International Journal of Computer Applications (0975 - 8887)
Volume 52 - No. 21, August 2012

Solving Sudoku with Boolean Algebra

Abu Sayed Chowdhury
Faculty, Department of Computer Science & Engineering
Dhaka University of Engineering & Technology (DUET), Gazipur
Gazipur-1700, Bangladesh

Suraiya Akhter
Faculty, Department of Computer Science & Engineering
Uttara University
Dhaka-1230, Bangladesh

ABSTRACT

Sudoku is a very popular puzzle which consists of placing several numbers in a squared grid according to some simple rules. In this paper, we present a Sudoku solving technique named Boolean Sudoku Solver (BSS) using only simple Boolean algebras. Use of Boolean algebra increases the execution speed of the Sudoku solver. Simulation results show that our method returns the solution of the Sudoku in minimum number of iterations and outperforms the existing popular approaches.

Keywords:

Sudoku, Boolean algebra, Memory representation

1. INTRODUCTION

Sudoku, English pronunciation: (/su:'do U ku:/ soo-doh-koo) is a logic-based, combinatorial number placement puzzle [2, 4, 11]. The word "Sudoku" is short for *Su-ji wa dokushin ni kagiru*, which means "the numbers must be single". The objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 sub-grids that compose the grid (also called "boxes", "blocks", "regions", or "sub-squares") contains all of the digits from 1 to 9. Fig. 1 shows an example of Sudoku puzzle. On July 6, 1895, Le Siecle's rival, La France, refined the puzzle so that it was almost a modern Sudoku. It simplified the 9×9 magic square puzzle so that each row, column and broken diagonals contained only the numbers 1-9, but did not mark the sub-squares. Although they are unmarked, each 3×3 sub-square does indeed comprise the numbers 1-9 and the additional constraint on the broken diagonals leads to only one solution [4]. The number of valid Sudoku solution grids for the standard 9×9 grid is 6, 670, 903, 752, 021, 072, 936, and 960. We can express a 9×9 Sudoku by putting 25-30 values in a 9×9 grids. It can be used in data compassion, in error checking and correction, in data encryption and deception etc.

Boolean algebra which deals with two-valued (true / false or 1 and 0) variables and functions find its use in modern digital computers since they too use two-level systems called binary systems. Boolean algebra, as developed in 1854 by *George Boole* in his book, "An Investigation of the Laws of Thought", is a variant of ordinary elementary algebra differing in its values, operations, and laws. Instead of the usual algebra of numbers, Boolean algebra is the algebra of truth values 0 and 1, or equivalently of subsets of a given set.

A Boolean function (or switching function) is a function of the form $f : Bk \rightarrow B$, where $B = \{0, 1\}$ is a Boolean domain and k is a non-negative integer called the arity of the function. In the case where k = 0, the "function" is essentially a constant element of B. A Boolean function describes how to determine a Boolean value output based on some logical calculation from Boolean inputs.

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

Fig. 1. Example of Sudoku puzzle.

After values, the next ingredient of any algebraic system is its operations. Whereas elementary algebra is based on numeric operations, multiplication: xy; addition: $x + y$; and negation: $\neg x$, Boolean algebra is customarily based on logical counterparts to those operations, namely conjunction: $x \wedge y$ or Kxy (AND); disjunction: $x \vee y$ or Axy (OR); and complement or negation $\neg x$ or Nx (NOT). In electronics, the AND is represented as a multiplication, the OR is represented as an addition, and the NOT is represented with an overbar: $x \wedge y$ and $x \vee y$, therefore, become xy and $x + y$. Conjunction is the closest of these three to its numerical counterpart: consider 0 and 1 = 0, and 1 and 1 = 1; it is multiplication. As a logical operation the conjunction of two propositions is true when both propositions are true, and otherwise is false. The first column of the Fig. 2 tabulates the values of $x \wedge y$ for the four possible valuations for x and y; such a tabulation is traditionally called a truth table. Disjunction, in the second column of the Fig. 2, works almost like addition, with one exception: the disjunction of 1 and 1 is neither 2 nor 0 but 1. Thus the disjunction of two propositions is false when both propositions are false, and otherwise is true. This is just the definition of conjunction with true and false interchanged everywhere; because of this we say that disjunction is the dual of conjunction. Logical negation however does not work like numerical negation at all. Instead it corresponds to incrimination: $\neg x = x + 1$ mod 2. Yet it shares in common with numerical negation the property that applying it twice returns the original value: $\neg\neg x = x$, just as $-(-x) = x$. An operation with this property is called an involution. The set $\{0, 1\}$ has two permutations, both involutary, namely the identity, no movement, corresponding to numerical

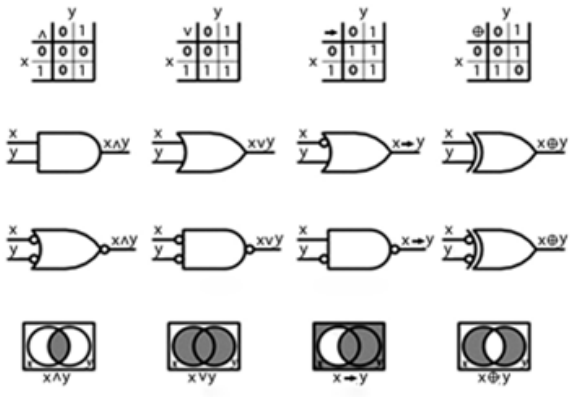

Fig. 2. Various representations of Boolean operations.

negation mod 2 (since +1 = -1 mod 2), and SWAP, corresponding to logical negation.
Using negation, we can formalize the notion that conjunction is dual to disjunction via De Morgan's laws, $\neg(x \vee y) = \neg x \wedge \neg y$ and $\neg(x \wedge y) = \neg x \vee \neg y$. These can also be construed as definitions of conjunction in terms of disjunction and vice versa: $x \vee y = \neg(\neg x \wedge \neg y)$ and $x \wedge y = \neg(\neg x \vee y)$. Fig. 2 shows the symbols used in digital electronics for conjunction and disjunction; the input ports are on the left and the signals flow through to the output port on the right. Inverters negating the input signals on the way in, or the output signals on the way out, are represented as circles on the port to be inverted.
In this paper, we provide a strategy to solve the Sudoku puzzle using the concepts of Boolean algebra.

2. RELATED WORK

The connection between decoding and Sudoku has been previously noted. In [7], Moon *et al.* give an explicit formulation of the BP algorithm for solving Sudoku. BP appears to work only for easier puzzles, with the probable cause being the "loopy" nature of the Tanner graph associated with the puzzle (all cells are in cycles of length four). The author of [6] also discusses BP. Other related works such as *Sinkhorn* balancing [8, 14] is a means of obtaining a unique doubly stochastic matrix from a (nearly) arbitrary matrix. Extensions to produce matrices with arbitrary row and column sums appear in [13]. Sinkhorn balancing (sometimes called Sinkhorn scaling) has been widely studied, and makes its appearance in a variety of applications (See [3]). The Sudoku is solved with genetic operations in [1, 10]. Solution of the Sudoku puzzle with membrane computing is discussed in [5]. The authors of [12] introduced solution of the Sudoku using graphs with deriving Constraint Satisfaction Problem (CSP).
In this paper, we present a solution algorithm based on basic Boolean algebra, which has both lower computational complexity (per iteration) than BP and does not apparently suffer from cycles in the graph.

3. PUZZLE DESCRIPTION AND REPRESENTATION

In this paper, we consider the most popular format of Sudoku puzzle. It is a 9×9 grid with nine rows and nine columns and the 9×9 grid is divided into nine 3×3 sub-grid.

3.1 Memory Representation

Bitwise operation is much faster than arithmetic operation. We use Boolean operation to determine the possible value of a cell. For this reason, we use a new approach to represent the values of

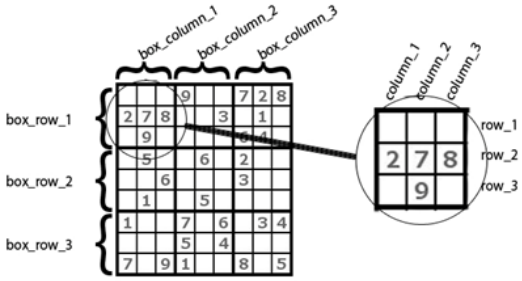

Fig. 3. Memory representation.

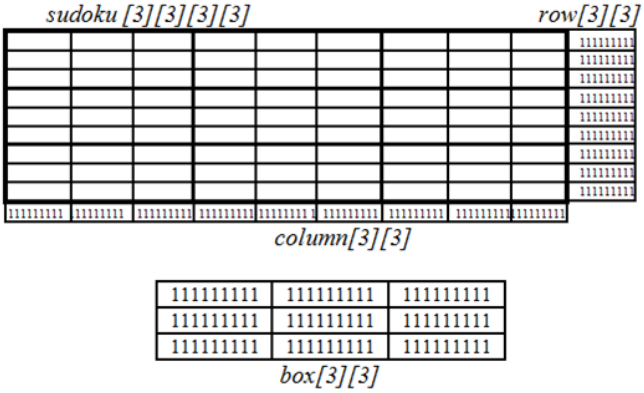

Fig. 4. Primary initialization.

the Sudoku. We use 9 variables that represent the values required in a sub-grid, 9 variables to represent the rows and 9 variables to represent the columns.
In our proposed method, we use $3 \times 3 \times 3 \times 3$ integer type variable, $Sudoku[3][3][3][3]$ to store the values of the Sudoku puzzle. We represent the Sudoku in a 4D (four dimensions) view and divide the Sudoku into 9 boxes. Each box has 3 box-row, 3 box-column and two suffix called box_row_number (*BR*) and box_column_number (*BC*). Each box has 3×3 matrix, 3 rows and 3 columns. Fig. 3 shows the memory representation of a 9×9 Sudoku. We consider 27 integer type variables named $row[3][3]$,$column[3][3]$ and $box[3][3]$ to store the values required in every row, column and box respectively in Boolean form.

3.2 Indexing

The indexes used for $suduko[I][J][K][L]$ are Box indexing ($box[3][3]$), Row indexing ($row[3][3]$) and Column indexing ($column[3][3]$). As each integer variable takes 2 bytes of memory, total allocated memory to solve a 9×9 sudoku is $3 \times 3 \times 3 \times 3 \times 2 + 3 \times 9 \times 2 = 216$ bytes. Space complexity is $O(n^2)$ for a $n \times n$ Sudoku puzzle.

3.3 Solving Technique

At first, we need initialization to the arrays $row[I][K]$, $column[J][L]$ and $box[I][J]$. Primarily initialization with value 1 for 9 x 9 Sudoku is shown in Fig. 4. We calculate all the values of each array using the following equations.
For $box[I][J]$:

$$box[I][J] = box[I][J] - \sum_{I,J,K,L=0}^{2} 2^{suduko[I][J][K][L]-1} \quad (1)$$

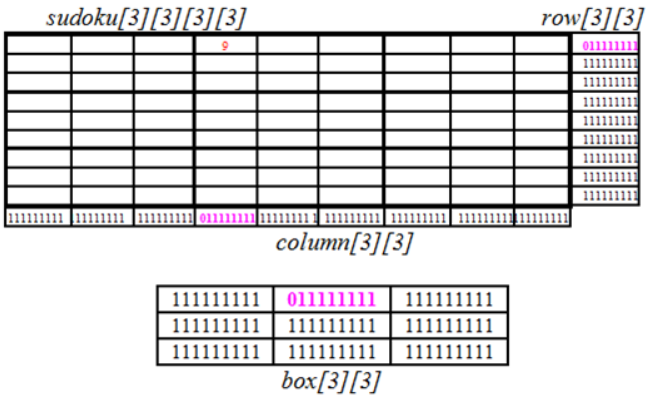

Fig. 5. Update of a variable.

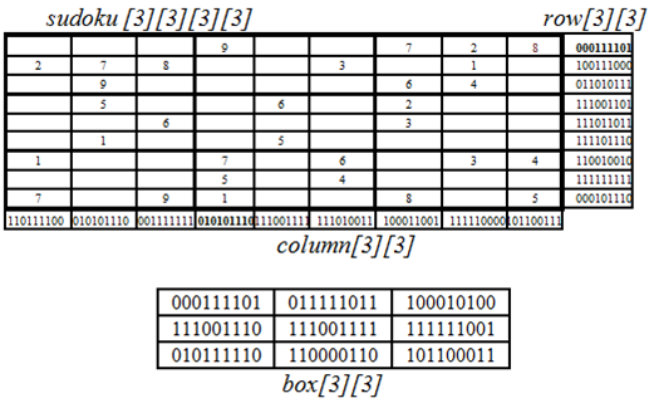

Fig. 6. Update of all variables.

For $row[I][K]$:

$$row[I][K] = row[I][K] - \sum_{I,J,K,L=0}^{2} 2^{suduko[I][J][K][L]-1} \quad (2)$$

For $column[J][L]$:

$$column[J][L] = column[J][L] - \sum_{I,J,K,L=0}^{2} 2^{suduko[I][J][K][L]-1} \quad (3)$$

When a value is entered as input, changes are shown in Fig.s 5 and 6.

4. SOLVING TECHNIQUES

Two basic techniques are used in solving Sudoku described as follows.

4.1 Technique 1: Naked Single

We select a blank cell randomly and count from 1 to 9. Reject the value which are available in that sub-grid plus row and column, where it is situated. In this approach, the arrays $box[][]$, $row[][]$ and $column[][]$ represent the values between 1 to 9 which can be set along with their related rows, columns, and boxes for any cell of $suduko[I][J][K][L]$. We have to find out what can be set in $row[I][K]$, $column[J][L]$ and $box[I][J]$. So, the required value for cell $suduko[I][J][K][L]$ is-

$$R_v = row[I][K] \ \& \ column[J][L] \ \& \ box[I][J] \quad (4)$$

If $\log_2 R_v$ is an integer value, it indicates that the possible number for the cell $suduko[I][J][K][L]$ have only one and it is $\log_2 R_v + 1$ As for example, referring Figure 1, 1st value is found in 6th iteration when R_v = 32. Hence, $\log_2 32 = 5$ and $I = 0$, $J = 1$, $K = 1$, $L = 0$. In this situation, the $suduko[I][J][K][L]$ updated as follows-
$suduko[0][1][1][0] = 5 + 1 = 6$.

4.2 Technique 2: Hidden Single

During playing Sudoku, sometime we may have no such a blank cell and we can not define a value. In that case, we try to find out a blank cell and a value, which is not suitable for the other blank cells either in the row or in the column or in the box. Using Boolean algebra, we can implement such a technique. We can determine the required value of a cell (except for a selected cell in a row or in a column or in a box) using OR operation basis on the blank cell requirements. If the requirement of our selected cell is A, the requirement of other cells that are also blank and in a same row or in a same column or in a same box is B then $A.\overline{B}$ is either 0 or $\log_2 A.\overline{B}$ is a integer number between 0 to 8 and $\log_2 A.\overline{B} + 1$ is the value for our selected cell.

• For a box:
Let $suduko[3][3][3][3]$ is 9×9 Sudoko puzzle. Possible value for $suduko[I][J][K][L]$ is Equation 4. Required value for other blank cell is -

$$\begin{aligned} R_v = {} & R_v \& \sim (box[I][J] \& ((row[I][K] \& (column[J][(L+1)\%3] * !(sudoku[I][J][K][(L+1)\%3]) | column[J][(L+2)\%3] * !(sudoku[I][J][K][(L+2)\%3])))) | (row[I][(K+1)\%3] \& (column[J][(L+1)\%3] * !(sudoku[I][J][(K+1)\%3][(L+1)\%3]) | column[J][(L+2)\%3] * ! \\ & (sudoku[I][J][(K+1)\%3][(L+2)\%3]) | column[J][L] * ! \\ & (sudoku[I][J][(K+1)\%3][L]))) | (row[I][(K+2)\%3] \& \\ & (column[J][(L+1)\%3] * ! \\ & (sudoku[I][J][(K+2)\%3][(L+1)\%3]) | \\ & column[J][(L+2)\%3] * ! \\ & (sudoku[I][J][(K+2)\%3][(L+2)\%3]) | column[J][L] * ! \\ & (sudoku[I][J][(K+2)\%3][L])))) \end{aligned} \quad (5)$$

• For a column:
Let $suduko[3][3][3][3]$ is 9×9 Sudoko puzzle. Possible value for $suduko[I][J][K][L]$ is Equation 4. Required value for other blank cell is -

$$\begin{aligned} R_v = {} & R_v \& \sim (column[J][L] \& ((box[I][J] \& (row[I][(K+1)\%3] * !(sudoku[I][J][(K+1)\%3][L]) | row[I][(K+2)\%3] * !(sudoku[I][J][(K+2)\%3][L])))) | (box[(I+1)\%3][J] \& (row[(I+1)\%3][(K+1)\%3] * !(sudoku[(I+1)\%3][J][(K+1)\%3][L]) | row[(I+1)\%3][(K+2)\%3] * !(sudoku[(I+1)\%3][J][(K+2)\%3][L]) | row[(I+1)\%3][K] * !(sudoku[(I+1)\%3][J][K][L])))) | (box[(I+2)\%3][J] \& (row[(I+2)\%3][(K+1)\%3] * !(sudoku[(I+2)\%3][J][(K+1)\%3][L]) | row[(I+2)\%3][(K+2)\%3] * !(sudoku[(I+2)\%3][J][(K+2)\%3][L]) | row[(I+2)\%3][K] * !(sudoku[(I+2)\%3][J][K][L]))))) \end{aligned} \quad (6)$$

• For a Row:
Let $suduko[3][3][3][3]$ is 9×9 Sudoko puzzle. Possible value for $suduko[I][J][K][L]$ is Equation 4. Required value for other

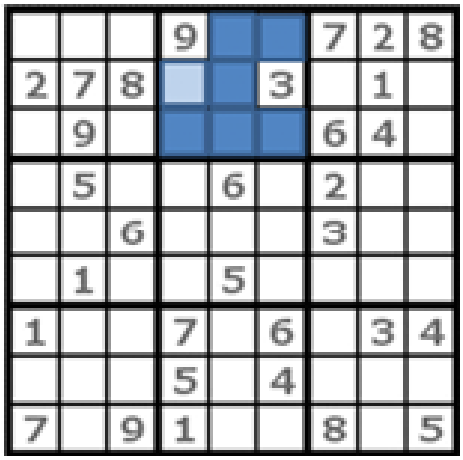

Fig. 7. A typical scenario.

blank cell is -

$$R_v = R_v\& \sim (row[I][K]\&((box[I][J]\&(column[J][(L+1)\%3]*!(sudoku[I][J][K][(L+1)\%3])|column[J][(L+2)\%3]*!(sudoku[I][J][K][(L+2)\%3])))|(box[I][(J+1)\%3]\&(column[(J+1)\%3][(L+1)\%3]*!(sudoku[I][(J+1)\%3][K][(L+1)\%3])|column[(J+1)\%3][(L+2)\%3]*!(sudoku[I][(J+1)\%3][K][(L+2)\%3])|column[(J+1)\%3][L]*!(sudoku[I][(J+1)\%3][K][L]))))|(box[I][(J+2)\%3]\&(column[(J+2)\%3][(L+1)\%3]*!(sudoku[I][(J+2)\%3][K][(L+1)\%3])|column[(J+2)\%3][(L+2)\%3]*!(sudoku[I][(J+2)\%3][K][(L+2)\%3])|column[(J+2)\%3][L]*!(sudoku[I][(J+2)\%3][K][L])))))) \quad (7)$$

Considering the fig. 7, Possible values are-

$$sudoku[1][2][1][2] = 000001001 \quad (8)$$
$$sudoku[1][2][1][3] = 000010001 \quad (9)$$
$$sudoku[1][2][2][2] = 100001000 \quad (10)$$
$$sudoku[1][2][3][1] = 010000010 \quad (11)$$
$$sudoku[1][2][3][2] = 011000011 \quad (12)$$
$$sudoku[1][2][3][3] = 011010011 \quad (13)$$
$$sudoku[1][2][2][1] = 000101000 \quad (14)$$

Using equations 8-13, we get,

$$(Eqn.(8) \vee Eqn.(9) \vee Eqn.(10) \vee Eqn.(11) \vee Eqn.(12) \vee Eqn.(13) = 111011011 \quad (15)$$

Now by AND-ing the value of Equation 14 and complement value of equation 15, we find the value- $000100000 = 32_{(10)}$ and $\log_2 32 = 5$. Therefore, $sudoku[1][2][2][1] = 5 + 1 = 6$.

5. BOOLEAN SUDOKU SOLVER (BSS)

The algorithm, Boolean Sudoku Solver (BSS), for solving Sudoku with Boolean algebra is given in Algorithm 1. Let us explain how the algorithm works.
The Sudoko is inputted to the array $Sudoku[][][][]$ and initialization is done for $box[][]$, $row[][]$ and $column[][]$. Then, the algorithm looks for the empty cell. If any empty cell found then it checks whether *Technique 1* or *Technique 2* will be applied. Otherwise, the algorithm is terminated.

Algorithm 1 Boolean Sudoku Solver (BSS)

Input: $Sudoku[BR][BC][R][C]$, $box[BR][BC]$, $row[BR][R]$ and $column[BC][C]$.
Jump1:
Find a empty cell;
if not found **then**
  go to *Jump3*;
end if
Check for *Technique 1*;
if found **then**
  go to *Jump2*;
end if
Check for *Technique 2*;
if no value found **then**
  go to *Jump1*;
end if
Jump2:
Set the value;
Update $box[BR][BC]$, $row[BR][R]$, $column[BC][C]$;
Go to *Jump1*;
Jump3:
Terminate;

6. SIMULATION RESULTS AND ANALYSIS

The objectives of the experimental work are to verify the feasibility and efficiency of Boolean Sudoku Solver (BSS). The experimental evaluation has been performed with 200 Sudoku puzzles. Our aim is to find out the solving rate, number of iterations and execution time, and compare BSS with other existing popular techniques. The BSS has been implemented over Borland c++ 5.02, 32 bit compiler, running on 2.13 GHz Core-i3 2nd generation processor with 2GB main memory. The operating system used is windows 7 ultimate. The results for various solving techniques are compared at the same hardware configuration. We have chosen Sudoku from newspaper. There are 4 categories of Sudoku- Very difficult level Sudoku, Difficult level Sudoku Medium level Sudoku, and Easy level Sudoku.

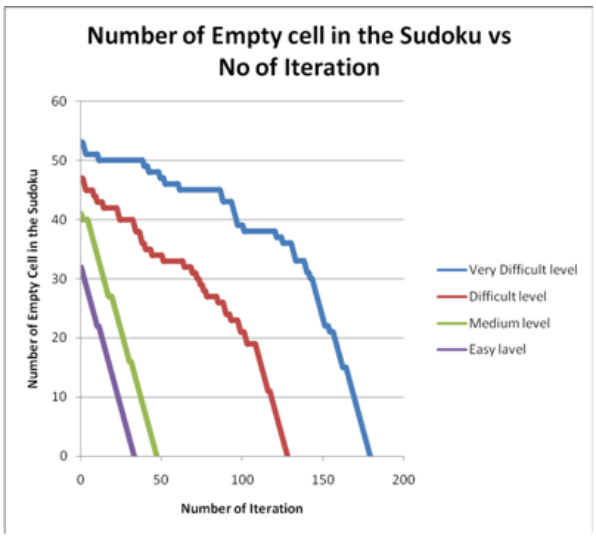

Fig. 8. No. of blank cells remaining with No. of iterations.

Number of iterations depends on the complexity and the number of free cells. Fig. 8 shows that the easy Sudoku takes 35 iterations to find out 33 values. The number of iterations increases with increase in difficulty level. Form the figure, we see that very

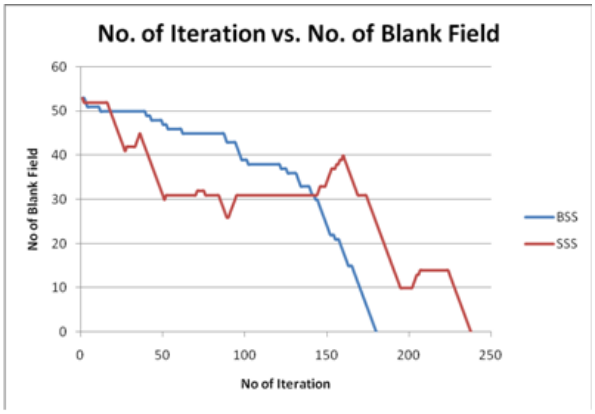

Fig. 9. Number of Iteration vs. number of empty cell for a very difficult Sudoku using BSS and SSS.

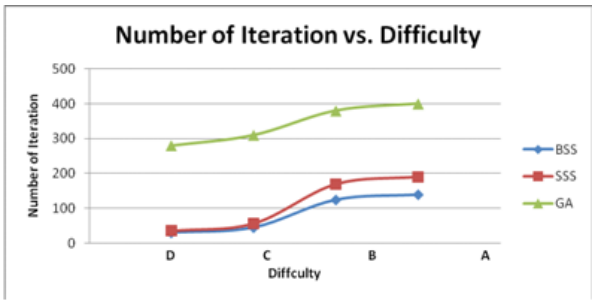

Fig. 10. The number of iteration vs. difficulty level between different methods of Sudoku solver.

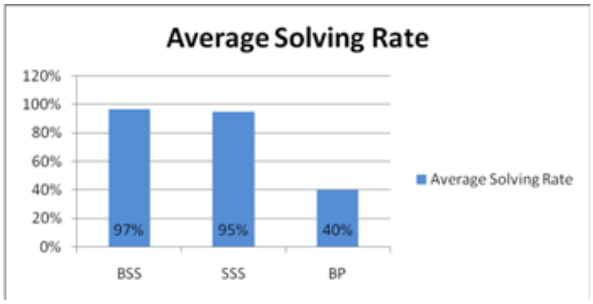

Fig. 11. Average solving rate of BSS, SSS and BP.

difficult Sudoku takes about 140 iterations to find out first 15 values and takes more 40 iterations to find out 37 values.

We solve a very difficult Sudoku using our proposed BSS, and the existing popular approach- Sinkhorn Sudoku Solution (SSS) [8] and count iterations and number of empty cells as shown in Fig. 9. BSS solves that Sudoku in 179 iterations, but SSS takes 234 iterations. Therefore, BSS takes less number of iterations than SSS.

Fig. 10 shows the number of iteration vs. difficulty level between different methods of Sudoku solver. Sudoku solved by Genetic Algorithm (GA) [1] can solve very difficult level Sudoku by taking 400 (average) iterations and SSS can solve it by taking 190 (average) iterations but BSS can solve it by taking only 140 (average) iterations. This also indicates the superiority of our approach.

Fig. 11 shows the solving rate of Sudoku using BSS, SSS and Belief Propagation (BP) [7]. The solving rates of BSS, SSS and BP are 97%, 95% and 40% respectively for our given Sudoku.

7. CONCLUSION AND FUTURE WORK

We present a new strategy to solve Sudoku using Boolean algebra. The main contribution of this paper is the solution of Sudoku with a reduction in consumption of memory and execution time. Our Boolean Sudoku solver (BSS) outperforms the existing popular approach with better accuracy. Simulation results depict that BSS is more faster in solving Sudoku due to usage of bitwise operation instead of arithmetic operation. The implemented strategy is enough to find the solution of many Sudoku problems. Parts of our future works are adding features of artificial intelligence to solve super difficult level Sudoku.

8. REFERENCES

[1] J. Almog, "Evolutionary Computing Methodologies for Constrained Parameter, Combinatorial Optimization: Solving the Sudoku Puzzle," in Proc. of IEEE AFRICON, pp. 1-6, September 2009, Nairobi, Kenya.

[2] B. Arnoldy, "Sudoku Strategies," The Home Forum (The Christian Science Monitor), Schaschek, Sarah, March 2006.

[3] H. Balakrishnan, I. Hwang and C. J. Tomlin, "Polynomial approximationalgorithms for belief matrix maintenance in identity management," in Proc. of IEEE Conf. Decision Control, pp. 4874-4979, December 2004.

[4] C. Boyer,"Sudoku's French ancestors," http://web.archive.org. Retrieved on 3 August 2009.

[5] D. Diaz-Pernil, C. M. Fernandez-Marquez, M. Garcfa-Quisrnondo, M. A. Gutierrez-Naranjo and M. A. Martinez-del-Arnor, "Solving Sudoku with Membrane Computing," in Proc. of IEEE BIC-TA, pp. 610-615, Changsha, November 2010.

[6] S. Kirkpatrick, "Message-passing and Sudoku," Available in http://www.www.cs.huji.ac.il/ kirk/Sudoku.ppt.

[7] T. K. Moon and J. H. Gunther, "Multiple constraint satisfaction by belief propagation: An example using Sudoku," in Proc. of SMCals/IEEE Mountain Workshop on Adaptive and Learning System, pp. 122-126, Logan, UT, July 2006.

[8] T. K. Moon, J. H. Gunther and J. J. Kupin , "Sinkhorn Solves Sudoku," Transactions On Information Theory, Vol. 55, No. 4, pp. 1741-1746, April 2009.

[9] Pegg, The Mathematical Association of America, http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html. Retrieved on October 3, 2006.

[10] Y. Sato and H. Inoue, "Solving Sudoku with Genetic Operations that Preserve Building Blocks," in Proc. of IEEE CIG, pp. 23-29, Dublin, September 2010.

[11] S. Schaschek, "Sudoku champ's surprise victory". The Prague Post. Retrieved on August 13, 2006.

[12] P. S. Segundo and A. Jimnez, "Using graphs to derive CSP heuristics and its application to Sudoku," In Proc. of IEEE ICTAI, pp. 538-545, Newark, NJ, November 2009.

[13] R. Sinkhorn, "A relationship between arbitrary positive matrices and oubly stochastic matrices," Ann. Math. Statist., vol. 35, pp. 876-879, June 1964.

[14] R. Sinkhorn, "Diagonal equivalence to matrices with prescribed row and column sums," Amer. Math. Monthly, vol. 35, pp. 876-879, 1967.