Journal of Computer Science

Original Research Paper

Algorithm for Compressing/Decompressing Sudoku Grids

[1]Zhivko Nedev, [2]Murtala Adamu Zungeru, [3]Suykhun Khov, [3]Daravisal Dy and [3]Kimkhung Sov

[1]Department of Computer Science and Information System,
Botswana International University of Science and Technology, Botswana
[2]Department of Electrical, Computer and Telecommunications Engineering,
Botswana International University of Science and Technology, Private Bag 16, Palapye, Botswana
[3]Department of Computer Science, Zaman University, Cambodia

Article history
Received: 26-04-2020
Revised: 02-09-2020
Accepted: 01-10-2020

Corresponding Author:
Zhivko Nedev
BIUST, Palapye, Botswana
Email: znedev@gmail.com

Abstract: We describe a way to transfer efficiently Sudoku grids through the Internet. This is done by using linearization together with compression and decompression that use the information structure present in all sudoku grids. The compression and the corresponding decompression are based on the fact that in each Sudoku grid there are information dependencies and so some of the information is redundant.

Keywords: Sudoku, Compression, Decompression, Algorithms, Data Structure

Introduction

In this study, we use the established terminology (Delahaye, 2006) - a Sudoku grid is a square 9×9 table with 81 cells. In each cell, there is a single digit from 1 to 9 and each Sudoku grid fulfills three types of constraints: (1) Each row has each of the digits from 1 to 9 exactly once; (2) each column has each of the digits from 1 to 9 exactly once; and (3) each of the small 3×3 squares has each of the digits from 1 to 9 exactly once. In the following, the figure on the left is an example of a Sudoku grid and on the right is a sudoku grid with all the small 3×3 squares denoted by $B_{i,j}$ with $1 \leq i, j \leq 3$. We use this notation in the rest of this paper.

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

$B_{1,1}$	$B_{1,2}$	$B_{1,3}$
$B_{2,1}$	$B_{2,2}$	$B_{2,3}$
$B_{3,1}$	$B_{3,2}$	$B_{3,3}$

Sudoku puzzles show incomplete grids, with a number of cells pre-filled with fixed, or given digits, chosen to ensure that there is a unique solution. That is, for each Sudoku puzzle, there is only one way to fill the empty cells so to form a complete Sudoku grid.

One Sudoku grid can be used as a base for many Sudoku puzzles. Therefore the grids and the puzzles might have to be stored and transferred separately. In this study, we investigate what is an efficient way to transfer through the Internet a big number of previously generated Sudoku grids. Moreover, the receiving device might have limited memory and computational resources as in the case of a mobile phone.

The most compact representation (https://www.quora.com/What-can-be-the-most-compact-representation-for-a-solved-Sudokupuzzle) of a sudoku grid would use $\log_2 n$ bits, where n is the number of possible Sudoku puzzles. For the first time (Felgenhauer and Jarvis, 2005), it was computed that n = 6, 670, 903, 752, 021, 072, 936, 960 ≈ 6.67×10^{21}. The same result was confirmed in (Mishra *et al*., 2016). So to uniquely identify any Sudoku grid we need only 73 bits. Although this encoding is optimal (we cannot represent one of n possible values with less than $\log_2 n$ bits), both the sending and the receiving device must contain a database of all possible Sudoku grids which put heavy memory and computational burden on the two devices. Given the grid, the compression algorithm searches the database in order to find the 73-bit representation of the grid. Given the 73-bit representation of the grid, the decompression algorithm searches the database to find the grid.

As a second approach, we can represent any sudoku grid with one of its sudoku puzzles; the solution of the puzzle is the grid. So at the sending device, given a grid, we compute one puzzle (this is the compression algorithm), then we send an encoding of the puzzle through the Internet. At the receiving device, we first

decode the puzzle and then solve it (the decompression algorithm) to get the sudoku grid.

Since we want the strongest compression, one could use a puzzle with minimal number of clues. In (McGuire *et al*., 2014), it was proved that any solvable puzzle needs at least 17 clues. Although this approach raises two questions: (1) Does a 17-clue puzzle exist for any sudoku grid and (2) given a sudoku grid, how to efficiently compute at least one of its puzzles with minimal number of clues, they are not considered in our paper.

In general, solving $n^2 \times n^2$ sudoku puzzles is an NP-complete problem (Yato and Seta, 2003). But for small n (in our case $n = 3$), there are a number of practical algorithmic approaches for solving such puzzles. In (Crook, 2009), a simple backtracking algorithm is given based on preemptive sets and random choice. In (Eppstein, 2012), a new algorithm is developed by using full Nishio deduction rules. The computations use directed acyclic graphs and depth-first search. In (Lewis, 2007), a stochastic search-based algorithm is given, which uses simulated annealing. In (Perez and Marwala, 2008), the following stochastic search techniques are used: Cultural Genetic Algorithm, Quantum Simulated Annealing and the Hybrid method that combines Genetic Algorithm with Simulated Annealing. In (Santos-García and Palomino, 2007), it is shown how a sudoku puzzle can be solved with the use of rewriting logic.

We have not done yet any run time comparisons between our algorithm and any of the compression/decompression algorithms based on the above techniques. Looking at the above references (Crook, 2009; Eppstein, 2012; Lewis, 2007; Perez and Marwala, 2008; Santos-García and Palomino, 2007), our algorithm is considerably simpler and has much smaller number of operations.

In this study, we encode any grid to be transferred through the Internet or to be stored in a database by using one fixed universal puzzle. A puzzle is universal, if its pattern of empty cells can be imposed on any grid and the result is a valid sudoku puzzle. Then the puzzle derived from the grid is linearized starting from the top left corner, going from left to right and from top to bottom and finishing at the bottom right corner. Since we use a fixed pattern for the empty cells, the linearization contains only content of the clue cells.

Universal Compression Algorithm for Sudoku Grids

Definition

A universal construction (U for short) for a Sudoku puzzle is a fixed pattern with positions for the empty cells with the property that when the pattern is applied to any sudoku grid, we get a puzzle which is uniquely solvable.

To describe a universal construction, we have to give the positions of all empty cells. In this study, we will describe them visually by giving a Sudoku grid with the non-empty cells filled with the star symbol (∗); all non-filled positions are the empty one. Here is one universal construction:

∗	∗	∗	∗	∗	∗	∗	∗	
∗	∗	∗	∗	∗	∗	∗	∗	
	∗	∗		∗	∗	∗	∗	
∗	∗	∗	∗	∗	∗	∗	∗	
∗	∗	∗	∗	∗	∗	∗	∗	
	∗	∗		∗	∗	∗	∗	
∗	∗	∗	∗	∗	∗	∗	∗	
∗	∗	∗	∗	∗	∗	∗	∗	

The empty cells do not have a star inside. In the above, the empty cells are all cells in the top row and in the last column plus the four empty cells - one in each of the four 3×3 squares in the lower left part of the grid. The total number of empty cells is 9+8+4 = 21.

When we impose the above pattern to any Sudoku grid, we get a Sudoku puzzle with a unique solution: First we fill the four empty cells in each of the four 3×3 squares in the lower left part of the grid; then we fill all cells in the top row except the cell in the top right corner; finally we fill all cells in the last column. That is why the pattern is a universal construction - it has a unique solution for any Sudoku grid.

We can also encode any universal construction with a 9×9 binary matrix where zeros indicate empty positions. So the above universal construction is represented as:

$$U = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

We can use any universal construction to make a compression and its corresponding decompression algorithm.

An input for our compression algorithm is a 9×9 matrix G representing the Sudoku grid and the 9×9 binary matrix U representing the universal construction.

To get the compressed image of *G*, we linearize the grid *G* ignoring all positions for which the corresponding entry in *U* is zero and we get a one dimensional array *C*.

Algorithm 1 Compressing Sudoku Grid

```
Result: Outputs the compressed image of the Grid G
  C = new Vector() // will contain compress image of G
  for i = 1..9 do
     for j = 1..9 do
        if U[i, j] = 1 then
           C.add(G[i,j])
  return C
```

So the compression algorithm runs in 81 = 9×9 steps.

The corresponding decompression algorithm has two inputs: The output array *C* from the above algorithm and the same matrix *U* representing the universal construction. The decompression has two steps. First, it delinearizes the one dimensional array to get to the sudoku puzzle. Then it solves the puzzle to get its underlying grid.

Algorithm 2 Decompressing Sudoku Grid

```
Result: Outputs a decompressed image of the Grid G
  G = new Matrix(1..9, 1..9) // will contain decompress image of G
  l = length(C)
  k = 0 // will represent running index in the array C
  for i = 1..9 do
    for j = 1..9 do
       if U[i,j] = 1 then
            k ← k +1
            G[i,j] ← C[k]
       else
            G[i,j] ← 0 // 0 represents empty position in the Sudoku puzzle
  // the delinearization is done; now we need to solve the puzzle
  G ← SolvePuzzle(G)
  return G
```

Notice that for the puzzle coming from the above universal construction, if we follow the empty positions in the right order, we can use only simple elimination logic – for each empty position we can eliminate the eight values that cannot be there. The same is true for each puzzle coming from the universal construction bellow.

The delinearization in the above algorithm runs in 81 = 9×9 steps. The number of steps to solve the puzzle coming from the above is equal to the number of empty positions in the puzzle 21. So the total is 102 steps to run the decompression algorithm.

For the universal construction bellow, there is no difference in the number of steps to run the compression algorithms. Similar calculations as above for the decompression algorithm show that the number of steps is 81+33 which is 114.

Notice also that with the above universal construction, we always get a compression of $\frac{81-21}{81} * 100 = 74\%$. (21 is the number of empty cells in the construction.)

In the rest of the paper, we try to optimize our compression ratio.

Optimization Problem and a Lower Bound

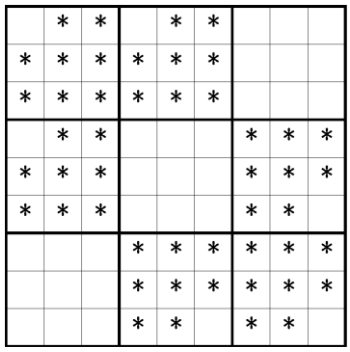
Definition

If UC is a universal construction, then let $\alpha(\mathbb{UC})$ be the number if its empty cells. For example, the α of the above universal construction is 21.

Problem 1

Let $\alpha_{\max}$ be the biggest value for α over all universal constructions. What is $\alpha_{\max}$? Can we find at least one universal construction, $\mathbb{UC}$, for which $\alpha(\mathbb{UC}) = \alpha_{\max}$?

After many trials and errors and improvements, we found the following universal construction. It's α is 33.



The encoding of the above universal construction as a binary matrix is:

$$U = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

The above pattern is a universal construction; imposing the pattern on any sudoku grid, we always get a puzzle that has a unique solution. To solve the puzzle,

first, we fill the six empty cells in the six small 3×3 squares that have only one empty cell each. Then we fill the 3 empty small squares using a simple deductive logic. This is possible because for each of these small empty squares, there are two completely filled small squares in the same row and in the same column. Notice that if we have enough processors (≥27), solving the puzzle can take only two steps: One step for filling the six empty cells in the six small 3×3 squares and one step for filling all 27 cells of the 3 empty small squares.

The important idea in the above construction is that for each of the three small empty squares, there are two completely filled small squares in the same row and in the same column. Can we have four completely empty small squares that are surrounded in the same row and in the same column by four completely filled small squares? No, because by the pigeonhole principle, there will be at least one row with two completely empty small squares and then these cannot be surrounded in the same row by two completely filled small squares.

Despite further efforts, we were unable to find any universal construction with a bigger α. So for now $\alpha_{max} \geq 33$.

Maximality by Inclusion

Our proof for an Upper bound utilizes the notion of a negative pattern.

Definition

A pattern of empty cells that cannot be part of any universal construction is called negative pattern.
We will use the following two negative patterns.

Negative Pattern 1

The following pattern cannot be part of any universal construction.

		*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
		*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*

This is because of the following counter example: When the pattern is applied to the following grid, the resulting puzzle has two solutions: In the first two columns, the missing 1s and 2s are interchangeable.

		3	4	5	6	7	8	9
4	5	6	7	8	9	1	2	3
7	8	9	1	2	3	4	5	6
		4	3	9	8	5	6	7
8	6	5	2	7	1	3	9	4
9	3	7	6	4	5	8	1	2
3	4	1	8	6	2	9	7	5
5	7	2	9	1	4	6	3	8
6	9	8	5	3	7	2	4	1

Negative Pattern 2

The following pattern cannot be part of any universal construction.

*		*		*	*	*	*	*
*	*			*	*	*	*	*
*	*	*	*	*	*	*	*	*
*			*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*
*	*	*	*	*	*	*	*	*

This is because of the following counter example: When the pattern is applied to the following grid, the resulting puzzle has two solutions: The missing 1s and 2s (in each of the three 3×3 squares: $B_{1,1}$, $B_{1,2}$, $B_{2,1}$) are interchangeable.

3		4		7	6	8	9	5
5	6			8	9	3	4	7
7	8	9	5	4	3	1	2	6
8			7	9	4	6	5	3
9	4	3	8	6	5	7	1	2
6	7	5	3	2	1	4	8	9
1	3	6	4	5	2	9	7	8
4	5	8	9	3	7	2	6	1
2	9	7	6	1	8	5	3	4

The above two negative patterns have many equivalent forms through the following transformations:

- Permuting rows in the same band
- Permuting bands
- Permuting columns in the same stack
- Permuting stacks
- Mirroring the grid by one of the diagonals

- Rotating the grid on 90, 180, or 270°C

Next, we prove that if we add one or more empty cells in the universal construction from section 3, then the new pattern is not anymore a universal construction. That's it, for any new pattern by inclusion, always there is at least one Sudoku grid such that the resulting puzzle does not have a unique solution.

Theorem 1

The universal construction from section 3 is maximal by inclusion.

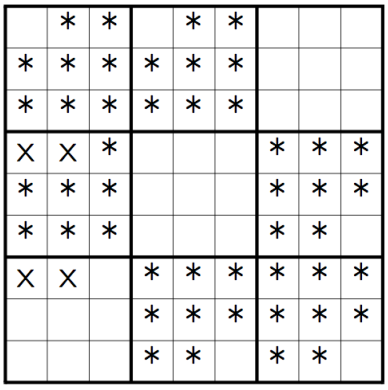
Proof. (We use Proof by Contradiction.)

Assume the we could add at least one more empty cell to the universal construction from section 3. We will prove that the new pattern is not any more a universal construction.

To add one more empty cell, we need to choose which cell with a star to make empty. So first, we need to choose one of the six small squares that each has one empty cell. Then we can make empty one of the cells with a star in that square. Each of the six small squares has already one empty cell. We will call them the old empty cells. There are two cases for the new empty cell.

Case 1.)

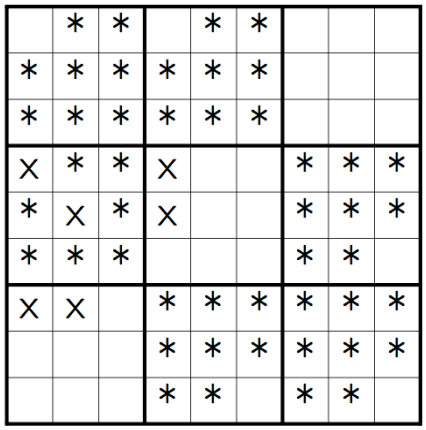
If the new empty is in the same row or in the same column as the old empty cell in the same small square, then the new pattern contains the Negative Pattern 1 as a sub pattern and so there exist at least one Sudoku grid so that imposing the new pattern on it, we get a puzzle with no unique solution. Here is an example:



In the above, the Negative Pattern 1 is formed by the two empty cells in $B_{2,1}$ and any of the three pairs of cells in $B_{3,1}$ that are parallel to the two empty cells in $B_{2,1}$.

Case 2.)

If the new empty is not in the same row or in the same column as the old empty cell in the same small square, then the new pattern contains Negative Pattern 2 as a sub pattern; so there exist at least one Sudoku grid such that that imposing the new pattern on it, we get a puzzle with no unique solution. Here is an example:



Simple Upper Bound for α

In (McGuire *et al*., 2014), it is given that the smallest Sudoku puzzle has 17 clues. Since every universal construction can be used to derive Sudoku puzzles, it follows that there is no universal construction with $\alpha >$ 81-17. So $\alpha_{max} \leq 64$.

Conclusion and Open Problems

Q1: Our best universal construction (from section 3) has $\alpha = 33$. Is there a universal construction with $\alpha > 33$?

Q2: Can we improve the upper bound of 64 from section 5?

Q3: At the moment, we have only two Negative Patterns. Can we find more? Can we find all possible Negative Patterns with small number of empty cells?

Q4: Instead of looking for a universal construction with big α, for a given sudoku grid we could try to find a puzzle with a smallest number of initially given cells with numbers (also called clues). What is the complexity of this minimization problem? What is the best algorithm that finds a puzzle with minimal number of clues for a given Sudoku grid? Then, to transfer a given grid, we could only transfer one of its puzzles with smallest number of clues. At the destination, to decompress, we could solve the puzzle. How efficient could be such a transfer compared with our transmission using our best universal construction?

Author's Contributions

All authors equally contributed in this work.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

References

Crook, J. F. (2009). A pencil-and-paper algorithm for solving Sudoku puzzles. Notices of the AMS, 56(4), 460-468.

Delahaye, J. P. (2006). The science behind Sudoku. Scientific American, 294(6), 80-87.

Eppstein, D. (2012, June). Solving single-digit sudoku subproblems. In International Conference on Fun with Algorithms (pp. 142-153). Springer, Berlin, Heidelberg.

Felgenhauer, B., & Jarvis, F. (2005). Enumerating possible Sudoku grids. Preprint available at http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf.

https://www.quora.com/What-can-be-the-most-compact representation-for-a-solved-Sudokupuzzle

Lewis, R. (2007). Metaheuristics can solve sudoku puzzles. Journal of heuristics, 13(4), 387-401.

McGuire, G., Tugemann, B., & Civario, G. (2014). There is no 16-clue Sudoku: Solving the Sudoku minimum number of clues problem via hitting set enumeration. Experimental Mathematics, 23(2), 190-217.

Mishra, P., Gupta, D. K., & Badoni, R. P. (2016). A new algorithm for enumerating all possible Sudoku squares. Discrete Mathematics, Algorithms and Applications, 8(02), 1650026.

Perez, M., & Marwala, T. (2008). Stochastic optimization approaches for solving Sudoku. arXiv preprint arXiv:0805.0697.

Santos-García, G., & Palomino, M. (2007). Solving Sudoku puzzles with rewriting rules. Electronic Notes in Theoretical Computer Science, 176(4), 79-93.

Yato, T., & Seta, T. (2003). Complexity and completeness of finding another solution and its application to puzzles. IEICE transactions on fundamentals of electronics, communications and computer sciences, 86(5), 1052-1060.