Chapter 5

Techniques for Generating Sudoku Instances

Overview

Sudoku puzzles become worldwide popular among many players in different intellectual levels. In this chapter, we are going to discuss different ways of creating numerous Sudoku instances of varying levels of difficulty, in terms of number of clues only and not by their location. There are several ways to generate a Sudoku instance. The most popular one is to consider one solved Sudoku puzzle, and remove some of the numbers from the cells based on the required difficulty level and after each removal, check whether there is a unique solution for the created Sudoku instance, though no such technique found yet that tells about the uniqueness of a solution. The *Digging Hole* strategy is one of the most popularly methods that generates Sudoku instances. Other methods include creation of a Sudoku instance from an existing Sudoku instance using shuffling techniques like *digit exchanging*, *rotation*, *rows-in-a-band exchanging*, *columns-in-a-stack exchanging*, *band exchanging*, *stack exchanging*, and the combination of some or all of these methods mentioned herein. For creating an instance of the puzzle as per different levels of difficulty, a proper understanding about the difficulty level is very much important. So in the next section, we are going to discuss on the different difficulty level of the Sudoku puzzle.

5.1 Metrics of Difficulty Level

In this section, we develop the metrics to determine the difficulty level of a Sudoku puzzle from both computing as well as human logic deducing perspective. The following two factors [15] as metrics are taken into consideration that affecting the difficulty level:

- The total amount (or number) of given cells (or clues), and
- The lower bound of given cells in each row, column, and minigrid.

Based on the above two factors with scores, we grade a Sudoku puzzle in five different levels as follows:

Level 1: Extremely Easy,

Level 2: Easy,

Level 3: Medium,

Level 4: Hard, and

Level 5: Evil.

5.1.1 The Total Amount of Given Cells

As the first factor that affecting the level estimation is the total amount of given cells in an initial Sudoku puzzle. This factor can significantly eliminate potential choices of digits in each cell by three constraints in the game rule such that each row, column, and minigrid would contain 1 through 9 exactly once. In general, it is reasonable to argue that the more empty cells (and fewer clues) provided at the beginning of a Sudoku game, the higher level the puzzle graded in. Different researchers have moderately scaled the amount of ranges of givens for each level of difficulty; in brief such an accumulated data are shown in Table 5.1 [15].

Table 5.1: The amount ranges of givens in each difficulty level.

Level	Number of Clues	Score
1 (Extremely easy)	More than 46	1
2 (Easy)	36-46	2
3 (Medium)	32-35	3
4 (Hard)	28-31	4
5 (Evil)	17-27	5

5.1.2 The Lower Bound on the Number of Clues in Each Row, Column, and Minigrid

The arrangement of empty cells significantly affect the difficulty level if two puzzles provide the same amount givens (or in slight difference) at the beginning of a Sudoku game. The puzzle with the givens in clusters is graded in higher level than that with the givens in scattered form. Based on the row, column, and minigrid constraints, we normalize the lower bound on the number of given cells in each row, column, and minigrid for each level of difficulty as shown in Table 5.2.

Table 5.2: The lower bound on the number of clues in each row and column for each difficulty level.

Level	Lower Bound on the Number of Givens in Each Row, Column, and Minigrid	Score
1 (Extremely easy)	5	1
2 (Easy)	4	2
3 (Medium)	3	3
4 (Hard)	2	4
5 (Evil)	0	5

5.2 Generating a Sudoku Instance from a Solved Sudoku Puzzle

In this method an already solved Sudoku puzzle has been taken as input. Then using *Digging Hole* strategy [52] some of the values (or digits) are removed from the filled in solved Sudoku grid and then in the end it is checked for conflicts, if any. If there is no more conflict and the instance generated is solvable, then it is taken as a valid Sudoku instance. For an example consider the solved Sudoku puzzle shown in Figure 5.1.

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

Figure 5.1: A solved Sudoku puzzle.

Now we can remove the values from any of the cells and check for the conflicts, if arises. *Conflict checking* means to judge whether the generated Sudoku instance provides only one solution after removal of a number of values from the cells (from a solved Sudoku solution in some desired fashion) [52], though before our research we did not found any existing article that can compute two or more solutions of a Sudoku instance, if it really has. The number of values that is supposed to be there as given clues entirely depend on the difficulty level under consideration, as per the guidelines shown in Tables 5.1 and 5.2. That means if we want to create puzzles of size 9×9, with the difficulty level Extremely Easy, there must be more than 46 values and at most 34 values can be removed from the Sudoku solution using *Digging Hole* strategy. Similarly, in the case of generation of Easy puzzles, we can remove 35-45 values whereas in the case of Medium we can remove 46-49 values. In case of difficult puzzles, we can remove 50-53 values whereas in case of generating Evil instances we can remove 54-64 values. At the time of removing the values it must satisfy the constraints shown in Table 5.2. That means, in case of

Extremely Easy puzzles, there should be at least five clues in each row/column / minigrid, whereas only four, three, two, and zero clues must be present in the cases of generating Easy, Medium, Hard, and Evil puzzles, respectively, as mentioned in Table 5.2.

Now we summarise the processes that could be adopted in removing the values from the cells while generating instances from a solved Sudoku puzzle as follows:

1) Randomized, and

2) Sequential.

5.2.1 Randomized Selection of Cell Location

In case of randomized site selection, we can remove the values from any of the locations from a solved Sudoku puzzle. For example, we have removed values from 34 random cell locations from the solved Sudoku puzzle shown in Figure 5.1, and obtain a generated Sudoku instance as shown in Figure 5.2. Here we have taken away the values from cells [1,2], [1,3], [1,6], [1,7], [2,1], [2,4], [2,7], [2,8], [3,3], [3,5], [3,6], [3,9], [4,1], [4,3], [4,5], [4,8], [5,3], [5,5], [5,8], [6,2], [6,4], [6,7], [6,8], [7,2], [7,5], [7,9], [8,1], [8,4], [8,6], [8,7], [9,1], [9,4], [9,6], and [9,9].

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

Figure 5.2: An Extremely Easy Sudoku instance created after randomly digging holes in the solved Sudoku puzzle shown in Figure 5.1.

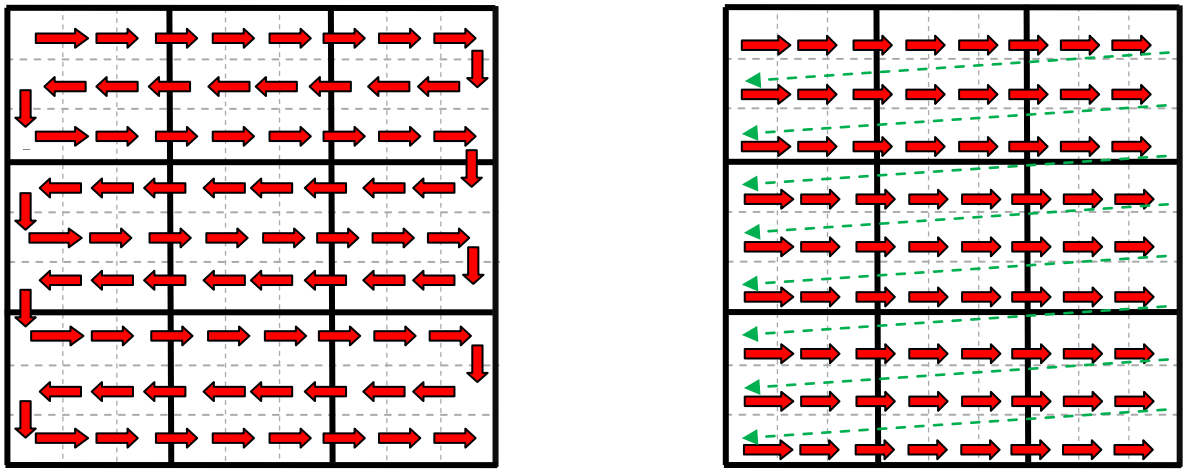
In Figure 5.2, we can say that the generated Sudoku instance is extremely easy as there are 47 givens / clues now and each row, column, and minigrid is having at least five givens / clues. In a similar way, we can create puzzles with different other levels of difficulty satisfying the constraints mentioned in Tables 5.1 and 5.2.

5.2.2 Sequential Selection of Cell Location

In this method of generating Sudoku instances, the removal of values from cells can follow certain sequence along rows or columns. In case of rowwise direction, the selection path could be zigzag or like English letter (capital) 'S' or the reverse and only rowwise or columnwise or the reverse, as discussed in brief as follows:

5.2.2.1 Wandering along S (or Zigzag) Path

In this process, the cells to be dug out are chosen from the left to right direction for the first row. Then for the next row it starts moving from the right to left. Then for the successive rows, the direction of selection of cells changes alternatively. The process is shown in Figure 5.3(a). In a similar way, the direction can also be adopted from the right to left for the first row, from the left to right for the second row, and so on, i.e. the opposite (or reverse) direction of the above.



(a) **(b)**

Figure 5.3: Cells are dug in rowwise direction while **(a)** wandering along 'S' path and **(b)** moving from the left to right direction.

5.2.2.2 Wandering from Left to Right or the Reverse

In this process cells to be dug out are chosen from the left to right direction for each of the rows. It starts choosing the cell from the top left corner then move to the right, again starts moving from left to right for the next row, and so on. The process is depicted in Figure 5.3(b). It can also follow the reverse sequence.

Likewise, the cells to be dug can also be chosen in the following fashion when we follow columnwise direction.

A) Top to Bottom or the Reverse

In this case, the direction of considering the path could be from the top to bottom always for each of the columns, either from left to right or from right to left, or the reverse fashion in all respects.

B) Wandering along S in top to bottom direction and vice versa

In this case, for a solved Sudoku solution, the instance generator may follow a path alternatively from the top to bottom and from the bottom to top like a sideways 'S' (or zigzag), or the reverse.

Instead of removing the values from the cells randomly or any well-defined fashion, they can also be removed in a symmetrical way for matching pairs of rows, columns, and minigrids. These are also briefly discussed and exemplified as follows.

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

Figure 5.4: A Sudoku puzzle instance is created by symmetric removal of values (once from top-left and then from bottom-right) from the solved puzzle shown in Figure 5.1.

5.2.2.3 Symmetrical Removal of Values from Rows

In this method, values are removed in a symmetrical fashion. As for example, if the top row is considered from the left to right, then the bottom row is considered from the right to left. Then the locations are removed simultaneously and either the corresponding values are both kept at their own locations or both of them are removed. The same fashion of considering rows (from the top and the bottom) and keeping or removing values in a symmetrical manner is executed till

the middle of the Sudoku instance is reached. The same process can also be executed columnwise in achieving a new Sudoku instance.

As for example, a generated instance is shown in Figure 5.4, which is created from the solved Sudoku puzzle shown in Figure 5.1. In the figure, Row numbers and column numbers are shown on the left and top side of the puzzle. This is by definition a puzzle with Medium level of difficulty as there remain only 35 clues (or givens) and each row, column, or minigrid contains at least three among the clues. Here we may observe that in the first row (from the left to right) the status of clues is ×2××5×78× (including blank cells being represented by '×') and the status of clues in the last row (from the right to left) is ×5××3×89× (that are symmetric). Such similarity could be observed for each pair of equidistant rows from the top as well as from the bottom, and also for each pair of equidistant columns from the left as well as from the right in the generated Sudoku instance.

5.2.2.4 Symmetrical Removal of Values from Columns

If we deeply observe the way of creating symmetrical Sudoku instances as has been described in the previous section when we removed (or kept) values rowwise for a solved Sudoku puzzle, that automatically generates instances where it is symmetric columnwise as well. Rather, we may execute the tasks that we followed in generating Sudoku instances by removing (or keeping) values in symmetrical fashion by rows, can also be obtained by executing this method too. As for example, we may observe the third column from left (from the bottom to top) and the third column from right (from the top to bottom); the status of the said columns are 81×2×97×× and 73×4×81×× that are absolutely symmetric.

5.2.2.5 Symmetrical Removal of Values from Minigrids

In a similar fashion, a new instance of the Sudoku puzzle can also be generated by symmetric removal of values from the minigrids. Based on the symmetry of a Sudoku puzzle, we may observe that if either of the techniques discussed in both of the above two sections is executed, a Sudoku instance is generated wherein a symmetric fashion could be obtained between pairs of minigrids (1, 9), (2, 8), (3, 7), and (4, 6) in some fashion. We may observe that the created Sudoku instance in Figure 5.4 follows these minigrid pairs (in reverse direction). Another minigrid pairs could be formed where either columnwise (1, 3), (4, 6), and (7, 9) are symmetric

where the values from the top to bottom along columns 4 and 6 are also symmetric, or rowwise (1, 7), (2, 8), and (3, 9) are symmetric where the values from the left to right along rows 4 and 6 are also symmetric.

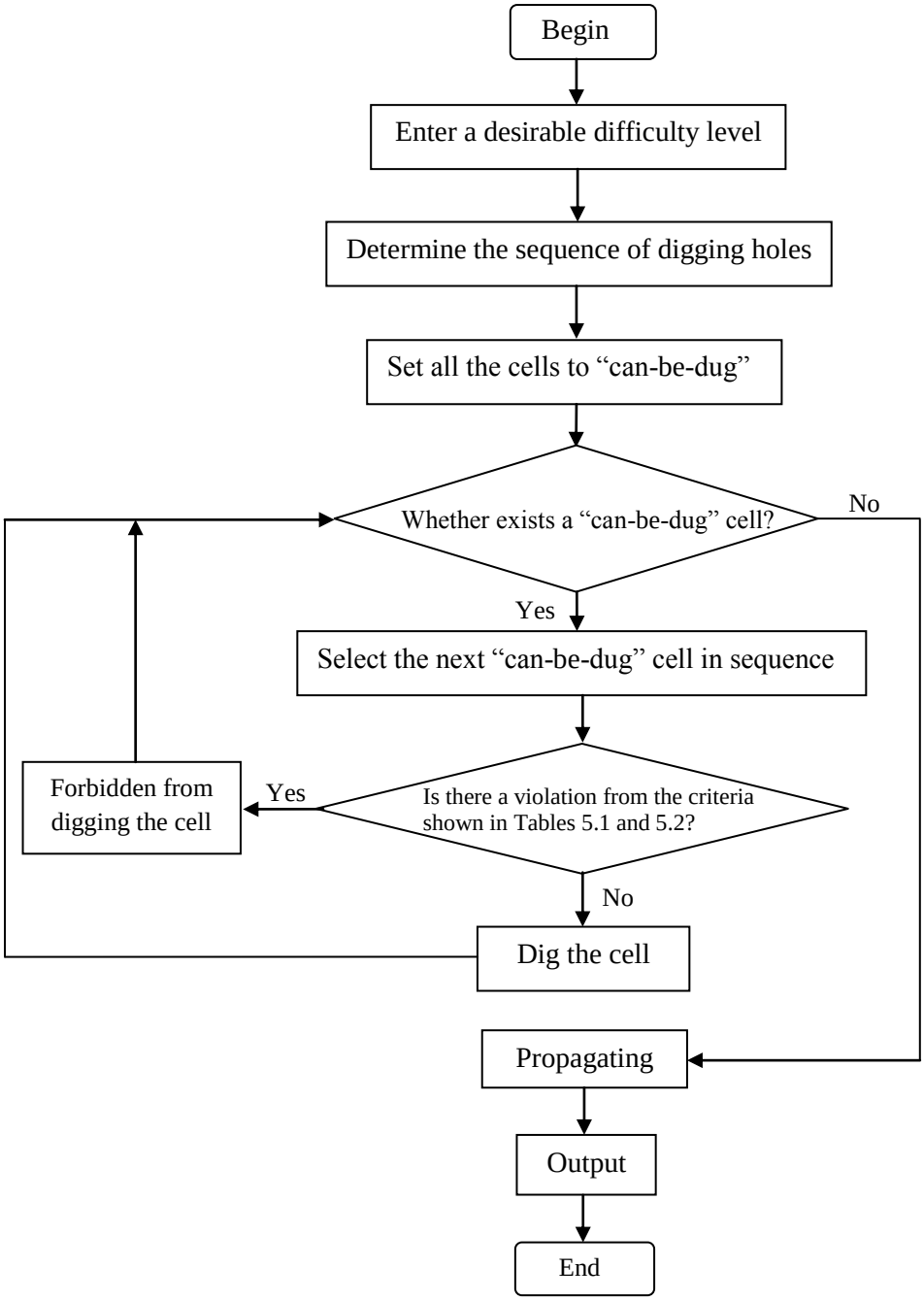


Figure 5.5: Flowchart for the *Digging Hole* strategy.

5.2.3 Flowchart at a Glance for the *Digging Hole* Strategy

The steps for the said algorithm are depicted in the flowchart shown in Figure 5.5. From the flowchart, it is very clear that first we need to input one solved Sudoku puzzle. Then we are supposed to provide the level of difficulty for which an instance is supposed to be created. From the difficulty level the algorithm finds out the "can-be-dug" cells (in some fashion, well-defined or arbitrary) based on the constraints and criteria shown in Tables 5.1 and 5.2. Then from the "can-be-dug" cells, cells are chosen in some fashion and follow a sequence of cells based on the techniques discussed in previous sections. Next the chosen cells made blank by removing values from each of them and checked for if a unique valid solution for the created Sudoku instance is achieved.

Using this method, we eventually can create a Sudoku instances though this algorithm has not stated clearly from which locations the values should be removed. We can remove them from any place we like to, either randomly or by following a definite sequence. Then after removing a value each time it is checked that whether there exists a unique solution of the generated instance. So again checking for the solution is also a very time consuming task. Incidentally, there does not be any suitable technique in the literature which can successfully check whether the generated Sudoku instance is truly having more than one solution. Another prime limitation is that this strategy needs a solved Sudoku puzzle for generating a Sudoku instance or more.

5.3 Generation of Sudoku Instances by Transformation of Puzzles

The other method of creating a Sudoku instances is to transform an existing Sudoku puzzle into a new one. In this type of method, an existing Sudoku instance is taken as input. Then the instance is transformed by using any of the methods as discussed below to get a new instance of the puzzle. For transformation, we use the following methodologies [53] such as (i) *Digit exchanging*, (ii) *Rotation*, (iii) *Rows-in-a-Band exchanging*, (iv) *Columns-in-a-Stack exchanging*, (v) *Band exchanging*, (vi) *Stack exchanging*, and (vii) *The combination of all six methods*. Now we consider each of these methods and briefly discuss as follows.

5.3.1 Digit Exchanging

It is simple to accomplish the method of digit exchanging because what is necessary here is to exchange all the digits in the cells of one existing Sudoku instance in some well-defined fashion.

Interestingly, this exchange does not influence the uniqueness of a Sudoku instance. Thus a new instance of the Sudoku puzzle is produced.

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

Figure 5.6: An instance of Sudoku puzzle.

Figure 5.6 shows an instance of Sudoku puzzle. Let us replace all 1's belonging to this puzzle by 9 and the reverse. Then the modified Sudoku instance can be as shown in Figure 5.7. We can see in this figure, all 9's present in cells [6,1], [2,6], and [7,9] are now replaced by 1's, whereas all 1's present in cells [1,1], [7,2], [6,8], and [9,9] are now replaced by 9's. All these replacements are highlighted in the figure. This process of exchanging digits is valid throughout the puzzle as a new Sudoku instance is generated, and it is valid for exchanging any number of values among themselves in the puzzle.

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

Figure 5.7: A new instance is generated from the Sudoku instance shown in Figure 5.6 after interchanging of 1 and 9.

Now, the same method can be carried out for a multiple pairs of numbers (to be exchanged) as well. Figure 5.8 shows a newly generated Sudoku instance from the Sudoku instance shown in Figure 5.6, where values 1 and 9, 5 and 6, and 7 and 8 are pair wise exchanged in order to acquire a new Sudoku instance. All the replacements are highlighted in the said figure.

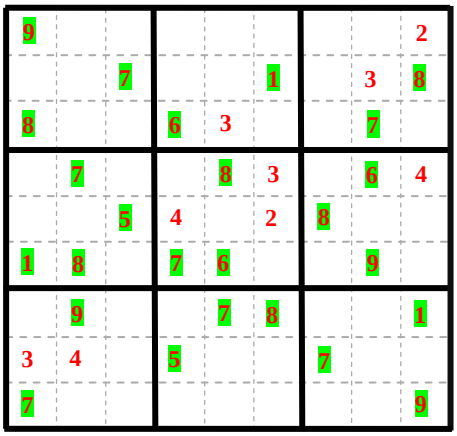


Figure 5.8: A new instance is generated from the Sudoku instance shown in Figure 5.6 after exchanging three pairs of values.

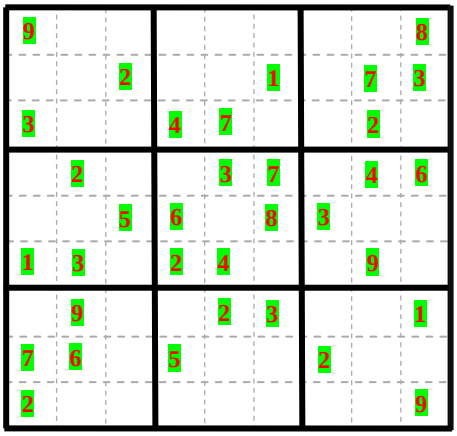


Figure 5.9: A new Sudoku instance is generated after replacement of all the digits for the Sudoku instance shown in figure 5.6.

5.3.1.1 Replacement of All Digits

A new Sudoku instance can be produced by replacing values of all clues of a Sudoku instance among themselves. Consider the same Sudoku instance shown in Figure 5.6. Here we may exchange the following values pair wise: (1, 9), (2, 8), (3, 7), (4, 6), (5, 4), (6, 5), (7, 3), (8, 2), and (9, 1). Then, we can get a new instance as shown in Figure 5.9. Here we may observe that

the 1's present in cells [1,1], [7,2], [6,8], and [9,9] are now replaced by 9's, the 2's present in cells [1,9] and [5,6] are replaced by 8's, and so on that have been depicted in this figure.

5.3.2 Rotation

In this method, an existing Sudoku instance is rotated by a certain angle (with multiple of unit value 90 degree) to produce a new instance. By the application of the angle of rotation the newly generated instance can be characterized as follows:

(i) Rotation by 90 degree,

(ii) Rotation by 180 degree,

(iii) Flipping vertical rotation, and

(iv) Flipping horizontal rotation.

5.3.2.1 Rotation by 90 Degree

The Sudoku instance can be rotated by an angle of 90 degree. The direction of rotation can be of two types:

(a) Left Rotation (or in anticlockwise direction), and

(b) Right Rotation (or in clockwise direction).

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

Figure 5.10: A new Sudoku instance generated after left rotation of 90 degree of the Sudoku instance shown in Figure 5.6.

In Figure 5.10, a newly generated Sudoku instance is shown, which is produced after left rotation of 90 degree of the Sudoku instance shown in Figure 5.6.

We can observe that the clues present in minigrids 3, 6, 9, 2, 5, 8, 1, 4, and 7 in Figure 5.6 become the clues of minigrids 1 through 9, respectively, as shown in Figure 5.10. The row numbers of the clues now become the corresponding column numbers of the same.

After a right rotation of 90 degree of the same Sudoku instance in Figure 5.6, the newly transformed instance is depicted in Figure 5.11. We can observe that clues present in minigrids 7, 4, 1, 8, 5, 2, 9, 6, and 3 become the clues of minigrids 1 through 9, respectively. Here the column numbers of the clues now become the corresponding row numbers of the same.

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

Figure 5.11: A new Sudoku instance is generated after right rotation of 90 degree of the Sudoku instance of Figure 5.6.

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

Figure 5.12: A new Sudoku instance is generated after rotation of 180 degree of the Sudoku instance shown in Figure 5.6.

5.3.2.2 Rotation by 180 Degree

In the similar way, after 180 degree of rotation of the Sudoku instance shown in Figure 5.6, the transformed instance obtained is shown in Figure 5.12. We can observe that the clues present in minigrids 9, 8, 7, 6, 5, 4, 3, 2, and 1 have become the clues of minigrids 1 through 9,

respectively, and the initial instance has been toggled, that means the top rows (from the left to right) have now been converted to as the bottom rows (from the right to left), and the reverse.

5.3.2.3 Flipping Vertical Rotation

The Sudoku instance can be flipped vertically (keeping the clues of the fifth row unchanged) to produce a new instance of the puzzle. The Sudoku instance shown in Figure 5.13 is produced after flipping vertically of the Sudoku instance shown in Figure 5.6. We can observe that the clues present in minigrids 7, 8, 9, 4, 5, 6, 1, 2, and 3 have become the clues of minigrids 1 through 9, respectively, as well as the top row of the instance now becomes the bottom one, and vice versa, keeping the row information from the left to right as it was.

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

Figure 5.13: A new Sudoku instance is generated after flipping vertically of the Sudoku instance shown in Figure 5.6.

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

Figure 5.14: A new Sudoku instance is generated after flipping horizontally of the Sudoku instance shown in Figure 5.6.

5.3.2.4 Flipping Horizontal Rotation

An existing Sudoku instance can be flipped horizontally (keeping the clues of the fifth column unchanged) to generate a new instance. In case of flipping horizontal rotation, the clues present in minigrids 3, 2, 1, 6, 5, 4, 9, 8, and 7 now become the clues of minigrids 1 through 9, respectively, as well as the left column of the instance now becomes the tight column, and vice versa, keeping the column information from the top to bottom as it was. In other words, each minigrid is now the mirror of the given (or original) one. The transformed minigrid of the Sudoku instance shown in Figure 5.6 is depicted in Figure 5.14.

5.3.3 Rows-in-a-Band Exchanging

Rows-in-a-band exchanging means interchanging two or three rows in the same band in any fashion we like to (or randomly). Three consecutive minigrids in a row form a band as shown in Figure 5.15. We can observe from the figure that minigrids 1, 2, and 3 form band 1, whereas minigrids 4, 5, and 6 form band 2, and minigrids 7, 8, and 9 form band 3. Now we may examine that all the rows in each band are interchangeable among themselves in order to generate newer Sudoku instances.

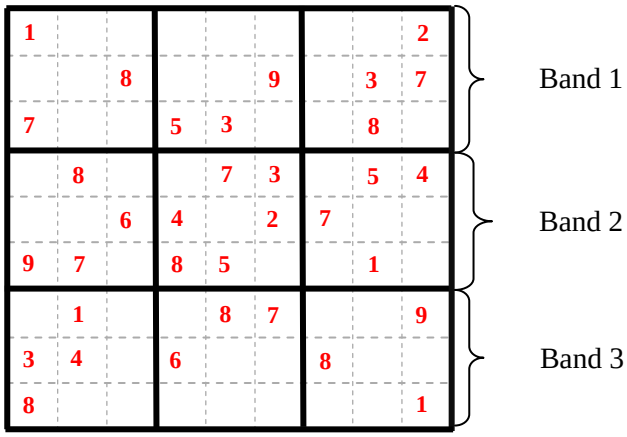


Figure 5.15: Concept of band in Sudoku puzzle.

After the exchange of the rows in the same band, the newly made puzzle obtained is still a valid one. Figure 5.16 shows a new Sudoku instance, which is created by exchanging the first row with the second row in band 1 for the Sudoku instance shown in Figure 5.6. This idea can be generalized in exchanging rows in respective bands, in isolation or in combination, in order to produce more and more Sudoku instances.

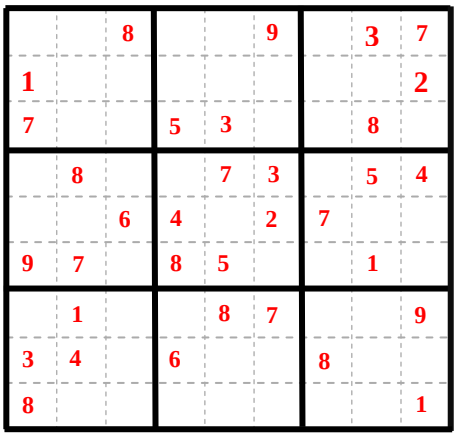


Figure 5.16: A new Sudoku instance is created after exchanging the values present in rows 1 and 2 in band 1 for the Sudoku instance shown in Figure 5.6.

5.3.4 Columns-in-a-Stack Exchanging

Here we adopt the same concept that has been discussed in the previous section. Columns-in-a-stack exchanging means interchanging of two or three columns in the same stack randomly (or in some well-defined fashion). Let us first understand the concept of stack. The entire minigrids present in the same column form a stack. The concept of stack is shown in Figure 5.17.

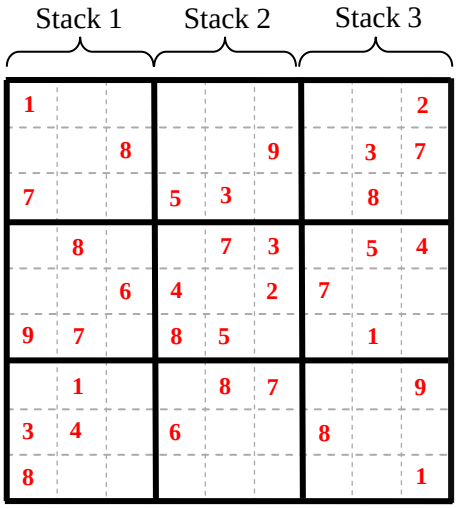


Figure 5.17: Concept of stack in Sudoku puzzle.

From the figure, we can observe that minigrids 1, 4, and 7 form stack 1, minigrids 2, 5, and 8 form stack 2, and minigrids 3, 6, and 9 form stack 3. The columns of each stack are interchangeable among themselves. A newly generated Sudoku instance is depicted in Figure 5.18 after exchanging the first two columns of stack 1 of the instance shown in Figure 5.6.

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

Figure 5.18: A new Sudoku instance is created after exchanging the values of column 1 and column 2 of the Sudoku instance shown in Figure 5.6.

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

Figure 5.19: A new Sudoku instance is created after exchanging the values in band 1 with that of band 3 for the Sudoku instance shown in Figure 5.6.

5.3.5 Band-Exchanging

Now it is intuitively obvious that we can also interchange the position of an entire band with another one to acquire a new Sudoku instance. Thus, as shown in Figure 5.19, bands 1 and 3 of the Sudoku instance in Figure 5.6 have been interchanged to realize this instance. Here we can observe that minigrids 7, 8, and 9 in Figure 5.6 have been exchanged with minigrids 1, 2, and 3 to get this new Sudoku instance.

5.3.6 Stack-Exchanging

We can also exchange the position of a whole stack with another stack to get a new Sudoku instance. Hence, if stacks 1 and 3 of the Sudoku puzzle shown in Figure 5.6 are swapped, then a new Sudoku instance is obtained as shown in Figure 5.20.

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

Figure 5.20: A new Sudoku instance is created after exchanging the values in stack 1 and stack 3 present in the Sudoku instance shown in Figure 5.6.

We may observe that the clues present in minigrids 1, 4, and 7 are now the clues in minigrids 3, 6, and 9, respectively, and the clues present in minigrids 3, 6, and 9 are now the clues in minigrids 1, 4, and 7, respectively, in the newly generated instance.

5.3.7 A Combination of All Six Methods

If the all six methods or any two or more of them among *digit exchanging*, *rotation*, *rows-in-a-band exchanging*, *columns-in-a-stack exchanging*, *band exchanging*, *stack exchanging* are combined and acted upon an existing puzzle instance, a new instance can be produced (with no influence on its correctness). This fact is implicit as all these methods in isolation starts from a complete Sudoku solution, and thus, any instance that is created from a complete Sudoku solution must have at least that solution as the outcome of the newly generated instance.

Now here we briefly elucidate a case to show how a new Sudoku instance could be obtained from a given Sudoku instance where several such methods are taking their role in a sequence. Say, first of all, we replace digits 1, 2, 3, 4, 5, 6, 7, 8, and 9 by 7, 3, 6, 5, 4, 8, 2, 9, and 1, respectively, of the Sudoku puzzle shown in Figure 5.21(a) and then rotate the obtained puzzle at an angle of 90 degree in anticlockwise direction. Then we pair wise exchange rows 4 and 6, rows

7 and 8, columns 1 and 2, and also columns 8 and 9 one after another. In the end, we swap the first band with the second band, and also the second stack with the third stack in sequence. After execution of all these stages, a new solution grid as an instance of a Sudoku puzzle is created, which is shown in Figure 5.21(b).

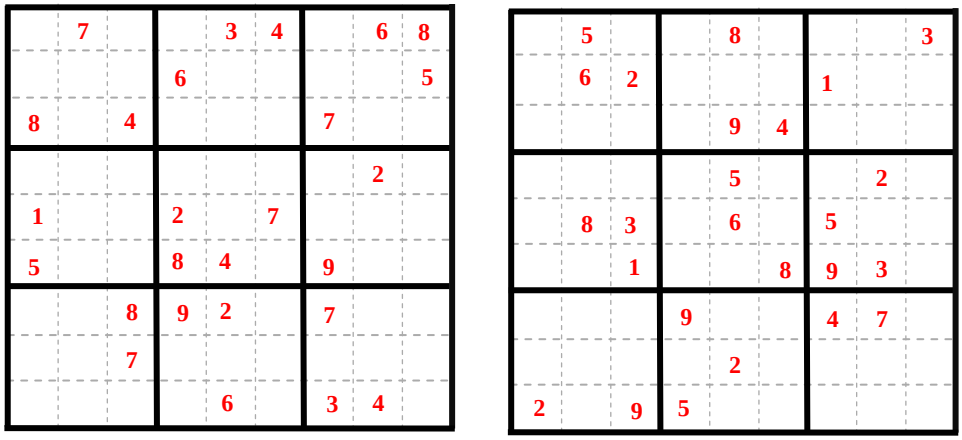


Figure 5.21: (a) A given Sudoku instance. **(b)** A newly generated Sudoku instance after application of several methods of transformation.

5.4 Verifying the Uniqueness of the Solution

As discussed in the introductory chapter, a well-formed Sudoku puzzle must have a unique solution [25]. But incidentally, we found no article in literature that guarantees the uniqueness of a Sudoku instance. Nevertheless, though we are not convinced, what we have obtained in literature does the following towards their claim. After realizing a newly generated instance through the application of the *Digging Hole* strategy and/or transformation executed over a complete Sudoku solution on the way to create a new instance, it is necessary to verify whether the created puzzle can produce one or more solutions. As claimed, researchers usually apply the method of backtracking [52] hoping that a new solution is obtained. If a new solution is obtained, definitely the puzzle instance created is not unique. But, in general, one cannot guarantee a new solution is likely, even if a new solution of the created Sudoku instance could be there [14].

In this respect, the graph theoretic algorithms developed in this thesis (in Chapter 3 and 4) are truly meticulous and authentic that it certifies the uniqueness of any Sudoku instance. From that point of view as well, we dare to generate a large number of Sudoku instances based on the ideas

partially existing in literature [14, 53] and experimenting a lot thereafter as has been included in this chapter. In any case, as the total number of possible Sudoku instances of size 9×9 is known by a mathematical measure (which is 6,670,903,752,021,072,936,960 [37]), as mentioned in Chapter 2, any number of instances that we may generate in practice, is really very few for any experimentation in this field of research. Next, in future, we like to modulate all these ideas towards generation of Sudoku instances for larger Sudoku puzzles as mentioned in Chapter 1 of this thesis.

5.5 Summary

In this chapter we have discussed several methods for creating new Sudoku instances. Essentially two methods are discussed here. The first method is known as the *Digging Hole* strategy and the second method is known as *transformation*. In *Digging Hole* strategy, an already solved Sudoku puzzle is taken as input. Then a number of values from cells are removed in some specified sequence based on the level of difficulty, and then the generated puzzle is checked for having a unique valid solution or more. In the second method, new instances are generated based on transformation(s). Here an existing Sudoku instance of certain difficulty level is taken as input. Then different kinds of transformation are applied on it, and as a result new Sudoku instances are generated. Again in this case as well, generated Sudoku instances are verified for multiple solutions, if any. For checking this, our proposed graph theoretic Sudoku solver (in Chapter 4) is greatly helpful. So we can conclude that, our proposed solver and the instance generation technique discussed in this chapter complement to each other, as our devised Sudoku solver is truly needed to confirm if multiple solutions are there for a generated instance. At the same time it is also verified whether the newly generated Sudoku instance is having only a unique solution, and for all these a Sudoku instance generation technique is very much necessary as the input to any of our algorithms (or versions of an algorithm developed in Chapters 3 and 4) is nothing but a Sudoku instance only.