Journal of Game Theory 2014, 3(1): 1-6
DOI: 10.5923/j.jgt.20140301.01

Mathematical and C Programming Approach for Sudoku Game

Sanjay Jain[1,*], Chander Shakher[2]

[1]Department of Mathematical Sciences, Government College, Ajmer Affiliated to M. D. S. University Ajmer, Ajmer-305001, India
[2]Research Scholar, Mahatma Gandhi University, Meghalaya, India

Abstract A lesser-known fact is that Sudoku is a special case of Latin squares, and hence the enumeration of the total number of possible grids proves to be an interesting combinatorial problem. Previous researchers have come up with an accurate answer to this question through various reduction methods as well as computer-based programming: they derived a way to place all Sudoku grids into 44 different classes, after which each class was enumerated separately. Here we develop a mathematical C-approach to solve specific Sudoku problems.

Keywords Sudoku, Number place, Puzzle game, Board game, Mathematical game, Matrix game, Grid game

1. Introduction

Sudoku puzzle, as a widely popular intellectual game in recent years, was invented in Swiss in 18th century. Then, it initially harvested well development in Japan in the past decades. The name Sudoku actually derives from Japanese that means "**Number Place**". Sudoku is a popular Japanese game with the objective of filling a 9x9 grid of cells with the numbers 1-9 without entering the same number in a column, row or and 3x3 sub-box of the grid. Some boxes have already been filled in by the setter to serve as clues. The rules are simple, but some puzzles can get very difficult, and it attracts fans from all over the world. Generally, Sudoku game is started with such a situation in grid that some of the cells have already been confirmed by digits known as givens. The task for Sudoku players is to place a digit from 1 to 9 into each cell of the grid, and meanwhile each digits can only be used exactly once in each row, each column and each block. Additionally, all the nine rows, nine columns and nine blocks are respectively ensured to contain all the digits from 1 through 9. These limitations for placing digits in three locations are respectively called row constraint, column constraint and block constraint; as in[2].

Based on the rules that we mentioned above, Sudoku players are commonly inspired to complete the placement of digits into all empty cells using various techniques as soon as possible. Previous research by Felgenhauer and Jarvis (Mathematics of Sudoku I, 2006) puts the total number of possible sudoku grids as 6670903752021072936960. This is approximately 6.671×10^{21}. This number was calculated using various reduction methods and computer programs; as in[9]. This figure ha s been confirmed independently by numerous others[8]. Later Russell and Jarvis[5] showed that when symmetries were taken into account, there were, of course, many fewer solutions; 5,472,730,538 to be exact.

Another powerful conclusion shows that the minimal amount of givens in an initial Sudoku puzzle that can yield a unique solution is 17 given cells; as in[10]. Recently, researchers engaged in algorithm development issues some breakthroughs in Sudoku solving, grading and generating by Genetic Algorithm, evolutionary method with geometric crossover, belief propagation etc.; as in[11],[12],[13].

The researchers first counted the total number of possibilities for the first three rows, which turned out to be 948109639680. To make enumeration easier, th ey used many reduction methods such as relabeling, putting the first grid into standard form, and lexicographical reduction. Firstly, they reduced 948109639680 by 9! in order to put the first grid in standard form, yielding the answer 2672136. This can be further reduced by lexicographical reduction. This cuts down the number of top 3 rows that we are required to count by a factor of 72, to 36288. This was eventually reduced to 1296, 71 and finally 44; as in[1].

The final 44 classes can represent all the 2612736 different top 3 rows. The significance of the 44 classes is that every possible configuration of the top 3 blocks will have the same number of completions as one of the classes, and each member in every class has the same number of completions to a full grid. Thus, they can simply be placed into the same class for easier calculation. Each of the 44 classes is then enumerated, to find out the number of different completions to a full grid. Following that, the

* Corresponding author:
drjainsanjay@gmail.com (Sanjay Jain)
Published online at http://journal.sapub.org/jgt

number is multiplied by the number of equivalent configurations to a full grid, totaled up and multiplied by 1881169920 (9!x72x72) for the total number of valid Sudoku grids; as in[8],[9],[10].

Sudoku most commonly appears in its 9x9 matrix form. The rules are simple: fill in the matrix so that every row, column, and 3x3 sub matrix contains the digits 1 through 9 exactly once. Each puzzle appears with a certain number of "givens." The number and location of these determines the game's level of difficulty. Figure 1 is an example of a 9x9 Sudoku puzzle. This puzzle idea can accommodate games of other sizes. Of course, a 4x4 puzzle would be easier and a 16x16 puzzle, harder. In general, any n x n game can be created, where $n=m^2$ and m is any positive integer. In fact, the 25 x 25 puzzle is sometimes called Samurai Sudoku, because it is much more challenging and time-consuming. And there are numerous other variants of the game; as in[15].

2. Problem Analysis

Since Sudoku game is hugely attractive for worldwide players in different intellectual levels, it is significantly necessary for researchers to create puzzles tailored to the difficulty requests of players in a tolerable time. Meanwhile, these created puzzles must yield a unique solution so that players can complete the puzzles based on confirm ed cells using logic-deducing step by step. A puzzle with unique solution also shows sufficient intellectual challenges in the pursuit of terminal answer, and highlights the inherent charm of Sudoku game.

Thus, we develop a program to construct specific Sudoku puzzles with the consideration of the following three requirements: as in[2].

2.1. Varying Difficulty

Essentially, the program should be able to create puzzles in different levels of difficulty. It regulates that the program for creating must be designed by means of two jobs as follows:

- Difficulty level: define what the difficulty level is, and evaluate a fixed Sudoku puzzle by a grading program.
- Extensibility: A varying number as a metric of difficulty request from a play can be input by the player. Based on this number, the algorithm for creating must be applicable to generate diverse puzzles satisfying the difficulty request of the player.

2.2. Unique Solution

All generated Sudoku puzzles must be guaranteed to yield a unique solution by a solving algorithm.

2.3. Minimizing Complexity

The programs of all these algorithms must finish their jobs in a short time accepted by users or players.

2.4. Assumptions

- We scale the game environment of a Sudoku puzzle within a 9 -by-9 grid, and take no consideration of the Sudoku puzzle in other size of the grid.
- All the statistic data of the running time in this paper are counted by our computer. These data have reliable statistic values and comparability since they are collected under the same computational condition.

2.5. Computational complexity of the Sudoku Problem

The computational complexity for a Sudoku problem is analytically valuable, in fact, at every step there is always a cell that can be filled only with a number, than the complexity results O(n^2),where n is the number of cells for each row or column,(in our case n=9).

Instead, for a generalized Sudoku problem (a Sudoku problem that has not a uniqu e solution) the computational complexity is not easily valuable analytically but may be evaluated by comparison with the corresponding one of a problem, the mathematical complexity of which has been evaluated and is known; as in[3],[4] .

3. Mathematical Model to Describe Sudoku

3.1. Variables in the Mathematical Models

To describe a mathematical model first of all variables and their kind (for example binary, integer.) should be defined.

In this model a 3D matrix 9x9x9 of variables is defined, with a total amount of 729 elements. Each so defined variable is binary and characterized by three indexes: i , j , k: the index i and j define respectively the row and the column of a generic position in a Sudoku schema; the index k represents integer in the range 1 to 9 which may be present in a generic position. Being the variables binary they can assume only the values 0 or 1: 0 value is assumed if in the position i, j, the k integer is not included and the value 1 if the position i,j is filled with the k integer.

3.2. The Objective Function

In this point an "objective function" to be minimized or maximized should be define d. But one must remark that in our case one need to determine only an admissible solution, because every Sudoku schema has one and only one solution: then the admissible solution is automatically the optimal one.

As a consequence the value of objective function is out from our interest and a generic constant value may be assumed like "objective function", for instance the unit value.

3.3. Constraints

The constraints, necessary to limit the field of the

problem, are linear combinations of variables giving a constant value.

The model contains the following typology of constraints:

1. Having fixed a j column, at the crossing of each row i with the j column, the sum of variables included within correspondent k column shall assume unit value;

2. Within the matrix i,k the sum of variables along i for each k assumes unit values.

3. Having fixed an i row, at the crossing of each column j with the i row, the sum of variables included with i correspondent k column shall assume unit value

4. Within the matrix j,k the sum of variables along i for each k assumes unit value

5. For every sub-matrix 3x3 (i,j) the sum of variables for each k shall assume unit value.

Of course all the previous constraints define an empty Sudoku schema ; but if we want to introduce in the schema some predefined values in predefined position, the corresponding values shall be also predefined assuming the unit value; nothing to say that such predefined values shall respect the above defined restraints; as in[3],[4],[6],[7].

4. Mathematical Formulation of the Model for Sudoku Game

The mathematical model of Sudoku game above defined in terms of variables, objective function and restraints is described by the following expressions

Variables: $X_{i,j,k}$ binary i=1...9, j=1...9, k=1...9

Objective Function: min 1

Constraints:

$$\sum_i X_{i,j,k} = 1 \forall j, \forall k \ \{\forall \text{ stands "For All"}$$

$$\sum_j Xi, j, k = 1 \forall i, \forall k$$

$$\sum_j Xi, j, k = 1 \forall i, \forall j$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; i \in [1,3] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [1,3] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [1,3] \ j \in [7,9]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [7,9]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [7,9]$$

5. Implementation of the Model with Instance Generation Program

To apply practically the Sudoku program, an instance generation program has been written in C++ language; this program is able to generate a Sudoku game having chosen the number of fixed parameters to be introduced, the position of which is defined by the program by means of a random generator; a second program assigns the predefined integers to the position chosen as above described. In the following pages the instruction of the programs are reproduced.

```
# include <stdio.h>
# include <stdlib.h>
# include <time.h>
#define nummax 5
int main()
{
        int num,i,j;
        int a[9][9];
        char z[9][9];
        FILE *fi;
        FILE *fo;
        fi = fopen("input.txt","r");
        fo = fopen("output.txt","w");
        for(i=1;i<=9;i++)
        {
                for(j=1;j<=9;j++)
                {
                        a[i-1][j-1]=0;
                        z[i-1][j-1] = fscanf(fi,"%c");
                        fprintf(fo,"%d",z[i-1][j-1]);
                }
                fscanf(fi,"%c");
        }
        srand(time(NULL));
        for(num=1;num<=nummax;num++)
        {
                i=1+rand()%9;
                j=1+rand()%9;
                a[i-1][j-1]=1;
                printf("%d%d\n",i,j);
        }
        for(i=1;i<=9;i++)
        {
                for(j=1;j<=9;j++)
                {
                        if ( a[i-1][j-1]==0)
                                z[i-1][j-1]=0;
                        fprintf(fo,"%d",z[i-1][j-1]);
                }
                fprintf(fo,"\n");
        }
        fclose(fi);
        fclose(fo);
```

```
        return 0;
}
```

This program gives in output a generalized Sudoku instance, but we cannot be sure that this instance has a unique solution.

5.1. Comparison with Existing One

To generate an instance with unique solution the following method, developed by Professor Della Croce and professor Ferro of Politecnico di Torino is used.

Suppose that the model presented above has been solved and a feasible solution has been found. Denote by SOL(i,j) the value of the element (i,j) of the schema in the feasible solution. Solve the following model P2:

Variables: $X_{i,j,k}$ binary i=1...9, j=1...9, k=1...9

Objective Function: min $z = \sum_{i,j,k\,:\,SOL(i,j)=k} X_{ijk}$

Constraints:

$$\sum_i X_{i,j,k} = 1 \forall j, \forall k \ \{\forall \text{ stands "For All"}$$

$$\sum_j Xi, j, k = 1 \forall i, \forall k$$

$$\sum_j Xi, j, k = 1 \forall i, \forall j$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; i \in [1,3] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [1,3] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [1,3] \ j \in [7,9]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [4,6] \ j \in [7,9]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [1,3]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [4,6]$$

$$\sum_i \sum_j Xi, j, k = 1 \ \forall k; \ i \in [7,9] \ j \in [7,9]$$

The solution of this model provides a feasible solution also to the previous model. The objective function of model P2 minimizes the sum of the elements in the grid having the same value obtained in the solution of the previous model. Hence, as each grid is composed by 81 elements, if the objective function value of model P2 is equal to 81 (Z=81), then the Sudoku problem has unique solution, else it has not.

5.2. Solving Algorithm: Depth-first Search

An effective solving algorithm for searching out all the feasible solutions of a Sudoku puzzle is indispensible for the generating algorithm to judge whether a created puzzle has a unique solution. Since a Sudoku puzzle is shown as a 9 -by-9 grid with sufficient information from givens and strict constraints of game rules, it is feasible to search out all the solutions of the puzzle using enumerating search.

For finding out a solution as a terminal pattern or judging whether a created puzzle is unique-solvable, we build a Sudoku solver by the mechanism of Depth -first search. The solver searches empty cells from left to right, top to bottom in the grid. It attempts to fill a potential 1-through-9 digit in each empty cell while satisfies the three constraints of game rules. Once none of digits from 1 through 9 can be filled in an empty cell to meet the constraints, the solver backtracks to the previous empt y cell and substitutes the filled digit there into another untried potential digit. In this way the solver continuously performs the search until all the potential solutions are searched out and recorded.

6. Results

Based on the specification of algorithms above, we create Sudoku puzzles in different levels respectively within tolerable time using our developed solving and generating algorithms **while guarantees each of these puzzles has a unique feasible solution**.

Our algorithm finds at least one feasible solution, if it exists. Originally, we assumed each Sudoku puzzle has one unique solution since usually an answer is provided with each puzzle. However, after playing a few games, we discovered that uniqueness is not a requirement for puzzle creators. Some puzzles have multiple solutions. In such cases, our solution was correct but distinct from the solution provided by the puzzle creators.

7. Analysis of Algorithm Complexity

For generating Sudoku puzzles we develop an algorithm with negligible space co mplexity. Thus we concern significantly about time complexity of our algorithm, and focus on reducing it in order to achieve the generation of Sudoku puzzle within tolerable time for players.

The entire algorithm for generating Sudoku puzzle is divided into two parts: to create a terminal pattern and then to dig holes on it. The time complexity of creating a terminal pattern proves to be $O(1)$, which indicates that a terminal pattern can be produced within constant-magnitude time. In the algorithm of generating puzzles, the program does trials of digging a hole at most 81 times in the terminal pattern. For **the uniqueness feasible solution** derived from the constructed puzzle, a solver built by the **Depth-First Search** is called to find out all potential solutions of the dug-out puzzle once a trial of digging a hole is done at an unexplored cell. The time complexity of the Depth -First Search proves to be $\Theta(V + E)$ as in[14]. The amount of $(V + E)$ in the problem of solving a Sudoku puzzle is estimated to be less than 1500000.

8. Examples

In this paper some examples of Sudoku instance on which our model has been tested will be given.

Example of Sudoku and its solution.

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

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

Another example with their solution is

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

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

9. Conclusions and Future Developments

The Sudoku mathematical game, born about two centuries ago, but largely spread in World only during the last years, has been studied and analysed in details.

Our model is very efficient, because it is able to solve every Sudoku instance in a very short time; it takes 0.2 seconds to 0.5 seconds for the diabolic ones according to their difficulty levels. This fact shows that the different difficulty between two levels affects the performance of a human solver but not that of the program.

A mathematical model has been formulated. In addition, the mathematical model has been implemented in a computer solver and in parallel a program has been developed able to generate Sudoku games. Some applications of this analytical and numerical work have been presented too.

Further developments in this field should consider the uniqueness of solution in relation to the minimum number of input data necessary for this condition.

ACKNOWLEDGEMENTS

The authors are thankful to the reviewers for their suggestions to improve the present paper.

REFERENCES

[1] Bertram Felgenhauer & Frazer Jarvis, 2006. Mathematics of Sudoku I. Mathematical Spectrum 39 (2006): 15-22.

[2] Wei-Meng Lee, *Programming Sudoku (Technology in Action)*, Apress. 2006.

[3] F. Della Croce, G. Ferro, I love Sudoku, Mondatori, Torino, 2006.

[4] R. Tadei, F. Della Croce, Ricerca operativa e ottimizzazione, Progetto Leonardo, Esculapio, Bologna, 2002.

[5] Ed Russell and Frazer Jarvis. There are 5472730538 essentially different Sudoku grids . . . and the Sudoku symmetry group.http://www.afjarvis.staff.shef.ac.uk/sudoku/sudgroup.html.

[6] site web: www.nonzero.it.

[7] site web: www.polito.it/didattica/polymath.

[8] http://en.wikipedia.org/wiki/Sudoku.

[9] Bertram Felgenhauer and Frazer Jarvis. http://www.shef.ac.uk/~pm1afj/sudoku/.

[10] Gordon Royle, The University of Western Australia. *MinimumSudoku*.http://people.csse.uwa.edu.au/gordon/sudokumin.php.

[11] Timo Mantere, Janne Koljonen. Solving, *Rating and Generating Sudoku Puzzle with GA*. IEEE Congress on Evolutionary Computation, 25-28 Sept. 2007, pp. 1382-1389.

[12] Alberto Moraglio, Julian Togelius, Simon Lucas. *Product Geometric Crossover for the Sudoku Puzzle.* Proceedings of the IEEE Congress on Evolutionary Computation, 2006. http://privatewww.essex.ac.uk/~amoragn/sudoku.pdf.

[13] Todd K. Moon, Jacob H. Gunther. *Multiple Constraint Satisfaction by Belief Propagation: An Example Using Sudoku*. Utah State University. Adaptive and Learning Systems, 2006 IEEE Mountain Workshop on. 24-26 July 2006, pp. 122-126.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. 2002. *Introduction To Algorithms*, Seccond Edition. CA: The MIT Press.

[15] Ed Pegg Jr. Sudoku Variations. MAA Online. Sept. 6, 2005. http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html.

[16] website: www.nikoli.com.