Sudoku puzzles

Abstract: Targeting the unique numerical Rubik's cube – Sudoku, we have studied a variety of heuristic intelligence algorithm, metrics of difficulty levels, a variety of Sudoku derivative algorithm as well as a range of principles that are sufficient to ensure the unique solution of Sudoku puzzles. Finally, we established a highly efficient randomized Sudoku puzzle generator.

First of all, we developed a computer backtracking algorithm of relatively high efficiency (see 4.5) which can quickly resolve all Sudoku puzzles(including those with multiple solutions) and developed basic metrics for mechanically assessing the difficulty degrees of Sudoku puzzles. On the basis of that, we created a heuristic intelligence algorithm system (see section 4) with multi-level self-detective mechanism targeting real Sudoku puzzles that have unique solutions. Such intelligence algorithm system can, in addition, stimulate manual puzzle-solving steps and record each step.

Next, we considered comprehensively the pros and cons of the long term development of Sudoku games and defined metrics of five difficulty levels with vague boundary standards. Within the levels we established, we have provided numerical and quantitative descriptions for the difficulties of the puzzles in the same level.

We believe, to generate a real Sudoku puzzle, uniqueness of solution is crucial. Therefore, it is the main and strict constraint of creating algorithm. In 5.2, we have listed the three principles that have to be followed to ensure the uniqueness of the solution.

Another key point of our considerations is to generate Sudoku puzzles of diversity, and we believe the diversity of the puzzles we generate bare direct influence to the market outlook for the algorithm. Therefore, regarding the diversity of the puzzles generated, we have provided up to 8 different transformed and derived methods (see 5.1), which will meet the diversity requirements of Sudoku puzzles.

Based on the above studies, the algorithm we have established can generate a variety of Sudoku puzzles with different difficulty levels subject to individual requirements of players. Our algorithm can also be used as a Sudoku puzzle generator.

Finally, we have, through the establishment of the optimal model (see 6.2), studied the complexity of the puzzle generating system and the changing value of the corresponding parameter setting at different degrees of difficulty.

Keywords: Numerical Rubik's cube, Heuristic intelligence algorithm, Derivatives, Backtracking algorithm, Self-detective mechanism, Optimal model

Contents

1 Introduction

Sudoku puzzle is a challenging mathematic problem and it is also a popular logic reasoning game. The idea of the puzzle is extremely simple; the player is faced with a 9 × 9 grid divided into nine 3 × 3 blocks. In some of these boxes, the setter puts some of the numbers 1~9 whereas other entries are left blank to form puzzles with various difficulty levels; the aim of the solver is to complete the grid by placing in a number in every box in such a way that each row, each column, and each block contains each of the numbers 1~9 exactly once by logic reasoning according to the number distribution of the puzzle.

It is not only solving Sudoku puzzle is interesting but also creating Sudoku puzzles is a hobby of many people. Meanwhile, it is a favorite of many people to develop algorithms of creating and solving Sudoku puzzles and make them carry out by computer.

2 Definitions

- Block(i.e.mini-square): A 3×3grid with 9 boxes.
- Row and column: There are 9 rows and 9 columns in a Sudoku puzzle.
- Degree of freedom: The amount of available number of a box.
- Fixed box: Box with number in the initial Sudoku puzzle.
- Undetermined box: Box with the degree of freedom more than one.
- Proper puzzle: Sudoku puzzle with a unique solution.

3 Problem Analysis

In this paper, the main goal is to develop an algorithm and difficulty levels metrics to construct Sudoku puzzles of varying difficulty. The algorithm and metrics should possess the following character:

- There are at least 4 difficulty levels and the difficulty degree with numerical and quantitative descriptions.
- Algorithm and metrics are extensible.
- Algorithm should guarantee a unique solution.
- Complexity of the algorithm should be minimized and measurable.
- It can create Sudoku puzzles efficiently and quickly.
- Symmetric Sudoku puzzles are better although there are lots of asymmetric puzzles.

4 Heuristic intelligence algorithm system

To create a Sudoku puzzle, we believe, the primary step is to know how to resolve a Sudoku puzzle. Kinds of solving approaches are introduced in this section. We class them into three types as A, B and C according to their own character.

4.1 Preparatory work

4.1.1 Notes

Notes	Definitions
$P(x,y)$	Position of each box; x is the row number and y is the column number. $(x, y = 1\cdots 9)$
$Receive(x,\ y)$	Keep the number filled in box $P(x,y)$
$Record(x,y)$	Number in box $P(x,y)$ is determined or not
$Possible(x,y,k)$	Keep the possibility of box $P(x,y)$ $(k=1\cdots 9)$

Specific functions of variables:

- $Receive(x,\ y)$: This variable is to keep the fixed boxes of an initial puzzle and the final results. Blank boxes are initialized to zero particularly.
- $Record(x,y)$: If $Receive(x,y)\neq 0$, $Record(x,y)=1$ else $Record(x,y)=0$. We can fill in box with an assured number when $Record(x,y)=1$ else we cannot confirm which number is suitable to the box.
- $Possible(x,y,k)$: This variable is to record possibility of the undetermined box. The method of records is to initialize variable as

$$Possible(x,y,k)=k\,(x,y,k=1\cdots 9)$$

During the process of computation, $Possible(x,y,k)$ changes to -1 as a mark when we find number k $(k=1\cdots 9)$ is not suitable to fill in this box.

4.1.2 Data pretreatment

Well posed puzzles should have a unique solution and the task is to find it without guessing when men are the players. Logic reasoning all of its possible numbers placed into whichever box is an indispensable job by the constrained rule of Sudoku. This job can be done by computer (the function is *modifyPb*()). All of these possible numbers are called as *hint* numbers (possible numbers).

We present Sudoku square figures with *hint* numbers to illustrate algorithm of solving and creating Sudoku puzzles.

4.2 Type A approaches

4.2.1 Item A1

If the degree of freedom of a box λ is 1, the sole relevant number should be placed into this box.

As shown in the gray box of Figure 1, 8 is the unique choice that satisfies Sudoku rule, and thus it should be placed in $P(1,5)$ shown in Figure 1.

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

Figure 1

Algorithm details:

***Step* 1**: Search $Possible(x, y, k)$. If the three conditions:

$$Record(x, y) = 0$$

$$Possible(x, y, k) = k,$$

$$Possible(x, y, t) = -1 \ (t = 1 \cdots 9, t \neq k)$$

are satisfied, the next step continues，else jump to ***Step* 3**.

***Step* 2**: Assign $Receive_{ij} = k$ and $Record_{ij} = 1$.

***Step* 3:** Stop searching and exit.

4.2.2 Item A2

If a possible number only appears once in a row, column or block, it must be the sole suitable number that can be placed in this box.

As shown in the box of Figure 2, in column 3, number 4 appears only once in box $P(9,3)$.

So this box can only be placed with number 4.

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

Figure 2

Algorithm details (We take an example that a number only appears once in a column to illustrate it**)**:

***Step* 1**: Search $Possible(x, y, k)$. If there is a box $P(x, y)$ with conditions:

$$Record(x, y) = 0 \ ,$$

$$Possible(x, y, k) = k \ ,$$

$$Record(i, m) = 0 \ (m = 1 \cdots 9, m \neq j) \ ,$$

next step continues, else jump to ***Step* 4**.

***Step* 2**: If $Possible(x, y, k) \neq k (m = 1 \cdots 9, m \neq j, n = 1 \cdots 9)$, jump to ***Step* 3**, else jump to ***Step* 4**.

***Step* 3**: Assign $Receive_{ij} = k$ and $Record_{ij} = 1$.

***Step* 4:** Stop searching and exit.

4.2.3 Item A3

If a box has two possible numbers and another box in the same row (or column, block) has the same possible numbers, the two possible numbers can be only placed in these two boxes respectively. We call this kind of two boxes as ***number couple***.

As shown in Figure 3, a number couple appears in $P(2,1)$ and $P(7,1)$ which leads 7 and 8 can be excluded in other boxes of this column.

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

Figure 3

Algorithm details (We take an example that a number couple appears in a column to illustrate it**)**:

***Step*1**：Search $Possible(x, y, k)$. If there are two undetermined boxes $P(m, y)$ and $P(n, y)$ that satisfies

$$Possible(m, y, p) = p,$$
$$Possible(n, y, p) = p,$$
$$Possible(m, y, q) = q$$
$$Possible(n, y, q) = q$$

where $m \neq n,\ p \neq q\ (m, n, p, q = 1 \cdots 9)$ and no other possible numbers in the two boxes., jump to ***Step* 2**, else jump to ***Step* 4**.

***Step*2**：Search $Record(x, y)$. If there is

$$Record(t, y) = 0\ (t = 1 \cdots 9, t \neq m, t \neq n),$$
$$Possible(t, y, p) = p$$
$$Possible(t, y, q) = q.$$

jump to ***Step* 3**, else to ***Step* 4**.

***Step*3**：Assign $Possible(t, y, p) = -1$ and $Possible(t, y, q) = -1$.

***Step* 4:** Stop searching and exit.

4.3 Type B approaches

Chain-number algorithm B1 and Connotative Chain-number algorithm B2 are this type. Chain-number algorithm is used to exclude the same possible numbers in other chain boxes; Connotative Chain-number algorithm is used to exclude other possible numbers in this box in

order to avoid collision.

Definitions:

- Chain: It is an extension for number couple. Some possible numbers only distributes among certain boxes in the same row (or column, block) and don't appear in the left boxes. These possible numbers are called chain numbers. These certain boxes (at least three) consist of a chain. The length of chain is the amount of chain boxes i.e. the amount of chain numbers. Another important character of a chain is that chain boxes have no redundant possible numbers except chain numbers.
- Connotative Chain: Connotative chain has the same character with chain except two differences. One is connotative chain boxes may have other number except chain numbers; the other is the minimum amount of connotative chain boxes are two not three.

4.3.1 Item B1

At first, we introduce the easiest approach item B1of Chain-number algorithm with the length of chain being three. In other words, the three chain numbers can be placed only in these three chain boxes to avoid collision.

For example, in Figure 4, chain boxes are $P(8,4)$, $P(8,5)$and $P(9,5)$. Chain numbers are 2, 6, and 8. It can be concluded that number 6 and 8 should be excluded in the left boxes.

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

Figure 4

Algorithm details (We take an example that there is a chain in a row to illustrate it; chain length is 3**)**

***Step* 1**：Provided the chain numbers are r, s and t, search $Possible(x, y, k)$. If

$$Possible(i, u, w) = -1 (w = 1 \cdots 9, w \neq r, w \neq s, w \neq t),$$

and there is at least one variable value among $Possible(i, u, r)$, $Possible(i, u, s)$ and $Possible(i, u, t)$ $(u = 1 \cdots 9)$ to be -1, jump to ***Step 2***, else to ***Step* 4**.

Step 2：Search $Record(x, y)$. If

$$Record(i, k) = 0 \ (k = 1 \cdots 9, k \neq r, k \neq s, k \neq t),$$

and there is at least one variable value among $Possible(i, k, r)$, $Possible(i, k, s)$ and $Possible(i, k, t)$ to be -1, jump to ***Step 3***, else to ***Step* 4**.

Step 3：If $Possible(i, k, g) = g$ $(g = r, s, t)$, make $Possible(i, k, g) = -1$.

Step 4：Stop searching and exit.

4.3.2 Item B2

At first, we introduce an easy approach of Chain-number algorithm B2 which the length of chain is two. In other words, in the same row (or column, block), some two numbers appears in two certain boxes. Now other possible numbers in the two chain boxes can be excluded to avoid collision.

For example, in row 1of Figure 5 with gray shading, number 2,5,6,8 and 9 are the possible numbers of box $P(1,7)$; number 4, 5, 6, and 8 are the possible numbers of box $P(1,8)$. It can be reasoned that number 6 and 8 are chain numbers, and thus 2, 5, 9 are excluded in box $P(1,7)$, so do number 4 and 5 in box $P(1,8)$.

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

Figure 5

Algorithm details (We take an example that there is a connotative chain in a row to illustrate it; the length of chain is 2**)**:

***Step 1*:** Search $Possible(x, y, k)$. If there are two undetermined boxes $P(x, m)$ and $P(x, n)$ where

$$Possible(i, m, p) = p,$$

$$Possible(i, n, p) = p,$$

$$Possible(i, m, q) = q,$$

$$Possible(i, n, q) = q,$$

$$m \neq n, p \neq q.$$

jump to ***Step 3***, else to ***Step 4***.

Step 2*:** Search $Record(i, k) = 0$ $(k = 1 \cdots 9, k \neq j)$. If $Possible(i, k, p) = p$ or $Possible(i, k, q) = q$, jump to ***Step 3, else to ***Step 4***.

***Step 3*:** Make $Possible(i, m, t) = -1$ and $Possible(i, n, t) = -1 (t = 1 \cdots 9, t \neq p, t \neq q)$.

***Step 4*:** Stop searching and exit.

4.3.3 Derivative of algorithm of Type B

We can get derivative approaches of B1 and B2 by logic analogism. Algorithms of type B are extended.

- Derivative approach of item B1

When there is a chain in the same row (or column, block) with four chain numbers, the four chain numbers can be placed only in these four chain boxes to avoid repetitive.

- Derivative approach of item B2

When there is a connotative chain in the same row (or column, block) with three (or four) chain numbers, the redundant possible numbers except chain numbers can be excluded to avoid collision.

4.4 Type C approaches

4.4.1 Item C1

In this case, some possible numbers are in the intersection of a row (or column) and a block but never appears in the same row again. If so, these possible numbers cannot be placed in

other boxes of this block.

As in the following example in Figure 6, number 6 in boxes $P(4,2)$, $P(4,3)$ can be excluded.

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

Figure 6

Algorithm illustration (We take an example that there is an intersection of a row and a block to illustrate it):

Step1: Search $Possible(x, y, k)$. As to the same row，if number k is the possible number of boxes in column $y_1, y_2 \cdots y_n$， and $(y_1 - 1)/3 = (y_2 - 1)/3 = \cdots = (y_n - 1)/3$, jump to ***Step 2***， else to ***Step 1***.

Step2: Determine the intersection. If *k* is also in boxes which are not in the intersection, jump to ***Step 3***, else to ***Step 4***.

Step3: If $Possible(x, y, k) = k$， make $Possible(x, y, k) = -1$.

Step 4: Stop searching and exit.

4.4.2 Item C2

If there is a communal possible number in four boxes that are intersections of two rows and two columns, this possible number can be excluded in all the undetermined boxes left of the two rows and columns.

As shown in Figure 7, boxes $P(3,1)$, $P(3,8)$, $P(6,1)$ and $P(6,8)$ are intersections of row 3,

row 6 , column 1 and column 8. Number 3 is the communal possible number. Now boxes $P(4,1)$ $P(4,8)$ $P(9,1)$ $P(9,8)$ can exclude 3 as the possible number.

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

Figure 7

Algorithm illustration：

Step 1：Search $Possible(x, y, k)$. If possible number k appears twice respectively in column y and y' i.e. $Possible(x, y, k) = k, Possible(x, y', k) = k$, jump to ***step 2,*** else to ***step 4***.

Step 2：Search the possible numbers in column y and y'. If

$$Possible(x', y, k) = k,$$

$$Possible(x', y', k) = k,$$

jump to ***step 3***, else to ***step 4***.

Step 3： If

$$Record(m, n) = 0 \ (m = x, x'; n = y, y')$$

$$Possible(m, n, k) = k,$$

make $Possible(m, n, k) = -1$.

Step 4: Stop searching and exit.

4.4.3 Derivative approach of item C2

This algorithm can be extended to three-order as shown in Figure 8. Now number 6 in boxes $P(2,1), P(2,6)$ and $P(2,9)$ can be excluded.

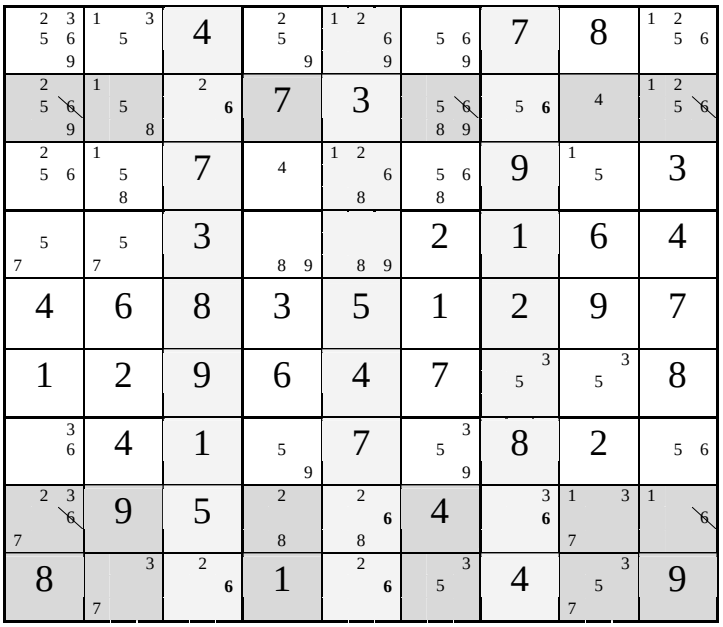

Figure 8

Similarly, we can extend it to four-order. For example Figure 9, 4 is no longer the possible number in boxes $P(2,5)$ and $P(2,8)$ by this approach.

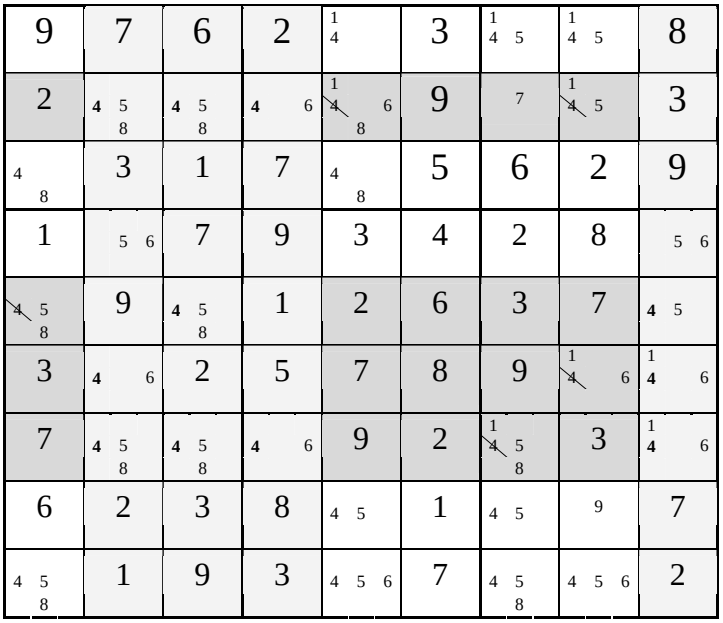

Figure 9

These algorithms above are the excellent artificial intelligence approaches[2][8] to solve Sudoku puzzles. There are still many practical and efficient artificial intelligence methods can improve the speed of resolving Sudoku but we don't display them due to it's hard to described them with computer languages and algorithms. And the methods mentioned can resolve the vast majority of the Sudoku puzzles already.

4.5 Backtracking Algorithm [3] to solve Sudoku puzzle

It's notable that in this section we use *i* (*i* from 1 to 81) to number the eighty-one boxes in turn from the top to the bottom and from the left to right in order to state the Backtracking algorithm conveniently.

4.5.1 Definitions of data structure

In this algorithm, four integer variables are used.

- $Receive_i \left(i = 1 \cdots 81\right)$

We use this variable to keep the fixed boxes of an initial puzzle and the final results, and blank boxes are initialized to zero particularly.

- $Record_i \left(i = 1 \cdots 81\right)$

$Record_i$ is one to one correspondence with $Receive_i$: If $Receive_i \neq 0$, correspondingly $Record_i = 1$（this suggests we can fill the box with an assured number）, else $Record_i = 0$, and in this case we cannot confirm which number is suitable to the box.

- $Possible_{ij} \left(i = 1 \cdots 81, j = 1 \cdots 9\right)$

We use this variable to keep record of possibility the entire undetermined box. The recording method is: Initialize variable as

$$Possible_{ij} = j \left(i = 1 \cdots 81, j = 1 \cdots 9\right)$$

During the process of computation, $Possible_{ij}$ changes to -1 as a mark when we find number j $\left(j = 1 \cdots 9\right)$ is not suitable to fill in this box.

- $Stack_i \left(i = 1 \cdots 81\right)$

Variable $Stack_i \left(i = 1 \cdots 81\right)$ is used to simulate a stack with size of 81, which plays an important role in the main backtracking process. Before backtracking, assign to $Stack_i$ with location of all the boxes with $Record_i = 0$ i.e. $Stack_i = i$.

In the backtracking process, we confirm the suitable number of these boxes. If there is no suitable number from 1 to 9, we should backtrack to the former location and revise its hypothetic value. Analogically, backtracking process is continued until all the boxes are filled in the right number or we backtrack to the one before the initial box.

4.5.2 Functions

Now we introduce three important functions that will be used in section 5.3 and 5.4, and other easier functions are not displayed.

- **[ ] = *modifyPb*( )**

This function is used to modify the variable $Possible_{ij} \left(i = 1 \cdots 81, j = 1 \cdots 9\right)$. When we query

all the possible number that can be placed in the box of a *undetermined* box ($Record_i = 0$), the existed number now in it's row, column and block must be excluded. We keep the query information to realize the modification of variable $Possible_{ij}$.

- **[r1] = *recordAll*( )**

This function is used to modify variable $Record_i$ and $Possible_{ij}$. The return value of function means whether the modification is implemented each time when this function is called.Its specific function is as follows:

We assign $Record_i = 1$ and $Receive_i$ equals to the sole suitable number when the degree of freedom λ_i of *undetermined* box *i*. The return value is:

$$r1 = \begin{cases} TRUE & sole \text{ possible number exists} \\ FALSE & \text{none box with one degree of freedom} \end{cases}$$

- **[r2] = *Exist*(i , j)**

This function is used to judge whether $Receive_i$ equals to *j* or not, that is to say, it can make sure of there is number *j* already or not in the row, column and block that this box lies in. The return value r2 is:

$$r2 = \begin{cases} TRUE & j \text{ exists already} \\ FALSE & \text{else} \end{cases}$$

4.5.3 "Finite recusive" Pretreatment

After calling function ***recordAll*( )**, some boxes are determined resulting in the changing of the value of $Possible_{ij}$. The more determined boxes, the less possible numbers of some boxes. Then function ***modifyPb*( )** is called to modifying $Possible_{ij}$. However, some other boxes with the degree of freedom being one arise after modification. Now it's time to call function ***recordAll*( )** again…… Obviously, it is more helpful when more boxes are determined. The loop ends only in the condition that all of the boxes are determined or the return value of function ***recordAll*( )** is FALSE.

4.5.4 Backtracking Algorithm

Step 1 Scan $Record_i$ according to the suffix in an increase order (*i* from 1 to 81), and then keep record of box location in an increase order by suffix to $Stack_i$ if $Record_i = 0$. Finally, integer variable *max* is used to count $Record_i = 0$ and then 1 is added.

Step 2 Integer variable *Top* is initialized to zero; $Stack_1 \cdots Stack_{81}$ are viewed as stack; *Top* is used to trace the stack top.

Step 3 Boolean variable *flag* is initialized to TRUE. The following steps are running in a unlimited loop. If *Top* reaches the current position by backtracking, *flag* is assigned to FALSE (Step 4 continues), else *flag* is TRUE (Step 5 continues). The end condition is

$top = max$ i.e. final solution is available or none solution at all. At the end, $top < 0$.

Step 4 It means *Top* reaches the current position by normal hypothesis while not backtracking due to *flag* being TURE. Therefore, the possible numbers of the box located by stack top can be available according to variable $Possible_{ij}$ and function *Exist*. Make $Receive_i$ equals to the possible number that first fall across, and let $Record_i = 1$. Meanwhile, determine whether the final solution is obtained i.e. whether $top = max$ or not. If $top = max$, print the solution and exit. If 1~9 are not suitable to this box, it means the previous hypothesis is wrong and needs backtracking. So $Record_i$ and $Receive_i$ are reset to zero, variable *Top* deducts 1 and *flag* is assigned to FALSE. This loop body is over and next loop begins.

Step 5 It means *Top* reaches the current position by backtracking due to *flag* being FALSE. If stack pops up the next box position when there is still no possible numbers can be placed in the box located by stack top, else make $Receive_i$ equals to the possible number that first fall across, let $Record_i = 1$, *Top* adds 1 and *flag* is assigned to TRUE. This loop body is over and next loop begins.

5 Creating a Sudoku puzzle

We use "Number Coverage" method to create a Sudoku puzzle so that it is need to know abundant Sudoku solutions first. The uniqueness and difficulty level of the Sudoku puzzles are tested and recorded in each time of coverage.

5.1 Derivative Rubik's cube

Definitions:

- Cognate row[2]: Row 1, 2, 3, row 4, 5, 6 and row 7, 8, 9 are cognate rows.
- Cognate column[2]: Column 1, 2, 3, column 4, 5, 6 and column 7, 8, 9 are cognate columns.
- Sub-row and sub-column: There are 3 sub-rows and 3 sub-columns in a block. Three elements are included in each sub-row or sub-column.

On the basis of analysis at the beginning of this section, how to create a new Sudoku puzzle, deriving from an intact numerical Rubik's cube or placing numbers stochastically? The former one is our choice because the successful probability is too tiny. The starting point is how to get suitable Rubik's cube to create Sudoku puzzle with our algorithm.

5.1.1 Choosing feasible Rubik's cube basement

A well-regulated Rubik's cube (Figure 10) is not suitable —— it is not an ideal basement for generating puzzles , because its regulation is likely to be used by players. It is believed a

failure if we use this kind of basement to create a Sudoku puzzle. The better ones are Rubik's cube with large discrete level relatively (Figure 11). In our opinion, 100 Rubik's cube basements are enough for a puzzle generating library to get derivative Rubik's cubes.

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

Figure 10

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

Figure 11

We develop function **FunInit1(*data)** to choose a Rubik's cube basement and its return value is a 9×9 Rubik's cube recorded in matrix **data**.

In sections from 5.1.2 to 5.1.6, eight kinds of derivative approaches of Rubik's cube are described.

5.1.2 Swapping elements in cognation rows or cognation columns

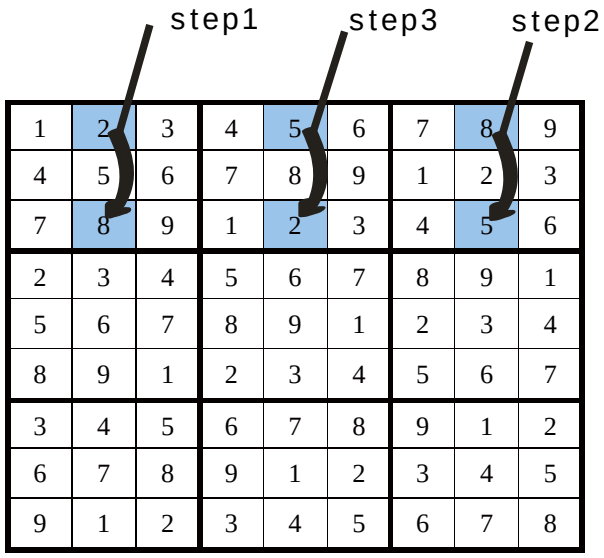

Figure 12 Swapping steps

Firstly, we find a Rubik's cube arbitrarily as a basement. Secondly, adjust some numbers in the basement to obtain different Rubik's cubes. Confined to Sudoku rules, all the adjustments can only take place between two numbers in a cognation row (cognation column). The principle of adjustment is:

- **One-time interchanging**: Take Figure 12 for example to illustrate it. Take two numbers respectively in the same column and block (or the same row and block) and interchange

them. This is done by **step 1**. The result of interchanging is a number appears twice in the same row, which disobeys the rule of Sudoku; it thus needs to continue to interchange in the collision row done by **step 2** and **step 3**.

- Implement one-time interchanging until the whole solution satisfies Sudoku rule.

Finally, millions of Rubik's cubes can be derived by this changing method. We develop a function **FunInit2(*data)** to transmogrify cognate rows by this method. Function **FunInit3(*data)** is to transmogrify cognate columns in the same way. Matrix **data** saves the input and output.

5.1.3 Numbers mapping

Briefly speaking, this is one of the branches of number replacement. Now we let （9,8,7,6,5,4,3,2,1）substitute for （1,2,3,4,5,6,7,8,9）and thus two Rubik's cubes(Figure 13 and Figure 14) are obtained. Subsequently, two different Sudoku puzzles are generated by covering the same places in the two Rubik's cubes. Vice versa, we can get puzzle solutions by this kind of number replacement.

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

Figure 13 The original Rubik's cube

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

Figure 14 Rubik's cube after mapping

Rubik's cube in Figure 14 is derived from Figure 13. Problem arises when two numbers don't appear in a Sudoku puzzle, which destroys the uniqueness of solution. Because a puzzle created on the base of this Rubik's cube has two solutions only if we exchange the two numbers.

The original Rubik's cube transforming to a new one is called the Rubik's cube derivative and number mapping method is one of them.

We develop function **FunInit4(*data)** that can generate mapping sequences stochastically and transform matrix **data.** Matrix **data** saves the input and output.

5.1.4 Rotation

Rotate the primary Rubik's cube with 90^o , 180^o and 270^o anticlockwise rotation respectively to derive new independent Rubik's cube.

This does not change the uniqueness of solution as well as the difficulty and solutions. Solutions can be obtained by the same rotation method. Rubik's cube in Figure 16 is the result with a 90^o counterclockwise rotation in Figure 15.

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

Figure 15 The original Rubik's cube

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

Figure 16 Rubik's cube after rotation

We develop function **FunInit5(*data)** to rotate matrix **data**. Rotation angles are generated stochastically in this function. Matrix **data** saves the input and output.

5.1.5 Exchange of cognate rows and cognate columns

As the definitions in 5.1, the three columns can be exchanged arbitrarily in a cognate row or column. This does not change the uniqueness of solution as well as the difficulty and solutions. Solutions can be obtained by the same exchange method.

- Exchanging of cognate row

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

Figure 17 The original Rubik's cube

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

Figure 18 Rubik's cube after exchange

- Exchanging of cognate column

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

Figure 19 The original Rubik's cube

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

Figure 20 Rubik's cube after exchange

We develop function **FunInit6(*data)** to complete the exchange of cognate row and

FunInit7(*data) to complete the exchange of cognate column. Matrix **data** saves the input and output.

5.1.6 Exchange of blocks

Definitions:

- Lateral block groups: 3 blocks in the same row.
- Vertical block groups:3 blocks in the same column.

Any two lateral block groups can be exchanged to generate a new Rubik's cube. This does not change the uniqueness of solution as well as the difficulty and solutions; solutions can be obtained by the same exchange method. For example, the first lateral block groups and the second lateral block groups are exchanged in Figure 21. We don't give examples of the exchange of vertical block groups because both the principles are similar.

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

Figure 21 The original Rubik's cube

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

Figure 22 Rubik's cube after block exchange

Function **FunInit8(*data)** is applied into complete the exchange of lateral block groups and **FunInit9(*data)** to complete the exchange of vertical block groups. Matrix **data** saves the input and output.

5.2 Guarantee of Uniqueness

In this paper, we only discuss the proper Sudoku. As to a proper Sudoku, its solution must be unique.

Fixed boxes cannot be too few in order to create a proper puzzle. The more fixed boxes the easier of this puzzle in some cases, however, it's not absolute. It's more suitable when the quantity of fixed boxes in a Sudoku puzzle between 20 to 45 or more if people play it by hand. Besides, the covered boxes are centre symmetry to meet aesthetic requirements. After considering these factors, we develop three principles to guarantee uniqueness of solutions. All of these principles must be satisfied when we create a Sudoku puzzle.

- Principle(1)

It is not permitted that more than one number from 1 to 9 don't appear in a Sudoku puzzle. Reason about this can be referred in 5.1.3 Number mapping.

- Principle(2)

Because there must be multiple solutions, it is not permitted that two or three rows in a cognate row are blank, so does a cognate column.

- Principle(3)

Three elements analogical row couple and three elements analogical column couple of a Rubik's cube have six numbers in all. At least one of the six numbers cannot be covered at the same time in the process of generating a Sudoku puzzle. We deal with two elements analogical row couple and two elements analogical column couple in the same way.

We illustrate these principles with some cases. In Figure 23, there is a three elements analogical column couple and both two elements analogical couple of the Rubik's cube. We create a Sudoku puzzle (Figure 24) from Figure 23. This means Rubik's cube in Figure 23 is the solution of puzzle in Figure 24. If players don't conform to the principles we mentioned above in the process of resolving, it is obviously that the second solution (Figure 25) is obtained only if we exchange both two elements analogical couple in the first cognate. The third solution (Figure 26) is obtained if we exchange the three elements analogical column couple. These solutions are feasible. Furthermore, there are more potential solutions.

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

Figure 23 Three elements analogical column couple , two elements analogical couple

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

Figure 24 Analogical elements coverage

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

Figure 25 Solution 2

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

Figure 26 Solution 3

Principle (1)-(3) are true by researching the cases above. We also develop function **FunInit10(*data)**, **FunInit11(*data)** and **FunInit12(*data)** corresponding to the three principles. Matrix **data** is the input of functions and a Boolean value is returned to indicate whether it satisfies these principles or not.

5.3 Developing metrics of difficulty levels

5.3.1 Weakness of general metrics

Well posed puzzles should have a unique solution and the task is to find it without guessing. A kind of illusion often occurs to a beginner is that the more fixed boxes in the initial puzzle, the harder of the puzzle. In fact, there is no definite relation between them. So it is unadvisable to divide difficulty levels just by the amount of fixed boxes in the initial puzzle.

But there is no uniform metrics to divide difficulty levels to date. The existing metrics of difficulty levels are 4 or 5 based on methods (such as section 4 introduced) applied into successful solving process. In general, the more complex methods are adopted, the level is higher i.e. the puzzle is harder.

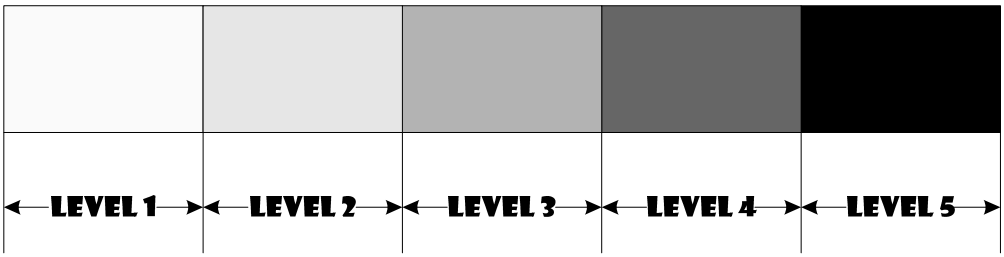

Figure 27 Sketch map of five difficulty levels

However, 5 levels have distinct boundary and independent in general metrics, that is to say, the player only contacts these fixed methods such as he cannot contacts methods of level 2 when he plays puzzles of level 1 .His ability is promoted slowly and it takes a lot of time before he moves on to the higher level. This metrics is inefficient and some of boring for players.

5.3.2 Fuzzy metrics

When we develop our own metrics of difficulty levels, some humanization factor is considered besides the general metrics. It is more reasonable and attractable that some methods in the higher level applied into the lower level occasionally to promote player to have command of more complex method.

So making the level boundary be vague is necessary to extend the interest and universality of Sudoku puzzle. Fussy process method can be set artificially. We make the quantity of methods of Level (i+1) appearing in Level (i) be an acceptable range such as twice. But these method crossovers are only permitted between adjacent levels.

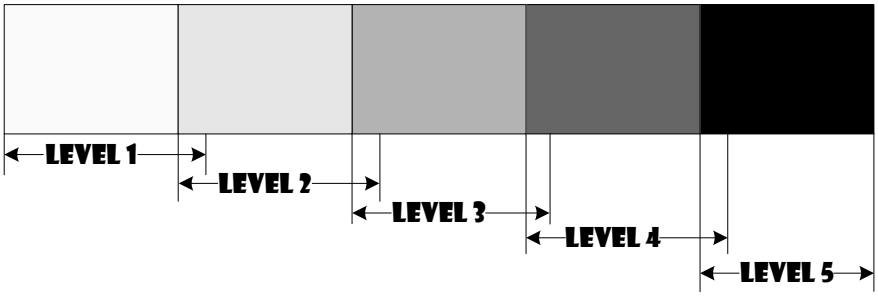

Figure 28 Sketch map of five difficulty levels after adjustment

Under this metrics, a Sudoku puzzle is more convenient generated by computer programs and thus the ratio of feasible puzzle increases.

Difficulty levels:

- **Level 1** (Rookie): Puzzles resolved by method of type A.
- **Level 2** (Beginner): Puzzles resolved by method of type A and B.
- **Level 3** (Professional): Puzzles resolved by method of type C1or C2.
- **Level 4** (Expert): Puzzles resolved by method of type C1 and C2.
- **Level 5** (Evil): Puzzles resolved only by method of backtracking algorithm. Uniqueness of solutions cannot be guaranteed.

5.3.3 Difficulty coefficient of Sudoku puzzle in the same level

In this paper, Numerical quantitative approach is used to reflect varying difficulty of Sudoku puzzle in the same level. We set a difficulty coefficient to measure every Sudoku puzzle.

We can see from 4.5 that the main work is searching and backtracking[1] if there are more than one solutions of a Sudoku puzzle. We denote searching times as a and backtracking times as b. Difficulty of puzzle increases when b becomes larger. But difficulty of puzzle also varies when a changes. As to the same backtracking times, the more times of searching, the more difficult of a puzzle. We take backtracking as invalid searching, and thus $b < a$. There is no feasible Sudoku solution when $b = a$.

From analysis above, we define β as the difficulty coefficient.

$$\beta = \frac{a}{a-b}$$

After puzzles are created, the difficulty coefficient can be presented to players. This can help players play well and they can choose the puzzles they want with the clear difficulty coefficient.

5.4 Algorithm of creating Sudoku puzzles

There are ten steps in the Algorithm of creating Sudoku puzzles as follows:

Step0： Input the difficulty level L and the quantity of fixed boxes N and the quantity of difficulty crossover procedures M required by players; record the current time T of computer.

Step1： Run **FunInit1(*data)**to get a basic feasible discrete Rubik's cube.Copy **data** to **datat**;

Step2： Derive Rubik's cube **data** and run **FunInit2(*data)～FunInit9(*data)** stochastically.

The running times of each approach are controlled by time and the derivative time is 0.1s in general.

Step3：Cover about 15 boxes stochastically. Run **FunInit10(*data)～FunInit13(*data)**. If the return value is small, it suggests the degree of disperse and jump back to Step1.

Step4：Copy **data** to **datat**. Initialize $M'=0$ and $m'=0$;

Step5：Compare the current time of computer and **T**. If the time cost is beyond 2s, the program is over and requires players to input again.

Step6：Run the random generator, choose an uncovered box and change the state of this box in **datat** being *covered*.

Step7：Run **FunInit10(*data)～FunInit13(*data)**. If the return value is bigger than zero, jump to Step4 .

Step8.1：According to the input, the level is L. It need to choose heuristic intelligence approaches in level L+1.

Step8.2：If it cannot resolve successfully, jump back to step5 because it is too hard to generate a puzzle by cover this box.

Step8.3：If approaches in L+1 are used, $M'=M'+1$.
If $M'>2$, $M'=M'-1$ and jump back to Step5.

Step8.4：If approaches with difficulty level less than L, $m'=m'+1$; if $\frac{m'}{81-N}>\frac{2}{3}$, $m'=m'-1$, and jump back to Step5.

Step9：Copy **datat** to **data**. If more than ***N*** of the determined boxes in **data** appear, jump back to Step5.

Step10: The program is over. Output the Sudoku puzzle to user terminal.

Notes:

- M' is the times of using approaches in level L+1.
- m' is the times of using approaches with difficulty level less than L.
- About random generator: In general, we use the random generator to produce a puzzle at each time, but we can store enough random positions of boxes from1 to 81 to improve efficiency.
- About symmetric initial puzzle: A symmetric Sudoku puzzle is easily obtained in our program only if we generate the random position Z from 1 to 41 and its symmetric position is $Z'=81-Z+1$. It must be the symmetric Sudoku puzzle by covering the two boxes .

6 Optimization of the complexity of creating algorithm

6.1 Analysis

It is clear that the main process of creating algorithm is similar to an inverse process of solving. We believe that the more derivative times and precise of each step, the better the puzzles can meet the requirements of plays. However, it takes more time to generate a Sudoku puzzle, which is a contradiction to players' requirements. An excellent algorithm is not the fastest but considering multiple factors to make the computing load distributed reasonably to each computing procedure. The complexity of creating algorithm contains time complexity and space complexity. In this paper, we only have a quantitative measurement of time complexity.

- **Derivative time cost**: The time complexity of kinds of derivative methods analyzed in 5.1 has no big differences. So the computing load depends on the derivative times denoted as Y. We let T be the time cost of computer in one derivative process. The average value of T can be monitored by programs. We believe $Y \times T$ is the **Derivative time cost**.

- **Difficulty limitation cost**: Among the seven intelligence algorithms introduced in section 4 , the time complexity of determining difficulty is different. We define T_i' as the time cost of the i_{th} approach to determine the current difficulty. The average value of T_i' can be monitored by programs. Let Y_i' be the time cost in one called process of the i_{th} approach. Which one is the more important aspect players care, metrics of difficulty of or diversity of a puzzle? In general, the degree of recognition of these aspects varies as to different players. We denote α and β to measure the need of players. These parameters can be controlled i.e. we can generate the more suitable Sudoku puzzles.

6.2 Building the linear programming model

Based on the analysis above, we build a linear programming model. The objective is the maximum utility of computer algorithm and the constraint is the generating time U.

$$Max \quad \alpha \times Y \times T + \beta \times \sum_{i=1}^{7} Y_i' T_i'$$

$$S.T. \begin{cases} T \times Y + \sum_{i=1}^{7} Y_i' T_i' \le U \\ \sum_{i=1}^{7} \left(10^i \times sign(Y_i')\right) \le 10^L \\ Y > 81 - N \\ Y_i' > 81 - N \quad i = 1...7 \\ T \ge 0 \\ T_i' \ge 0 \qquad i = 1...7 \end{cases}$$

Notes:

L: The requiring difficulty levels of players when a puzzle begins.
N: The amount of fixed boxes of an initial puzzle.
U: The acceptable generating time of players when a puzzle begins.
Sign ( ): Do the operation to the input number such as Sign(5)=1,Sign(0)=0.
Conclusion: A Sudoku puzzle generator is better when it can satisfy the need of players well but not the speed of generating.

7 Strengths and weakness

- Strengths

Our algorithm can be extended into multi-order $n \times n$ Sudoku, which has a universal significance. The backtracking algorithm can solve all the Sudoku puzzles in theory though the time complexity varies due to difficulty of puzzles. We define metrics of difficulty according to the intelligent approaches used in solving.

- Weaknesses

The algorithm of the game mainly concludes 7 kinds of methods. From the perspective of perfecting the game, the number of intelligent algorithm is not enough because more intelligent algorithms can optimize the complexity of the algorithm. The discrete degree of numbers in initial puzzle will affect the efficiency of generating.

8 References

[1]. Sudoku Alignment website:http://www.51sudoku.com
[2]. YAN De Ren. Competitive Sudoku. *China Yanshi press. 2007-Dec.*
[3]. LEI Lei, SHEN Fu-ke. The Design and Implementation of the Algorithm about Sudoku. *Journal of Research and Development and Design Technology.*
[4]. Richard Johnsonbaugh, Marcus Schaefer. Algorithms. *Copyright 2004 PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.*
[5]. Duane Hanselman, Bruce Littlefield. Mastering Matlab 7. *Copyright 2004 PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.*
[6]. Robert L.Kruse . Data Structures & Program Design in C(Second Edition). *Copyright 2004 PEARSON EDUCATION ASIA LIMITED and TSINGHUA UNIVERSITY PRESS.*
[7]. Meng Qing ling. Sudoku Puzzle of Artificial Solution with Computer. GanSu Science and Technology, 2006Sep, Vol.22, NO.9
[8]. Mathe's blog: http://blog.csdn.net/mathe/archive/2007/08/23/1755672.aspx
[9]. Gotoread.com: http://www.gotoread.com/article/bbs.aspx?id=563463
[10]. Fuji's Website: http://www.pro.or.jp/~fuji/numplace/index.html
[11]. http://www.shes.hcc.edu.tw/~oddest/su303.htm#op1

9 Appendix

9.1 Sudoku puzzles created by our algorithm

Rookie

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

Beginner

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

Professional

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

Export

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

Evil

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

9.2 Some of the procedures

BackTracking.m

```
sd=reshape(xlsread('data.xls','A1:I9')',1,81);
fix=sd~=0;
poss=zeros(81,9);
for i=1:81
  poss(i,1:9)=1:9;
end
stack=zeros(1,81);
t=1;
for i=1:81
    if fix(i)==0
      stack(t)=i;
      t=t+1;
    end
end
Max=t-1;
% small sq list
Glist= xlsread('data.xls','s1','Ran');
%% preDoing
disp('preDoing time')
tic
while 1
    poss=setPb(sd,poss,fix,Glist);
    [F,sd,poss,fix]=fixAll(sd,poss,fix);
```

```
    if F~=1
        break;
    end
    if sum(sd==0)==0
        break;
    end
end
toc
if sum(sd==0)==0
  return;
end
%% BackTracking
t=1;
for i=1:81
    if fix(i)==0
      stack(t)=i;
      t=t+1;
    end
end
stack=stack(1:t-1);
Max=t;
top=1;
F=1;%Normal in
tic
while 1
    if top<1
        disp('error')
        break;
    end
    if F==1
        j=1;
      while j<=9
        if poss(stack(top),j)~=-1 && beExist(sd,Glist,stack(top),j)==0
            fix(stack(top))=1;
            sd(stack(top))=j;
            top=top+1;
            if top>=Max
                toc
                disp(sd)
                return;
            end
            break;
        end
        j=j+1;
      end
      if j>9
        top=top-1;
        F=0;
      end
    else
      if sd(stack(top))==9
          fix(stack(top))=0;
          sd(stack(top))=0;
          top=top-1;
      else
          temp=sd(stack(top))+1;
          while poss(stack(top),temp)==-1 ||
beExist(sd,Glist,stack(top),temp)==1
              temp=temp+1;
              if temp > 9
                  break;
              end
          end
          if temp>9
            fix(stack(top))=0;
            sd(stack(top))=0;
            top=top-1;
```

```
          else
            sd(stack(top))=temp;
            top=top+1;
            F=1;
          end
      end
    end
end
toc
disp('Result....')
reshape(sd,9,9)'
```

ModifyPb.m

```
function poss=setPb(sd,poss,fix,Glist)
for i=1:81
   if fix(i)==0
       %Row
        R=ceil(i/9);
        for j=R*9-8:R*9
            if sd(j)~=0
                poss(i,sd(j))=-1;
            end
        end
       %Col
        if mod(i,9)==0
           C=9;
        else
           C=mod(i,9);
        end
        for j=C:9:81
            if sd(j)~=0
                poss(i,sd(j))=-1;
            end
        end
        %Block
        for j=1:4
            if sd(Glist(i,j))~=0
                poss(i,sd(Glist(i,j)))=-1;
            end
        end
    else
        poss(i,:)=-1;
    end
end
```

Generator.m

```
Bas=xlsread('data.xls','A23:I31');
Bas3=zeros(9,9,30);
Bas3d=Bas3;
Ob_Num=40;
for k=1:20
% row 1 change
t=zeros(3,9);
for i=1:3:9
   t=Bas(i:i+2,:);
   n1=floor(2*rand(1,30))+1;
   n2=floor(8*rand(1,30))+1;
   n3=ones(1,length(n1));
   for j1=1:length(n1)
       if n1(j1)==1
           n3(j1)=2;
       elseif n1(j1)==2
           n3(j1)=3;
       else
```

```
            n3(j1)=1;
        end
    end
    for j1=1:length(n1) %all
      for j2=1:length(n2)
          x1=n1(j1);
          y1=n2(j2);
          x2=n3(j1);
          while 1
            t1=t(x2,y1);
            t(x2,y1)=t(x1,y1);
            t(x1,y1)=t1;
            t2=find(t(x1,:)==t1);
            if length(t2)==1
                break;
            else
                if y1==t2(1)
                    y1=t2(2);
                else
                    y1=t2(1);
                end
            end
          end
      end
    end
    Bas(i:i+2,:)=t;
end
Bas3d(:,:,k)=Bas;
n2=floor(40*rand(1,100))+1;
Cposs=zeros(Ob_Num,81,'uint8');
i=1;
Bast=reshape(Bas',1,81);
 while i<=length(n2)
  if Idt1(n2(i))==1
      i=i+1;
      continue;
  else
    Idt1(n2(i))=1;
    Idt1(81-n2(i)+1)=1;
    Cposs(ceil(sum(sum(Idt1))/2),:)=Cpc(Bast.*Idt1);
  end
  i=i+1;
  if sum(sum(Idt1))>=Ob_Num
      Bas3(:,:,k)=reshape(Bast.*Idt1,9,9)';
      break;
  end
 end
end
```