For office use only	Team Control Number	For office use only
T1 _____	**233666**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**B**	F4 _____

2016

Mathematical Contest in Modeling (MCM/ICM) Summary Sheet

Sudoku Analyzing

Summary

here is the abstract! !!

Keywords: keyword1; keyword2

Sudoku Analyzing

Kai Feng, Song Lu, Yutao Zeng

October 5, 2016

Summary

here is the abstract! !!

Keywords: keyword1; keyword2

1 Introduction

1.1 Statement of Problem

We set out to design an algorithm that would construct unique sudoku puzzles of various difficulties as well as to develop metrics by which to measure the difficulty of a given puzzle. In particular, our algorithm must admit at least four levels of difficulty while minimizing its level of complexity.

1.2 Significance of Sudoku Research

Sudoku is one of the most pupular puzzle games of all time, which is famous for its simple rules and puzzle diversity. We think that this problem is insteresting and of great significance, due to its inherently mathematical, and offers us an oppotunity to explore new mathematical techniques. After studying, we found out that this problem could be regarded as a derivation of the Latin Square puzzle, and can be solved by using the exact-cover method.

Meanwhile, both solving and building the Sudoku puzzles are proved to be NP-Complete problems, which means that there is no known efficient way to locate an exact solution in the first place. It is apparently a great challenge for us to solve the problem in the polymonial time right now. But if we can make some progress,we may also expand into other and more practical problems. However, we shall restrict our focus directly to the problem at hand, and be content to leave these reasons, along with sudoku's entertainment value, as our motivation for exploring the game.

1.3 Sudoku Introduction

Sudoku, is a logic-based,combinatorial number-placement puzzle. The objextive is to fill a 9×9 grid with digits so that each column, each row, and each of the nine 3×3 subgrids that compose the grid contains all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a unique solution. Figure.1(a) is a typical example of sudoku puzzle.

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

(a) Typical sudoku puzzle

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) The same puzzle with solution

Figure 1: sudoku puzzle instance

Completed games are always a type of Latin square with an additional constraint on the contents of individual regions. For example, the same single integer may not appear twice in the same row, column, or any of the nine 3×3 subregions of the 9×9 playing board.

1.4 Notations and Terminologies

It is difficult to discuss our solution to the proposed problem without understanding some common terminology. Moreover, since we will apply more mathematical formalism here than in most documents dealing with sudoku, it will be helpful to introduce notational conventions.

- **Cell.** The basic unit of Sudoku puzzle. A square in the grid which may contain one digit(1-9). The grid is composed of 81 cells.
- **Block.** A 3×3 array of cells. Normally, the boundaries of the blocks are marked by slightly darker or thicker lines than the lines separating the cells. The grid is composed of 9 non-overlapping blocks. Each block must contain all the digits(form 1 to 9) and may not contain more than one of each digit.
- **Column.** A verticle line of 9 cells. The grid is composed of 9 columns. Each column must contain all the digits(1-9) and may not contain more than one of each digit.
- **Row.** A horizontal line of 9 cells. The grid is composed of 9 rows. Each row must contain all the digits (1-9) and may not contain more than one of each digit.
- **Grid.** The 9×9 array of cells that compose a Sudoku puzzle. The grid contains 9 rows, 9 columns and 9 blocks.
- **Puzzle.** A 9×9 matrix of cells, with at least one empty and at least one filled cell. For our purposes, we impose the additional requirement that all puzzles have exactly one solution.
- **House.** The column, row and block are collectively called the House.
- **Peer.** If two cells are in the same house (same row, same column, or same block) they are said to see each other, or to be peers.
- **Candidate.** Any digit that may be placed in an empty cell based on current state of the puzzle. If a digit is present in one or more of a cell's buddies, it cannot be a candidate for that cell. Analysis may further reduce the candidate set to a signle candidate, that candidate must be the sulution for that cell.
- **Analysis.** Any technique that eliminates candidates. Techniques of analysis do ultimately lead to solutions for cells, but it may take the application of multiple techniques or multiple applications of the same technique to reach a solution for a single cell. The point of analysis then, is to eliminate candidates, not look for solutions. Looking for solutions is scanning.

1.5 Common Solving Strategy

1.5.1 Full House/Last Digit

A Full House is simply the last digit that can be placed in a row,column or block. If it is the last digit for the whole grid, it is sometimes called "Last Digit".

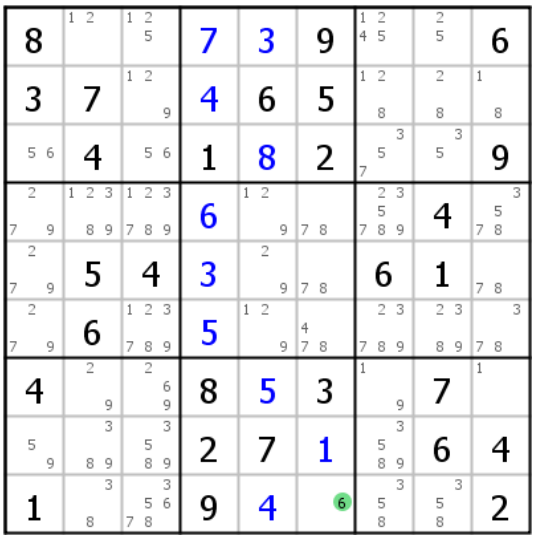
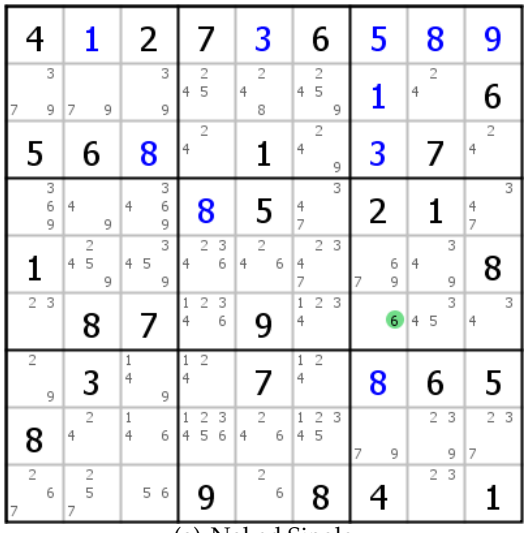


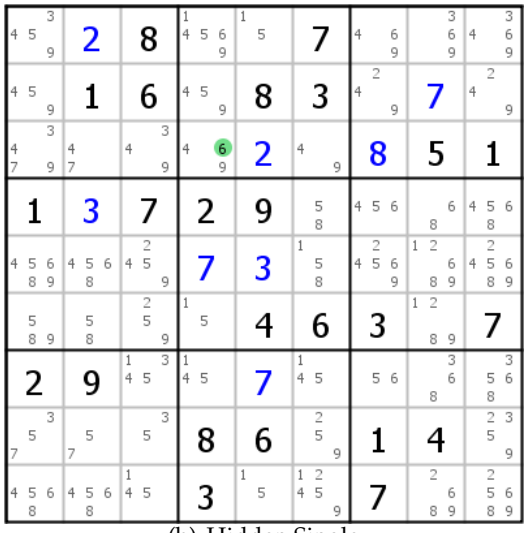
Figure 2: an example of full house

1.5.2 Naked Single

Naked Single means that in a specific cell only one digit remains possible (the last remaining candidate has no other candidates to hide behind and is thus naked). The digit must then go into that cell.



(a) Naked Single



(b) Hidden Single

Figure 3: Single Strategy

1.5.3 Hidden Single

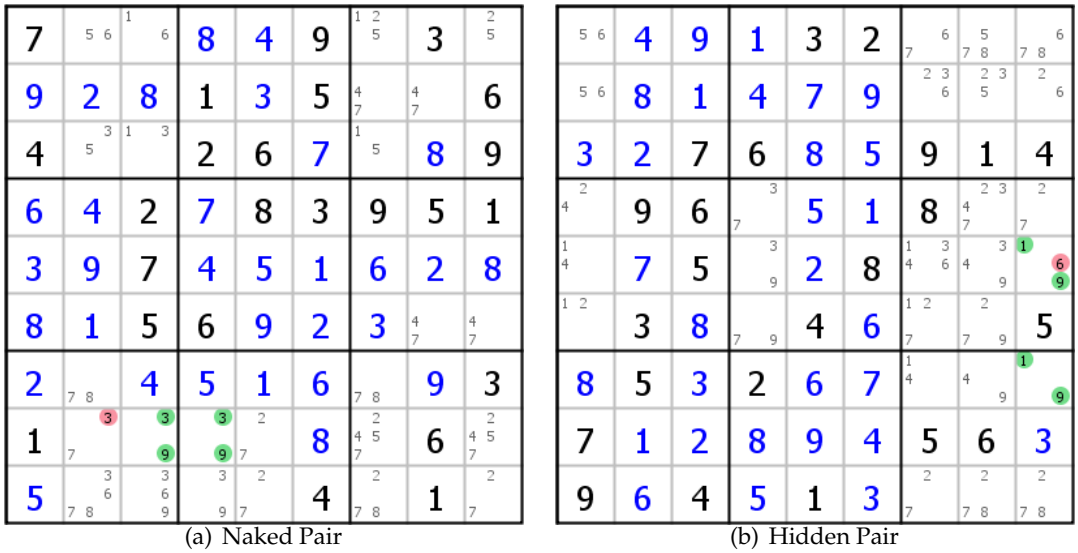
Hidden Single means that for a given digit and house only one cell is left to place that digit. The cell itself has more than one candidate left, the correct digit is thus hidden amongst the rest.

1.5.4 Naked Pair

If you can find two cells, both in the same house, that have only the same two candidates left, you can eliminate that two candidates from all other cells in that house.

1.5.5 Hidden Pair

All Hidden Subsets work the same way, the only thing that changes is the number of cells and candidates affected by the move. Take Hidden Pair: If you can find two cells within a house such as that two candidates appear nowhere outside those cells in that house, those two candidates must be placed in the two cells. All other candidates can therefore be eliminated.



(a) Naked Pair (b) Hidden Pair

Figure 4: Pair Strategy

1.5.6 Hidden Triple

Hidden Triples work in the same way as Hidden Pairs only with three cells and three candidates.

All the strategies above will be used as the benchmark to test the difficulty of new-created sudoku puzzles.

2 Analysis of the Problem

2.1 Background Knowledge

2.1.1 Exact Cover

In mathematics, given a collection $\mathcal{S}$ of subsets of a set $\mathcal{X}$, an **exact vocer** is a subcollection $\mathcal{S}^*$ of $\mathcal{S}$ that satisfies two conditions:

- The intersection of any two sistinct subsets in $\mathcal{S}^*$ is empty, i.e., the subsets in $\mathcal{S}^*$ are pairwise disjoint. In other words, each element in $\mathcal{X}$ is contained in at most one subset in $\mathcal{S}^*$.
- The union of the subsets in $\mathcal{S}^*$ is $\mathcal{X}$, i.e., the subsets in $\mathcal{S}^*$ cover $\mathcal{X}$. In other words, each element in $\mathcal{X}$ is contained in at least one subset in $\mathcal{S}^*$.

In short, an exact cover is "exact" in the sense that each element in $\mathcal{X}$ is contained in exactly one subset in $\mathcal{S}^*$. Figure 5 is an example of exact cover. As we can see the set(row 1,4 and 5) is the exact cover of the origin matrix.

$$\begin{pmatrix} 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

Figure 5: an example of exact cover

2.1.2 Algorithm X

To solve the exact cover problem, we need to introduce the methodology called Algorithm X. "Algorithm X" is the name D.E Knuth used in his paper "Dancing Links" to refer to "the most obvious-trail-and-error approach" for finding all solutions to the exact cover problem. Technically, Algorithm X is a recursive, nondeterministic, depth-first, backtracking algorithm. While Algorithm X is generally useful as a succinct explanation of how the exact cover problem may be solved, Knuth's intent in presenting it was merely to demonstrate the utility of the dancing links technique via an efficient implementation he called DLX.

The exact cover problem is represented in Algorithm X using a matrix A consisting of 0s and 1s. The goal is to select a subset of the rows so that the digit 1 appears in each column exactly once. Algorithm X functions as follows:

Algorithm X

1. If the matrix A has no columns, the current partial solution is a valid solution; terminate successfully.

2. Otherwise choose a column *c* (deterministically).
3. Choose a row *r* such that $A_{r,c} = 1$ (nondeterministically).
4. For each column *j* such that $A_{r,j} = 1$,
   for each row *i* such that $A_{i,j} = 1$,
   delete row *i* from matrix *A*,
   delete column *j* from matrix *A*.
5. Repeat this algorithm recursively on the reduced matrix *A*.

For better understanding the procedure of Algorithm X, let's see the following example. Consider the exact problem specified by the universe U = {1, 2, 3, 4, 5, 6, 7} and the collection of sets $\mathcal{S}$ = {*A, B, C, D, E, F*}, where:

- *A* = {1, 4, 7};
- *B* = {1, 4};
- *C* = {4, 5, 7};
- *D* = {3, 5, 6};
- *E* = {2, 3, 6, 7};
- *F* = {2, 7};

This problem is presented by the matrix:

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Table 1: Origin Matrix

For the least recursive time, we often select the column with the least 1s to start. Let's go on and solve this problem.

Level 0

Step 1 – The matrix is not empty, so the algorithm proceeds.

Step 2 – The lowest number of 1s in any column is two. Column 1 is the first column with two 1s and thus is selected(deterministically):

Step 3 – Rows *A* and *B* each have a 1 in column 1 and thus are selected (nondeterministically).

The algorithm moves to the first branch at level 1.

Level 1: Select Row A

Step 4 – Row *A* is included in the partial solution.

Step 5 – Row *A* has a 1 in columns 1, 4, and 7:

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Column 1 has a 1 in rows *A* and *B*; column 4 has a 1 in rows *A*, *B*, and *C*; and column 7 has a 1 in rows *A*, *C*, *E*, and *F*. Thus rows *A*, *B*, *C*, *E*, and *F* are to be removed and columns 1, 4 and 7 are to be removed:

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

Row D remains and columns 2, 3, 5, and 6 remain:

	2	3	5	6
D	0	1	1	1

Step 1 – The matrix is not empty, so the algorithm proceeds.
Step 2 – The lowest number of 1s in any column is zero and column 2 is the first column with zero 1s:

	2	3	5	6
D	0	1	1	1

Thus this branch of the algorithm terminates unsuccessfully. The algorithm moves to the next branch at level 1.

Level 1: Select Row B
Step 4 – Row *B* is included in the partial solution.Row *B* has a 1 in columns 1 and 4:

	1	2	3	4	5	6	7
A	1	0	0	1	0	0	1
B	1	0	0	1	0	0	0
C	0	0	0	1	1	0	1
D	0	0	1	0	1	1	0
E	0	1	1	0	0	1	1
F	0	1	0	0	0	0	1

After several steps, we can get the matrix:

	2	7
F	1	1

Notice that the next step we can clear all the rest 1s in the matrix. So it is a correct answer to this exact cover problem. As rows B, D, and F are selected, the final solution is:

	1	2	3	4	5	6	7
B	1	0	0	1	0	0	0
D	0	0	1	0	1	1	0
F	0	1	0	0	0	0	1

2.1.3 Sudoku to Exact Cover

2.2 Model Building

3 Matrix Design

4 The Model Results

5 Evaluate of the Mode

5.1 Advantage of the Model

5.2 Disadvantage of the Model

6 Validating the Model

7 Conclusions

in short but accurate

8 A Summary

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

lipsum latex

9 Strengths and weaknesses

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

9.1 Strengths

- **Applies widely**
  This system can be used for many types of airplanes, and it also solves the interference during the procedure of the boarding airplane,as described above we can get to the optimization boarding time.We also know that all the service is automate.
- **Improve the quality of the airport service**
  Balancing the cost of the cost and the benefit, it will bring in more convenient for airport and passengers.It also saves many human resources for the airline.
-

References

[1] Donald E. Knuth: Dancing Links, Oxford-Microsoft Symposium on Computer Science, 2000

[2] Wikipedia: Sudoku

[3] `http://blog.gssxgss.me/use-dlx-to-solve-sudoku-1/`

Appendices

Appendix A First appendix

Here are simulation programmes we used in our model as follow.

Listing 1 : Implement of Sudoku Creation

```
package mcm;

public class Sudoku {
    public SudokuPoint[][] matrix;

    public Sudoku(){
        matrix = new SudokuPoint[10][10];
    }

    public boolean testRow(){
        boolean b = true;
        boolean[] test = new boolean[10];
        boolean[] vaild = new boolean[10];
        for(int i = 1; i <= 9 ; i++){
            for(int j = 1; j <= 9; j++){
                if(test[matrix[i][j].value] == false){
                    test[matrix[i][j].value] = true;
                }
                else {
                    break;
                }
            }
            boolean s = true;
            for(int k = 1; k <= 9; k++){
                s = s && test[k];
            }
            if(!s){
                break;
            }
            else {
                vaild[i] = true;
                test = new boolean[10];
            }
        }
        for(int i = 1; i <= 9; i++){
            b = b && vaild[i];
        }
        return b;
    }

    public boolean testColumn(){
        boolean b = true;
        boolean[] test = new boolean[10];
        boolean[] vaild = new boolean[10];
        for(int j = 1; j <= 9 ; j++){
            for(int i = 1; i <= 9; i++){
                if(test[matrix[i][j].value] == false){
                    test[matrix[i][j].value] = true;
                }
```

```
                else {
                    break;
                }
            }
            boolean s = true;
            for(int k = 1; k <= 9; k++){
                s = s && test[k];
            }
            if(!s){
                break;
            }
            else {
                vaild[j] = true;
                test = new boolean[10];
            }
        }
        for(int i = 1; i <= 9; i++){
            b = b && vaild[i];
        }
        return b;
    }

    public boolean testGrid(){
        boolean b = true;
        boolean[] test = new boolean[10];
        boolean[] vaild = new boolean[10];
        for(int i = 1; i <= 3; i++){
            for(int j = 1; j <= 3; j++){
                for(int k = 1; k <= 9; k++){
                    int offset = k <= 3 ? 1 : k <= 6 ? 2 : k <= 9 ? 3 : -1;
                    if(test[matrix[3 * (i - 1) + offset][(3 * (j - 1) + k - (offset - 1) * 3)].
                        test[matrix[3 * (i - 1) + offset][(3 * (j - 1) + k - (offset - 1) * 3)]
                    }
                    else {
                        break;
                    }
                }
                boolean s = true;
                for(int l = 1; l <= 9; l++){
                    s = s && test[l];
                }
                if(!s){
                    break;
                }
                else{
                    vaild[3 * (i - 1) + j] = true;
                    test = new boolean[10];
                }
            }
        }
        for(int i = 1; i <=9; i++){
            b = b && vaild[i];
        }
        return b;
    }

    public boolean test(){
        return testColumn() && testRow() && testGrid();
    }
}
```

```
package mcm;

public class SudokuPoint {
    public SudokuPoint topPoint;
    public SudokuPoint bottomPoint;
    public SudokuPoint leftPoint;
    public SudokuPoint rightPoint;
    public int value;

    public SudokuPoint(){
        topPoint = null;
        bottomPoint = null;
        leftPoint = null;
        rightPoint = null;
        value = 0;
    }

    public SudokuPoint(SudokuPoint topPoint, SudokuPoint bottomPoint, SudokuPoint leftPoint, Su
        this.topPoint = topPoint;
        this.bottomPoint = bottomPoint;
        this.leftPoint = leftPoint;
        this.rightPoint = rightPoint;
        this.value = value;
    }
}
```

Appendix B Second appendix

some more text **Input C++ source:**

```
//============================================================================
// Name        : Sudoku.cpp
// Author      : wzlf11
// Version     : a.0
// Copyright   : Your copyright notice
// Description : Sudoku in C++.
//============================================================================

#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int table[9][9];

int main() {

    for(int i = 0; i < 9; i++){
        table[0][i] = i + 1;
    }

    srand((unsigned int)time(NULL));

    shuffle((int *)&table[0], 9);

    while(!put_line(1))
```

```
    {
        shuffle((int *)&table[0], 9);
    }

    for(int x = 0; x < 9; x++){
        for(int y = 0; y < 9; y++){
            cout << table[x][y] << " ";
        }

        cout << endl;
    }

    return 0;
}
```