Sudoku Puzzles Generating: from Easy to Evil

Abstract

As Sudoku puzzle becomes worldwide popular among many players in different intellectual levels, the task is to devise an algorithm that creates Sudoku puzzles in varying level of difficulty. With the analysis of the game rules, we first define the difficulty level from four aspects as: total given cells, distribution of given cells, applicable techniques of logic deduction and complexity of enumerating search.

By the guidance from the definition of difficulty level, the algorithm for generating puzzles is developed with the “dig-hole” strategy on a valid grid. Thus, the algorithm developed in two steps: to create a valid grid by Las Vegas algorithm, and then to generating puzzles by erasing some digits using five operators:

- Determine a sequence of digging holes according to the desirable difficulty level,
- Set two restrictions to guide the distribution of given cells,
- Judge whether a puzzle being dug out has a unique solution by a solver built using Depth-First Search,
- Add pruning technique to avoid digging an invalid cell, and
- Perform propagating at a dug-out puzzle to raise the diversity of the output puzzle.

Using our developed algorithm, we generate Sudoku puzzles in any five difficulty levels. The difficulty level of output puzzles can be adjusted by a desirable difficulty value input by players. The complexity of the algorithms in space and time is analyzed to demonstrate the effectiveness of the algorithms.

Our main contributions in exploring the “dig-hole” strategy are summarized as following three works: to do a massive research on the sequence of digging holes and how it affects the algorithm to create a evil-level puzzle with minimal given cells, to invent a skill for judging the solution’s uniqueness of a puzzle being dug out by the reduction to absurdity, and to reduce the computational time by avoiding backtracking to an explored cell and refilling an empty cell.

Keywords: Dig-hole Strategy, Las Vegas Algorithm, Pruning, Reduction to Absurdity

Contents

1 Background

1.1 Sudoku is Hot

Sudoku puzzle, as a widely popular intellectual game in recent years, was invented in Swiss in 18th century. Then, it initially harvested well development in Japan in the past decades. The name Sudoku actually derives from Japanese that means "number place" [1].

Due to its simple and friendly rules for beginners and the charm from intellectual challenge, Sudoku becomes welcome recently for players of various ages. You are even able to solve a Sudoku puzzle easily without any mathematical knowledge.

1.2 How to Play

How is the Sudoku game played? You only need to know where you play the game and what your goal is. The both simple aspects that help you join the game are specified as follows:

- **Game Environment:** you may first get a general overview of this game board in **Figure 1.1**.

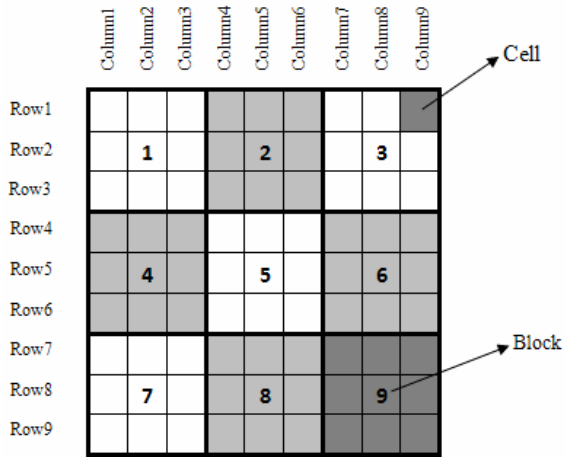


Figure 1.1: General view of Sudoku game environment

We define several basic components of the board as **Figure 1.1** illustrates. The whole board is actually a 9-by-9 **grid** made of nine smaller 3-by-3 grids called **blocks**. The smallest unit square is called a **cell** which has two types of states: empty, and confirmed by a digit from 1 through 9. We mark the whole gird with rows and columns from top-left corner.

- **Goal of the Game:** generally, Sudoku game is started with such a situation in grid that some of the cells have already been confirmed by digits known as **givens**. The task for Sudoku players is to place a digit from 1 to 9 into each cell of the grid, and meanwhile each digits can only be used exactly once in each row, each column and each block. Additionally, all the nine rows, nine columns and nine blocks are respectively ensured to contain all the digits from 1 through 9. These limitations for placing digits in three locations are respectively called **row constraint**, **column constraint** and **block constraint**.

Based on the rules that we mentioned above, Sudoku players are commonly inspired to complete the placement of digits into all empty cells using various techniques as soon as possible.

2 Problem Analysis

2.1 Our Tasks

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

Thus, we develop an algorithm to construct specific Sudoku puzzles with the consideration of the following three requirements:

1. **Varying difficulty**: essentially, the algorithm should be able to create puzzles in different levels of difficulty. It regulates that the algorithm for creating must be designed by means of two jobs as follows:
   - **Difficulty level:** define what the difficulty level is, and evaluate a fixed Sudoku puzzle by a grading algorithm.
   - **Extensibility:** a varying number as a metric of difficulty request from a play can be input by the player. Based on this number, the algorithm for creating must be applicable to generate diverse puzzles satisfying the difficulty request of the player.
2. **Unique solution**: all generated Sudoku puzzles must be guaranteed to yield a unique solution by a solving algorithm.
3. **Minimizing complexity**: the programs of all these algorithms must finish their jobs in a short time accepted by users or players.

In sum, we need to solve the entire problem of creating Sudoku puzzles by developing the three algorithms respectively for **solving**, **grading** and **generating**.

2.2 Related Works

By reviewing the history of Sudoku development, Sudoku puzzles grow up from an intellectual game for humans to a challenging problem for algorithm development with the participation of computer science. A famous demonstration in [2] claims that the total number of valid 9-by-9 Sudoku grids is 6,670,903,752,021,072,936,960. Another powerful conclusion shows that the minimal amount of givens in an initial Sudoku puzzle that can yield a unique solution is **17** given cells [3]. Recently, researchers engaged in algorithm development issues some breakthroughs in Sudoku solving, grading and generating by Genetic Algorithm, evolutionary method with geometric crossover, belief propagation etc.[4], [5], [6].

Amount these algorithms, the simplest and easiest one for implementation of generating puzzles is digging-hole method. The idea of this method begins with a valid grid that seems the same as a terminal pattern of puzzle. Then, a fixed amount of confirmed cells are dug into empty cells by some mechanism or sequence. While a confirmed cell is dug out, the remaining pattern of grid as an incomplete puzzle must be judged whether the puzzle can yield a unique solution. The difficulty level is mainly determined by the amount of empty cells.

2.3 Basic Ideas

Our generating algorithm stands on the shoulders of “giant” who initially proposed the idea of digging holes. In this paper, our main efforts are paid to research how to dig holes. For instance, given a terminal pattern, which cell do you plan to dig at first? What about next one? Can the puzzle being dug yield a unique solution after a cell is dug out? Therefore, constructing a mechanism for the operation of digging is worthy of further developing. Our creative works that answer the above questions well is distinguished from the existing methods in the following critical aspects:

★ Explore how to assign the sequence of digging holes according to different levels of difficulty.

★ Find an easy method, reduction to absurdity, to judge the solution’s uniqueness of the puzzle after a cell is dug out.

★ Reduce the computational time spent in digging holes skillfully by pruning and the avoidance of backtracking.

3 Assumptions and Definitions

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

4 Metrics of Difficulty Level

In this section, we develop a metrics to determine the difficulty level of a Sudoku puzzle from both computing perspective and human logic deducing perspective. Four factors affecting the difficulty level are taken into consideration in this metrics respectively as follows:

- The total amount of given cells,
- The lower bound of given cells in each row and column,
- Applicable techniques by human logic thinking, and
- Enumerating search times by computer.

By weighting the above four factors with scores, we grade a Sudoku puzzle in five levels as follows:

Level 1 Extremely Easy
Level 2 Easy
Level 3 Medium
Level 4 Difficult
Level 5 Evil

1. The total amount of given cells

As the first factor affecting the level estimation, the total amount of given cells in an initial Sudoku puzzle can significantly eliminate potential choices of digits in each cell by the three constraints in the game rules. In general, it is reasonable to argue that the more empty cells provided at the start of a Sudoku game, the higher level the puzzle graded in. We moderately scale the amount ranges of givens for each difficult level in **Table 1**.

Table 1: The amount ranges of givens in each difficult level

Level	Givens Amount	Scores
1 (Extremely easy)	more than 50	1
2 (Easy)	36-49	2
3 (Medium)	32-35	3

4 (Difficult)	28-31	4
5 (Evil)	22-27	5

2. The lower bound of given cells in each row and column

The positioning of the empty cells significantly affect the difficulty level if two puzzles provide the givens in the same amount or in slight difference at the start of a Sudoku game. The puzzle with the givens in clusters is graded in higher level that that with the givens in scattered distribution. Based on the row and column constraints, we regulate the lower bound of given cells in each row and column for each difficulty level in **Table 2**.

Table 2: The lower bound of given cells in each row and column for each difficulty level

Level	Lower bound of givens amount in rows and columns	Scores
1 (Extremely easy)	5	1
2 (Easy)	4	2
3 (Medium)	3	3
4 (Difficult)	2	4
5 (Evil)	0	5

3. Applicable techniques by human logic thinking

The popular techniques solving Sudoku puzzle using human logic thinking are summarized in [1]. These techniques have been graded in basic, intermediate and advanced levels according to the efforts of logic thinking. We score all these techniques and grade them into the five levels as **Table 3** shows.

Table 3: Scoring all the techniques

Techniques	Scores
Row, Column and Block Elimination	1
Lone rangers in block	2
Lone rangers in column	2
Lone rangers in row	2
Twins in block	3
Twins in column	3
Twins in row	3
Triplets in block	**4**
Triplets in column	4
Triplets in row	4
Brute-force elimination	5
Backtracking in brute-force elimination	5

Then, we can partially grade a puzzle in such a way that if a puzzle can be solved by a technique, it will be rewarded with the scores of the technique. In this way, the

high scores the puzzle is rewarded, the higher level it will be probably graded into.

4. Enumerating search times by computer

Apart from logic-thinking techniques, it is verified by our practice that a Sudoku puzzle can be solved by enumerating search within tolerable time. The computer does a trial that filling an empty cell with a digit from 1 through 9, then check whether the filled digit can meet the three constraints in the game rules. Next trial will be done if the previous one is successful. We define searching once during the enumerating search as the computer does such a trial. In this way, the enumerating search can find out all the valid solutions after it tried all the potential combinations of digits by the trials. Since the total search times indicate the scope of potential combinations, they imply how many trials a player has to explore if he does not know any logic technique. We estimate the difficulty of a Sudoku puzzle partially using the total enumerating search times as **Table 4** indicates.

Table 4: Enumerating search times in each difficulty level

Level	Enumerating search times	Scores
1 (Extremely easy)	less than 100	1
2 (Easy)	100-999	2
3 (Medium)	1000-9999	3
4 (Difficult)	10000-99999	4
5 (Evil)	more than 100000	5

Based on the four factors that we analyze above, we sample how a Sudoku puzzle is graded into a difficulty level.

We deploy our developed algorithm following specified in Section 5 to create a Sudoku puzzle in the evil level, shown in **Figure 4.1**. This puzzle is scored based on the four factors, which display in **Table 5** as below.

5		8						
			2					1
			5				9	
					1			6
9		6				4		
	3	1	48	458	6			7
			7	2		8		
	8	2		9	45			3

Figure 4.1: An evil-level Sudoku puzzle created by our algorithm

Notice: the digits with smaller size are the selectable ones based on the three constraints of the game rules. Using the technique, Triplets in block, we claim that the digits of 4, 5 and 8 cannot be filled in the other empty two cells in the block.

Table 5: A puzzle scored based on the four factors

Factor	Interpretation	Weights	Scores
Total givens	total 22 givens, within the range in level 5	0.4	5
Lower bound in rows and columns	0 givens in Row 1, reach the lower bound in Level 5	0.2	5
Applicable technique	Triplets in block is applied, high technique in Level 5	0.2	5
Search times	search 263734 times, Level 4	0.2	4
Global evaluation	**Level 5: evil**	**total:**	**4.8**

5 Specification of Algorithms

Our entire algorithm for creating a Sudoku puzzle in fixed level of difficulty is performed by creating a terminal pattern, digging holes based on the difficulty level while checking whether the constructed puzzle can yield a unique solution.

5.1 Solving Algorithm: Depth-first Search

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

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

5.2 Generating Algorithm

With the both tools, the metrics of difficulty level and the solver of Sudoku puzzle, the generating algorithm constructs a varying-difficulty Sudoku puzzle in two steps as follows:

- Step 1: Create a terminal pattern;
- Step 2: Digging holes to generating a puzzle.

5.2.1 Creating Terminal Pattern: Las Vegas Algorithm

The operator of creating a terminal pattern (known as a pattern of solution) stands ahead in the whole performance of generating algorithm. We randomly create a terminal pattern by means of a randomized algorithm, **Las Vegas Algorithm** [7].

Obviously, a terminal pattern can be generated from an empty pattern by means of the solver. For enhancing the diversity of the generated puzzles, we first randomly locate n cells in an empty grid and then fill these empty cells with random 1-to-9 digits while satisfy the game rules. This step yields a puzzle with ***n*** givens. Whether this puzzle can be solved within tolerable time (for example 0.1s) is determined by the positions and values of these givens.

Suppose $P(x)$ is the probability that the puzzle with a set of givens is solvable. The Las Vegas algorithm performs the event ***x*** continuously until the event happens, where the event stands for completing the generation of a terminal pattern within the fixed time. In programming words, the function las_vegas($P(x)$) will return TRUE if the event happens with the probability $P(x)$, otherwise will return FALSE and performs the event again until it returns TRUE as the following codes:

```
while (!las_vegas(P(x)))
```

The probability of the event $P(x)$ is affected by the amount of random givens ***n***. Thus, we fix several ***n*** and then perform the event 10000 times for each ***n***. A statistic result about the relationship between the probability that the event happens and the amount of givens ***n*** is illustrated in **Figure 5.1**.

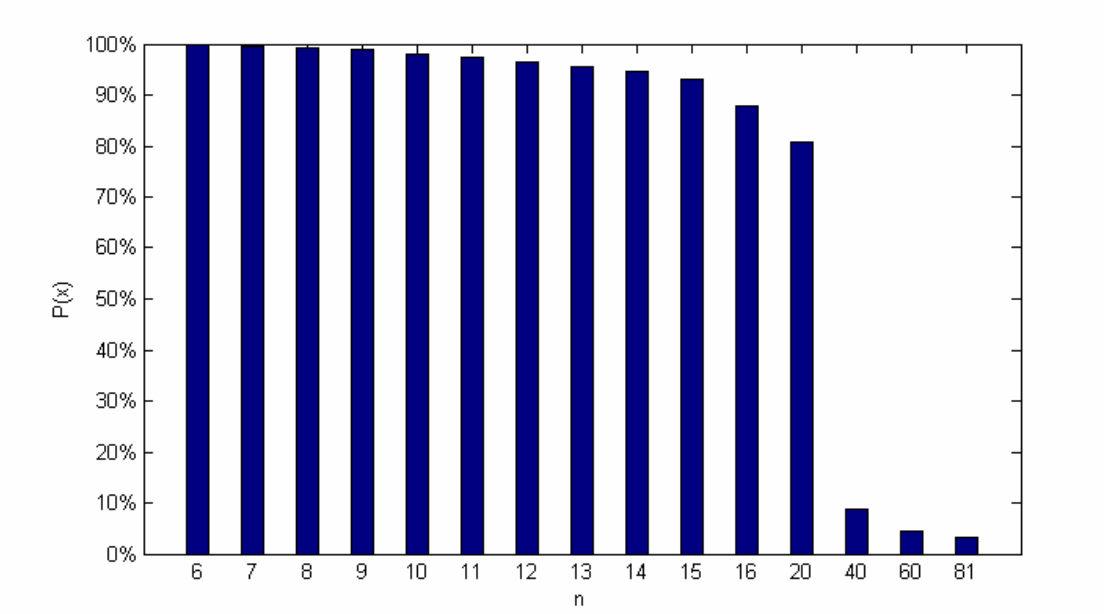


Figure 5.1 the relationship between the probability that the event happens and the amount of givens *n*.

Finally, we find that randomly creating a puzzle with **11** givens can help to minimize the computational time and meanwhile enhance the diversity of the generated puzzles.

5.2.2 Digging Holes

The strategy of digging holes method is, as its name implies, to generate a Sudoku puzzle by erasing several digits in confirmed cells of a terminal pattern. In addition, different mechanisms of digging holes lead to the diversity of constructed puzzles with various patterns or in varying difficulty level.

For speeding up the generation of a puzzle, we build the mechanism of digging holes with greedy strategy. Once a confirmed cell is dug out into an empty cell, the following operation is forbidden from filling another digit into this cell again.

The procedure of the digging holes method for generating a varying difficulty puzzle is outlined by the following flow chart (**Figure 5.2**):

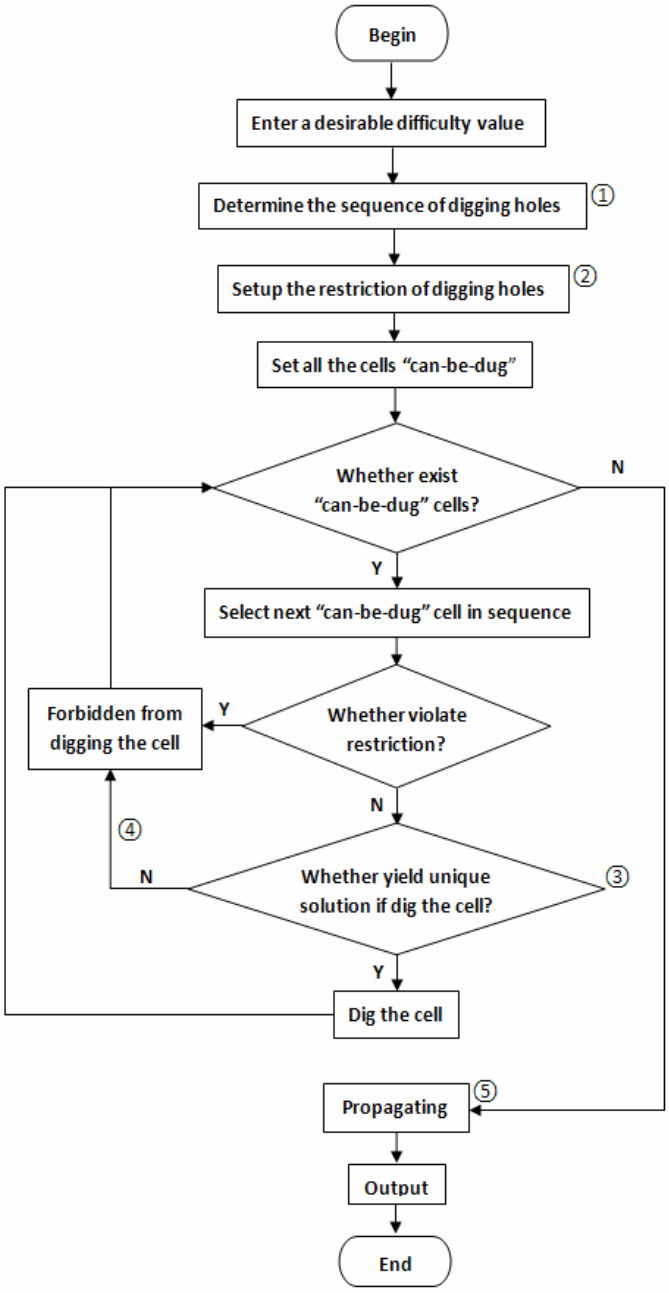
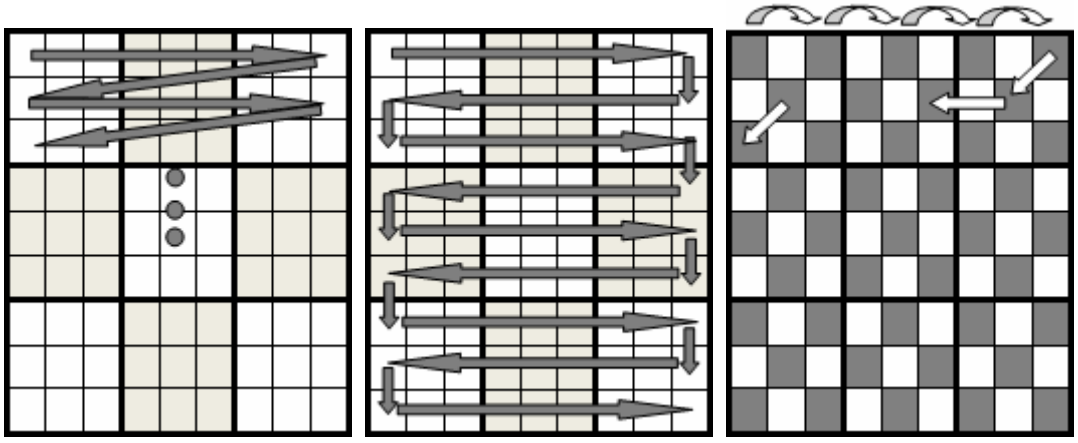


Figure 5.2: Flow chart

★ Operator ①: The sequence of digging holes

The sequence of digging holes regulates that which cell in a terminal pattern is first explored by the operation of digging holes, and which cell is the next. Four types of sequence are concluded from our quantities of trials as below:

Sequence **1**: Left to Right then Top to Bottom (seen in **Figure 5.3 (a)**)
Sequence **2**: Wandering along "S" (seen in **Figure 5.3 (b)**)
Sequence **3**: Jumping one cell (seen in **Figure 5.3 (c)**)
Sequence **4**: Randomizing globally



(a) **(b)** **(c)**

Figure 5.3: The illustrations of the four types of sequences

With these four sequences a Sudoku puzzle can be generated in any difficulty level within tolerable time (1s) excluding the evil. Meanwhile, we find that puzzles with fewest givens can be generated by Sequence 1. However when we make a puzzle of evil level based on the four sequences respectively, there is an obvious difference between the results which are drawn from average running time and the rate of success. The statistic output is shown in **Table 6**.

Table 6: Comparison of the effectiveness of the four sequences in generating an evil-level puzzle

Sequence type	Average running time	Rate of success
Randomizing globally	12723.00ms	13.33%
Jumping one cell	1224.73ms	89.66%
Wandering along “S”	1375.52ms	100%
Left to Right then Top to Bottom	1088.35ms	100%

Thus, taking consideration of the feasibility of the methods and the diversity of the produced puzzles, we assign these four sequences to the five difficulty levels as **Table 7** indicates.

Table 7: the assignments of sequences for different levels

Level	Sequence type
1 (Extremely easy)	Randomizing globally
2 (Easy)	Randomizing globally
3 (Medium)	Jumping one cell
4 (Difficult)	Wandering along “S”
5 (Evil)	Left to Right then Top to Bottom

Operator ②: The restriction of digging holes

When completing one digging trial, the rest confirmed cells can be regarded as

givens and the entire grid seems to be a puzzle. Based on the metrics of difficulty level, we set two restrictions on the amount of givens remained by each digging operation. The range of total amount of givens and the lower bound of givens in rows and columns, as the both restrictions, are determined by the desiring difficulty value input by players.

- Restriction 1: randomize a bound value within the range of the total givens, and the remained cells must be more than that bound value.
- Restriction 2: the remained cells in each row and column must be more than the lower bound of givens in rows and columns.

A trial of digging hole proves to be illegal once violating either of the two restrictions.

The both restrictions guarantee sufficient information implied in a constructed puzzle is supportive for human logic deducing.

★Operator ③: Judge the uniqueness of solution by reduction to absurdity

We develop a new strategy to judge whether the puzzle with remained cells as givens can yield a unique solution after digging a hole in one trial. This strategy is interpreted step by step as below:

In one trail of digging a hole, suppose we try to dig a cell filled with the digit 6, then

Step 1: substitute the digit 6 into another new one from 1 through 9 one by one excluding 6 while meeting the game rules;

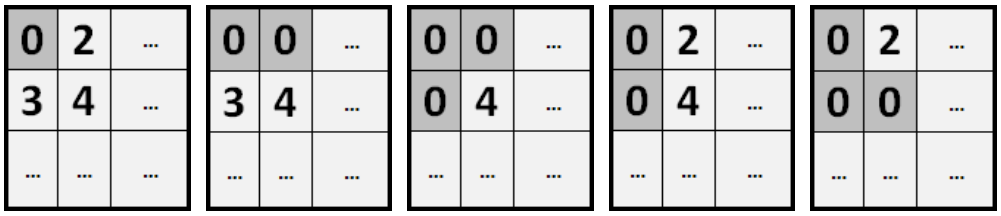
Step 2: call the solver to solve the puzzle with the givens including the new digit.

Step 3: once the solver reports a solution, terminate the solver and claim that the puzzle generated by digging out the digit 6 into empty cell has two solutions at least because originally a solution exists when the cell is filled with the digit 6.

Step 4: only if all rest 8 digits excluding 6 are used to do such a trial in step 1 and 2 and the solver reports none solution, it is safe to claim that the puzzle generated by digging out the digit 6 into empty cell has a unique solution, which means that the operation of digging out the digit 6 is feasible and legal.

★Operator ④: Pruning optimization

Using pruning technique we reduce the running time spent in doing massive trials of digging holes. We interpret the process of digging holes in details, which is illustrated in **Figure 5.4**.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 5.4: Process of pruning technique

Suppose we finish digging a hole at the top-left corner and indicate this empty cell by filling it with 0 (seen at (a)). We do a trial of digging a hole at the digit 2 (seen at (b)), and then find that the puzzle with remained cells as givens can yield at least two solutions by the solver. Since the operation of digging out 2 is unfeasible, we claim that the pattern in which more cells are dug out than pattern (b) is unfeasible as well (seen at (c)). Thus we refill the digit 2 into the cell at (R1,C2) (known as Row 1, Column 2), and try to dig the cell (R2,C1) (seen at (d)). Suppose we are then told that the pattern (d) has a unique solution by the solver. When we try to select next cell to dig from pattern (d), the cell (R1,C2) are eliminated from the selectable cells because the pattern (d) returns to the unfeasible pattern (c). Thus the next cell tried to dig may be any cell in the grid excluding the cells ever tried to dig as (R1,C2) and the empty cells as (R1,C1). Based on this analysis, we regulate that a cell can only be tried to dig once, which concludes that we only need to try to dig a cell at most 81 times for each cell from a terminal pattern, and any empty cells should not be refilled again. This strategy significantly differs from the “Backtrack” idea.

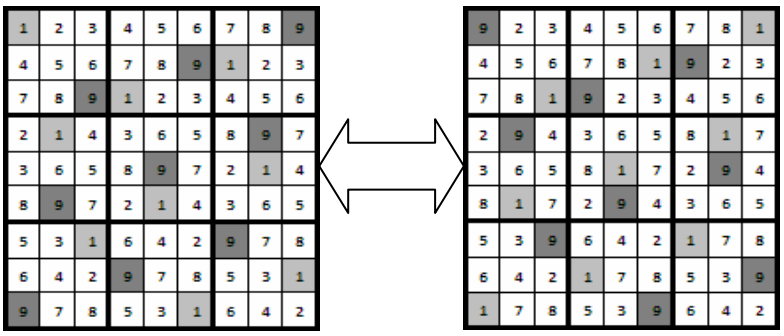
By testifying the effectiveness of the pruning technique, we perform massive operations of digging holes with the pruning technique and without the pruning technique. Finally we find the pruning technique is significantly effective in reducing the running time as **Table 8** indicates.

Table 8: The affection of pruning technique on running time

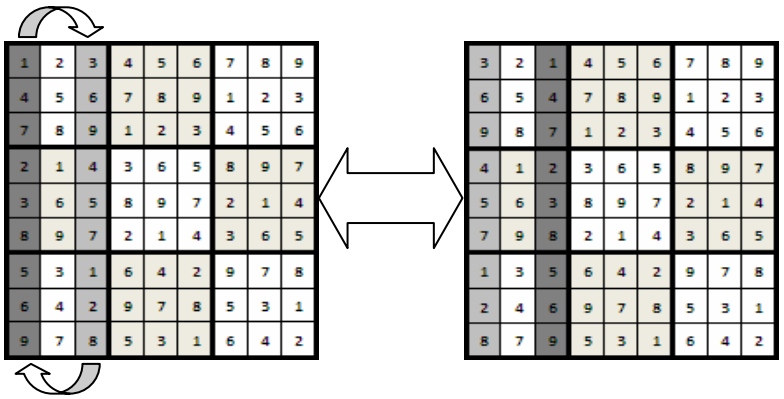
Pruning technique	Average running time
Without	55106.23ms
With	1088.35ms

Operator ⑤: Equivalent propagation

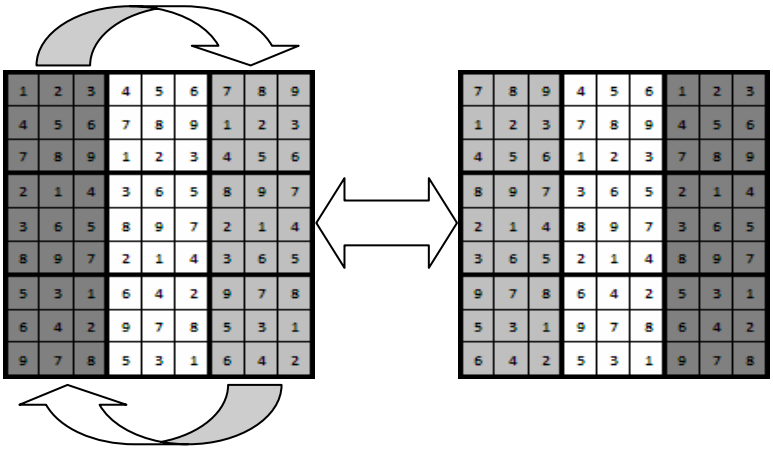
By analyzing the properties of Sudoku puzzles based on the game rules, four types of propagation in a valid grid (known as terminal pattern) can change the puzzle pattern while obeys the game rules. We illustrate these propagations in **Figure 5.5**.



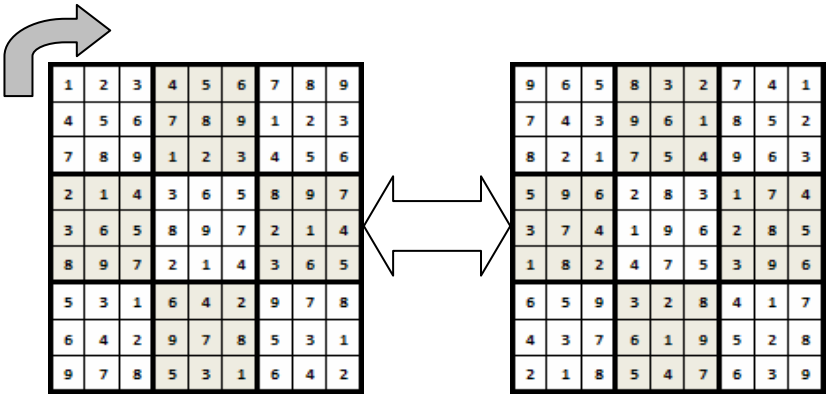
(a) Propagation 1: mutual exchange of two digits



(b) Propagation 2: mutual exchange of two columns in the same column of blocks



(c) Propagation 3: mutual exchange of two columns of blocks



(d) Propagation 4: grid rolling

Figure 5.5: The illustrations of the propagations

6 Results

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

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

Level 1: Extremely easy

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

Level 2: Easy

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

Level 3: Medium

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

Level 4: Difficult

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

Level 5: Evil

Figure 6: The results

7 Analysis of Algorithm Complexity

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

The entire algorithm for generating Sudoku puzzle is divided into two parts: to create a terminal pattern and then to dig holes on it. The time complexity of creating a terminal pattern proves to be $O(1)$, which indicates that a terminal pattern can be produced within constant-magnitude time. In the algorithm of generating puzzles, the program does trials of digging a hole at most 81 times in the terminal pattern. For guaranteeing the uniqueness of solution derived from the constructed puzzle, a solver built by the Depth-First Search is called to find out all potential solutions of the dug-out puzzle once a trial of digging a hole is done at an unexplored cell. The time complexity of the Depth-First Search proves to be $\Theta(V+E)$ in [8]. The amount of $(V+E)$ in the problem of solving a Sudoku puzzle is estimated to be less than 1500000.

8 Strengths and Weaknesses

8.1 Strengths

- Basically, we have devised an extensible algorithm to generate Sudoku puzzles in different levels of difficulty, which is adaptive well the demands from many players in different intellectual levels.
- After massive efforts paid in exploring the relationship between the sequence of digging holes and the difficulty levels of the generated puzzles, we generate Sudoku puzzles using different sequences of digging holes to adapt the characters of different levels of difficulty.
- By introducing the pruning technique, we minimize the running time spent in digging holes to generate Sudoku puzzles.
- Using a randomized algorithm called Las Vegas algorithm and equivalent propagating transformation, we raise the diversity of Sudoku puzzles generated by our algorithms.

8.2 Weaknesses

- Unfortunately, we are not able to build a superb generator of Sudoku puzzle to simulate all the existing techniques by human logic thinking in grading a Sudoku puzzle.
- As a sub-goal that is to generate an evil-level Sudoku puzzle with minimal 17 givens, we can only achieve the generation of the evil-level Sudoku puzzle with 22 givens at least.

9 Conclusions

With the task that is to create Sudoku puzzles of varying difficulty, we construct the metrics to define a difficult level, and develop an algorithm to generate Sudoku puzzles in different levels of difficulty by means of "dig-hole" strategy. In this paper, our massive works in exploring the mechanism of digging holes are concluded as below:

- Conclusion 1: the Left to Right then Top to Bottom sequence of digging holes is helpful to generate an evil-level puzzle with clustering distribution of givens; correspondingly the full-randomized sequence for an easy-level one with scattered distribution.
- Conclusion 2: it is simply demonstrated by the reduction to absurdity that whether an unexplored cell can be dug out legally meanwhile guarantees the solution's uniqueness of the puzzle being dug out.
- Conclusion 3: the running time spent in digging holes can be optimized by avoiding backtracking to an explored cell and refilling a cell dug into empty.

By comparison with modern optimization algorithms like Genetic Algorithm, our generating algorithm with "dig-hole" strategy performs better in consumed computational time and earns higher practical value for business implementation.

References

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

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

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

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

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

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

[7] M.H.Alsuwaiyel, *Algorithms Design Techniques and Analysis.* London: World Scientific Publishing Company. 1998.

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