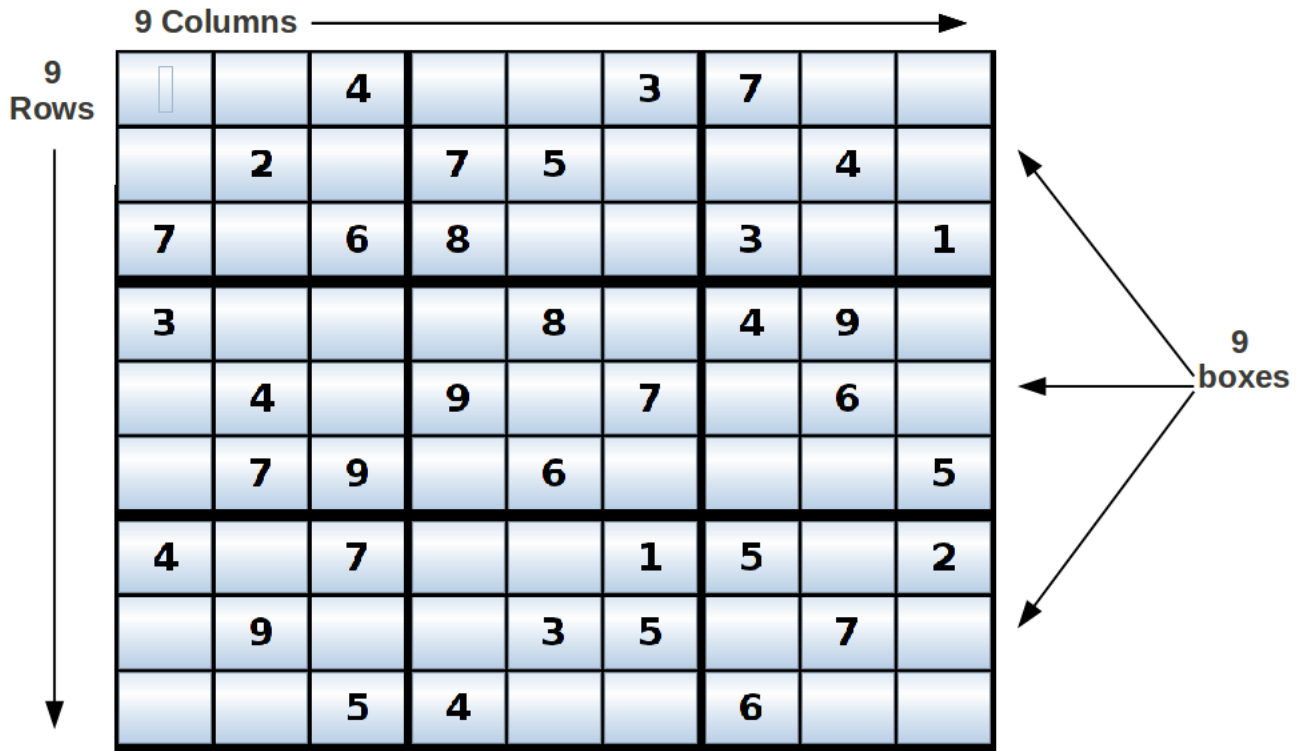


# Contents

<b>1</b>	<b>Chapter : INTRODUCTION</b>	<b>2</b>
1.1	Aim / Objective . . . . .	2
1.2	Existing System . . . . .	3
<b>2</b>	<b>Chapter: Propsed System</b>	<b>4</b>
2.1	Why sets? . . . . .	4
2.2	Introduction to BitVectors . . . . .	4
2.3	Advantages of BitVector . . . . .	5
2.4	Implementation details of Union ( $\cup$ ) operation . . . . .	5
2.5	Implementation details of Intersection ( $\cap$ ) operation . . . . .	6
2.6	Implementation details of Difference ( $-$ )operation . . . . .	6
<b>3</b>	<b>Chapter : System Design and Implementation</b>	<b>7</b>
3.1	Tools and Technologies used . . . . .	7
3.1.1	Languages Used . . . . .	7
3.1.2	Tools Used . . . . .	7
3.2	System Design . . . . .	7
3.2.1	Defining PossibleValues of a Cell . . . . .	8
3.2.2	Basic Filling . . . . .	8
3.2.3	Filling by Elimination . . . . .	8
3.2.4	Filling by Row and column Analysis . . . . .	9
3.3	System Implementation . . . . .	9
3.3.1	Algorithm . . . . .	10
3.3.2	Snapshots . . . . .	13
<b>4</b>	<b>System Testing and Results</b>	<b>15</b>
4.1	System Testing . . . . .	15
4.2	Results . . . . .	15
<b>5</b>	<b>Conclusions and Furture work</b>	<b>15</b>
5.1	Conclusions . . . . .	15
5.2	Future work . . . . .	16
<b>6</b>	<b>References</b>	<b>17</b>

# 1 Chapter : INTRODUCTION

Sudoku is a *mathematical, logical and permutational puzzle* which contains 9x9 grid of cells. Each of nine row, column and box (3x3) grid contains value from 1-9 without any repetitions. Using these properties one can try solving the puzzle. Puzzle is said to be solved if the entire 9x9 cells are filled with values with out any conflicts (any value repetiting in the same row, coloumn or box). There any many types of sudokus. Generic sudoku (the one discussed below and the most widely used), word sudoku, Jigsaw or Squiggle Sudoku, samurai sudoku etc.



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

Figure 1: Typical Sudoku example

## 1.1 Aim / Objective

The main endeavor of this project is to find a mathematical solution for solving sudoku. There exist many algorithms for generating sudokus, but not for solving them (only few exist). Hence we give a computerized solution to sudoku. The technique discussed in this report is a new approach which tries to reduce the space and time complexity with still having ability solve sudoku. The technique uses **SET Operations** to solve sudoku. This approach also determines whether a sudoku is solvable *without guess* or *not*.

## 1.2 Existing System

As specified earlier there exist many system which efficiently generate sudokus of different levels (easy, hard etc), but there are only few techniques which concentrate on providing solution to them. This is because it is generated so that it can be solvable by humans. But there are ceratain sudokus which require guess and are complex to solve. The traditional approaches were in the  $O(n^n)$  (**backtracking**) or  $O(n^5)$  (**brute force techniques**). These approaches have high time and space complexities. As there are only few techniques for solving sudoku, there is a lot of scope for new approaches for solving sudoku with having lower space and time complexities.

## 2 Chapter: Propsed System

The system proposed uses **SET operations** to solve sudoku. In this approach the main endeavor is to reduce the space and time complexity of the algorithms. A new Data Structures for SET is designed so that it reduces the complexities. Set operations such as *Union*, *Intersection*, *Difference*, *Compliment* are implemented. These set operations are applied on the cells of the sudoku to determine the values to be filled in these cells. The Set Data Structure is implemented using **BitVectors**. BitVector is another Data structure which uses only few bits to hold certain values and it best suits for implementing SET Data Structure.

### 2.1 Why sets?

The properties of Sodoku such as “**No Repetition**” of values in row, column and box and each of the cells can have a “**Set**” of possible values which can appear in the cell (because the order of the possible values does not matter). It is obvious that if a cell has a possible values ‘x’ and if it does not appear as a possible value in any of the other cell in the same box then ‘x’ can be filled in that cell. Determining such things can be easily done using “*union and difference*”. SETs are one of the important fields of discrete mathematics. Though it is not widely used in applications, but still has its own importance. Using SETs as a tool for solving Sudoku increases the efficiency and elegance.

### 2.2 Introduction to BitVectors

“**BitVectors**” is a Data Structure created specifically for implementing ‘SET’ and its associated operations. The importance of the Data Structure to solve a problem can be realized in *BitVectors*. As Sudoku can have a maximum of 9 digits the *Union*, *Intersection* and *Difference* can be implemented in  $O(1)$ . BitVector can be considered as a linked list of bits corresoponding to a particular number. **Base** represents the base value to the node and the **9-bits** represent nine numbers starting from *Base*. i.e., if a bit ‘*i*’ is set then a number ‘*i*’ + ‘*Base*’ exists. For e.g. Consider base value as ‘1’ and a 5<sup>th</sup> bit is set then a number ‘5’ exists in the set. It also contains a pointer which point to the next node in the list of BitVector.

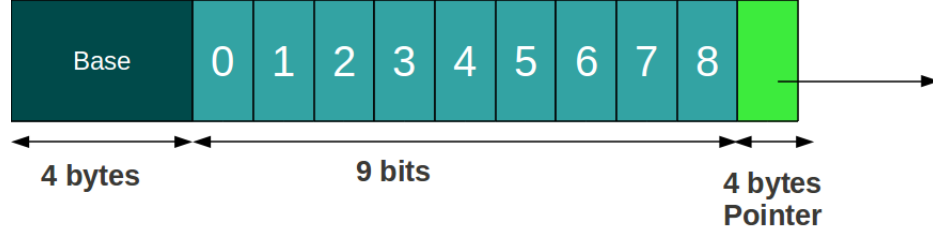


Figure 2: Representation of BitVector

## 2.3 Advantages of BitVector

Specific to this application only a single node with 9-bits are used. This representation of data structure is important in order to reduce the space and time complexity. If we want to store 9 values and decide to represent it in an array then it requires 9 positions in memory and each of which would require 4 bytes. So totally the array representation requires  $9 \times 4 = 36$  bytes, whereas BitVector requires only **8 bytes plus 1 bit** which saves around **77.5%** of space. We know that implementing Union, intersection and difference in an array is of  $O(n^2)$ . But the main advantage of BitVector lies here. **BitVectors implement the SET operation in  $O(1)$** . Let us see how BitSets implement Set operations in  $O(1)$ .

## 2.4 Implementation details of Union ( $\cup$ ) operation

The figure 3 illustrates how *union* operation is implemented in  $O(1)$  using BitVectors with the help of *OR* operation. As we know that union of two sets would result in a set which includes all the elements in both the sets and the repeated values are eliminated, similarly when we perform OR operation, if bits in two corresponding BitVectors are set, then the resulting BitVector will also have the same corresponding bit set, so it is counted only once. So to perform Union of two BitVectors it requires only one *OR* operation.

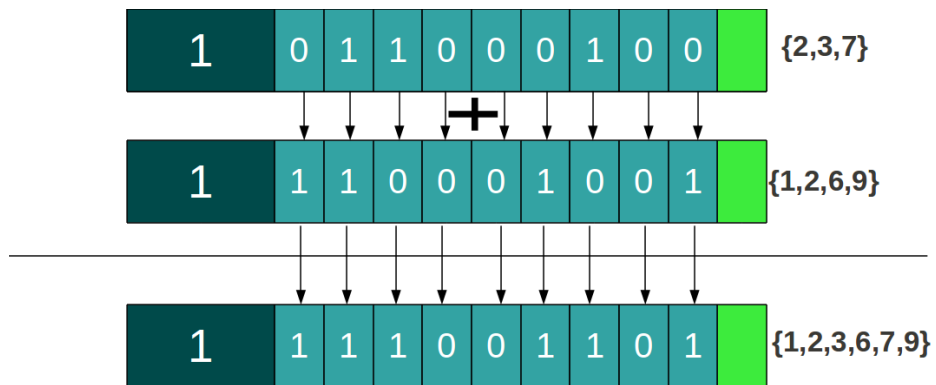


Figure 3: Union operation achieved by OR-ing two BitVectors

## 2.5 Implementation details of Intersection ( $\cap$ ) operation

The figure 4 illustrates how *intersection* operation is implemented in  $O(1)$  using BitVectors with the help of **AND** operation. *Intersection* of two sets would result in a set which would include elements which are there in both the sets. To achieve this **AND** operation is performed on the two BitVectors. If corresponding bits in both the BitVectors are set then only the corresponding bit in the resultant BitVector is set. So inorder to get Intersection of two BitVectors we need just one **AND** operation.

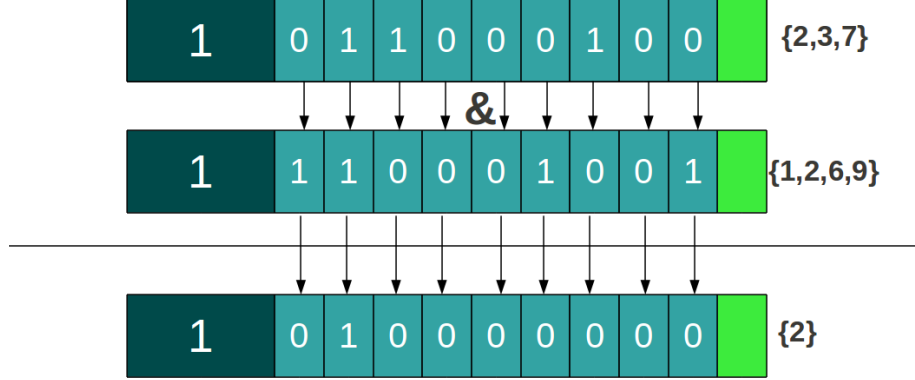


Figure 4: Intersection operation achieved by AND-ing two BitVectors

## 2.6 Implementation details of Difference ( $-$ ) operation

The figure 5 illustrates how *set difference* operation is implemented in  $O(1)$  using BitVectors by **ANDing** first BitVector with the *Complement* of second BitVector. *Set difference* of two sets would result in a set that includes elements from the first set which are not present in the second set. To achieve *Set difference* we need one **AND** and one *complement* operation.

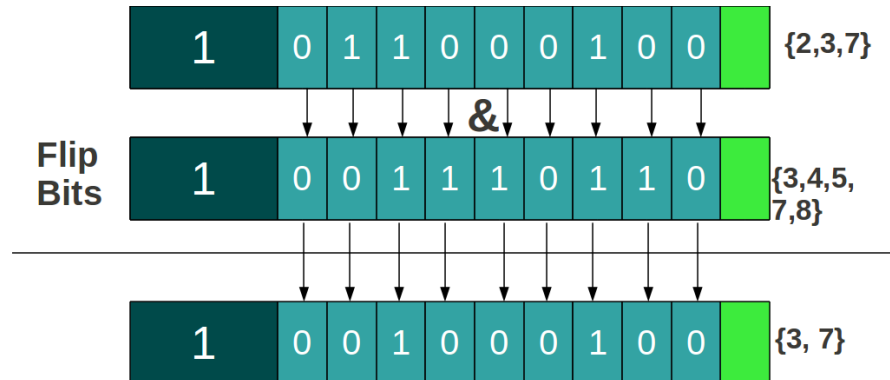


Figure 5: Set difference operation achieved by AND-ing first BitVector with complement of second BitVector

## 3 Chapter : System Design and Implementation

The System Design emphasizes on four techniques using which the sudoku is solved. each of the technique is discussed in the System design section. The four techniques to fill the cells of a sudodku are *Basic Filling*, *Elimination technique*, *Row and Column Analysis*. Each of the above techniques will be discussed in detailin System Design section.

### 3.1 Tools and Technologies used

#### 3.1.1 Languages Used

- C++ (Back end)
- Java (Front end) Swings

#### 3.1.2 Tools Used

- L<sup>A</sup>T<sub>E</sub>X(LaTeX for reporting)
- OpenOffice (for presentation)
- sudoku (terminal version of Sudoku)

Implementation of *BitVector* data structure and associated *SET* operations are implemented in *C++*. The front end (*GUI*) is implemented in *Java Swings*. Java program calls the C++ program and gives the unsolved sudoku as input. The C++ program dumps the solution to a text file. Java then parses that text file and stores them in the *solution string*. When Java GUI is asked to solve the Sudoku automatically then it uses that *solution string* to show solution to the sudoku in GUI. GUI also includes features for user to solve on his/her own combined with auto solver. A count down timer is provided to count the time taken to solve the Sudoku. User is provided controls to pause/resume the auto solver and also to clear the current sudoku to start from the beggining (clear the entries made by the user).

### 3.2 System Design

System Design as specified earlier concentrates on four techniques for solving sudoku. Each of the four techniques are independent and execute on their own, but solutions from each technique will help other techniques to determine values to the remaining unfilled cells. Each of the four techniques are dependent on the ***PossibleValues*** of each of the cell. In understanding the algorithm certain variables and Data structures need to be understood and are explained below.

**Set** - defines the Set Data Structure and its associated operations.  
**cells[n][n]** - defines the 9x9 grid cells.  
**row[i]** - defines row 'i' (A set)  
**column[n][i]** - defines column 'i'(A Set)  
**box[n]** - defines the 3x3 grid.  
**cells[i][j].PossibleValues** - define the possible values which might come in the cell.  
**box[n].cells[i]** - defines the cell 'i' in the box 'j'

### 3.2.1 Defining PossibleValues of a Cell

**PossibleValues** of a cell is also a Set containing the possible values of a sudoku cell. Possible Values of a cells is defined as set of values which are not there in the row, column and box, where the cell exists. Possible Values of a cell is defined mathematically as

Let  $C = \{ 1,2,3, \dots, 9 \}$

$\forall 0 \leq i, j \leq 8$

$cells[i][j].PossibleValues = C - (row[i] \cup column[j] \cup BOX(boxval(i, j)))$

where,  $boxval(i, j) = (\lfloor i/3 \rfloor) * 3 + (\lfloor j/3 \rfloor)$

### 3.2.2 Basic Filling

*Basic filling* is a filling technique where a cell is filled with a value if the cardinality of the set PossibleValues is equal to one. In other words If the cell can have only one value then just fill it. i.e. if  $|PossibleValues| = 1$  then fill that cell with PossibleValue.

### 3.2.3 Filling by Elimination

In *Filling by Elimination* technique, a cell 'c' of a box 'b' is filled with a value 'x' if 'x' is not a Possible Value in any of the other cell in the same box. i.e. A cell in a box is filled with a value if it cannot appear in any other cell in the same box. Filling By elimination can be mathematically defined as

if a cell 'j' is under consideration in the box 'i'

$Result = box[i].cell[j].PossibleValues - \bigcup_{q=0}^8 box[i].cell[q].PossibleValue, \text{ where } q \neq j$   
 if  $|Result| = 1$  then  $cell[j] = Result$



### 3.2.4 Filling by Row and column Analysis

This is to *Filling by elimination*, but instead of applying the technique to a box, it is applied to row and column. Each row and column is scanned for cells which satisfy the criteria to fill the cells.

#### Row Analysis

In *Filling by Elimination* technique, a cell 'c' of a row 'r' is filled with a value 'x' if 'x' is not a Possible Value in any of the other cell in the same row. i.e. A cell in a box is filled with a value if it cannot appear in any other cell in the same row. Filling By row analysis can be mathematically defined as

if a cell 'j' is under consideration in the row 'r'

$$Result = cell[r][j].PossibleValues - \bigcup_{q=0}^8 cell[r][q].PossibleValue, \text{ where } q \neq j$$

if  $|Result| = 1$  then  $cell[r][j]=Result$

#### Column Analysis

In *Filling by Elimination* technique, a cell 'c' of a column 'l' is filled with a value 'x' if 'x' is not a Possible Value in any of the other cell in the same column. i.e. A cell in a box is filled with a value if it cannot appear in any other cell in the same Column. Filling By column analysis can be mathematically defined as

if a cell 'j' is under consideration in the row 'l'

$$Result = cell[j][l].PossibleValues - \bigcup_{q=0}^8 cell[q][l].PossibleValue, \text{ where } q \neq j$$

if  $|Result| = 1$  then  $cell[j][l]=Result$

## 3.3 System Implementation

Implementation is done in two different languages. Front end in *Java Swings* and back end in *C++*. The four techniques discussed in the previous section are applied iteratively until solved. The *Basic Filling* is applied to all the cells iteratively. If none of the cells are solved with this technique we go for the next technique (*Filling by Elimination*) else we will repeat the same technique. And the same applies for other three techniques. If none of them were able to solve even one of the cell, then it is decided that the sudoku cannot be solved without guess. As explained earlier all these four techniques are independent of each other and they won't call each other. But output of one technique will be helpful for other techniques to determine values further.

### 3.3.1 Algorithm

In the algorithms discussed below operator use '+' for union, '-' for Set difference operation and it is a sudo code. i.e. it is not a source code of any language. Some of the simple function are not shown here such as *setValueAt(i,j,k)* means that set value 'k' to *cell[i][j]*, *getUnion(i,j)* means that get the union of possible values of other cells in the same box except the cell *cell[i][j]*.

```
filter()
{
    for(i=0 to 8)
        for(j=0 to 8)
            cell[i][j].PossibleValues = {1,2., .. 9} -
                                         (row[i] + column[j] + BOX(i,j))
}
```

The above algorithm is for generating possible values of all cells.

```
SolveSudoku()
{ do
    { do
        { n=0
          n=BasicFilling()
        }while(n>0)
      do
        { m=0
          m=FillByElimination()
        }while(m>0)
      do{ o=0
          o=RowAndColumnAnalysis()
        }while(o>0)
      if(n=0 AND m=0 AND o=0 ) then
      { print("Sudoku Cannot be solved Without Guess")
        return 0
      }
    } while ( Sudoku not solved )
  return 1
}
```

This algorithm applies all the four techniques iteratively through all the cells. It keeps on solving sudoku in one technique. If that technique fails then moves for the next

technique. If the last technique cannot determine further then again the first technique is started. If none of them were able to solve even a single cell then it is determined that sudoku is not solvable without guess and terminates.

```

FillByElimination()
{
    count=0;
    for(i=0 to 8)
        for(j=0 to 8)
            {
                if( cell not filled)
                {
                    temp=cells[i][j].PossibleValues
                    temp = temp - getUnion(i,j)
                    if( |temp| = 1)
                    {
                        setValueAt(i,j,k)
                        count++
                    }
                }
            }
        return count
}

```

The above algorithm determines value of cells using Elimination technique.

```

BasicFilling()
{
    count=0;
    for(i=0 to 8)
        for(j=0 to 8)
            {
                if(cells[i][j].value == 0)
                {
                    if( |cells[i][j].PossibleValues| = 1)
                    {
                        setValueAt(i,j,k)
                        count++
                    }
                }
            }
        return count;
}

```

The above algorithm determines value of cells using Basic filling technique.

```

fillUsingRowColumnAnalysis()
{
    n=0;
    for(i=0 to 8)
    {
        value=0
        Set currentRow=rows[i],temp=Set(1,2,...,9), temp2
        temp= temp-currentRow
        while((value=temp.getNextElement()))
            for(j=0 to 8)
                if( cells[i][j] to be filled)
                    if(cells[i][j].PossibleValues.search(value)= found)
                        {
                            temp2=cells[i][j].PossibleValues - getRowUnion(i,j)
                            if( |temp2| = 1)
                                {
                                    setValueAt(i,j,temp2)
                                    n++
                                }
                        }
    }
    for(j=0 to 8)
    {
        int value=0
        Set currentColumn=columns[j],temp=Set(1,2,...,9), temp2
        temp= temp-currentColumn
        while((value=temp.getNextElement()))
            for(int i=0 to 8)
                if(cells[i][j] to be filled)
                    if(cells[i][j].PossibleValues.search(value)= found)
                        {
                            temp2=cells[i][j].PossibleValues - getColumnUnion(i,j)
                            if( |temp2| = 1)
                                {
                                    setValueAt(i,j,temp2)
                                    n++
                                }
                        }
    }
    return n
}

```

The above algorithm determines value of cells using both row and column analysis

### 3.3.2 Snapshots

The figure 6 shows the empty sudoku where all the cells are empty. We can see a drop down menu which has different levels of difficulties such as *Easy*, *Medium*, *Hard* and *Very Hard*. The generate button would generate a sudoku depending upon the selection in the difficulty drop down menu. Time taken to solve the sudoku is also shown. The button *SolveAutomatically*, *Pause* and *Stop* are disabled initially. When a sudoku is generated, *SolveAutomatically* button and clear is enabled. When *SolveAutomatically* button is clicked the GUI automatically starts solving the sudoku and *Pause* button is enabled. When the entire sudoku is solved a popup will appear showing that sudoku is solved.

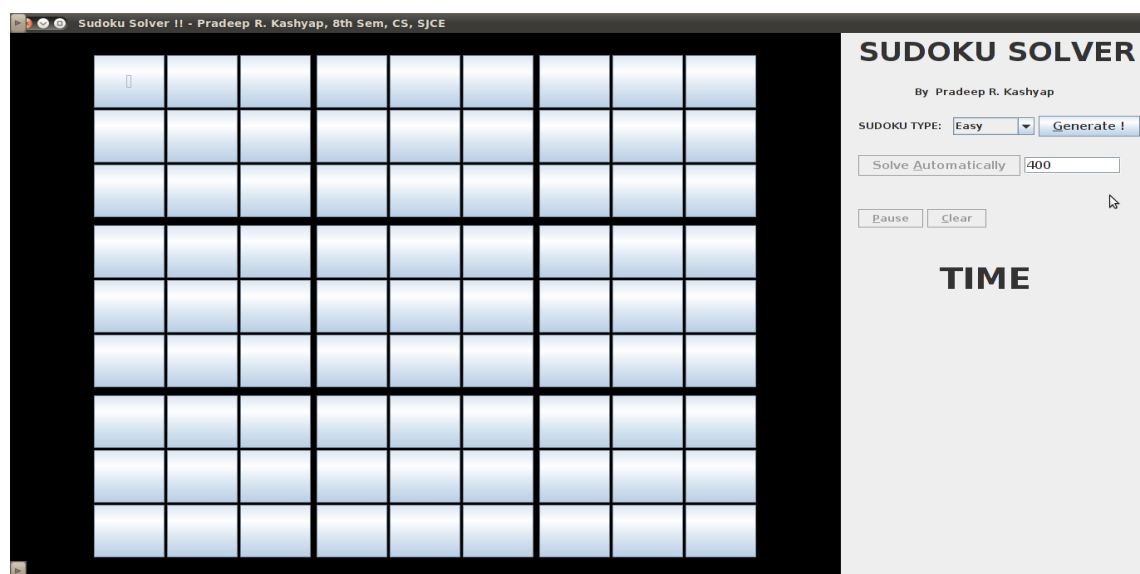


Figure 6: This figure shows the Empty Sudoku

When *SolveAutomatically* is clicked the *Solution String* stored is used to solve sudoku in GUI. The solution string is calculated at the time of generation of a sudoku. A text box is provided to give in the delay in how fast the computer should solve the sudoku. The *Pause* button is used to pause the automatic solving. *Clear* Button can be used to clear all the user made entries takes the sudoku to the initial state of the current problem. As as when the *Generate* button is clicked the counter starts ticking and when the sudoku is completed, it stops.

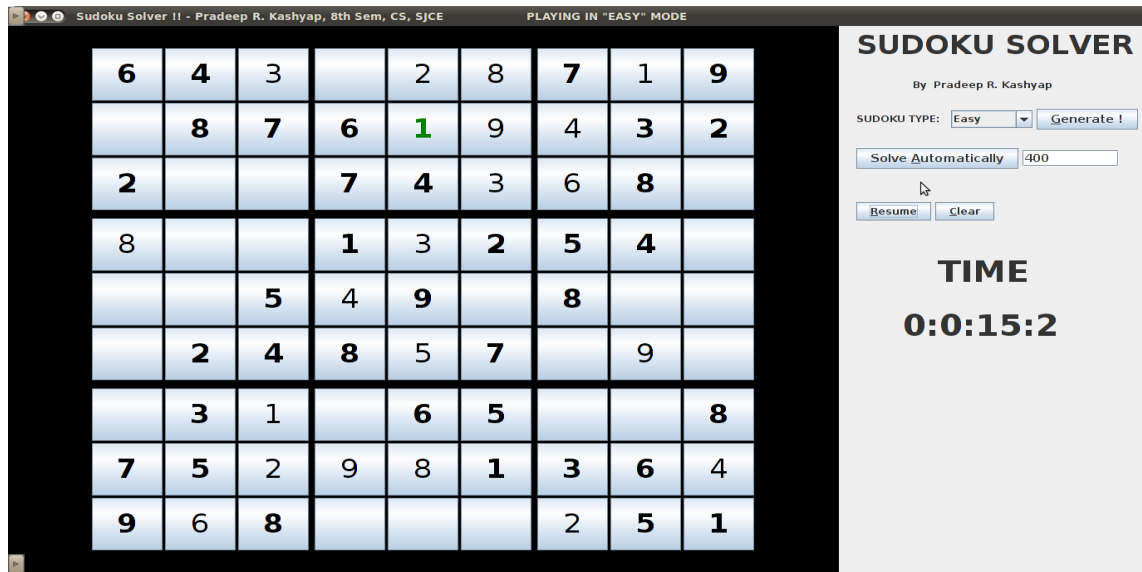


Figure 7: This figure shows GUI when SolveAutomatically Button is clicked

Sudoku can be solved using both *SolveAutomatically* button as well as the user can give in the solution. A cel can be filled by taking the control on to that cell and typing the value to be placed. When all the cells are filled without any conflicts.

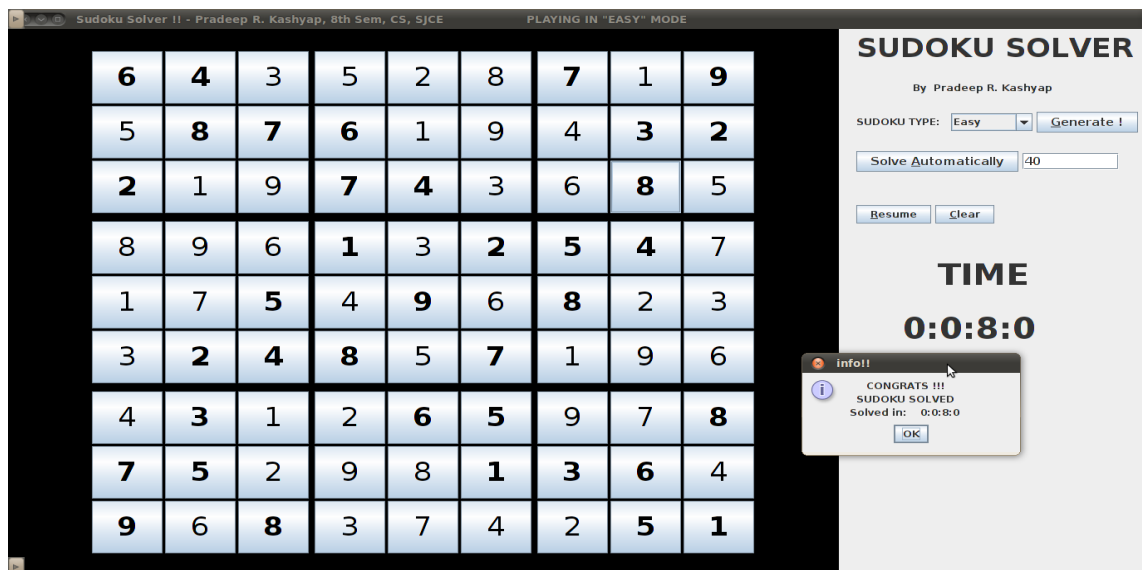


Figure 8: This figure shows the state of GUI when sudoku is completely solved.

## 4 System Testing and Results

### 4.1 System Testing

The System has been test for more that **1000** test cases including all the difficulty levels. The System works perfectly and provides complete solution to all the sudoku's till *Hard* level. The *Very Hard* level sudokus are not solvable without guess. Every partial solutions for such sudokus are provided. The system has been implemented such that whenever a wrong entry is made in a cell it shows the error. Hence we can verify whether the given solution (Automated or Mannual) is correct or not.

### 4.2 Results

The technique discussed provides solution to the sudoku in  $O(n^3)$ , where as other techniques give solution in  $O(n^5)$  (Brute force technique) and  $O(N^n)$  (for backtracking technique). The explanation for how  $O(n^3)$  is given below

$$O(n^2 \times (4(1) \times (3n))) = O(n^3)$$

where  $n^2$  is for each cell,  $4(1)$  is for 4 techniques and 1 is order of 1 for set operations,  $3n$  is for accessing cells in row, column and box. where as brute force technique without Set Data Structure requires  $O(n^5)$  and is explained below

$$O(n^2 \times (4(n^2) \times (3n))) = O(n^5)$$

where  $n^2$  is for each cell,  $4(n^2)$  is for 4 techniques and  $n^2$  is order of set operations (general array implementation of sets),  $3n$  is for accessing cells in row, column and box.

## 5 Conclusions and Furture work

### 5.1 Conclusions

- A mathematical appraoach was given to solve sudoku.
- Set operations are extensively applied and solutions are obatained.
- The technique is able to decide whether a sudoku is solvable without guess or not
- Implementation Set operations BitVector helped to reduce the order of set operations from  $O(n^2)$  to  $O(1)$ .
- This technique is efficeient interms of both space and time.

## 5.2 Future work

The proposed algorithm works at its best and  $O(n^3)$  is the best optimizations currently available. Due to the reduction in the order of set operations, this algorithm is able to run in  $O(n^3)$ . To improve the efficiency further any new algorithm introduced should be able to access  $\log(n)$  cells and no new technique can have order less than that.



## 6 References

- Discrete And combinatorial Mathematics,by Raplh P. Grimaldi
- <http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/WhiteSpace.html>
- <http://www.artofproblemsolving.com/Wiki/index.php/LaTeX:Symbols>
- <http://amath.colorado.edu/documentation/LaTeX/reference/figures.html>
- <http://tex.stackexchange.com/questions/8625/force-figure-placement-in-text>