IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 January 2023, accepted 19 February 2023, date of publication 14 March 2023, date of current version 22 March 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3256420

RESEARCH ARTICLE

Design and Analysis of a Modified 3D Sudoku Solver

SUNANDA JANA[1], (Graduate Student Member, IEEE), MANJARINI MALLIK[2], ABHINANDAN KHAN[3], (Member, IEEE), ARNAB KUMAR MAJI[4], (Senior Member, IEEE), AND RAJAT KUMAR PAL[2], (Member, IEEE)

[1]Department of Computer Science and Engineering, Haldia Institute of Technology, Haldia, West Bengal 721657, India
[2]Department of Computer Science and Engineering, Acharya Prafulla Chandra Roy Shiksha Prangan, University of Calcutta, Kolkata 700106, India
[3]ARP Engineering, Kolkata 700056, India
[4]Department of Information Technology, North-Eastern Hill University, Shillong, Meghalaya 793022, India

Corresponding author: Arnab Kumar Maji (akmaji@nehu.ac.in)

ABSTRACT Sudoku is a complicated multidimensional mathematical structure with several applications in various computer science domains. 3D Sudoku, compared to 2D, has one more dimension that can potentially provide an extra edge in the field of different application areas. Several researchers have developed various types of 2D Sudoku solvers using different methodologies. However, there is very limited research in the area of developing 3D Sudoku solvers. Thus, two different solvers for solving 3D Sudoku puzzles of size $9 \times 9 \times 9$ are proposed in this work. Both solvers provide all possible solutions for solving a 3D Sudoku puzzle. 2D Sudoku puzzles are applied in different research domains with different purposes. Recently, 3D structure of Sudoku has been applied in several areas to achieve more effectiveness compared to 2D Sudoku. Additionally, it can also be used to solve problems in 3D space. Again, solving an NP-complete puzzle by considering its 3D structure is a challenging job. Thus, we endeavoured to achieve all probable solutions for a 3D Sudoku instance in this work. In the first version of our proposed algorithm, all possible values for each blank cell are computed and stored. Subsequently, a few elimination-based methods are used to reduce the number of probable values (if possible) for each blank cell. Finally, the solutions are computed using the backtracking method. In the second version of our proposed algorithm, the nine 2D Sudoku puzzles, lying in the xz-plane one above the other, which form the 3D puzzle are fed as the input. All possible solutions are obtained for each of the nine puzzles. Then, the obtained solutions are mapped to achieve one or more solutions for the 3D Sudoku instance. Thus, our proposed techniques provide a new approach for solving 3D Sudoku. In addition, applying the obtained solutions provides us with an advantage over 2D Sudoku, in solving problems in the 3D space and where more data is required.

INDEX TERMS Backtracking, cell, grid, minigrid, mini-cube, puzzle, sudoku.

I. INTRODUCTION

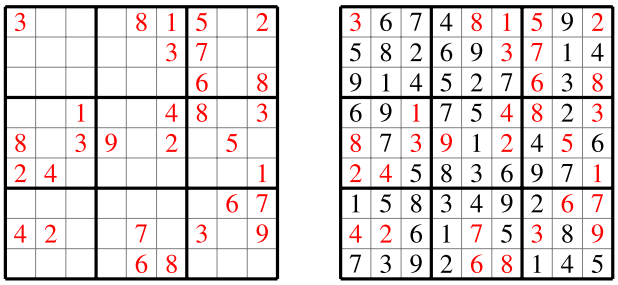
Sudoku puzzle is represented by a $n \times n$ grid in 2D, where n is a perfect square integer so that we can get n number of minigrids, each of size $\sqrt{n} \times \sqrt{n}$. The logic behind solving this puzzle is that each row, column, and minigrid contains an integer between 1 to n without repetition. Based on different grid sizes some of the possible Sudoku instances [1] are:

- 4×4
- 9×9 also called standard Sudoku
- 16×16 also called Super Sudoku
- 25×25, also known as Giant Sudoku

Apart from these, there are also several other types of Sudoku [2], which are not so familiar, namely three-dimensional Sudoku, which was invented by Dion Church and first published in The Daily Telegraph in May 2005, and Wordoku, also known as Godoku, where alphabetical placement is required instead of numerical placement.

Nowadays, Sudoku is not only treated as a puzzle game, but it has also become popular because of its application in

The associate editor coordinating the review of this manuscript and approving it for publication was P. K. Gupta.



(a) (b)

FIGURE 1. (a) A 2D Sudoku of size 9 × 9. (b) Solution of the instance.

various fields. Again, in some application areas, 3D can perform more efficiently compared to 2D, especially for solving 3D related problems. For example, in aircraft scheduling, the challenge is to maximise the number of aeroplanes in a limited 3D space. We can divide the 64 planes into four groups based on speed and place them in four elevations so that their positions do not come into conflict, using a $4 \times 4 \times 4$ Sudoku structure.

Until recently, no extensive research work was done on 3D Sudoku, even though its application can potentially revolutionise different application areas. Various 3D puzzles are being applied in ongoing research, including structural control in metal additive manufacturing [3], historical artefacts [4], image steganography [5], and security. Hence, an increase in the dimension of a Sudoku puzzle may also enhance many research areas.

A disadvantage of the 2D Sudoku problem is its limited number of elements. The grid size has to be increased for some application domains, like encryption of massive data. Hence, in such cases, implementing the compact structure of a 3D Sudoku would prove to be very useful for handling enormous data. Especially in data security, 3D Sudoku solutions are much more effective than 2D Sudoku. Nevertheless, solving a 3D Sudoku is challenging. We have proposed two versions of an algorithm that can solve 3D Sudoku in a simple and effective way. Our proposed solver can solve easy, moderate, and hard instances of 3D Sudoku in a comparatively lesser time and with a lower complexity.

The rest of the paper is structured in the following way. In Section II, we briefly describe preliminaries, which reflect the 2D and 3D structures, and a literature survey on the recent works related to Sudoku puzzle. The physical storage structure of the 3D Sudoku instance is described in Section III. Sections IV and V elaborate on our proposed algorithms, i.e., versions 1 and 2, with their respective pseudocodes. In Section VI, we present the experimental results and statistical analysis of the proposed algorithms. The paper concludes with Section VII, where we discuss the findings of our methods and future work possible in this area.

II. PRELIMINARIES

In this section, we elaborate the basic structure of 2D as well as 3D Sudoku.

A. TWO-DIMENSIONAL SUDOKU

The most used Sudoku puzzle is a two-dimensional or 2D structure. Each Sudoku constraint is applied for an individual row, column, and minigrid. Fig. 1 represents an example of such a Sudoku puzzle along with its solution. The 2D Sudoku structure can be extended to 16 × 16, 25 × 25, etc. Additionally, a 2D Sudoku can be extended by another dimension, resulting in a 3D Sudoku [2]. Adding an extra dimension will lead to a solid cubic structure of $n \times n \times n$ elements, where n itself is a perfect square integer.

B. THREE-DIMENSIONAL SUDOKU

Our proposed 3D Sudoku is represented as a solid cubic structure that consists of $n \times n \times n$ elements, where n is a perfect square integer. Let us consider as an example a $9 \times 9 \times 9$ Sudoku. In this case, we need to view nine 2D Sudoku maps, each of grid size 9×9. Each Sudoku puzzle lies in the xz-plane, one above another along the y-direction, as shown in Fig. 2.

If all elements in the ith ($1 \leq i \leq 9$) row of a 3D Sudoku map lying along the xy-plane are combined, a new 2D Sudoku will be obtained along the xy-plane. This new Sudoku is the ith 2D Sudoku along the xy-plane, as shown in Fig. 5. Similarly, if the elements in the ith ($1 \leq i \leq 9$) column of a 3D Sudoku map lying in the xz-plane are combined, the ith 2D Sudoku map along the yz-plane will be achieved. Thus, the 3D Sudoku of dimension $9 \times 9 \times 9$ contains $3^3 = 27$ 2D Sudoku grids. An example solution of a 3D Sudoku of dimension $9 \times 9 \times 9$ is presented in Fig. 2.

Nine grids represent the 2D Sudoku grids lying in the xz-plane, one above another as shown in Fig. 2. Grid 1 is the topmost among all other layers. Below grid 1, lies grid 2; below grid 2, lies grid 3; and so on. Grid 9 is the bottommost layer. Every cell $[i, j]$ of each grid has a different value. Here, a column of another 2D Sudoku is obtained in another dimension by combining each $[i, j]$ cell of the individual grid, which is represented as the third dimension, e.g., cell [2], [3] of grids 1 through 9 contains the values 5, 7, 6, 4, 2, 8, 3, 1, 9, respectively (see Fig. 2).

It is clear that all nine elements are present and they form a row in the 3rd dimension. Here 27 ($3 \times 3 \times 3$) mini-cubes are present in the 3D Sudoku of dimension $9 \times 9 \times 9$, and each mini cube is of dimension $3 \times 3 \times 3$. Consequently, each mini-cube contains 9 (i.e., 3^2) 2D minigrids of size 3×3. Each mini-cube has a total of six neighbouring mini-cubes, two each in the x-, y-, and z-directions. The values already placed in the neighbouring mini-cubes (in their appropriate positions) are considered for computing the value for a blank cell in a mini-cube. The connectivity graph of 27 mini-cubes is presented in Fig. 3. Each mini-cube is connected with its six neighbouring mini-cubes by edges. The mini-cube, S1, with its six neighbours, is shown in Fig. 4 to elaborate this connectivity further.

Moreover, grids 1, 2, and 3 form the three topmost layers lying in the xz-plane (according to Fig. 5), and the first

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

(a) Grid 1

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

(b) Grid 2

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

(c) Grid 3

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

(d) Grid 4

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

(e) Grid 5

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

(f) Grid 6

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

(g) Grid 7

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

(h) Grid 8

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

(i) Grid 9

FIGURE 2. 2D layers of a 3D Sudoku solution, each lying in the *xz*-plane one above another. Grid 1 is the topmost layer and Grid 9 is the layer at the bottom [2].

mini-cube, S1, can be obtained from the first minigrid (the topmost and leftmost) of these three layers (for the 3D Sudoku solution shown in Fig. 2), as shown in Fig. 6.

C. LITERATURE SURVEY

Nowadays, extensive research work is going on regarding Sudoku. However, many current 2D Sudoku solvers solve Sudoku using the guess-based method and hence are highly time consuming. Therefore, a guess-free Sudoku solver algorithm was proposed by Maji and Pal [6], which guaranteed all possible solutions for a valid 2D Sudoku instance. Thus, it is advantageous in different fields, like cryptography, where 2D Sudoku puzzles with more than one solution instances are highly appreciated.

There is no well-known existing algorithm for solving 3D Sudoku. Initially, a brute force algorithm using the backtracking method for 3D Sudoku was developed by Jana et al. [1]. Here, in this work, we propose an improved version compared to that developed by the authors [1], by applying specific methods and implementing a modified and updated interpretation. In addition, we also propose a second algorithm for hard Sudoku puzzles that guarantees all possible solutions for a given valid Sudoku instance.

For a valid Sudoku instance, it is possible to generate more than one solution, and because of this, it is very popular in the field of security. Again, in the case of 3D Sudoku, it is observed that a more significant number of possible solutions are usually obtained compared to 2D Sudoku. For this reason, the application of 3D Sudoku in the field of security has the

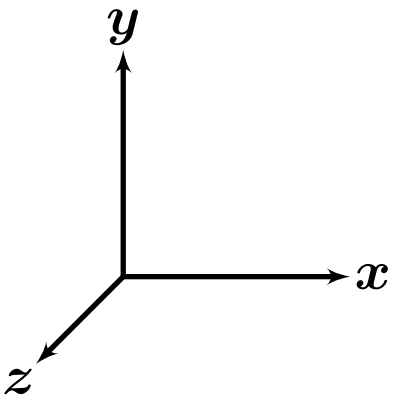


FIGURE 3. Alignment of the *x*-, *y*-, and *z*-planes [2].

potential of adding extra weight. There are no such rules that a Sudoku instance has to have only one solution [7].

The ASP-completeness (another solution possible completeness) of the Sudoku problem has been proved by some researchers [8]. According to the authors [9], a minimum of 17 clues is needed to obtain at least one solution for a particular 2D Sudoku instance. However, it is possible to obtain more than one solution when 17 or more clues are given. Different metaheuristic [10] techniques are there to solve Sudoku. In 2009, a new approach towards solving Sudoku puzzles was presented by exploring the idea of using an improved artificial bee colony (ABC) algorithm [11] which produced efficient results.

Methods based on soft computing have been developed to solve computational problems, leading to design solutions for Sudoku puzzles. Weyland [12] presented a critical analysis of the harmony search algorithm for Sudoku. Although the Sudoku solving capability was enhanced with the help of the harmony search algorithm, it had the tendency of getting trapped in local minima. Singh and Deep [13] presented a membrane algorithm using particle swarm optimisation rules for solving Sudoku puzzles. The algorithm was attached with mutation operators in cell-like P-systems. The search space was identified and used for solving the Sudoku problem.

Assad and Deep [14] proposed a hybrid method that combined the harmony search and hill climbing algorithms. The authors further modified their harmony search hill climber (HSHC) to create three different versions of the HSHC. The main aim of the authors was to improve HSHC. The authors [14] claimed that one of the modified HSHCs performed better than the other two modified versions, the standard HSHC and genetic algorithm (GA). The performance was comparable to a hybrid AC3-tabu search algorithm [15].

GA tends to converge prematurely at local optima [16]. Thus, Jana et al. [17] developed a new hybrid technique where the firefly mating algorithm was embedded with GA. The hybridisation was done primarily to control the premature convergence at local optima. The proposed method required a lesser population and a lower number of generations. The application of 2D Sudoku instances with single or multiple solutions can be observed in many fields, such as artificial intelligence, encrypting biometric template [18], OTP (one time password) generation [19], image-video encryption [20], SMS encryption [21], steganography [5], [22], [23], digital watermarking [24], DNA computing [25], visual cryptography [26], image authentication [23], and in many other areas. Extending the 2D Sudoku structure with one more dimension can lead it to a superior application level.

Consequently, researchers are trying to explore Sudoku by considering its extra dimension. However, no such exploration has yet been observed as the third dimension makes Sudoku more complex and trickier. Motivated by this research gap, we attempt to develop a 3D Sudoku solver in this work. We are sure that applying it in different fields, where more data is required, can give outstanding performance compared to 2D Sudoku.

D. MATHEMATICAL CHARACTERISATION OF 3D SUDOKU SOLVING PROBLEM

A Sudoku puzzle is solved based on logical constraints, and mathematical operations or functions are not directly involved in the solution. Still, it poses a variety of interesting mathematical problems. To represent this puzzle using a mathematical concept, researchers have reinterpreted it as the well-known vertex colouring problem [27] of graph theory [28], [29]. The vertex colouring or graph colouring problem has been revisited with a new perspective from the point-of-view of solving a Sudoku puzzle [30]. However, all of these reinterpretations using mathematical concepts have been carried out for 2D Sudoku puzzles. In a similar way, we reinterpret the problem of 3D Sudoku in the mathematical context of graph colouring in this work.

A $9 \times 9 \times 9$.3D Sudoku puzzle can be represented as a graph colouring problem. The goal is to build a 9-colouring of a specific graph, with 9 specific given colours. The graph corresponding to a $9 \times 9 \times 9$.3D Sudoku has 729 vertices, each representing one unique cell. All the vertices are labelled using an ordered triplet (x, y, z), where x, y, and z are integers within $[1, 9]$. Here, x, y, and z represent the index numbers of a 3D puzzle in the x-, y-, and z-dimensions, respectively. In the graph, two distinct vertices having labels $(x1, y1, z1)$ and $(x2, y2, z2)$ are connected with an edge, if and only if:

- $x1 = x2$, i.e., same row in the x-dimension, or
- $y1 = y2$, i.e., same row in the y-dimension, or
- $z1 = z2$, i.e., same row in the z-dimension, or
- $\lceil \frac{x1}{3} \rceil = \lceil \frac{x2}{3} \rceil$ and $\lceil \frac{y1}{3} \rceil = \lceil \frac{y2}{3} \rceil$, i.e., same 3×3 minigrid in dimension 1, or
- $\lceil \frac{x1}{3} \rceil = \lceil \frac{x2}{3} \rceil$ and $\lceil \frac{z1}{3} \rceil = \lceil \frac{z2}{3} \rceil$, i.e., same 3×3 minigrid in dimension 2, or
- $\lceil \frac{y1}{3} \rceil = \lceil \frac{y2}{3} \rceil$ and $\lceil \frac{z1}{3} \rceil = \lceil \frac{z2}{3} \rceil$, i.e., same 3×3 minigrid in dimension 3.

A 3D Sudoku puzzle can be solved by assigning an integer within $[1, 9]$ in each vertex, maintaining the rule that the connected vertices do not have the same integer assigned to them.

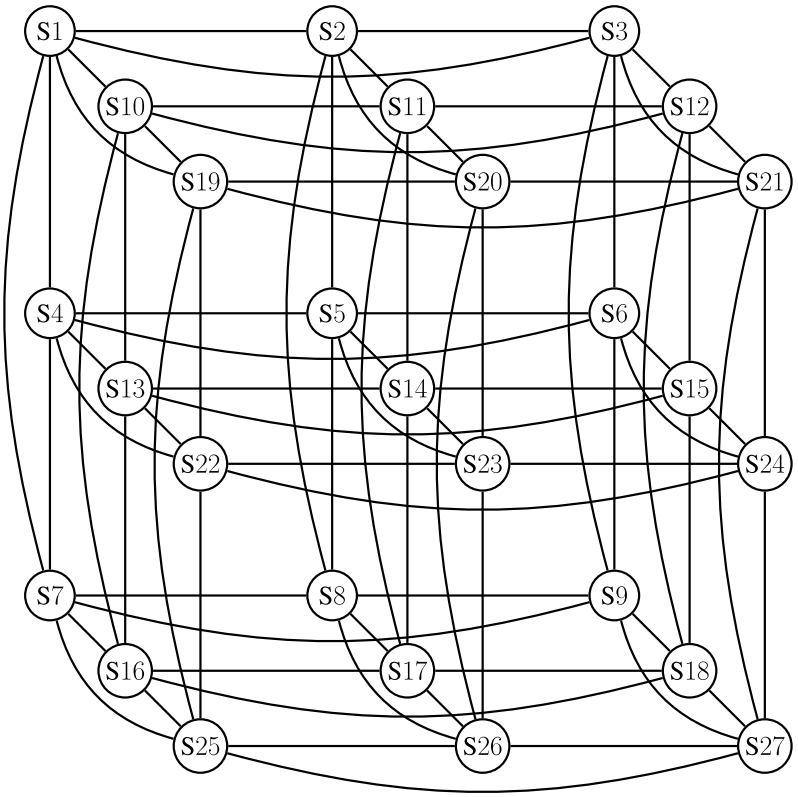


FIGURE 4. Connectivity graph of a 3D Sudoku of dimension 9 × 9×9 that represents relationships among the 27 mini-cubes, S1 through S27 [2].

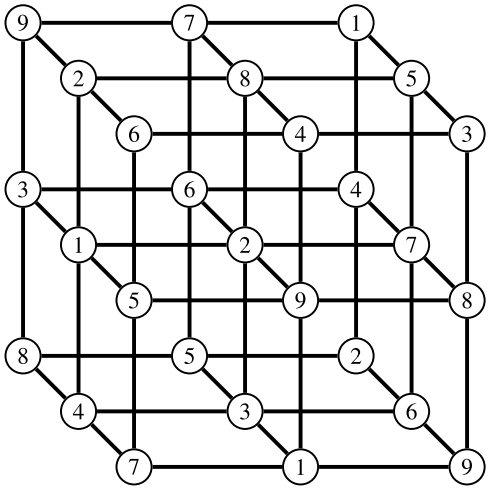


FIGURE 5. Mini-cube of position S1 (according to Fig. 3) of the Sudoku represented in Fig. 2 [2].

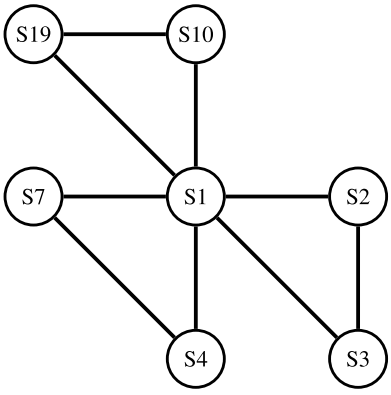


FIGURE 6. Connectivity graph of mini-cube S1 [2] (Fig. 4).

E. PHYSICAL STORAGE STRUCTURE OF THE THREE DIMENSIONAL SUDOKU

Table 1 represents the 3D array storage structure of the 3D Sudoku [2], shown in Fig. 2. This is a 2D visualisation of the 3D array. Each row of the table contains elements of a particular 2D Sudoku. For example, all the elements of grid 1 are represented by row 1, and for grid 2, all the elements are represented by row 2, and so on. Again, each row of each 2D Sudoku is stored in a single cell. Thus, each cell contains nine elements. For example, row 1 of grid 1 is stored in the first cell of row 1 of Table 1. Similarly, row 9 of grid 4 is stored in the ninth cell of row 4 in Table 1. In this way, all the nine 2D Sudoku maps lying in the xz-plane are stored in the individual rows of this table.

The nine elements in each cell represent the nine elements in the third dimension, (following the structure shown in Fig. 6). In this way, the elements of all 27 mini-cubes can be distributed in a 3D array. For example, the mini-cubes,

S1 through S27, are according to positions shown in Fig. 3. Six mini-cubes can be identified in Table 1 with the colour reference of Fig. 7.

III. PROPOSED ALGORITHM VERSION 1

We have improved the version of the algorithm developed by Jana et al. [2], using some functional constraints, and designed the improved version. In the proposed version of the algorithm, a 3D instance [2] is considered as the input stored in a 3D array. After storing the input, each blank cell is visited one by one, and all probable values for that particular cell are computed. Finally, the possible values of all empty cells are stored in the matrix, $\mathcal{P}$, which has been further explained in detail in the next section with the help of Algorithm 1.

A. PHYSICAL STORAGE STRUCTURE OF THE PROBABLE VALUES OF BLANK CELLS

In $\mathcal{P}$, each row contains the probable values for an individual blank cell. The first three columns indicate the x-, y-, and z-index values (in italics). The following nine columns store the nine possible values. Finally, the last column holds the number of probable values (in bold). Here, one thing that needs to be noted is that a blank cell may have less than nine possible values. In that case, the probable values are stored starting from the fourth column, and after all potential values have been stored, the remaining columns are filled with zeroes.

Let us consider the input puzzle in Table 2 and the matrix shown in Fig. 8(a). There are 16 blank cells in the input puzzle. Thus, there are 16 rows in the matrix. Each row refers to one individual blank cell. The first blank cell is indexed [0,3,2], i.e., row 0, column 3, position 2 in the cell [0,3]. For this particular blank cell, there are three probable values, 4, 8, and 9. The count, i.e., 3 (three), is stored in the last column.

In this way, all probable values are stored for every blank cell. Subsequently, the matrix data is sorted according to the count of probable values. This is done because when the algorithm begins implementing the ultimate tree structure using backtracking, it considers the blank cells stored in the proposed matrix, $\mathcal{P}$, row-by-row. According to the rule of minimum remaining value, the blank cells with the minimum number of probable values must be considered every time, as it helps in reducing the extent of backtracking. The tree structure usually contains a small number of branches at the upper level while a higher number at the lower level. Hence, in this way, backtracking is reduced.

Figure 8b represents the matrix, $\mathcal{P}$, after sorting it according to the count of probable values. After this, the five functional constraints, namely, naked single, hidden single, lone ranger, twin, and triplet, are applied, i.e., every blank cell is considered row-by-row from the generated matrix, $\mathcal{P}$, and the constraints applied. After using the five methods, the number of probable values is reduced for some blank cells. The updated matrix, $\mathcal{P}$, is shown in Fig. 9.

A backtracking tree is generated based on the reduced number of values, as shown in Fig. 12. Two solutions are highlighted in yellow in Fig. 12. Comparing the linked list structure given in Fig. 12 with the one shown in Fig. 11 (highlighted in blue), the reduction in the number of branches can be observed. This reduction has been achieved by implementing the five functional constraints, i.e., naked single, hidden single, lone ranger, twin, and triplet. The result is the same in both the cases.

We can observe that each node containing each possible value for each of the blank cell is connected by a right-directed arrow to the first possible value of the next blank cell. Now, this first possible value may or may not contribute to a solution path. For example, the first possible value, 3, of the blank cell, 051, does not contribute to a solution path. Now, the question may arise that if the second possible value, 8, of cell 050 is not directly connected by a link to the next contributing possible value, 4, of the node, 051, how can we traverse the solution path every time needed, after identifying the path? This is the reason why we store the different solution paths in the path-matrix, at the time they are being identified one by one.

However, as the blank cells are visited in a different order in the work done by Jana et al. [2] compared to the proposed versions of the algorithm, the solutions in the figures may look different. Both solutions are presented in Table 3 with the index values mentioned to clear this confusion. Table 3 provides a comparative view of the proposed algorithm (Version 1) and the one developed by Jana et al. [2].

B. APPLIED FUNCTIONAL CONSTRAINTS

In the first version of our proposed algorithm (i.e., Version 1), some functional constraints [31] are applied: naked single, hidden single, lone ranger, twin, and triplet. These constraints are explained below.

1) NAKED SINGLE AND HIDDEN SINGLE

Sometimes, when the probable values for each blank cell are computed, only one value is obtained for a specific empty cell. After placing that value in that blank cell permanently, other blank cells may remain in the same row in any three dimensions. Any such cell may hold the same value as its probable values. Thus, the particular value can be eliminated from the list of possible values for that blank cell. This condition is known as naked single. After this elimination, there may be only one probable value left in an empty cell. That value is known as a hidden single. For example, in Fig. 13, in the top row, cell 3 has two probable values, among which 4 is a hidden single because it is not present in any one of the x-, y-, or z-directions in the 3D Sudoku.

2) LONE RANGER

Sometimes, a particular value is present as the probable value for only one blank cell in an entire row in any of the three dimensions. This situation is known as a lone ranger. The other potential values can be eliminated from that particular cell. For example, in Fig. 14, 4 is present only in cell 3 as

Algorithm 1 Pseudocode of Algorithm Version 1

Input: $\mathcal{M}$: input 3D Sudoku matrix of dimension $9 \times 9 \times 9$
Output: $\mathcal{S}$: Set of solutions

Initialise $l = 1$
Initialise a matrix $\mathcal{P} = \left[p_{q,r}\right]_{l \times 13}$ to $\emptyset$ for storing all probable values pertaining to each blank cell $m_{i,j,k}$
for $i = 1$ to 9 **do** ▷ loop for the x-dimension
  for $j = 1$ to 9 **do** ▷ loop for the y-dimension
    for $k = 1$ to 9 **do** ▷ loop for the z-dimension
      if $m_{i,j,k} = 0$ **then**
        Assign $\left[p_{l,1}, p_{l,2}, p_{l,3}\right] \leftarrow [i, j, k]$
        Compute all probable values for $m_{i,j,k}$ and store them in $\left[p_{l,4}, p_{l,5}, p_{l,6}, \ldots, p_{l,12}\right]$
        Store the count of probable values for $m_{i,j,k}$ in $p_{l,13}$
        Increment l by 1
      end if
    end for
  end for
end for
Sort $\mathcal{P}$ in ascending order of $\left[p_{i,13}\right]$ $\forall i$, where $i = 1, 2, \ldots, l$
Implement Naked Single, Hidden Single, Lone Ranger, Twin, and Triplet on $\mathcal{P}$
Sort $\mathcal{P}$ in ascending order of $\left[p_{i,13}\right]$ $\forall i$, where $i = 1, 2, \ldots, l$
Create a new node that will act as the **`head`** of the search tree according to the structure defined in Fig. 10.
Initialise all values and pointers of the **`head`** to $\emptyset$
Create a new node with all empty fields and assign a pointer **`temp`** to point to that node
for $i = 1$ to l **do**
  for $j = 3$ to 11 **do**
    if $p_{i,j} \neq 0$ **then**
      $a \leftarrow p_{i,1}$, $b \leftarrow p_{i,2}$, and $c \leftarrow p_{i,3}$
      $id \leftarrow (a \times 100) + (b \times 10) + c$
      Create a new node
      Store $p_{i,j}$ to the VAL field of the node
      Store id to the INDEX field of the node
      Initialise all pointers of the node to $\emptyset$
      if DOWN pointer of **`head`** points to $\emptyset$ **then**
        Link node to DOWN pointer of **`head`**
      else
        Initialise a temporary 3D array $\mathcal{T}\_\mathcal{M}$ to $\mathcal{M}$
      end if
      for each branch in the search tree **do** ▷ traversal by backtracking
        if leaf node of the branch points to $\emptyset$ by NEXT pointer **then** ▷ i.e., open path
          for each node **`N`** in the branch **do**
            $x \leftarrow$ 1st digit of INDEX of **`N`**
            $y \leftarrow$ 2nd digit of INDEX of **`N`**
            $z \leftarrow$ 3rd digit of INDEX of **`N`**
            Place VAL of **`N`** to $t\_m_{x,y,z}$
          end for
          $t\_m_{a,b,c} \leftarrow p_{i,j}$
          if 3D Sudoku constraints are satisfied on $\mathcal{T}\_\mathcal{M}$ **then**
            if INDEX of the leaf node is id **then**
              Traverse by DOWN pointer of the leaf node that points to $\emptyset$
              Link node to that DOWN pointer pointing to $\emptyset$
            else
              Link node to the NEXT pointer of the leaf node
            end if
          else
            if INDEX of the leaf node $\neq id$ **then** ▷ i.e., this probable value leads to an END
              Create a new node **`N_end`**
              Store END to INDEX of **`N_end`** and assign all other fields to $\emptyset$
              link **`N_end`** to the NEXT pointer of the leaf node
            end if
          end if
        end if
      end for
    end if
  end for
end for

Red-coloured elements are from mini-cube S1
Green-coloured elements are from mini-cube S2
Blue-coloured elements are from mini-cube S3
Purple-coloured elements are from mini-cube S4
Orange-coloured elements are from mini-cube S10
Pink-coloured elements are from mini-cube S27

FIGURE 7. Identification of mini-cubes of Table 1.

one of its probable values. Thus, 4 can be placed in cell 3, and 5 can be eliminated from its list of possible values. This concept also holds in any of the x-, y-, and z-directions in a 3D Sudoku.

3) TWIN

In a row, in any of the three dimensions, two blank cells may hold a pair of probable values, say, $v1$ and $v2$ only. Hence, it is evident that $v1$ and $v2$ can only be placed in these two

TABLE 1. Storage structure representation of 3D Sudoku instance stored in 3D array.

971385642	285647931	643921785	769153428	152478369	438269157	897536214	526714893	314892576
364192578	197538264	528764193	843926715	976315842	215847936	431289657	789653421	652471389
852476319	436219857	719853426	521784693	384692571	697531284	265147938	143928765	978365142
583624791	674391582	291587634	135248967	748963125	962175348	356412879	417839256	829756413
746913825	912875346	835246917	487639251	629751483	351482679	174398562	298567134	563124798
129758463	358462179	467139258	296517834	513824796	874396512	982675341	635241987	741983625
215847936	843926715	976315842	652471389	431289657	789653421	528764193	364192578	197538264
697531284	521784693	384692571	978365142	265147938	143928765	719853426	852476319	436219857
438269157	769153428	152478369	314892576	897536214	526714893	643921785	971385642	285647931

TABLE 2. Input 3D Sudoku stored in a three-dimensional array. The *x*-, *y*-, and *z*-dimensions are represented by each row, column, and cell of this table. Each cell contains values 1 to 9, and 0 (zero) represents a missing value or blank cell [2]. The *i*th element of each column and row of the table contains nine elements i.e., 1 to 9 without any repetition. Hence, it is a 2D visualization of the 3D storage structure.

971385642	285647931	643921785	760153020	102478369	000269157	897536214	526714893	314892576
364192578	197538264	528764193	000926710	976315842	210847906	431289657	789653421	652471389
852476319	436219857	719853426	021784690	380692571	697531284	265147938	143928765	978365142
583624791	674391582	291587634	135248967	748963125	962175348	356412879	417839256	829756413
746913825	912875346	835246917	487639251	629751483	351482679	174398562	298567134	563124798
129758463	358462179	467139258	296517834	513824796	874396512	982675341	635241987	741983625
215847936	843926715	976315842	652471389	431289657	789653421	528764193	364192578	197538264
697531284	521784693	384692571	978365142	265147938	143928765	719853426	852476319	436219857
438269157	769153428	152478369	314892576	897536214	526714893	643921785	971385642	285647931

0	**3**	**2**	4	8	9	0	0	0	0	0	0	**3**
0	**3**	**6**	4	0	0	0	0	0	0	0	0	**1**
0	**3**	**8**	8	0	0	0	0	0	0	0	0	**1**
0	**4**	**1**	5	0	0	0	0	0	0	0	0	**1**
0	**5**	**0**	4	8	0	0	0	0	0	0	0	**2**
0	**5**	**1**	3	4	0	0	0	0	0	0	0	**2**
0	**5**	**2**	3	4	8	0	0	0	0	0	0	**3**
1	**3**	**0**	4	5	2	0	0	0	0	0	0	**3**
1	**3**	**1**	3	4	5	0	0	0	0	0	0	**3**
1	**3**	**2**	3	4	5	8	0	0	0	0	0	**4**
1	**3**	**8**	3	5	0	0	0	0	0	0	0	**2**
1	**5**	**2**	3	5	0	0	0	0	0	0	0	**2**
1	**5**	**7**	3	0	0	0	0	0	0	0	0	**1**
2	**3**	**0**	5	0	0	0	0	0	0	0	0	**1**
2	**3**	**8**	3	0	0	0	0	0	0	0	0	**1**
2	**4**	**2**	4	0	0	0	0	0	0	0	0	**1**

(a)

0	**3**	**6**	4	0	0	0	0	0	0	0	0	**1**
0	**3**	**8**	8	0	0	0	0	0	0	0	0	**1**
0	**4**	**1**	5	0	0	0	0	0	0	0	0	**1**
1	**5**	**7**	3	0	0	0	0	0	0	0	0	**1**
2	**3**	**0**	5	0	0	0	0	0	0	0	0	**1**
2	**3**	**8**	3	0	0	0	0	0	0	0	0	**1**
2	**4**	**2**	4	0	0	0	0	0	0	0	0	**1**
0	**5**	**0**	4	8	0	0	0	0	0	0	0	**2**
0	**5**	**1**	3	4	0	0	0	0	0	0	0	**2**
1	**3**	**8**	3	5	0	0	0	0	0	0	0	**2**
1	**5**	**2**	3	5	0	0	0	0	0	0	0	**2**
0	**3**	**2**	4	8	9	0	0	0	0	0	0	**3**
0	**5**	**2**	3	4	8	0	0	0	0	0	0	**3**
1	**3**	**0**	4	5	8	0	0	0	0	0	0	**3**
1	**3**	**1**	3	4	5	0	0	0	0	0	0	**3**
1	**3**	**2**	3	4	5	8	0	0	0	0	0	**4**

(b)

FIGURE 8. (a) Initial $\mathcal{P}$ for the puzzle shown in Table 1. (b) $\mathcal{P}$ for the puzzle shown in Table 2 after sorting.

cells and eliminated from all other cells in the related row. This occurrence is known as a twin. For example, in Fig. 15, cells 2 and 6 contain the pair of values (2,3) as their only probable values. Thus, it is evident that one of these cells will have 2 while the other will contain 3. Hence, 2 and 3 can be eliminated from the probable values of cells 5 and 8. This example row can be represented in any of the *x*-, *y*-, or *z*-directions in a 3D Sudoku.

4) TRIPLET

Similar to the occurrence of a twin, in a row in any of the three dimensions, three blank cells can hold a triplet of values, say,

0	**3**	**6**	4	0	0	0	0	0	0	0	0	**1**
0	**3**	**8**	8	0	0	0	0	0	0	0	0	**1**
0	**4**	**1**	5	0	0	0	0	0	0	0	0	**1**
1	**5**	**7**	3	0	0	0	0	0	0	0	0	**1**
2	**3**	**0**	5	0	0	0	0	0	0	0	0	**1**
2	**3**	**8**	3	0	0	0	0	0	0	0	0	**1**
2	**4**	**2**	4	0	0	0	0	0	0	0	0	**1**
1	**3**	**8**	5	0	0	0	0	0	0	0	0	**1**
1	**5**	**2**	5	0	0	0	0	0	0	0	0	**1**
0	**3**	**2**	9	0	0	0	0	0	0	0	0	**1**
0	**5**	**0**	4	8	0	0	0	0	0	0	0	**2**
0	**5**	**1**	3	4	0	0	0	0	0	0	0	**2**
0	**5**	**2**	3	8	0	0	0	0	0	0	0	**2**
1	**3**	**0**	4	8	0	0	0	0	0	0	0	**2**
1	**3**	**1**	3	4	0	0	0	0	0	0	0	**2**
1	**3**	**2**	3	8	0	0	0	0	0	0	0	**2**

FIGURE 9. Final $\mathcal{P}$ for the puzzle shown in Table 2 after applying functional constraints.

TABLE 3. The same two solutions obtained using algorithm [2] and our proposed algorithm (Algorithm 1) for the puzzle shown in Table 2.

Index	Solution 1	Solution 2
0 3 6	4	4
0 3 8	8	8
0 4 1	5	5
1 5 7	3	3
2 3 0	5	5
2 3 8	3	3
2 4 2	4	4
1 3 8	5	5
1 5 2	5	5
0 3 2	9	9
0 5 0	4	8
0 5 1	3	4
0 5 2	8	3
1 3 0	8	4
1 3 1	4	3
1 3 2	3	8

VAL	INDEX	PREV	NEXT	DOWN	UP

VAL	:	the probable value
INDEX	:	the x, y, z indices of the 3D Sudoku
PREV	:	points to previous node (linked to left)
NEXT	:	points to next node (linked to right)
DOWN	:	points to the down node
UP	:	points to the up node

FIGURE 10. Representing node fields of Algorithm 1.

$v1$, $v2$, and $v3$ as their probable values only. Therefore, $v1$, $v2$, and $v3$ can be placed in these three cells only and can be eliminated from all other cells in the related row. This occurrence is known as a triplet. For example, in Fig. 16, the three cells 2, 5, and 6 contain the triplet (2,3,6) as their only probable values. Therefore, 6 can be eliminated from the list of possible values in cell 8. This example row can be implemented in any of the x-, y-, and z-directions in a 3D Sudoku.

IV. PROPOSED ALGORITHM VERSION 2

A naive algorithm was proposed by Jana et al. [2] for solving 3D Sudoku using backtracking. We propose an improved version of the same (Version 1 explained in the previous sections). A second version (Version 2) is introduced in this section to solve 3D Sudoku puzzles, essentially using a bottom-up approach. In this case, the inputs are the nine individual 2D Sudoku puzzles lying in the xz-plane (as shown in Fig. 5), one above the other to form a 3D Sudoku puzzle. Each of these 2D Sudoku puzzles is stored separately in nine matrices. The term bottom-up refers to the fact that, at first, all possible solutions are obtained for an individual 2D Sudoku problem. Then, all possible combinations of these solutions are analysed by considering one solution at a time for each 2D Sudoku. This is done to check which combination of solutions works for the complete 3D Sudoku problem, maintaining all the necessary constraints. Finally, backtracking is used to obtain the solutions to the individual 2D Sudoku puzzles.

A. PHYSICAL STORAGE STRUCTURE OF INPUT AND INTERMEDIATE STRUCTURES

Simple 9×9 matrices have been utilised in this work to store 2D Sudoku puzzles as input (Fig. 16). To solve one 2D Sudoku, all probable values for each blank cell are computed. These computed values are stored in the corresponding matrix, $\mathcal{P}$ (Fig. 8a). Thus, nine $\mathcal{P}$ structures are generated, which hold all probable values of all blank cells for a 2D Sudoku problem. Next, empty cells in these matrices are sorted (in non-descending order) according to the number of possible values (Fig. 8b).

Finally, solutions are obtained by applying the backtracking method. After backtracking, nine linked lists (one for each 2D Sudoku) are obtained, similar to Fig. 11, and the corresponding linked list is considered. All solutions are fetched from the linked list and stored in a matrix, where each row stores a new solution. The number of columns is the same as the number of blank cells. The first row stores the indices of empty cells. From the second row onwards, accepted values of the blank cells are stored for each new solution. Nine matrices hold the solutions to the nine 2D Sudoku puzzles (Fig. 18). The structure of the solution matrix for each 2D Sudoku is shown in Fig. 17. Finally, each possible combination of solutions from these nine matrices is considered (as shown in Fig. 19) and checked for necessary 3D constraints.

In other words, all the accepted values are placed in their respective blank cells. Then, for each established value, it is checked whether distinct values from 1 through 9 are present in the corresponding rows, columns, and minigrids in all three dimensions. Since no 3D structure is used, a question may arise about how the rules for the third dimension can be

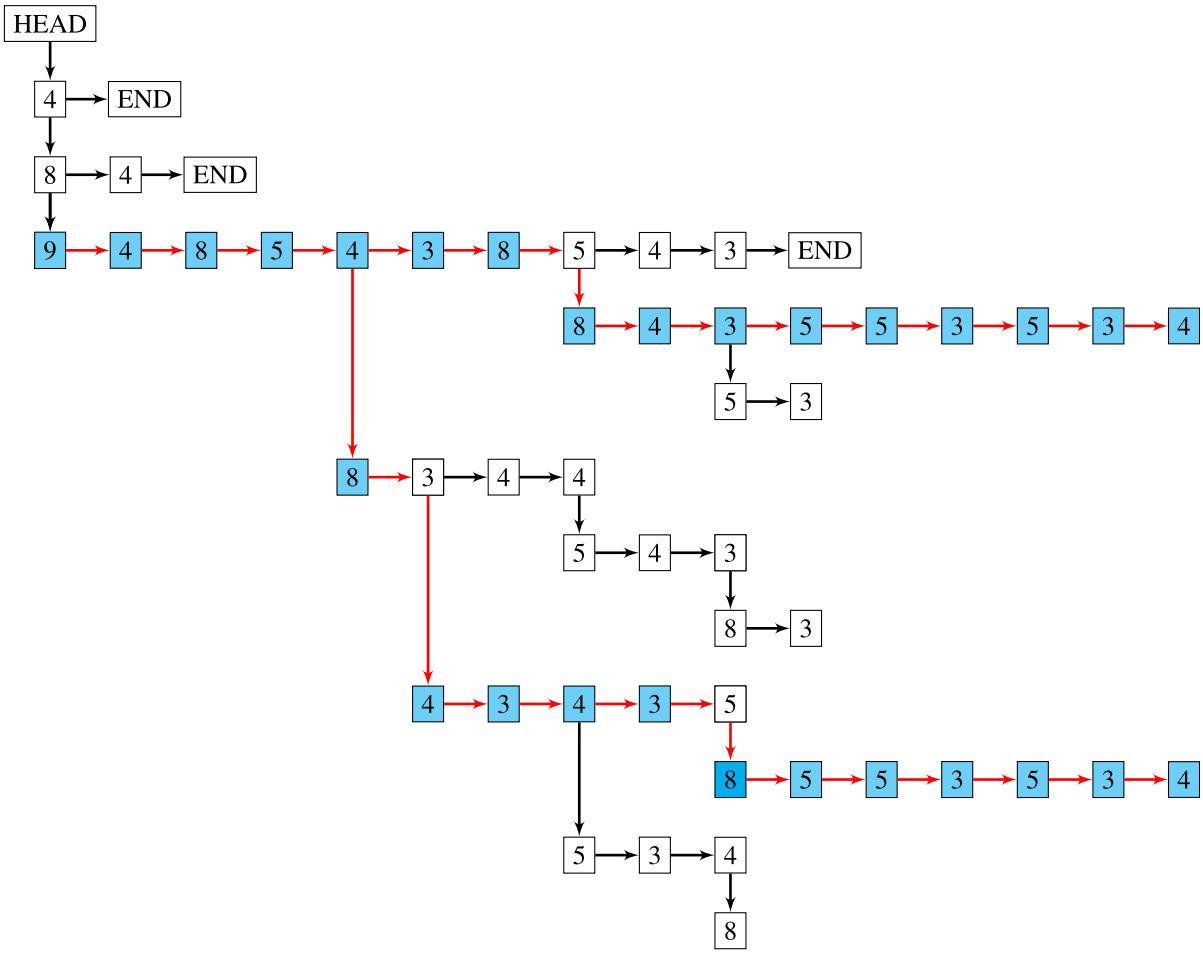


FIGURE 11. The linked list representation to store solution path for the given puzzle in Table 2.

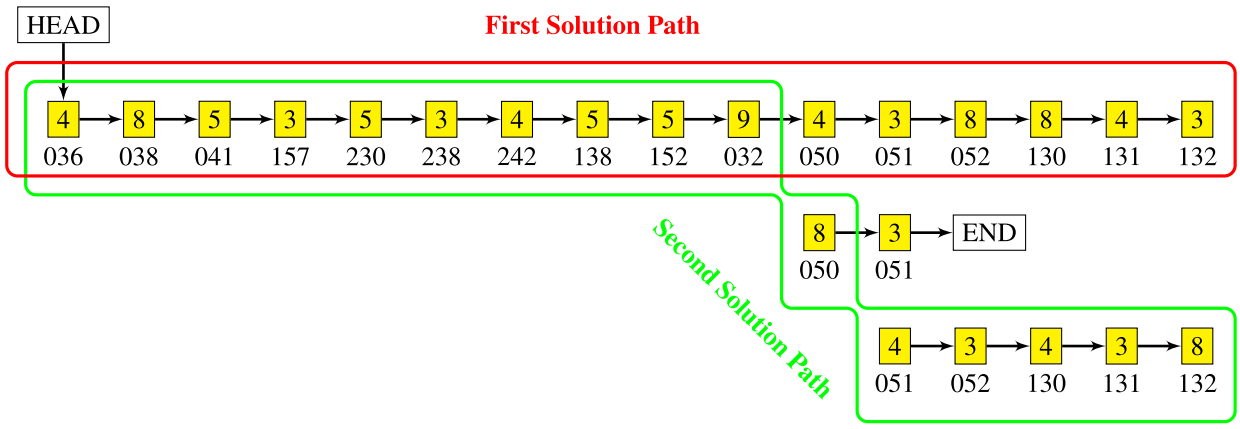


FIGURE 12. The linked list representation to store the solutions for the Sudoku instance is shown in Table 2.

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

FIGURE 13. Example of Naked single and Hidden Single.

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

FIGURE 14. Example of Long Ranger.

checked. This is possible as the relation among all the 2D Sudoku puzzles, i.e., how the 2D Sudoku puzzles lie on one another, is known.

For example, if 7 is placed in the blank cell [2], [3] in Sudoku 4, all the values in the cell [2], [3] in all the other eight Sudoku instances are checked, and it is made sure that

Algorithm 2 Pseudocode of Algorithm Version 2

Input: $\mathcal{M}$: input 3D Sudoku matrix of dimension $9 \times 9 \times 9$
Output: $\mathcal{S}$: Set of solutions

Extract nine 2D Sudoku puzzles along any one dimension from $\mathcal{M}$ and store each of them in $\mathcal{SP}^i = [sp^i_{j,k}]_{9\times 9}$ where $i = 1, 2, \ldots, 9$
Initialise $l = 1$
Initialise a matrix $\mathcal{P}^i = \left[p^i_{q,r}\right]_{l\times 12}$ to $\emptyset$ for storing all probable values pertaining to each $\mathcal{SP}^i$
Initialise individual $\mathcal{SM}^i$ for each puzzle $\mathcal{SP}^i$, as represented in Table. 4
for i =1 to 9 **do** ▷ each 2D matrix / Sudoku puzzle
  for j =1 to 9 **do**
    for k =1 to 9 **do**
      if $sp^i_{j,k} = 0$ **then**
        Assign $\left[p^i_{l,1}, p^i_{l,2}\right] \leftarrow [j, k]$
        Compute all probable values for $sp^i_{j,k}$ and store them in $\left[p^i_{l,3}, p^i_{l,4}, p^i_{l,5}, \ldots, p^i_{l,11}\right]$
        Store the count of probable values for $sp^i_{j,k}$ in $p^i_{l,12}$
        Increment l by 1
      end if
    end for
  end for
  Sort $\mathcal{P}^i$ in ascending order of $\left[p^i_{j,12}\right]$ $\forall j$, where $j = 1, 2, \ldots, l$
end for
for i =1 to 9 **do**
  Compute all possible solutions of $\mathcal{SP}^i$ using the corresponding $\mathcal{P}^i$ by backtracking
  Store all solutions in $\mathcal{SM}^i$
end for
for each solution in $\mathcal{SM}^1$ **do** ▷ each row represents a solution, except the 0th row
  Initialise a new matrix $\mathcal{T}\_\mathcal{M}^1 = \mathcal{SP}^1$
  Fill each blank cell of $\mathcal{T}\_\mathcal{M}^1$ with the current solution
  for each solution in $\mathcal{SM}^2$ **do**
    Initialise a new matrix $\mathcal{T}\_\mathcal{M}^2 = \mathcal{SP}^2$
    Fill each blank cell of $\mathcal{T}\_\mathcal{M}^2$ with the current solution
    for each solution in $\mathcal{SM}^3$ **do**
      Initialise a new matrix $\mathcal{T}\_\mathcal{M}^3 = \mathcal{SP}^3$
      Fill each blank cell of $\mathcal{T}\_\mathcal{M}^3$ with the current solution
      for each solution in $\mathcal{SM}^4$ **do**
        Initialise a new matrix $\mathcal{T}\_\mathcal{M}^4 = \mathcal{SP}^4$
        Fill each blank cell of $\mathcal{T}\_\mathcal{M}^4$ with the current solution
        for each solution in $\mathcal{SM}^5$ **do**
          Initialise a new matrix $\mathcal{T}\_\mathcal{M}^5 = \mathcal{SP}^5$
          Fill each blank cell of $\mathcal{T}\_\mathcal{M}^5$ with the current solution
          for each solution in $\mathcal{SM}^6$ **do**
            Initialise a new matrix $\mathcal{T}\_\mathcal{M}^6 = \mathcal{SP}^6$
            Fill each blank cell of $\mathcal{T}\_\mathcal{M}^6$ with the current solution
            for each solution in $\mathcal{SM}^7$ **do**
              Initialise a new matrix $\mathcal{T}\_\mathcal{M}^7 = \mathcal{SP}^7$
              Fill each blank cell of $\mathcal{T}\_\mathcal{M}^7$ with the current solution
              for each solution in $\mathcal{SM}^8$ **do**
                Initialise a new matrix $\mathcal{T}\_\mathcal{M}^8 = \mathcal{SP}^8$
                Fill each blank cell of $\mathcal{T}\_\mathcal{M}^8$ with the current solution
                for each solution in $\mathcal{SM}^9$ **do**
                  Initialise a new matrix $\mathcal{T}\_\mathcal{M}^9 = \mathcal{SP}^9$
                  Fill each blank cell of $\mathcal{T}\_\mathcal{M}^9$ with the current solution
                  if $\mathcal{T}\_\mathcal{M}^1$ through $\mathcal{T}\_\mathcal{M}^9$ validate 3D Sudoku rule **then**
                    Store $\mathcal{T}\_\mathcal{M}^1$ through $\mathcal{T}\_\mathcal{M}^1$ to $\mathcal{S}$ as a new 3D solution
                  end if
                end for
              end for
            end for
          end for
        end for
      end for
    end for
  end for
end for

7 is not present in any other cell. Cell [2], [3] is present in the first column and the last row of the second minigrid of Sudoku 4. Therefore, the first columns and last rows of the respective second minigrids of Sudokus 5 and 6 also need to be checked as these groups of three sub-rows and three sub-columns form minigrids in the other two dimensions. In this way, at least one solution can be obtained for any valid 3D puzzle using our proposed algorithm Version 2.

V. EXPERIMENTAL RESULTS

In this section, experimental results are presented. The performance of the proposed algorithms is compared based on Sudoku instances available at http://www.menneske.no/Sudoku3d. The comparison is made with respect to the time required and the number of blank cells in the input puzzle. The results are presented in Tables 5 through 7. To the best of our knowledge, no such work has yet been implemented in 3D Sudoku. The current research works focus on 2D Sudoku only. Consequently, the results obtained by our proposed methodology could not be compared with other such techniques.

Each row of the table indicates an experiment with a specific instance of 3D Sudoku and a certain number of blank cells, which are fed as inputs to the three algorithms. For some instances (easy), we obtain output (s) for all three algorithms, and for some cases (moderate and hard), we do not. This fact indicates the improvement in the algorithms, as represented in Fig. 21.

TABLE 4. Structure of the matrix that stores all the possible solutions of a 2D Sudoku having *n* blank cells and *m* solutions.

	Index of blank cell 1	Index of blank cell 2	Index of blank cell 3	.....	Index of blank cell n
solution 1	value	value	value	.....	value
solution 2	value	value	value	.....	value
solution 3	value	value	value	.....	value
.	.	.	.	.....	.
.	.	.	.	.....	.
.	.	.	.	.....	.
solution n	value	value	value	.....	value

TABLE 5. Comparative experimental results analysis for 3D Sudoku instances between our proposed methods and Jana et al. [2] for *easy* puzzles.

Instance #	# Blank Cells	Level of Difficulty	Jana et al. [2]		Algorithm 1		Algorithm 2	
			Time (seconds)	# Solution(s)	Time (seconds)	# Solution(s)	Time in seconds	# Solution(s)
1	180	Easy	1.74	1	1.62	1	1.40	1
2	185	Easy	1.82	1	1.71	1	1.44	1
3	190	Easy	2.40	1	1.86	1	1.82	1
4	198	Easy	1.37	1	1.02	1	3.82	1
5	202	Easy	2.32	1	1.42	1	1.07	1
6	207	Easy	1.64	1	1.52	1	1.20	1
7	216	Easy	1.40	1	1.32	1	1.08	1
8	225	Easy	1.87	1	1.42	1	1.06	1
9	234	Easy	1.52	1	1.07	1	1.05	1
10	239	Easy	1.82	1	1.64	1	1.53	1
11	243	Easy	1.55	1	1.02	1	1.22	1
12	248	Easy	1.72	1	1.36	1	1.02	1
13	252	Easy	1.46	1	1.27	1	2.87	1
14	257	Easy	1.44	1	1.25	1	1.16	1
15	261	Easy	1.98	1	1.06	1	1.52	1
16	270	Easy	2.70	1	2.10	1	1.52	1
17	274	Easy	1.86	1	1.47	1	1.16	1
18	279	Easy	1.92	1	2.74	1	1.20	1
19	284	Easy	2.20	1	1.14	1	1.34	1
20	288	Easy	1.62	1	1.18	1	2.98	1
21	294	Easy	2.22	1	1.87	1	1.17	1
22	299	Easy	3.10	1	2.46	1	1.24	1
23	306	Easy	4.45	1	1.83	1	3.09	1
24	315	Easy	1.51	1	1.03	1	2.17	1
25	320	Easy	2.45	1	1.87	1	1.18	1
26	324	Easy	2.62	1	2.20	1	2.74	1
27	333	Easy	2.74	2	2.22	2	2.35	2
28	342	Easy	2.74	2	2.45	2	2.01	2
29	351	Easy	2.69	2	2.61	2	2.98	2
30	360	Easy	2.51	4	2.11	4	3.48	4
31	369	Easy	8.84	8	2.41	8	3.55	8
32	387	Easy	19.99	1	12.18	1	3.39	1
33	390	Easy	20.25	1	13.16	1	4.47	1
34	395	Easy	19.10	1	13.51	1	4.96	1

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

FIGURE 15. Example of Twin.

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

FIGURE 16. Example of Triplet.

Another interesting fact is, the number of solutions for a single puzzle increases with an increase in the hardness of the puzzle. In Fig. 22, it can be observed that, for easy puzzles, maximum number of solutions obtained is 8, and for most (29) of the easy Sudoku puzzles, only one solution is obtained. In the case of moderate Sudoku puzzles, the number of solutions increases up to 64, although maximum number (14) of puzzles again provide a single solution, as represented in Fig. 23.

In Fig. 24, we can observe that number of solutions increases by a huge amount for the hard puzzles. Despite our best efforts, no 3D Sudoku solver algorithm could be found except [2]. There exist different algorithms for solving 2D Sudoku instances with varying difficulty levels. However, the instances they have used are not publicly available in many cases. Furthermore, even if they are available, we cannot feed 3D instances in infrastructure for solving 2D Sudoku puzzles. Hence, our comparison is limited to the three algorithms only.

TABLE 6. Comparative experimental results analysis for 3D Sudoku instances between our proposed methods and Jana et al. [2] for *moderate* puzzles. NW indicates Not Working.

Instance #	# Blank Cells	Level of Difficulty	Jana et al. [2]		Algorithm 1		Algorithm 2	
			Time (seconds)	# Solution(s)	Time (seconds)	# Solution(s)	Time in seconds	# Solution(s)
35	400	Moderate	NW		3.37	1	3.77	1
36	402	Moderate	NW		6.15	32	4.70	32
37	405	Moderate	140.51	1	4.13	1	2.81	1
38	405	Moderate	NW		4.55	16	4.11	16
39	407	Moderate	NW		4.86	1	4.20	1
40	410	Moderate	NW		4.48	1	3.25	1
41	413	Moderate	NW		4.78	1	3.50	1
42	415	Moderate	NW		4.36	1	3.27	1
43	417	Moderate	NW		4.17	1	3.12	1
44	420	Moderate	NW		4.81	1	3.56	1
45	423	Moderate	NW		5.61	1	3.45	1
46	425	Moderate	NW		5.52	1	3.92	1
47	426	Moderate	NW		4.89	1	2.75	1
48	429	Moderate	NW		10.72	1	3.84	1
49	432	Moderate	NW		5.28	2	3.10	2
50	435	Moderate	NW		6.25	2	3.92	2
51	437	Moderate	NW		17.06	64	4.41	64
52	441	Moderate	NW		6.84	32	4.11	32
53	445	Moderate	NW		14.33	1	3.69	1
54	448	Moderate	NW		16.33	1	3.56	1
55	450	Moderate	NW		7.30	2	2.45	2

TABLE 7. Comparative experimental results analysis for 3D Sudoku instances between our proposed methods and Jana et al. [2] for *hard* puzzles. NW indicates Not Working.

Instance #	# Blank Cells	Level of Difficulty	Jana et al. [2]		Algorithm 1		Algorithm 2	
			Time (seconds)	# Solution(s)	Time (seconds)	# Solution(s)	Time in seconds	# Solution(s)
56	500	Hard	NW		NW		4.08	6
57	524	Hard	NW		NW		7.26	8
58	524	Hard	NW		NW		5.29	2
59	525	Hard	NW		NW		4.70	2
60	527	Hard	NW		NW		5.69	2
61	527	Hard	NW		NW		6.62	2
62	528	Hard	NW		NW		8.85	12
63	529	Hard	NW		NW		142.13	720
64	529	Hard	NW		NW		6.95	16
65	531	Hard	NW		NW		2.40	3
66	531	Hard	NW		NW		7.58	4
67	532	Hard	NW		NW		4.36	3
68	533	Hard	NW		NW		5.71	1
69	533	Hard	NW		NW		6.72	1
70	533	Hard	NW		NW		160.40	768
71	534	Hard	NW		NW		6.62	1
72	536	Hard	NW		NW		8.87	3
73	537	Hard	NW		NW		7.48	15
74	540	Hard	NW		NW		7.71	3
75	542	Hard	NW		NW		8.26	3

Though the number of empty cells does not solely determine the difficulty level (the difficulty also depends on the positions of blank cells), we have divided the inputs into three clusters according to three difficulty levels based on the number of empty cells. The experimental results show that the algorithm proposed by Jana et al. [2] works only for easy instances. However, our proposed algorithm (Version 1) works for easy and moderate examples. The second version of our proposed algorithm (i.e., Algorithm 2) works for easy, moderate, and hard instances.

VI. COMPUTATIONAL COMPLEXITY

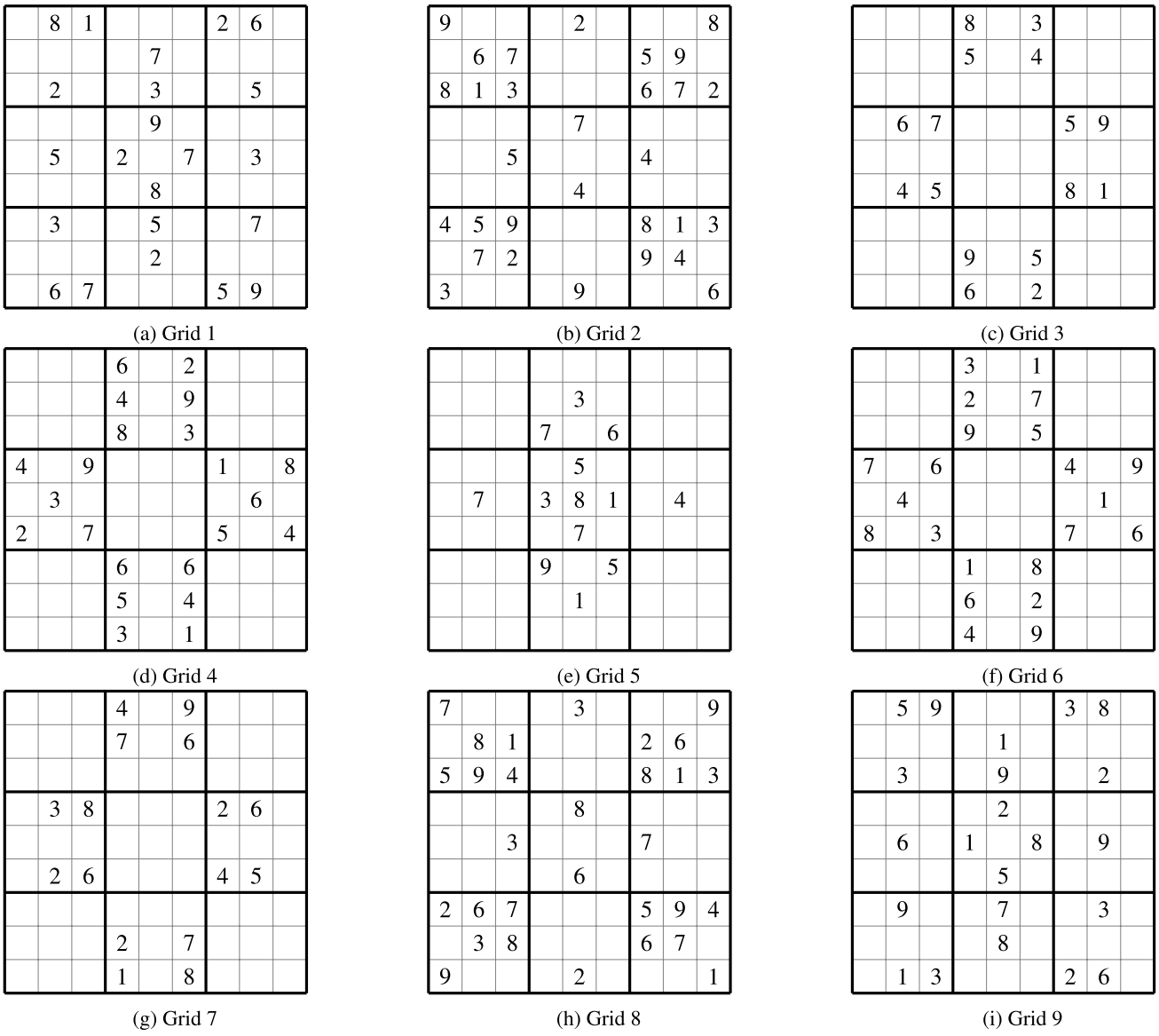
A full N-ary tree of height h contains

$$\frac{N^{h+1}-1}{N-1}$$

nodes. The number of branches is thus

$$\frac{N^{h+1}-1}{N-1}-1$$

In the case of Sudoku, the permutation tree contains h levels, where $h-1$ is the number of blank cells. The root node at



(a) Grid 1

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

(b) Grid 2

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

(c) Grid 3

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

(d) Grid 4

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

(e) Grid 5

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

(f) Grid 6

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

(g) Grid 7

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

(h) Grid 8

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

(i) Grid 9

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

FIGURE 17. 2D Sudoku layers that form the 3D puzzle fed as the input to Algorithm 2.

level 0 is the head node. Nodes present at level 1 denote the probable values of the first blank cell, nodes present at level 2 denote the possible values of the second empty cell, and so on. Therefore, the maximum number of branches is

$$\frac{9^{h+1}-1}{9-1}-1,$$

simplifying which we get:

$$\frac{9^{h+1}-9}{8}$$

Now, for a completely blank 3D Sudoku, h =729. Therefore, in the worst-case scenario, the number of branches is

$$\frac{9^{730}-9}{8},$$

which is enormous.

The brute force algorithm proposed in [2] faces this situation, thus working for only very easy instances. The number of probable values for each blank cell can be reduced so that the number of branches is also lowered. Whenever a new possible value is discovered for a particular empty cell, all the paths need to be traversed one-by-one to insert the found value at the end of each path.

Theoretically, each path may contain a valid solution for a 2D Sudoku puzzle. All combinations of such solutions are compared to achieve the final answer to a 3D Sudoku problem. Practically, all cells are not blank in a 2D Sudoku, and thus, a limited number of solutions exist for each such 2D Sudoku. As a result, checking all possible combinations of solutions becomes much less time consuming than the previous methods. The second version of the algorithm follows

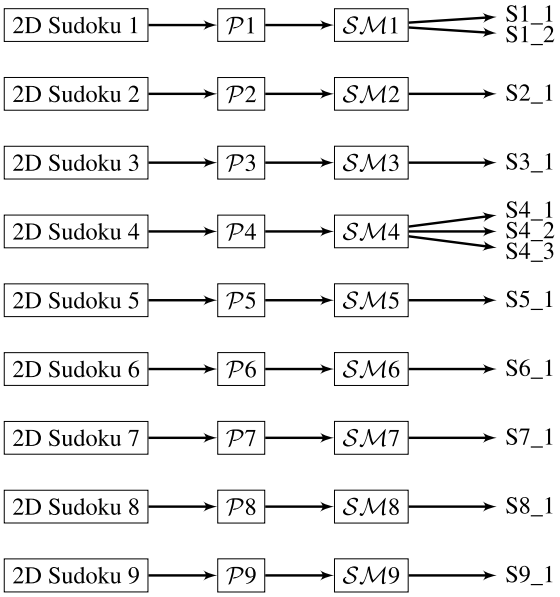


FIGURE 18. Flow of the algorithm, where S*i*_*j* denotes the *j*th solution of the *i*th 2D Sudoku.

S1_1	S2_1	S3_1	S4_1	S5_1	S6_1	S7_1	S8_1	S9_1
S1_1	S2_1	S3_1	S4_2	S5_1	S6_1	S7_1	S8_1	S9_1
S1_1	S2_1	S3_1	S4_3	S5_1	S6_1	S7_1	S8_1	S9_1
S1_2	S2_1	S3_1	S4_1	S5_1	S6_1	S7_1	S8_1	S9_1
S1_2	S2_1	S3_1	S4_2	S5_1	S6_1	S7_1	S8_1	S9_1
S1_2	S2_1	S3_1	S4_3	S5_1	S6_1	S7_1	S8_1	S9_1

FIGURE 19. All possible combinations of solutions of 2D Sudoku puzzles of Fig. 18.

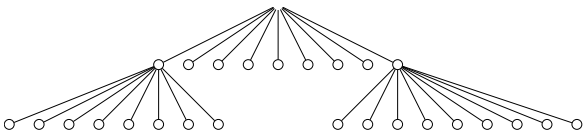


FIGURE 20. A 9-ary Tree Structure.

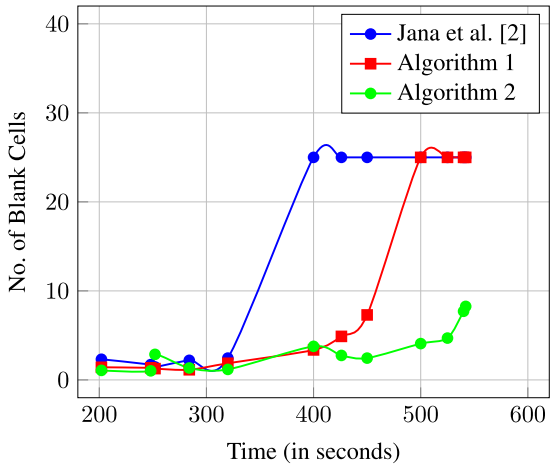


FIGURE 21. Performance of the three algorithms with respect to 3D Sudoku instances. No. of Blank Cells>=25 indicates stagnant level, where Jana et al. [2] and Algorithm 1 become stagnant.

a different way. For each permutation tree of a 2D Sudoku, in the worst case, 81 cells are blank. In that case, the number

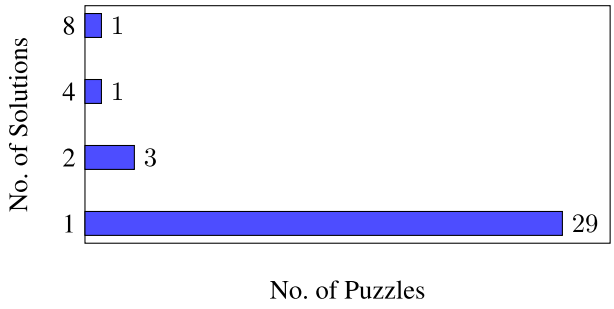


FIGURE 22. Number of solutions with respect to the number of *easy* puzzles.

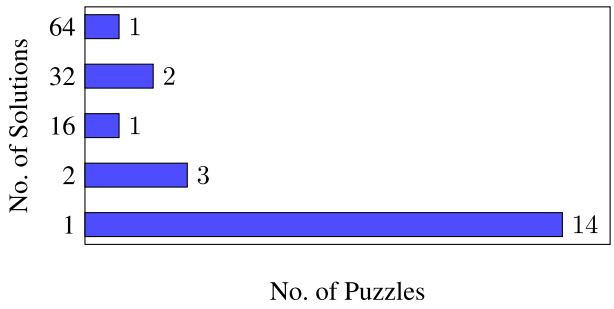


FIGURE 23. Number of solutions with respect to the number of *moderate* puzzles.

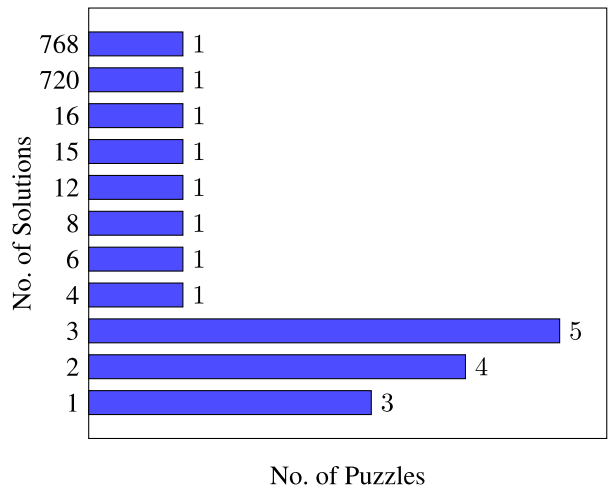


FIGURE 24. Number of solutions with respect to the number of *hard* puzzles.

of branches is

$$\frac{9^{82} - 9}{8}$$

VII. CONCLUSION AND FUTURE WORK

It can be concluded that by developing 3D Sudoku solvers using different methodologies not only enriches the research field of solving NP-complete constraint satisfaction problems, but immense benefits can also be achieved by applying this solution in different application areas. The backtracking-based algorithm proposed by Jana et al. [2]

works well only for easy 3D Sudoku puzzles. However, the algorithm does not perform well in the case of moderate or hard puzzles due to the vast number of complicated computations and recursions.

To solve this issue, we propose an improved algorithm (Algorithm 1) in this work, and the improvement is achieved by including five functional constraints. This helped our proposed algorithm work better for Sudoku puzzles with a higher number of blank cells and a higher difficulty level. However, Algorithm 1 does not perform well in the case of hard Sudoku puzzles. Although backtracking is reduced, it is still unable to make the algorithm suitable for more complex instances.

Some methodologies do exist that are specifically designed for solving hard Sudoku puzzles in 2D [32]. However, we proposed another algorithm capable of solving easy, medium, and hard instances in 3D, in this work. The proposed second version of the algorithm (Algorithm 2) follows a bottom-up approach, and backtracking is applied to individual 2D Sudoku puzzles rather than the whole 3D puzzle. For this reason, the quantum of backtracking required is not huge. Thus, many hard puzzles can be solved by Algorithm 2, along with easy and moderate puzzles. However, Algorithm 2 may not work if any puzzle (among the nine input 2D Sudoku puzzles) is blank, almost empty, or has very few clues.

Finally, it can be stated that the proposed algorithms, to the best of our knowledge, are the first ever to have been designed for solving 3D Sudoku puzzles. Compared to other algorithms, our proposed algorithms (both Algorithms 1 and 2) can solve many easy, medium, and hard 3D Sudoku puzzles with a controlled computational complexity. Furthermore, for any valid Sudoku instance, our proposed algorithms guarantee at least one solution, and if more than one solution exists, our proposed algorithms succeed in finding them.

In the future, we plan to generate a 3D Sudoku solver by using evolutionary algorithms, like genetic algorithm, ant colony optimization, etc., to check the effectiveness of evolutionary algorithms in developing a 3D Sudoku solver.

REFERENCES

[1] S. Jana, N. Dutta, A. K. Maji, and R. K. Pal, "A novel time-stamp-based audio encryption scheme using sudoku puzzle," in *Proc. Int. Conf. Frontiers Comput. Syst.* Berlin, Germany: Springer, 2023, pp. 159–169.

[2] S. Jana, M. Mallik, A. K. Maji, and R. K. Pal, "A novel search tree-based 3D Sudoku solver," in *Proc. Int. Conf. Comput. Commun. Syst.* Berlin, Germany: Springer, 2021, pp. 709–718.

[3] N. Ikeo, H. Fukuda, A. Matsugaki, T. Inoue, A. Serizawa, T. Matsuzaka, T. Ishimoto, R. Ozasa, O. Gokcekaya, and T. Nakano, "3D puzzle in cube pattern for anisotropic/isotropic mechanical control of structure fabricated by metal additive manufacturing," *Crystals*, vol. 11, no. 8, p. 959, Aug. 2021.

[4] N. Capece, U. Erra, M. Gruosso, and M. Anastasio, "Archaeo puzzle: An educational game using natural user interface for historical artifacts," in *Proc. Eurograph. Workshop Graph. Cultural Heritage*, M. Spagnuolo and F. J. Melero, Eds. Lecce, Italy: The Eurographics Association, 2020, pp. 99–102.

[5] B.-B. Xia, A.-H. Wang, C.-C. Chang, and L. Liu, "An image steganography scheme using 3D-sudoku," *J. Inf. Hiding Multimedia Signal Process.*, vol. 7, no. 4, pp. 836–845, Jul. 2016.

[6] A. K. Maji and R. K. Pal, "Sudoku solver using minigrid based backtracking," in *Proc. IEEE Int. Advance Comput. Conf. (IACC)*, Feb. 2014, pp. 36–44.

[7] A. K. Maji, S. Jana, and R. K. Pal, "A comprehensive sudoku instance generator," in *Advanced Computing and Systems for Security*. Berlin, Germany: Springer, 2016, pp. 215–233.

[8] T. Yato and T. Seta, "Complexity and completeness of finding another solution and its application to puzzles," *IEICE Trans. Fundam. Electron., Commun. Comput. Sci.*, vol. 86, no. 5, pp. 1052–1060, 2003.

[9] G. McGuire, B. Tugemann, and G. Civario, "There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration," *Exp. Math.*, vol. 23, no. 2, pp. 190–217, Apr. 2014.

[10] R. Lewis, "Metaheuristics can solve sudoku puzzles," *J. Heuristics*, vol. 13, no. 4, pp. 387–401, Jul. 2007.

[11] J. A. Pacurib, G. M. M. Seno, and J. P. T. Yusiong, "Solving sudoku puzzles using improved artificial bee colony algorithm," in *Proc. 4th Int. Conf. Innov. Comput., Inf. Control (ICICIC)*, Dec. 2009, pp. 885–888.

[12] D. Weyland, "A critical analysis of the harmony search algorithm—How not to solve sudoku," *Oper. Res. Perspect.*, vol. 2, pp. 97–105, Dec. 2015.

[13] G. Singh and K. Deep, "A new membrane algorithm using the rules of particle swarm optimization incorporated within the framework of cell-like P-systems to solve sudoku," *Appl. Soft Comput.*, vol. 45, pp. 27–39, Aug. 2016.

[14] A. Assad and K. Deep, "Harmony search based memetic algorithms for solving sudoku," *Int. J. Syst. Assurance Eng. Manage.*, vol. 9, no. 4, pp. 741–754, Aug. 2018.

[15] R. Soto, B. Crawford, C. Galleguillos, E. Monfroy, and F. Paredes, "A hybrid AC3-tabu search algorithm for solving sudoku puzzles," *Expert Syst. Appl.*, vol. 40, no. 15, pp. 5817–5821, Nov. 2013.

[16] T. Mantere and J. Koljonen, "Solving, rating and generating sudoku puzzles with GA," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 1382–1389.

[17] S. Jana, A. Dey, A. K. Maji, and R. K. Pal, "A novel hybrid genetic algorithm-based firefly mating algorithm for solving sudoku," *Innov. Syst. Softw. Eng.*, vol. 17, no. 3, pp. 261–275, Sep. 2021.

[18] A. K. Maji and R. K. Pal, "A novel biometric template encryption scheme using sudoku puzzle," in *Applied Computation and Security Systems*. Berlin, Germany: Springer, 2015, pp. 109–128.

[19] H. Okagbue, Z. Omogbadegun, F. Olajide, and A. Opanuga, "On some suggested applications of sudoku in information systems security," *Asian J. Inf. Technol.*, vol. 14, no. 4, pp. 117–121, 2015.

[20] M. Yang, N. Bourbakis, and S. Li, "Data-image-video encryption," *IEEE Potentials*, vol. 23, no. 3, pp. 28–34, Aug./Sep. 2004.

[21] M. H. Shirali-Shahreza and M. Shirali-Shahreza, "Steganography in SMS by sudoku puzzle," in *Proc. IEEE/ACS Int. Conf. Comput. Syst. Appl.*, Mar. 2008, pp. 844–847.

[22] A. K. Maji, R. K. Pal, and S. Roy, "A novel steganographic scheme using sudoku," in *Proc. Int. Conf. Electr. Inf. Commun. Technol. (EICT)*, Feb. 2014, pp. 1–6.

[23] W.-C. Wu and G.-R. Ren, "A new approach to image authentication using chaotic map and sudoku puzzle," in *Proc. 5th Int. Conf. Intell. Inf. Hiding Multimedia Signal Process.*, Sep. 2009, pp. 628–631.

[24] P. M. Naini, S. M. Fakhraie, and A. N. Avanaki, "Sudoku bit arrangement for combined demosaicking and watermarking in digital camera," in *Proc. 2nd Int. Conf. Adv. Databases, Knowl., Data Appl.*, 2010, pp. 41–44.

[25] Y. Erlich, K. Chang, A. Gordon, R. Ronen, O. Navon, M. Rooks, and G. J. Hannon, "DNA sudoku—Harnessing high-throughput sequencing for multiplexed specimen analysis," *Genome Res.*, vol. 19, no. 7, pp. 1243–1253, Jul. 2009.

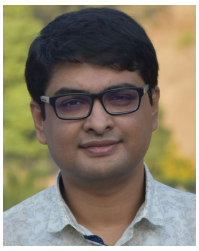
[26] C.-C. Chang, P.-Y. Lin, Z. H. Wang, and M. C. Li, "A sudoku-based secret image sharing scheme with reversibility," *J. Commun.*, vol. 5, no. 1, pp. 5–12, Jan. 2010.

[27] R. Lewis, *A Guide to Graph Colouring*, vol. 10. Berlin, Germany: Springer, 2015, pp. 978–983.

[28] J. F. Franco, O. G. Carmona, and R. A. Gallego, "Aplicación de técnicas de optimización combinatorial a la solución del sudoku," *Scientia et technica*, vol. 13, no. 37, pp. 151–156, 2007.
[29] A. M. Herzberg and M. R. Murty, "Sudoku squares and chromatic polynomials," *Notices AMS*, vol. 54, no. 6, pp. 708–717, 2007.
[30] G.-C. Lau, J. Maria Jeyaseeli, W.-C. Shiu, and S. Arumugam, "Sudoku number of graphs," 2022, *arXiv:2206.08106*.
[31] N. Jussien, *A to Z of Sudoku*. Washington, DC, USA: ISTE, 2007.
[32] H. Lloyd and M. Amos, "Solving *sudoku* with ant colony optimization," *IEEE Trans. Games*, vol. 12, no. 3, pp. 302–311, Sep. 2019.

SUNANDA JANA (Graduate Student Member, IEEE) received the B.Tech. degree from the Biju Patnaik University of Technology, Orissa, India, in 2007, and the M.Tech. degree from Berhampur University, Orissa, India, in 2010. She is currently an Assistant Professor with the Haldia Institute of Technology, West Bengal, India. Her research interests include the design and analysis of algorithms and network security.

MANJARINI MALLIK received the B.Sc. and M.Sc. degrees in computer science and the M.Tech. degree in computer science and engineering from the University of Calcutta, Kolkata, India, in 2016, 2018, and 2020, respectively. She is currently a Junior Research Fellow of the DST-Sponsored Project, Jadavpur University. Her research interests include the design and analysis of algorithms, machine learning, deep learning, and indoor localization.



ABHINANDAN KHAN (Member, IEEE) received the B.Tech. degree in electronics and communication engineering from the West Bengal University of Technology, Kolkata, India, in 2011, the M.E. degree in electronics and telecommunication engineering from Jadavpur University, Kolkata, in 2013, and the Ph.D. degree from the University of Calcutta, Kolkata, in 2020. He is currently the Head of product development and diversification with ARP Engineering, Kolkata. He has published more than 25 research articles. His research interests include computational biology, bioinformatics, artificial intelligence, and natural language processing. He received the University Gold Medal for securing the highest marks among all post-graduate engineering courses at Jadavpur University. He is also a recipient of the Junior and Senior Research Fellowships (NET) from the Council of Scientific and Industrial Research (CSIR), Government of India.

ARNAB KUMAR MAJI (Senior Member, IEEE) received the B.E. degree in information science and engineering from Visvesvaraya Technological University, in 2003, the M.Tech. degree in information technology from Bengal Engineering and Science University, Shibpur, in 2006, and the Ph.D. degree from Assam University, Silchar, India, in 2016. He has approximately 18 years of professional experience. He is currently an Associate Professor with the Department of Information Technology, North-Eastern Hill University, Shillong, India. He has published around 80 articles in reputed international journals and conferences, and authored book chapters. He has also coauthored three books (Springer). His research interests include computer vision, NP-complete puzzles, and natural language processing. He has also served as a Guest Editor for one of the Springer journals.

RAJAT KUMAR PAL (Member, IEEE) received the B.E. degree in electrical engineering from Bengal Engineering College, Shibpur, under the University of Calcutta, India, in 1985, the M.Tech. degree in computer science and engineering from the University of Calcutta, in 1988, and the Ph.D. degree from the Indian Institute of Technology, Kharagpur, India, in 1996. Since 1994, he has been a Faculty with the Department of Computer Science and Engineering, University of Calcutta. He took a leave of absence to become a Professor with the Department of Information Technology, Assam University, India, from 2010 to 2012. He is currently a Professor with the Department of Computer Science and Engineering, University of Calcutta. He has published nearly 250 research articles and has authored and coauthored two books. He also holds several international patents. His major research interests include VLSI design, graph theory and its applications, perfect graphs, logic synthesis, design and analysis of algorithms, computational geometry, and parallel computation and algorithms.