}\left( \frac{1}{|x_{\text{ori}}| + \epsilon} \right);
\]
Based on the method (LP2), \( x_{\text{new}} \) is obtained by solving the following linear programming problem.
\[
\min Wx, \quad \text{s.t.} \quad Ax = b, \quad x \in C;
\]
if \( \|x_{\text{new}} - x_{\text{ori}}\| < \text{tol} \)
break;
else
\[
x_{\text{ori}} = x_{\text{new}};
\]
End
End
Output: \( x_{\text{new}} \)

The 729 \( \times \) 1 vector \( x \) is a stack of 81 9 \( \times \) 1 sub-vectors, one for each of the 81 cells in the Sudoku puzzles. Each 9 \( \times \) 1 sub-vector is represented by a Boolean type value being zero except for a 1 in the position of the digit assigned to that cell. In practice, the \( \ell_1 \)-norm minimization problem (P1) and weighted \( \ell_1 \)-norm minimization problem (WP1) cannot always find a solution takes value exactly 0/1. We are required to transform the corresponding sub-vector.

For example, if we have a sub-vector \( e = (0.1, 0.11, 0.3, 0.4, 0.22, 0.211, 0.113, 0.122, 0.33) \), then we have
\[
e_i = \begin{cases} 1, & i = \text{I} \big( \text{max} \big( e_i \big) \big), \\ 0, & \text{otherwise}, \end{cases}
\]
where \( \text{I} \big( \text{max} \big( e_i \big) \big) \) denotes the position of the maximum number of vector \( e \). Then, we obtain a new sub-vector \( e = (0, 0, 0, 1, 0, 0, 0, 0, 0) \) which represents integer number 4.

3. An improved strategy for solving Sudoku

If the solved Sudoku puzzle is wrong, which means that there is at least one row, or column or 3 \( \times \) 3 sub-squares has repeated numbers. Then, we propose an improved strategy in order to obtain the correct Sudoku solution. Our strategies can be divided into the following steps:

Step 1. (Second Solver) Delete the same numbers appeared in a row, column and 3 \( \times \) 3 sub-squares. Keep remained numbers and view it as a new Sudoku puzzle. If it is solved successfully, then quit. Otherwise, go to Step 2.
Figure 2. (a) The original Sudoku puzzle; (b) Firstly solved solution, but it is wrong; The wrong numbers were drawn by circles. (c) A new Sudoku puzzle obtained by deleting the wrong numbers in (b); (d) The correct final solution.

**Step 2. (Third Solver)** Continue to delete the same numbers appeared in a row, column and 3 x 3 sub-squares. We can also obtain a new Sudoku puzzle, then if it is solved successfully, stop; otherwise, turns to Step 3.
Figure 3. (a) The original Sudoku puzzle; (b) Firstly solved solution, but it is wrong; (c) A new Sudoku puzzle by deleting the wrong numbers in (b); (d) Secondly solved solution, it is still not correct; (e) A new Sudoku puzzle obtained by deleting the wrong numbers in (d); (d) The correct final solution.

We found that if we repeat the step 2 again. There is no improvement. So we turn to the third step.

**Step 3. (Successive increase Solver)** First, we delete the same numbers which appeared in a row, column and $3 \times 3$ sub-squares and remove the original Sudoku clue numbers, then select one number added to the original Sudoku puzzle and hence obtain a new defined Sudoku puzzle. If this Sudoku is solved correctly, quit; otherwise, stop and return to this Sudoku is unsolvable.
Figure 4. (a) The original Sudoku puzzle; (b) Firstly solved solution, but it is wrong; (c) A new Sudoku puzzle by deleting the wrong numbers in (b); (d) Secondly solved solution, it is not correct; (e) A new Sudoku puzzle obtained by deleting the wrong numbers in (d); (f) Thirdly solved solution, it is still wrong; (g) A successive Sudoku puzzle obtained by adding a number to the original Sudoku puzzle; (h) The correct final solution.

In step 3, we pick up a number once and add to the original Sudoku puzzle. This complexity is linear because there are maximum 81 numbers. If we consider pick up two numbers, it will reduce to another combination problem and hence the complexity will increase accordingly. So we stop our method in step 3.
Figure 5. A flowchart of our strategy for solving Sudoku puzzles by sparse optimization methods.

Figure 5 gives a clear view of our proposed strategy for solving Sudoku which are based on sparse optimization methods. As a byproduct, we give a new definition of difficulty level of Sudoku as follows:
1) Easy. The given Sudoku was solved directly by the sparse optimization model (P₁) or (WP₁);
2) Middle. The given Sudoku was solved when Step 1 and Step 2 were used;
3) Hard. The given Sudoku was solved when the Step 3 was used;
4) Devil. The given Sudoku was still not solved with our proposed strategies.

4. Numerical experiments

In this section, we test the efficiency of our proposed methods. All the experiments were run on a standard Lenovo laptop with the Intel Core i7-4712MQ CPU 2.3 GHz and 4GB RAM. The software is MATLAB 2013a. We choose a Sudoku data-set with total 49,151 numbers. This dataset is downloaded from the website of Professor Gordon Royle [10]. All the Sudoku puzzles in the data-set are 17-clue and have a unique solution.

Since the solution of Sudoku puzzles are bounded above one, so we consider two constraint sets: 1) Nonnegative constraint, \( C = \{x|x \geq 0\} \); 2) Bounded constraint, \( C = \{x|0 \leq x \leq 1\} \). The results are reported in Table 2 and Table 3, respectively. In the Table 2 and Table 3, the first number in the column “first solving” is the total successfully solved Sudoku puzzles numbers and the second number is the corresponding percentage. Similarly, the two numbers in the column “Total”.

Table 2. The successfully solved Sudoku puzzles numbers by sparse optimization model (P₁) with and without our proposed strategy

| Methods (LP₁) | Constraint set C | First Solving | Our strategy | Total       | Time(s)       |
|---------------|------------------|---------------|--------------|-------------|--------------|
|               | {x ≥ 0}          | 41722/84.8%   | 2246         | 16          | 4713/99.08%  | 2.2676e+04   |
|               | {0 ≤ x ≤ 1}      | 41722/84.8%   | 2204         | 19          | 4755/99.08%  | 2.9375e+04   |
We can see from Table 2 that the total solved Sudoku puzzles are the same by using sparse optimization model (P$_1$) with different constraint sets. It only reaches 84.8% precision, which means that there is 15.2% of Sudoku puzzles unsolved. By using our proposed methods, we can see that the total solved Sudoku puzzles number is greater than before and it exceeds 99%. The highest is achieved by applying method (LP1) with constraint set 0 ≤ x ≤ 1.

For the weighted ℓ$_1$-norm minimization problem (WP$_1$), we find that the ε value affects the performance of the corresponding optimization algorithms. Although in the original (WP$_1$), the ε value is limited in [0,1] interval. We try large ε and the total solved Sudoku puzzles number by first running is increasing accordingly. The maximum solved Sudoku puzzle is obtained by applying (Weighted LP2) with bounded constraint. We also confirm that the weighted ℓ$_1$-norm minimization problem (WP$_1$) outperform the ℓ$_1$-norm minimization problem (P$_1$) for solving the Sudoku puzzles.

Table 3. The successfully solved Sudoku puzzles numbers by sparse optimization model (WP$_1$) with and without our proposed strategy

| Methods       | Constraint set C | ε   | First Solving | Our strategy | Total       | Time(s)  |
|---------------|------------------|-----|---------------|--------------|-------------|-----------|
|               |                  |     |               | Step 1 Step 2 Step 3 |             |           |
| (Weighted LP1)| {x ≥ 0}          | 0.5 | 45845/93.27%  | 80 0 3006    | 48931/99.55%| 1.3382e+04|
|               |                  | 1   | 45935/93.46%  | 52 5 2953    | 48945/99.58%| 1.9665e+04|
|               |                  | 30  | 46028/93.65%  | 85 0 2763    | 48876/99.44%| 2.1967e+04|
| (Weighted LP2)| {0 ≤ x ≤ 1}      | 0.5 | 45683/92.94%  | 37 1 3194    | 48915/99.52%| 1.8959e+04|
|               |                  | 1   | 45836/93.26%  | 105 0 2975   | 48916/99.52%| 1.9078e+04|
|               |                  | 30  | 45914/93.41%  | 114 0 2838   | 48866/99.42%| 2.5069e+04|
|               | {x ≥ 0}          | 0.5 | 45858/93.30%  | 167 0 2879   | 48904/99.50%| 2.4295e+04|
|               |                  | 1   | 45931/93.45%  | 110 0 2837   | 48878/99.44%| 2.7679e+04|
|               |                  | 30  | 46006/93.60%  | 130 0 2785   | 48921/99.53%| 2.9375e+04|
|               | {0 ≤ x ≤ 1}      | 0.5 | 45683/92.94%  | 121 1 3092   | 48897/99.48%| 2.3676e+04|
|               |                  | 1   | 45839/93.26%  | 163 0 2949   | 48951/99.59%| 2.2516e+04|
|               |                  | 30  | 45948/93.48%  | 114 0 2893   | 48955/99.60%| 2.2676e+04|
5. Conclusions

Sparse optimization method is a new method to solve Sudoku puzzles. We have proposed an effective strategy to improve the sparse optimization method. This idea is simple and easy to implement without increasing the complexity of the problem. We tested our method on a large Sudoku puzzles data-set. Numerical results showed that 99%+ Sudoku puzzles are solved by using our method. Although we can’t reach 100% precision, we have enhanced the performance of the original sparse optimization method from 84%+ to 99%+. Further, we present a new definition of difficulty level of Sudoku puzzles.

In order to achieve 100% precision, we believe that it may be possible by incorporating some logic techniques with sparse optimization methods. However, this exceeds the scope of this paper. We will consider this way in the future works.

6. References

[1] S.S. Skiena, The algorithm design manual, Spinger-Verlag, 2nd ed., 2008.
[2] R. Lewis, Metaheuristics can solve Sudoku puzzles, Journal of Heuristics, 13(4), 2007, 387-401.
[3] A. C. Bartlett, T.P. Chartier, A.N. Langville, T.D. Rankin, An integer programming model for the Sudoku problem, The Journal of Online Mathematics and its Applications, 8, 2008, Article ID 1798.
[4] T. K. Moon, J. H. Gunther, J. Kupin, Sinkhorn solves Sudoku, IEEE Transactions on Information Theory, 55(4), 2009, 1741–1746.
[5] P. Babu, K. Pelckmans, P. Stoica, J. Li, Linear systems, sparse solutions, and Sudoku, IEEE Signal Processing Letters, 17(1), 2010, 40-42.
[6] Y.D. Zhang, S.H. Wang, Y.K. Huo, L.N. Wu, A novel Sudoku solving methods based on sparse optimization, Journal of Nanjing University of Information Science and Technology: Natural Science Edition, 3(1), 2011, 23-47.
[7] J. Schaad, Modeling the 8-Queens problem and Sudoku using an algorithm based on projections onto nonconvex sets, Master of Science, The University of British Columbia, 2010.
[8] G. McGuire, B. Tugemann, G. Civario, There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration, Experimental Mathematics, 23(2), 2014, 190-217.
[9] E. Candes, M. Wakin, S. Boyd, Enhancing sparsity by reweighted l1 minimization, Journal of Fourier Analysis Applications, 14(5), 2008, 877-905.
[10] http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php.