American Journal of Computer Science and Technology
2021; 4(4): 119-128
http://www.sciencepublishinggroup.com/j/ajcst
doi: 10.11648/j.ajcst.20210404.15
ISSN: 2640-0111 (Print); ISSN: 2640-012X (Online)

Applications Based on a Novel Sudoku Solver Algorithm and Grid Based Models

Abhishake Kundu[1], Anand Sunder[2]

[1]Department of Industrial and Manufacturing Systems Engineering, Texas Tech University, Lubbock, USA
[2]Capgemini SE, Hyderabad, India

Email address:
abhishake.kundu@ttu.edu (A. Kundu), anand.sunder@ttu.edu (A. Sunder)

To cite this article:
Abhishake Kundu, Anand Sunder. Applications Based on a Novel Sudoku Solver Algorithm and Grid Based Models. *American Journal of Computer Science and Technology*. Vol. 4, No. 4, 2021, pp. 119-128. doi: 10.11648/j.ajcst.20210404.15

Received: August 18, 2021; **Accepted**: November 24, 2021; **Published**: December 2, 2021

Abstract: Numerous algorithms for solving sudoku puzzles have been explored, most of which use a backtracking approach. Thus computational efficiency of such algorithms can sometimes yield poor results. We propose a probabilistic solver algorithm which, iteratively fills the sudoku grid and solves the same. In this approach we make use of a dynamic random number set, we identify unassigned sudoku grids for a given puzzle where only one possible value can be filled in and iteratively identify and assign cells with least number of possible values. We not only elaborate on our solver algorithm logic, but also explore application areas based on algorithm devised, after reviewing relevant similar approaches illustrated in the referenced articles. We believe by extension of this algorithm, many combinatorial problems in the field of material characterization, cryptography, cybersecurity can be solved and advanced. We also envision that with application of neural networks, Machine Learning techniques the algorithm will take a very adaptive and robust form, useful for solving complex problems in accurate estimation of missing data, discrete event analysis and prediction. Uniqueness is the ability to use high probability for faster computation and low execution time. With cyberattacks of varied vectors and types, its important to devise a mechanism to create a deliberate mismatch every time a possible attack is detected.

Keywords: Sudoku Solver, Logistic Model, Backtracking, Algorithm

1. Introduction

Solving a sudoku puzzle using a deterministic method is similar to taking the right decisions after weighing the risks involved, instead of wasting resources trying and backtracking every time someone takes a wrong direction. Even the best of algorithms [1] do not prove ideal, alternative approaches such as the backtracking algorithm [2], exact cover approach [3], stochastic approaches [4, 5] and a deterministic approach [6] although each exhibiting a unique style and prowess based on varied computing metrics, thus a quest to unravel an approach that repeatedly seeks a naked single6 until the only option left is to make use of a random number generator as seen in 2 is explored based on shifting probabilities of assignment to an unassigned Sudoku cell. This approach opens avenue to understanding ways to avoid exhausting possible paths to solving a problem/completing a task.

Genetic algorithms [5] till date were acknowledged to have superior performance, compared to all of the state of the art till date. While we reviewed all approached, we realized that there was still no way to benchmark algorithms as the best achievable performance and do all comparisons based on the same.

The need for a comprehensive algorithm that worked based on assigning cells that had least number of possible values, or a dynamic probabilistic approach. Sudoku grids which generate a p=1 or single possible assignment value are considered as accelerators for our approach. Likewise, we in general look for unassigned cells with lest number of possible values, based on first cut grid tear down.

Our idea was to use this as a reference to compare human solvers and map it against this, to study how often the human mind gets close to the solution and misses it. Also where does the human mind surpass the algorithm.

To study these, we needed to phase the problem into phases and address each of them in order.

Following was the initial study undertaken:

1) Complete the working prototype solver in R and feed diverse, unbiased samples of Sudoku puzzles from a wide variety of sources.

2) Collect and save iteration statistics generated by R software and the 3D scatter plot visualizations showing probabilistic shifts for unassigned cells.

3) Train logic regression models based on unassigned versus assigned cell probabilities, to classify puzzles as unique or multi solution

4) Also identify diverse application possibilities for isomorphous decision structures.

Algorithm proof of concept for 2x2 grids:

We illustrate the approach using R software for a 2x2 grid and showing graphically how thealgorithm iteratively arrives at a solution.

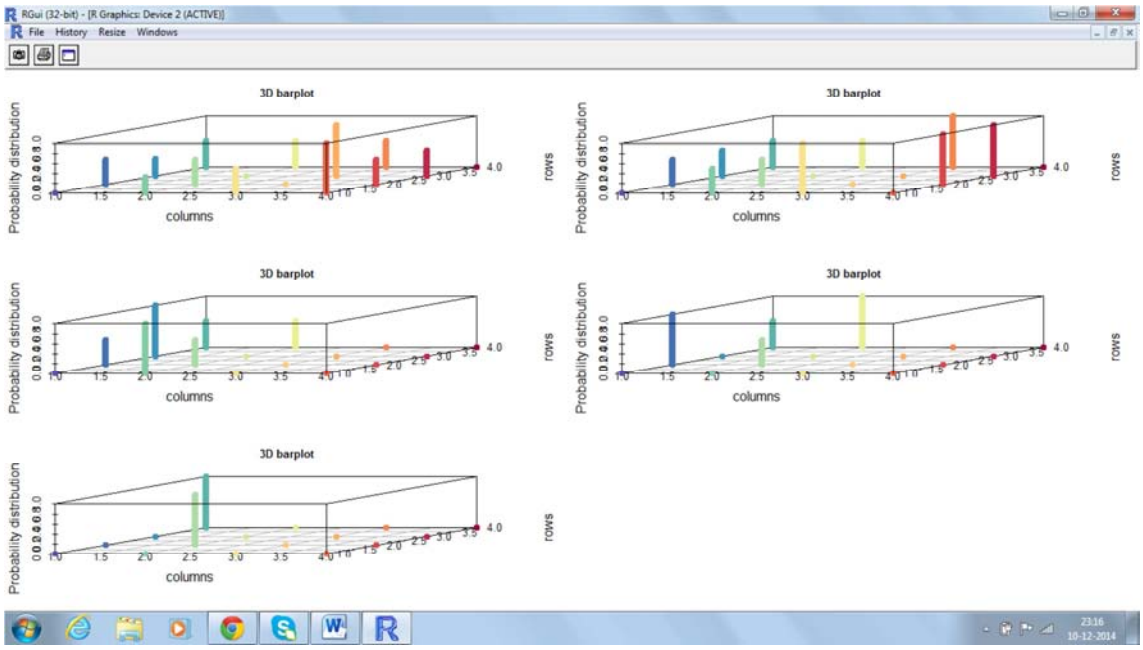

Figure 1. *Iterations of a 2x2 Latin square proof of concept showing solution at 5 th step.*

The algorithm illustrated for 4X4 puzzles turns out to be deterministic in comparison to 9x9 puzzles which turn stochastic in nature once a algorithm required run of random number generator to guess unassigned cells. However, with this limitation for 9x9 cells, we still are able to identify and reject multi solution grids and filter out sudoku puzzles. We haven't in our scope of study found a way of characterizing n 2Xn 2 type grids or studied effect of algorithm performance against increasing grid size.

Algorithm run in RStudio for a sudoku puzzle:

Combinatorics a forerunner to the modern day sudoku, has been under the attention of mathematicians like Euler [7], the first ever attempt to make a grid where a symbol or number occurs once in each row or column. For integers 1 through 9, a 9x9 such combinatorial unsolved grid yielding one unique solution became an intriguing class of puzzles, today we know these as sudoku.

Our initial hypothesis was that our greedy algorithm will identify cells in the unsolved Sudoku grid which have a unique possibility of assignment. In general, too for solving an incomplete Latin square this is a boon [7]. The next part of our study was based on the run of the algorithm for various sudoku puzzles and consolidate findings from the same.

We realize that the solver algorithm needed to continue without backtracking at points where there were multiple possible values of assignment, during the course of iterations. We introduced an explicit logic in code which flags off such points. We make use of a dynamic random number generator set S, obeying the constraints of a Latin square, here in specific the sudoku puzzle.

Excerpt from R programming showing a Random number generator runs:

Here the function sample (1:1:10,9) is a row generator that would form the base of a Latin square solver.

```
[1] 1 8 2 9 4 6 5 3 7
> b<-sample (1:1:9)
> b
[1] 3 5 8 9 2 1 7 4 6
> b<-sample (1:1:9)
> b
[1] 5 6 4 9 7 8 2 3 1
> b<-sample (1:1:9)
> b
[1] 3 1 2 6 4 8 5 7 9
> b<-sample (1:1:9)
> b
[1] 1 8 4 9 6 5 3 7 2
> b<-sample (1:1:9)
> b
[1] 6 7 3 2 1 5 4 9 8
> b<-sample (1:1:9)
> b
```

[1] 6 5 2 4 7 8 3 9 1
> b<-sample (1:1:9)
> b
[1] 9 2 6 4 5 3 1 8 7

Perhaps in the free form manner, we might have to end up running several runs of the function to obtain and isolate the Latin squares using this approach.

Given a Latin square puzzle, this approach would have to run several iterations and populate a set containing unique Latin squares. We would then have to make use of genetic algorithms [5] and neural networks, to arrive at a solution. In the evolutionary computing world this approach seemed ideal but would be computationally expensive.

Thus we needed a call for a straight forward approach without backtracking, in all of our literature survey we did find some approached were extremely efficient but all of the methods [1-6] called for backtracking at some stage of iteration. However, we can't deny that genetic algorithms would perhaps serve as the most efficient algorithms for puzzle generation [5], likewise these would also aid in solving complex challenges in pattern recognition, characterizing materials.

2. Results from the Run of Sudoku Solver Algorithm in R

Algorithm terminates when the iteration is left with cell values containing assignment probability all or most equal to 1, and zero's or assigned cells.

Interpolation studies would reveal the rate of shift in probabilities per iteration, computing $\Delta[p]9x9$, $\Delta 2[p]9x9$,.... $\Delta n - 1\,[p]9x9$, where n is the number of iterations taken to solve the sudoku grid will reveal complexity of the puzzles undertaken.

```
[1,] 0.3333333 0.25 0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.5000000 0.0000000
[2,] 0.2000000 0.25 0.2000000 0.2500000 0.0000000 0.3333333 0.2500000 0.0000000 0.2500000
[3,] 0.2500000 0.00 0.2000000 0.0000000 0.5000000 0.3333333 0.2500000 0.0000000 0.2500000
[4,] 0.0000000 0.25 0.0000000 0.3333333 0.3333333 0.0000000 0.5000000 0.0000000 1.0000000
[5,] 0.0000000 0.25 0.5000000 0.2500000 0.2000000 0.2500000 0.3333333 0.3333333 0.0000000
[6,] 0.5000000 0.00 0.3333333 0.0000000 0.5000000 0.2500000 0.0000000 0.3333333 0.0000000
[7,] 0.3333333 0.00 0.3333333 0.5000000 0.3333333 0.0000000 0.3333333 0.0000000 0.3333333
[8,] 0.3333333 0.00 0.3333333 0.3333333 0.0000000 0.2500000 0.2500000 0.3333333 0.3333333
[9,] 0.0000000 0.50 0.0000000 0.0000000 0.0000000 0.3333333 0.0000000 0.3333333 0.5000000
```

Figure 2. *Iteration 1.*

```
[1,] 0.3333333 0.2500000 0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.5000000 0.0000000
[2,] 0.2000000 0.2500000 0.2000000 0.2500000 0.0000000 0.3333333 0.2500000 0.0000000 0.3333333
[3,] 0.2500000 0.0000000 0.2000000 0.0000000 0.5000000 0.3333333 0.2500000 0.0000000 0.3333333
[4,] 0.0000000 0.3333333 0.0000000 0.5000000 0.5000000 0.0000000 1.0000000 0.0000000 0.0000000
[5,] 0.0000000 0.2500000 0.5000000 0.2500000 0.2000000 0.2500000 0.5000000 0.5000000 0.0000000
[6,] 0.5000000 0.0000000 0.3333333 0.0000000 0.5000000 0.2500000 0.0000000 0.5000000 0.0000000
[7,] 0.3333333 0.0000000 0.3333333 0.5000000 0.3333333 0.0000000 0.3333333 0.0000000 0.5000000
[8,] 0.3333333 0.0000000 0.3333333 0.3333333 0.0000000 0.2500000 0.2500000 0.3333333 0.5000000
[9,] 0.0000000 0.5000000 0.0000000 0.0000000 0.0000000 0.0000000 0.3333333 0.0000000 0.3333333 0.5000000
```

Figure 3. *Iteration 2.*

```
          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]      [,8]      [,9]
[1,] 0.3333333 0.2500000 0.0000000 0.2500000 0.0000000 0.0000000 0.0000000 0.5000000 0.0000000
[2,] 0.2000000 0.2500000 0.2000000 0.2500000 0.0000000 0.3333333 0.2500000 0.0000000 0.3333333
[3,] 0.2500000 0.0000000 0.2000000 0.0000000 0.5000000 0.3333333 0.2500000 0.0000000 0.3333333
[4,] 0.0000000 0.3333333 0.0000000 0.5000000 0.5000000 0.0000000 0.0000000 0.0000000 0.0000000
[5,] 0.0000000 0.2500000 0.5000000 0.2500000 0.2000000 0.2500000 0.5000000 0.5000000 0.0000000
[6,] 0.5000000 0.0000000 0.3333333 0.0000000 0.5000000 0.2500000 0.0000000 0.5000000 0.0000000
[7,] 0.3333333 0.0000000 0.3333333 0.5000000 0.3333333 0.0000000 0.5000000 0.0000000 1.0000000
[8,] 0.3333333 0.0000000 0.3333333 0.3333333 0.0000000 0.2500000 0.3333333 0.3333333 1.0000000
[9,] 0.0000000 1.0000000 0.0000000 0.0000000 0.0000000 0.5000000 0.0000000 0.3333333 0.0000000
```

Figure 4. *Iteration 3.*

```
     [,1] [,2] [,3] [,4]      [,5]      [,6] [,7] [,8] [,9]
[1,]  0.0  0.0    0  0.0 0.0000000 0.0000000  0.0    0    0
[2,]  0.5  0.0    0  0.5 0.0000000 0.5000000  0.5    0    0
[3,]  0.5  0.0    0  0.0 1.0000000 0.3333333  0.5    0    0
[4,]  0.0  0.5    0  0.5 0.5000000 0.0000000  0.0    0    0
[5,]  0.0  0.5    0  0.5 0.3333333 1.0000000  1.0    0    0
[6,]  0.0  0.0    0  0.0 0.0000000 1.0000000  0.0    0    0
[7,]  0.0  0.0    0  0.0 0.0000000 0.0000000  0.0    0    0
[8,]  0.0  0.0    0  0.0 0.0000000 0.0000000  0.0    0    0
[9,]  0.0  0.0    0  0.0 0.0000000 0.0000000  0.0    0    0
```

Figure 5. *Iteration 4.*

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0.0	0	0	0	0	0.0	0	0	0
[2,]	0.5	0	0	0	0	0.5	1	0	0
[3,]	1.0	0	0	0	0	0.5	0	0	0
[4,]	0.0	0	0	0	0	0.0	0	0	0
[5,]	0.0	0	0	0	0	0.0	0	0	0
[6,]	0.0	0	0	0	0	0.0	0	0	0
[7,]	0.0	0	0	0	0	0.0	0	0	0
[8,]	0.0	0	0	0	0	0.0	0	0	0
[9,]	0.0	0	0	0	0	0.0	0	0	0

Figure 6. *Iteration 5.*

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]	[,7]	[,8]	[,9]
[1,]	0	0	0	0	0	0	0	0	0
[2,]	1	0	0	0	0	1	0	0	0
[3,]	0	0	0	0	0	1	0	0	0
[4,]	0	0	0	0	0	0	0	0	0
[5,]	0	0	0	0	0	0	0	0	0
[6,]	0	0	0	0	0	0	0	0	0
[7,]	0	0	0	0	0	0	0	0	0
[8,]	0	0	0	0	0	0	0	0	0
[9,]	0	0	0	0	0	0	0	0	0

Figure 7. *Iteration 6 Penultimate step before solution.*

3. Regression Model Evolution to Classify Puzzles

There was a need to identify the most accurate classifier model, which can be used in conjunction with results of the first iteration of the algorithm to identify grids which qualify as sudoku puzzles, i.e, ones with unique solution.

Our initial premise is to evaluate a linear regression model to classify grids in this manner, We create the following variables y matrix showing unique versus multi solution grids.

Here unique solution is classified as 0, and multi solution as 1.

We take here for sampling 5 random puzzles, of which 3 are known to be unique solution and 2 are multi solution, train a regression model to classify puzzles.

3.1. Deviance Residuals

1 2 3 4 5
-0.08077 -0.60000 -0.19615 0.49615 0.38077

3.2. Coefficients

(1 not defined because of singularities)
Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.15000 6.44819 -0.489 0.674
P1 -0.25000 0.31547 -0.792 0.511
AC 0.07692 0.12374 0.622 0.598
UA NA NA NA NA
(Dispersion parameter for gaussian family taken to be 0.3980769)
Null deviance: 1.20000 on 4 degrees of freedom
Residual deviance: 0.79615 on 2 degrees of freedom
AIC: 13.002

3.3. Residuals Versus Fitted

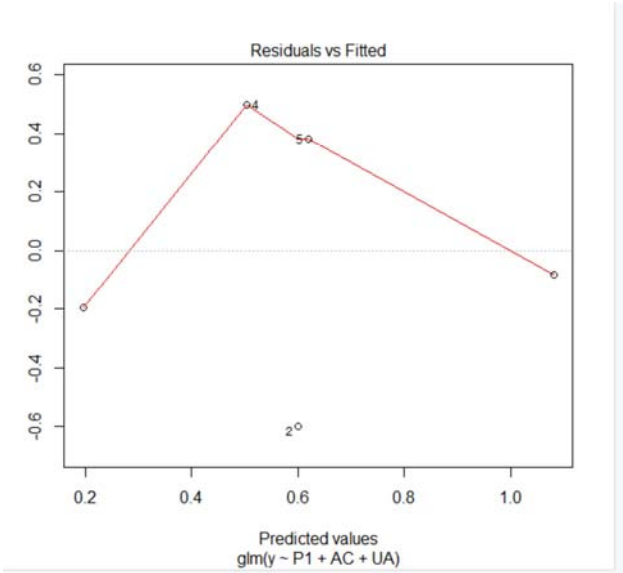

Figure 8. *Residual plot of fitted linear model.*

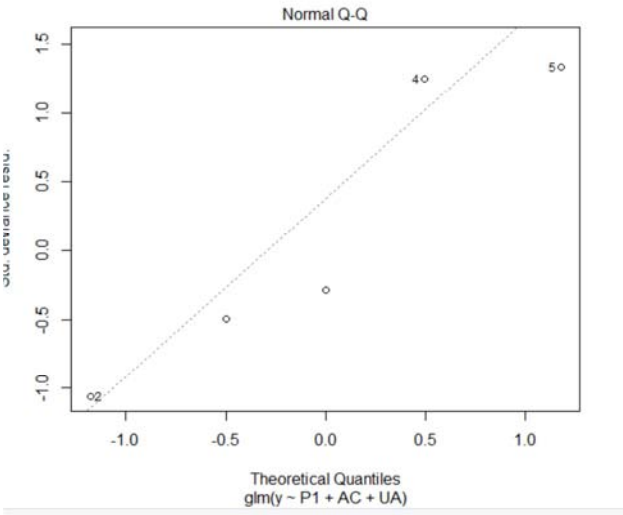

Figure 9*. Fitted line in contrast to the points varying as number of cells with single assignment possibility, unassigned cells, and assigned cells.*

From the Results Summary we obtain the classifier model as

$$Y=-0.25P1+0.077AC$$

But the classifier accuracy is low since 2,4 are classified as unique solution instead of 2,3. Meaning the accuracy is 50% and also our output value y is a Boolean.

4. Logit Regression Model

4.1. First Iteration Number of Cells with Unique Assignment Possibility

Call:

glm(formula=y ~ P1, family="binomial")
Deviance Residuals:
1 2 3 4 5
0.5837 -1.4036 -0.9172 1.4622 0.5837
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.683 1.779 0.946 0.344
P1 -1.166 1.214 -0.960 0.337
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 6.7301 on 4 degrees of freedom Residual deviance: 5.6307 on 3 degrees of freedom
AIC: 9.6307
Here the classification accuracy as highlighted by the Z value is at 94.6%.
Number of Fisher Scoring iterations: 4 where P1 is the number of cells in the grid after first iteration of the algorithm where only one value can be fitted.

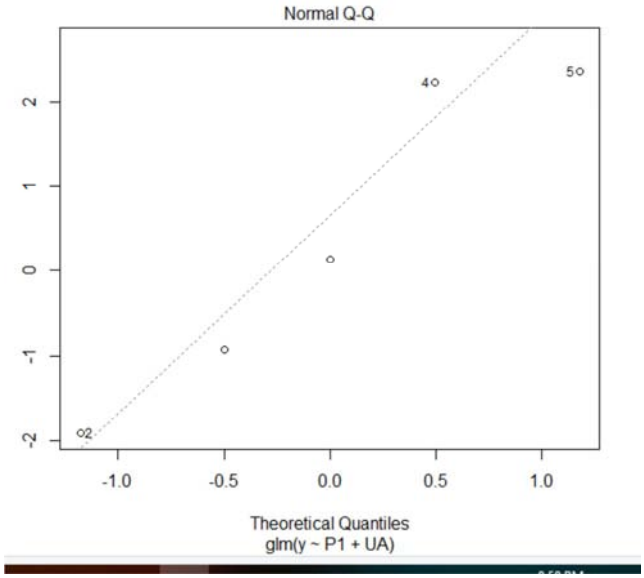

Figure 10. *Points in contrast to fitted line varying as uunassigned cells, number of unique assignment possibility cells.*

We thus conclude that number of unique assignment possibility cells, obtained in first iteration is the most accurate determining factor to classify and isolate sudokus from a grid generator.

To confirm we fit logit regressions to other factors individually listed and check

1) Number of Unassigned cells at first iteration:
Call:
glm(formula=y ~ UA, family="binomial")
Deviance Residuals:
1 2 3 4 5
0.6342 -1.3812 -1.0846 0.7395 1.4257
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 10.4668 13.1665 0.795 0.427
UA -0.3448 0.4472 -0.771 0.441
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 6.7301 on 4 degrees of freedom
Residual deviance: 6.0658 on 3 degrees of freedom
AIC: 10.066
Number of Fisher Scoring iterations: 4
Here the classification accuracy is about 79.5%

2) Number of Assigned Cells in the First Iteration:
Call:
glm(formula=y ~ AC, family="binomial")
Deviance Residuals:
1 2 3 4 5
0.6342 -1.3812 -1.0846 0.7395 1.4257
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -17.4626 23.1154 -0.755 0.450
AC 0.3448 0.4472 0.771 0.441
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 6.7301 on 4 degrees of freedom
Residual deviance: 6.0658 on 3 degrees of freedom
AIC: 10.066
Number of Fisher Scoring iterations: 4
Here the classification accuracy is about 77.1%

3) Unassigned Cells and No of cells with unique possibility at end of first iteration:
Call:
glm(formula=y ~ P1 + UA, family="binomial")
Deviance Residuals:
1 2 3 4 5
0.1236 -1.5454 -0.3968 1.0508 0.9131
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) 23.1191 28.3519 0.815 0.415
P1 -1.9316 2.1979 -0.879 0.379
UA -0.7019 0.8972 -0.782 0.434
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 6.7301 on 4 degrees of freedom
Residual deviance: 4.4990 on 2 degrees of freedom
AIC: 10.499
Number of Fisher Scoring iterations: 6
Here we get a 81.5% accuracy of classification

4) Assigned cells and no of cells with unique assignment possibility at first iteration:
Call:
glm(formula=y ~ P1 + AC, family="binomial")
Deviance Residuals:
1 2 3 4 5
0.1236 -1.5454 -0.3968 1.0508 0.9131
Coefficients:
Estimate Std. Error z value Pr(>|z|)
(Intercept) -33.7324 44.4275 -0.759 0.448
P1 -1.9316 2.1979 -0.879 0.379
AC 0.7019 0.8972 0.782 0.434
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 6.7301 on 4 degrees of freedom
Residual deviance: 4.4990 on 2 degrees of freedom
AIC: 10.499
Number of Fisher Scoring iterations: 6
We also run Chi-Square tests for getting models showing

highest

True positive percentage:

1) Unassigned cells at first iteration

Chi-squared test for given probabilities
data: UA
X-squared=0.89655, df=4, p-value=0.9251

2) Cells having unique possibility:

Chi-squared test for given probabilities
data: P1
X-squared=4, df=4, p-value=0.406

3) Assigned cells at first iteration

Chi-squared test for given probabilities
data: AC
X-squared=0.5, df=4, p-value=0.9735

Using randomly selected sample sudoku puzzles from various sources, we make sure that our classifier models are accurate and unbiased.

4.2. Scatter Plot Showing Probability Distribution of the Penultimate Iteration

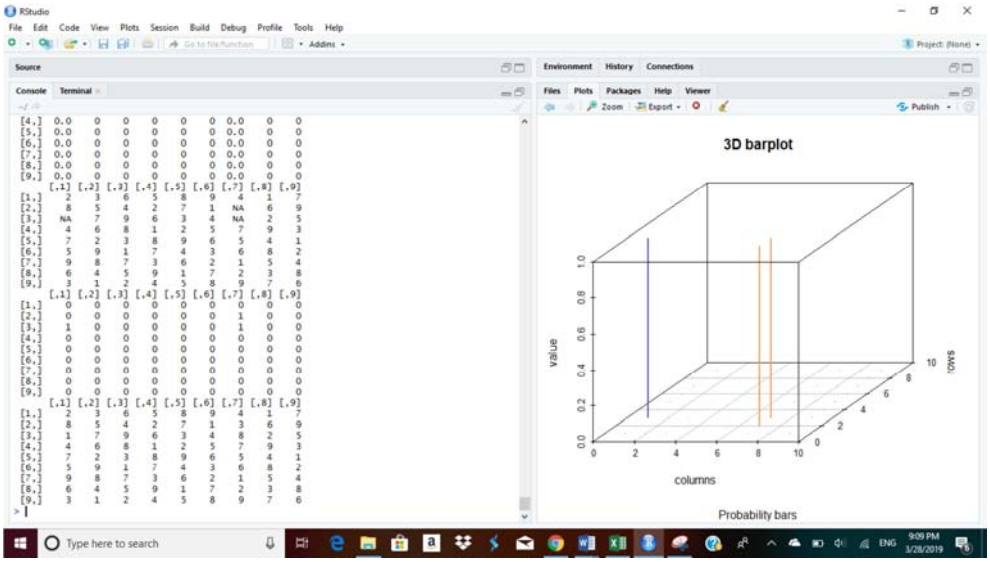

Figure 11. *Scatter plot showing assignment probability distribution at Penultimate iteration.*

4.3. Identifying a Multi Solution Puzzle

We have included also an identifying logic that isolates multi-solution puzzles and flags it through the iterations as seen in the snippet below.

Broadly three categories or cases can be charted out where grids can yield multiple solutions.

Random Number Generator Cannot Fill Assign any Value Or Runs Out

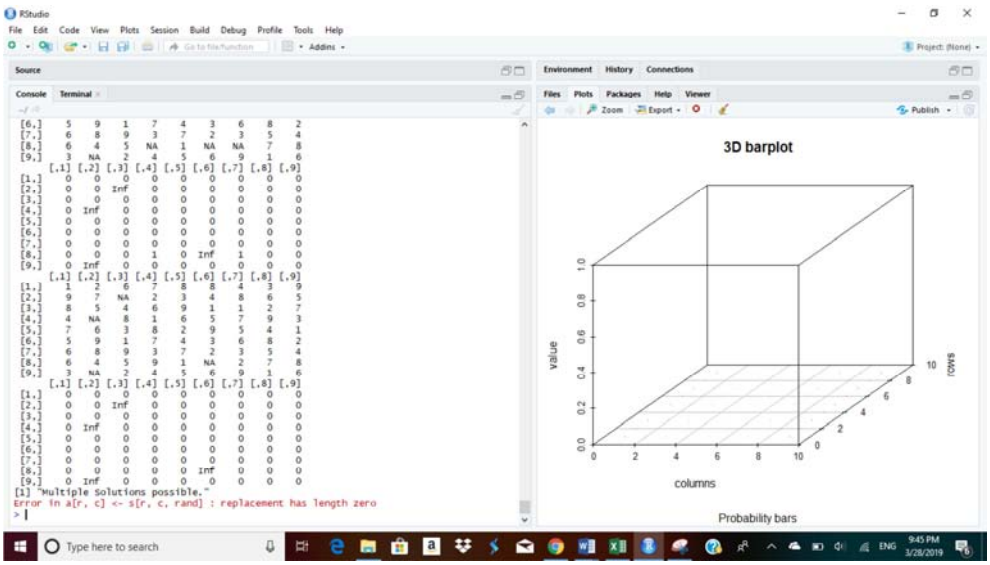

Figure 12. *Locked conflicting iteration block.*

4.3.1. Cell with Two Possible Assignment Values at the Penultimate Iteration

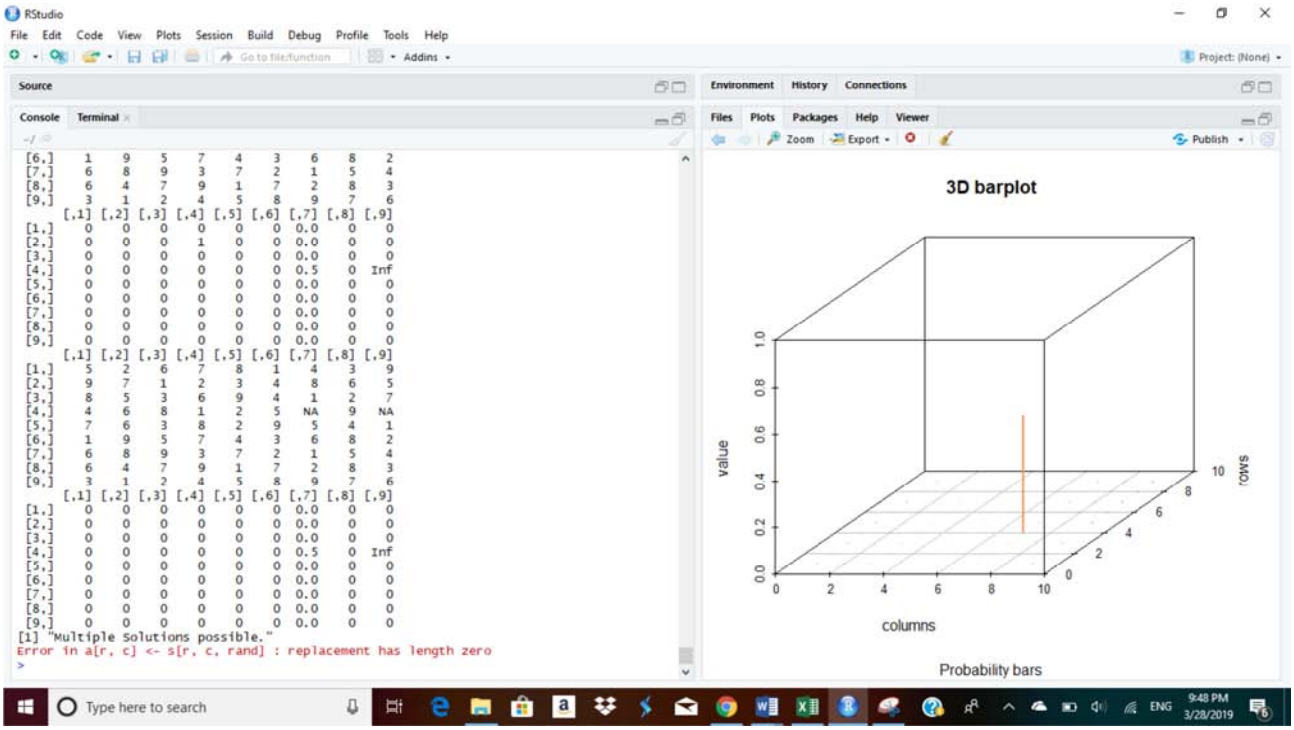

Figure 13. *Penultimate iteration with one locked cell and one with two assignment possibilities.*

4.3.2. All Unassigned Cells have Multiple Possibilities at Penultimate Iteration

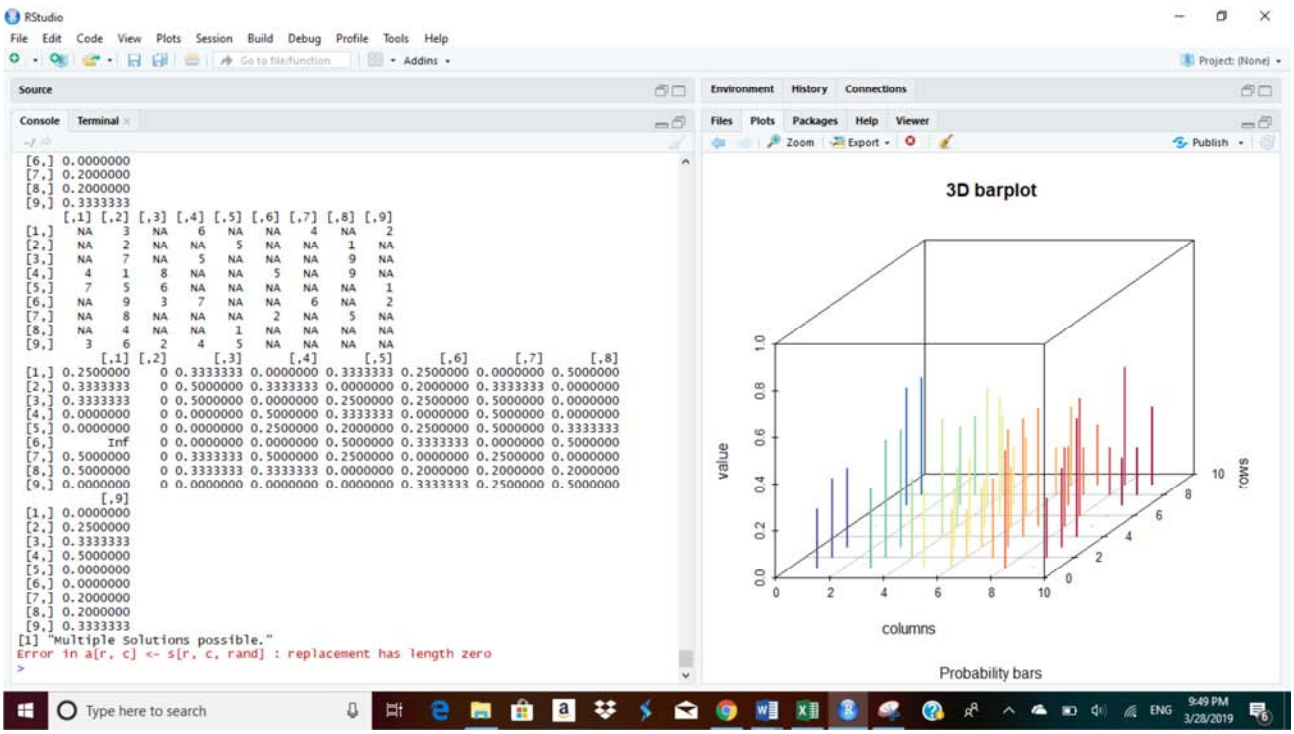

Figure 14. *Penultimate iteration with no cell values with unique assignment possibility.*

These are explicitly called in the working code to flag such grids, this also highlights the potential of a puzzle generator. In order to get to logically define a puzzle generator we thus need to firstly revisit the algorithm, in terms of a flow chart. Algorithm flowchart:

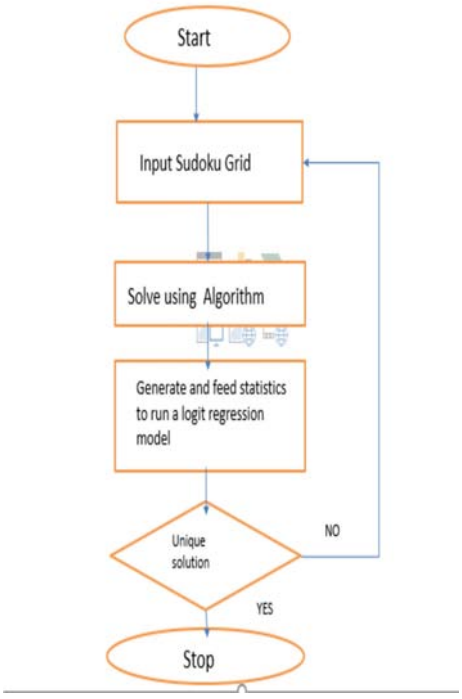

Figure 15. *Algorithm flow chart of the sudoku solver.*

4.4. Difficulty Rating of Sudoku Puzzles

Based on the proposed and running it on various puzzles we come across what is a rounded normalized score. The hypothesis or premise of the difficulty rating is based on the probability of assignment to unassigned grids before puzzle is solved by the algorithm.

NAs	% (P=1) change in iterations								EMA	NA/EMA	Normalized score	Rounded normalized	sqrt(NA/EMA)	Normalized score~root	Rounded Normalized~root
	Last Step														
12	0.17	0.17	0.17	0.33	0.17				0.25	48	2.296551724	2.3	6.92820323	3.01728799	3
10	0.50	0.50							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
11	0.36	0.45	0.18						0.295455	37.23076923	1.707692308	1.7	6.101702158	2.460478146	2.5
10	0.60	0.40							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
12	0.33	0.50	0.17						0.291667	41.14285714	1.921604504	1.9	6.414269806	2.671053485	2.7
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
12	0.50	0.33	0.17						0.291667	41.14285714	1.921604504	1.9	6.414269806	2.671053485	2.7
11	0.18	0.45	0.36						0.340909	32.26666667	1.436256158	1.4	5.680375574	2.176632429	2.2
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
7	0.14	0.29	0.00	0.14	0.29				0.125	56	2.733990148	2.7	7.483314774	3.391263998	3.4
12	0.17	0.50	0.25	0.00					0.291667	41.14285714	1.921604504	1.9	6.414269806	2.671053485	2.7
10	0.20	0.40	0.40						0.35	28.57142857	1.234201267	1.2	5.345224838	1.950842971	2
10	0.50	0.50							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
10	0.40	0.30	0.30						0.325	30.76923077	1.354376658	1.4	5.547001962	2.08677926	2.1
8	0.50	0.50							0.5	16	0.54679803	0.5	4	1.044571391	1
12	0.17	0.17	0.17	0.33	0.17				0.25	48	2.296551724	2.3	6.92820323	3.01728799	3
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
12	0.50	0.33	0.17						0.291667	41.14285714	1.921604504	1.9	6.414269806	2.671053485	2.7
7	0.29	0.71							0.5	14	0.437438424	0.4	3.741657387	0.8705272	0.9
7	0.29	0.71							0.5	14	0.437438424	0.4	3.741657387	0.8705272	0.9
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
6	1.00								1	6	0	0	2.449489743	0	0
10	0.40	0.60							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
10	0.40	0.60							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
10	0.50	0.50							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
10	0.10	0.40	0.50						0.375	26.66666667	1.130049261	1.1	5.163977795	1.828737699	1.8
10	0.30	0.40	0.30						0.325	30.76923077	1.354376658	1.4	5.547001962	2.08677926	2.1
10	0.30	0.20	0.50						0.375	26.66666667	1.130049261	1.1	5.163977795	1.828737699	1.8
10	0.40	0.60							0.5	20	0.765517241	0.8	4.472135955	1.36264714	1.4
10	0.20	0.40	0.40						0.35	28.57142857	1.234201267	1.2	5.345224838	1.950842971	2
10	0.10	0.40	0.50						0.375	26.66666667	1.130049261	1.1	5.163977795	1.828737699	1.8
8	1.00								1.00	8	0.109359606	0.1	2.828427125	0.255288313	0.3
8	1.00								1.00	8	0.109359606	0.1	2.828427125	0.255288313	0.3
11	0.09	0.18	0.36	0.27					0.25	44	2.077832512	2.1	6.633249581	2.818579118	2.8
12	0.17	0.33	0.33	0.17					0.291667	41.14285714	1.921604504	1.9	6.414269806	2.671053485	2.7
12	0.67	0.33							0.5	24	0.984236453	1	4.898979486	1.650209598	1.7
13	0.08	0.15	0.00	0.23	0.00	0.15	0.15	0.08	0.133413	97.44144144	5	5	9.871243156	5	5

Figure 16. *Normalized Difficulty rating of sudoku puzzles based on the solver.*

4.5. Applications of the Solver Algorithm

Based on the study conducted and insights on the algorithm run, following possible applications could be explores:

4.5.1. Cybersecurity Solutions

A dynamically generated incomplete latin square or sudoku can serve as a strong authentication means for accessing extremely confidential information. Traditional username password-based authentication will have vulnerabilities that could be overcome by this means.

4.5.2. Modelling Decision-making Problems

Real-life decision-making scenarios especially in complex projects involve multiple factors, after exploring the algorithm from various perspectives, it is not far that we are able to create algorithms [3, 8], that are able to model decision making and factors governing it.

4.5.3. Constraint Modelling for Complex Scenarios

High backtracking *or* failure prone projects or issues, where constraints are also dynamic. We can use Latin squares or rectangles to model or approximate the constraints themselves [8, 7].

4.5.4. Cybersecurity

Firewalls, antiviruses, token-based authentication mechanisms, make use of logic that could be manipulated and tampered. We thus need to integrate a mechanism that is feedback and input driven. The inputs to the mechanism being threat pattern or signature, this will also be used to feed a random number generator which feeds incomplete latin squares to our algorithm. A logic block that deliberately introduces a mismatch and fails an attempted authentication by the malicious software or virus program. The detailed logic for the system, needs to be developed in phases and calls for indigenously developing each of the logical blocks as shown below.

Flowchart of the proposed cybersecurity solution:

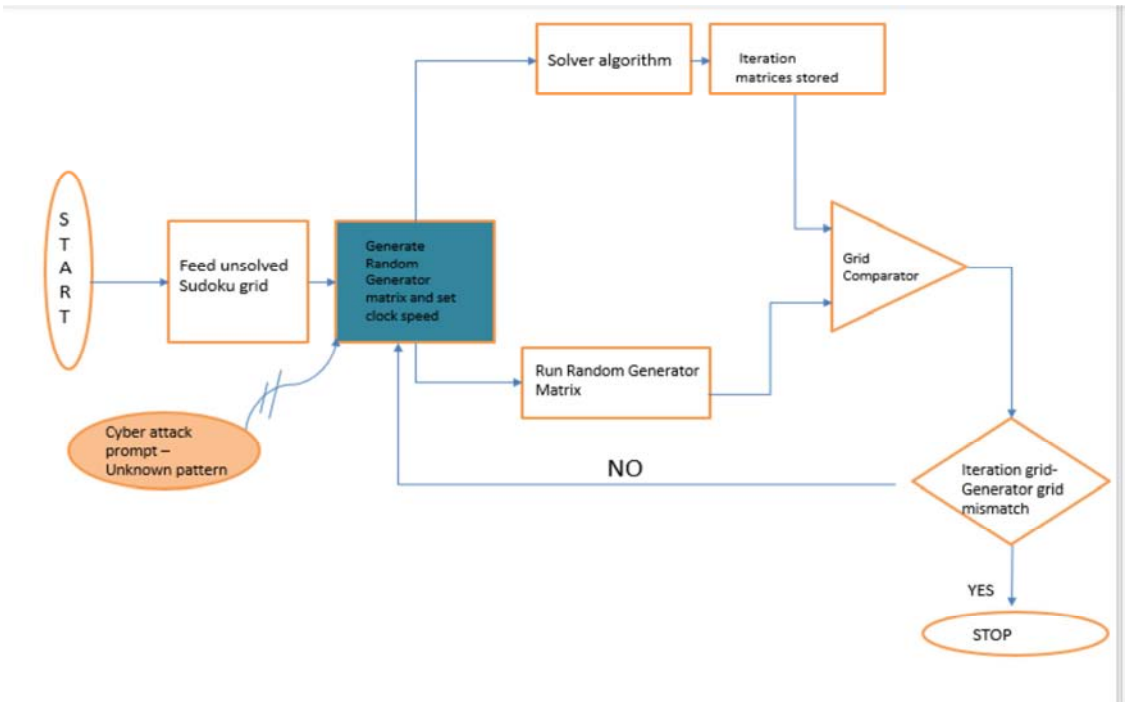

Figure 17. *Cybersecurity solution design based on solver algorithm.*

4.5.5. Characterizing Material Properties

The published article [9] talks about use of IFEA, characterizing methods based on conventional tests such as uniaxial tensile and compression tests. A closer thought to the process, we can see how the concept of an incomplete Latin square can find suitable use here. An incomplete Latin square can be shown analogous to an incompletely characterized material. Based on knowledge of permissible limits for properties such as uniaxial tensile strength, compression strength or shear strength we can model a random number generator, which can be run inside of the solver algorithm to completely decode the missing property values.

4.5.6. Design of Experiments for Fractional Factorial Designs with Blocking Uses

Latin squares for modelling scenarios when there are n distinct treatment types applied to n different subjects. Here balanced incomplete blocks [10] are used in situations where number of factors are less than number of treatment levels. Given that these form basis of designing incomplete blocks.

Statistical model defining response variable yijk, seen as

an effect of other variables such as μ (Overall average), Ri, column Cj, kth treatment Tk and error ∈ ijk [10].

$$y_{ijk} = \mu + R_i + C_j + T_k + e_{ijk}$$

We are able to use the above model also to evaluate the incomplete blocks, in conjunction with the Latin square solver algorithm. This is applicable in situations where all treatments are distinct, number of subjects are definitely lower than treatments.

Error modelling is cumbersome, but reduced factorials or blocking effects when known can be efficiently solved using our algorithm.

Thus, we reach a point where the concept of purely deterministic or purely statistical approaches to model completely fail, calling in for new approaches to tackle challenges in all the mentioned problem areas.

Reformulating (2), showing the error in predicting the outcome yijk ∈ ijk = yijk − μ − Ri − Cj − Tk.

Minimizing the above function given constraints, one of which is the Latin square that forms the design block.

One of the real-life problems that is intriguing and similar in model to factorial design, being discussed here is that of identifying factors that negatively impact performance of an indigenously developed application with multiple features. Here we can liken application features to elements or attributes in a grid, treatments as the distinct combination of test scenario settings that application is being subjected to.

For example, an application where multiple workflows are running concurrently, the number of factors causing variation are large, in most cases nested in structure too. Very often, the actual design block would often have missing elements, calling out more tests including these elements. In order to not run extra tests, and arrive at actual expected projection estimates, we need to use fractional factorials.

Decision modelling using design spaces, can make use of the solver to fill up missing parts of the decision model or outcome.

Even with use of non-parametric smoothing processes, for improving model accuracy, we need to make sure we reduce the space of unknowns, are left with most part of our experimental design known [10].

We can use the following excerpt showing a nested random number generator run to generate rows of varying sizes:

5. Conclusions

Applications of the generic Latin square or the solver algorithm could range from cybersecurity to frontiers in material science and material characterization. The possibilities are limitless, with the advent of modern AI/ML the algorithms can be further expanded and enriched to include multiple possibilities.

Acknowledgements

We would like to thank Prof Susan Urban for bringing in us an interest in the field of Data Science during our journey as graduate students in Texas Tech University.

References

[1] Arnab K Maji, Sudipta Roy and Rajat K Pal (2013). A Novel Algorithmic approach for solving Sudoku puzzle in Guess Free Manner.

[2] Backtracking approach to Sudoku solver: Geeks for Geeks.

[3] Exact cover algorithm to solving Sudoku puzzles: Wikipedia.

[4] Karimi-Dehkordi Z., Zaman far K., Baraani-Dastjerdi A., Ghasem-Aghaee N. (2010) Sudoku Using Parallel Simulated Annealing. In: Tan Y., Shi Y., Tan K. C. (eds) Advances in Swarm Intelligence. ICSI 2010. Lecture Notes in Computer Science, vol 6146. Springer, Berlin, Heidelberg.

[5] John M Weiss. (2009). "Genetic Algorithms and Sudoku".

[6] H. L Xao, X. S Ding (2014) "On the Generation and Evaluation of a Sudoku Puzzle".

[7] Maria-Ercsey-Ravasz & Zoltan Toroczkai (2012). "The Chaos within Sudoku".

[8] Alice H. Becker (2013). "Sudoku and Image Security".

[9] Guilio Maier, Vladmir Buljak, Giuseppe Cocheti (2012). "Mechanical Characterization of Materials and Diagnosis of Structures by Inverse Analysis: Some Innovative Procedures and Applications".

[10] Lei Gao (2005). "Latin Squares in Experimental Design".

Biography

Anand Sunder: Currently working as a Site Reliability Engineer with Capgemini. As a graduate student in Texas Tech University developed deep interest in Data Science, Analytics and this led to my onward journey in career further.

Abhishake Kundu: Pursuing a Ph.D. at Texas Tech University, Abhishake Kundu is a statistician, stochastics researcher and a football enthusiast.