International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 07 Issue: 05 | May - 2023 SJIF 2023: 8.176 ISSN: 2582-3930

ZIGUZAGU
Squiggly Sudoku

M.Sandeep
2111CS020475
School of Engineering
MallaReddy University

M.Sanjana
2111CS020476
School of Engineering
MallaReddy University

A.Sanjana
2111CS020477
School of Engineering
MallaReddy University

B.Sanjana
2111CS020478
School of Engineering
MallaReddy University

P.Sanjana
2111CS020479
School of Engineering
MallaReddy University

U.Sanjana
2111CS020480
School of Engineering
MallaReddy University

Guide: Dr G Hariharan
Associate Professor, Department of AIML
School of Engineering
MallaReddy University

Abstract : Abstract In moment's world of excited life people tend to use more phones and lose the capability of introductory logical logic and don't do important brain related exercise due to which our design Sudoku Game is veritably important as it helps in logical logic and brain development. currently Sudoku mystifications are getting decreasingly popular among people each around the world. The Sudoku game is now veritably notorious again and thus numerous inventors have tried to induce indeed more complicated and more intriguing mystifications. Hence the purpose of this design is to produce a Sudoku game that would help the stoner to break their mystification in certain time and also would give hints if the stoner could not crack it. In this report, we present the detailed development and perpetration of a simple Sudoku game. The Sudoku game consists of a graphical stoner interface, and a mystification solver; enforced using python. The solver finds the result to the mystifications entered by the stoner. This design gives an sapience into the different aspects of python programming.

I.INTRODUCTION

To complete this mystification requires the conundrum to fill every empty cell with an integer between 1 and 9 in such a way that every number from 1 over to 9 appears formerly in every row, every column and every one of the small 3 by 3 boxes stressed with thick borders. Sudoku mystifications vary extensively in difficulty. Determining the hardness of Sudoku mystifications is a gruelling exploration problem for computational scientists. Harder mystifications generally have smaller specified symbols. still, the number of specified cells isn't alone responsible for the difficulty of a mystification and it isn't well understood what makes a particular Sudoku mystification hard, either for a mortal or for an algorithm to break. By Sudoku mystification of box size n, in this paper, is
meant a partial assignment of values from {1,., n 2} to the cells of an n 2 × n 2 grid in such a way that at most one of each symbols occurs in any row, column or box. A result of a Sudoku mystification is a complete assignment to the cells, satisfying the same conditions on row, columns and boxes, which extends the original partial assignment. Withsudoku.py, the process of structure models of Sudoku mystifications, which can also be answered using algorithms for calculating results of the models, is a simple matter. In order to understand how to make the models, first it's necessary to explain the two different representations of Sudoku mystificationsinsudoku.py. The dictionary representation of a mystification is a mapping between cell markers and cell values. Cell values are integers in the range { 1,., n 2} and cell markers are integers in the range{ 1,., n 4}. The labelling of a Sudoku mystification of boxsize n starts with 1 in the top- left corner and moves along rows, continuing to the coming row when a row is finished. So, the cell in row i and column j is labelled (i −1) n 2j. For illustration, the mystification from the preface can be represented by the wordbook .
> d = { 1 2, 2 5, 5 3, 7 9, 9 1, 50 1, 15 4, 19 4, 21 7, 25 2,. 27 8, 30 5, 31 2, 41 9, 42 8,. 43 1, 47 4, 51 3, 58 3, 59 6,. 62 7, 63 2, 65 7, 72 3, 73 9,. 75 3, 79 6, 81 4}
A Sudoku mystification object can be erected from such a wordbook. Note that the box size is a parameter of the mystification object constructor.
> for i in range(quantum)
> y = random.randint(0, len(grid)- 1)
>>> x = random.randint(0, len(grid)- 1)
>>> num = random.randint(1, len(grid))
>>> allow = 0

> for e in range(len(grid))

> if num not in grid(x) and num! = grid(e)(y)
> allow = 1
> grid(x)(y) = num

> tempo = grid
> tempo = rearrange(tempo)
> for e in range(len(grid))
> if(duplicate_checker(tempo(e)))
> allow = 0
> if allow! = len(grid)
> grid(x)(y) = 0 2 5.. 3. 9. 1 1... 4... 4 7... 2. 8.. 5 2..... 9 8 1... 4... 3..... 3 6.. 7 2. 7..... 3 9. 3... 6. 4

Random mystifications can be created insudoku.py by the function.

> import arbitrary
> q = scramble()
> q 5... 1 5..... 7 1 9. 7... 5... 7.. 6.... 9. 5... 5..... 4.. 1.....

The first argument to srumble() is the number of specified cells in the mystification. working of mystifications insudoku.py is handled by the break function. This function can use a variety of different algorithms, specified by an voluntary model keyword argument, to break the mystification. Possible values are CP for constraint propagation, lp for direct programming, graph to use a knot coloring algorithm on a graph mystification model and groebner to break a polynomial system model via a Groebner base algorithm. The dereliction geste is to use constraint propagation. > from sudoku import break.

> s = break(q)
> s

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

II. LITERATUREREVIEW

D.H. Lehmer was the first to introduce the countermanding algorithm in 1950. The countermanding algorithm is one of the problem- working styles included in a strategy grounded on searching the result space, but it doesn't have to examine all possibilities, only those that lead to only results will be reused. Algorithms countermanding is also an algorithm that workshop recursively, where the hunt process is grounded on the Depth-First Hunt(DFS) algorithm, which is to search for methodical results to all possible results and search for answers is done by tracing a tree- shaped structure embedded (2). thus this algorithm is relatively important and veritably good to be applied in problem- working and to give artificial intelligence in the game. Several types of digital games that are generally generally known by the public, similar as Chess, Math Maze, Tic Tac Toe, to Sudoku can be set up a result by enforcing the countermanding algorithm. The countermanding algorithm is an enhancement of the brute force algorithm, which is to find results to problems among all possible results totally. Backtracking is a typical form of recursive algorithm and is grounded on DFS(Depth-First Hunt) in chancing the right answer. In another sense, the countermanding algorithm works like experimenting with several possibilities that lead to the result until it finds the most applicable bone.

So there's no need to check all possible results, but it's enough that only leads to the result, videlicet by sorting pruning the bumps that don't lead to the result. therefore the hunt time can be saved. The difference with the brute force algorithm is the introductory conception, videlicet, in countermanding, all results are made in the form of a result tree(tree), and also the tree will be traced in DFS(Depth-First Hunt) to find the best- asked result.

Affiliated workshop

- The former studies related to exploration conducted by the author include Llyod(2019) Conduct exploration published by IEEE in transnational journals entitled " working Sudoku with Ant Colony Optimization. " Says that sudoku game is a notorious mystification game that's veritably computationally grueling , so it requires the most important and most sophisticated form of algorithm to break it.(2)

- Ghosh(2017) International journals published by IEEE did the exploration entitled " A SimpleRecursive Backtracking Algorithm for Knight's tenures Puzzle on Standard 8 × 8 Chessboard " says the countermanding algorithm isn't entirely the stylish result in the case of a knight's stint game because it executes too long, making it less effective and practical.(3)
- Schottlender(2014) Conduct exploration published by IEEE in transnational journals entitled " The Effect of Guess Choices on the effectiveness of a Backtracking Algorithm in a Sudoku Solver. " Using the original element of a aimlessly generated sudoku mystification is better in the countermanding algorithm compared to using figures that are formerly available as starting rudiments.(4)
- Szabó(2014) on International journals published by IEEE did the exploration entitled " Creation of the Chips Placement Game with Backtracking Method in Borland Pascal " the operation of countermanding algorithm in the creation of the chips placement game

is veritably effective because the algorithm can calculate all combinations with a terse law Fermuller(2014) International journals published by IEEE did the exploration entitled " Semantic Games with Backtracking for Fuzzy sense. "

- apply the countermanding algorithm in Hintikka's classical game. Li etal.,(2011) International journals published by IEEE did the exploration entitled " The Research on Departure Flight Sequencing Grounded on Improved Backtracking Algorithm. " The operation of the countermanding algorithm in scheduling has worked optimally and produced a way that will be and as anticipated.

III. PROBLEM STATEMENT

This section provides a clear and terse statement of the problem, this should include a description of the data used in the design and the exploration questions and suppositions that guided the design. This problem statement should easily identify the problem that the exploration paper is trying to break and how it'll be addressed in the design. A sudoku problem is a problem where there are is an deficient 9x9 table of figures which must be filled according to several rules Within any of the 9 individual 3x3 boxes, each of the figures 1 to 9 must be set up. Within any column of the 9x9 grid, each of the figures 1 to 9 must be set up. The ideal is to fill a 9 × 9 grid so that each column, each row, and each of the nine 3 × 3 boxes(also called blocks or regions) contains the integers from 1 to 9, only one time each(that is, simply). The mystification setter provides a incompletely completed grid.

IV. METHODOLOGY

enforcing working algorithms In order for there to be no difference other than the algorithms themselves, all algorithms were written in the same programming language(python) and all testing was done on the same device. To get as analogous conditions as possible, all algorithms were written as, which also had all the test cases defined. The test cases, after being constructed, were also transferred to each of the working algorithms in turn.

```
 > def solver()
 > global grid, done
>>> if( done ==  False)
 > for y in range( 9)
> for x in range( 9)
 > if grid( y)( x) ==  0
 > for num in range( 1,10)  still, x, y)
 >>> if stay( num.
>>> grid( y)( x) =  num
> solver()
> grid( y)( x) =  0
> return > done =  True
 > for a inentry_list
> for b in a
>b.delete( first =  0, last =  100)
>>>display_val()
> return
```

The break system uses countermanding algorithm. To insure as little bias as possible, test cases were also divided into categories depending on difficulty, and also these categories were answered both collectively and together. enforcing test cases In order to insure the quality of the test cases be sufficient, the sudoku mystifications used are taken from the book " A to Z of sudoku " by Narendra Jussien. The book divides mystifications into six orders grounded on difficulty but then we took only three orders. These orders can be set up in the result section, named " Easy ", " Medium " and " Hard ".

Creating Sudoku mystification

```
 > def scramble()
 > global grid
 > clear()
 > for a inentry_list
 > for b in a
>b.delete( first =  0, last =  100)
>>>  quantum =  20
> for i in range(  quantum)
> y = random.randint( 0, len( grid)- 1)
>>> x = random.randint( 0, len( grid)- 1)
>>> num = random.randint( 1, len( grid))
>>> allow =  0
> for e in range( len( grid))
> if num not in grid( x) and num! =  grid( e)( y)
> allow =  1
> grid( x)( y) =  num
> tempo =  grid
> tempo =  rearrange( tempo)
 > for e in range( len( grid))
 > if(duplicate_checker( tempo( e)))
 > allow =  0
> if allow! =  len( grid)
> grid( x)( y) =  0
>display_val()
```

For different position we change the quantum which is mentioned in the below algorithm.

These situations of difficulty are grounded on the number of different sudoku working ways necessary to break them. The veritably easy mystifications only need one or two of the most introductory working ways, whereas the Hard mystifications bear every available solving system. Ten mystifications from each order were used as test cases, and to make the figures easier to observe, each test case was run times, totalling one million answered mystifications per position of difficulty.

Comparing the results :

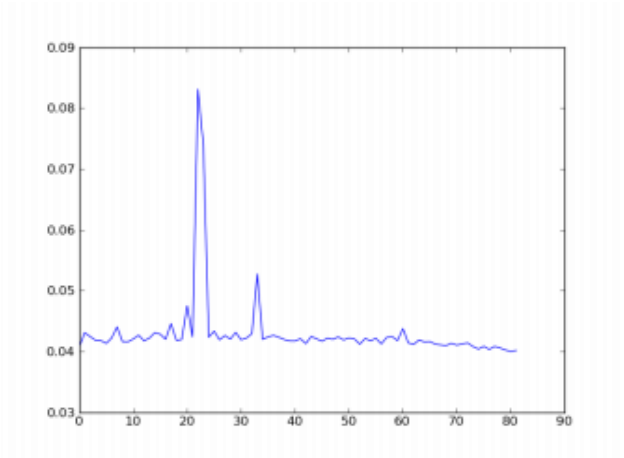
The results of the tests were compared primarily on how snappily the algorithms answered all the test cases. They were also reviewed to ascertain whether any mystifications were unattainable for any of the algorithms. With the countermanding algorithm, the alternate factor was spare as it tries every possiblesolution.However, it means that it was either inaptly enforced, or the test cases themselves were unattainable, If this algorithm returns unsolved test cases. The

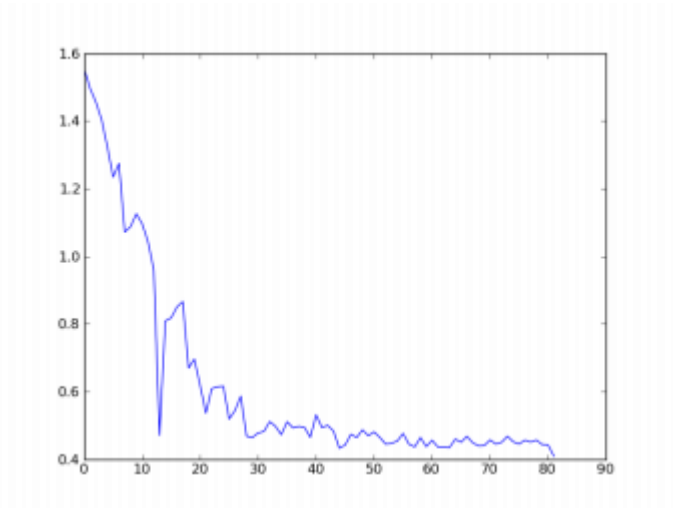
results were also given in seven different performances. First all of the test cases together, also the results for the testing the different situations of difficulty for the unsolved mystifications. These results were reviewed collectively as well as grouped together, to see if there was any difference between the different results.

V. EXPERIMENTAL RESULTS

We introduced therandom_puzzle function in the preface. The system by which this function produces a arbitrary mystification is fairly simple. A completed Sudoku mystification is first generated by working the empty mystification via constraint propagation and also from this completed mystification the applicable number of suggestions is removed. An intriguing problem is to probe the geste of different models on arbitrary mystifications. A simple script, available in the examinations brochure of the source law, has been written to time thesolution of models of arbitrary mystifications and compass the timings via matplotlib.

Two plots produced by this script punctuate the different geste of the constraint model and the integer programming model. The first plot has time on the perpendicular axis and the number of suggestions on the vertical axis. From this plot it seems that the constraint propogation algorithm finds mystifications with numerous or many suggestions easy. The delicate problems for the constraint solver appear to be clustered in the range of 20 to 35 suggestions. A different picture emerges with the direct programming model. With the same set of aimlessly generated mystifications it appears that the further suggestions the briskly the solver finds a result





VI. SYSTEM TESTING

System testing is done by running the Sudoku program using the Android platform. Several functions can be run on the system, including generating puzzles, checking the puzzle, andrunning the backtracking algorithm. Implementation of the operation carried out at Smartphone android with thefollowing specifications:

- CPU Quad-core 1.2 GHz Cortex-A7
- Memory RAM 2048 MB

Figure 1: The Sudoku generated is of Easy level

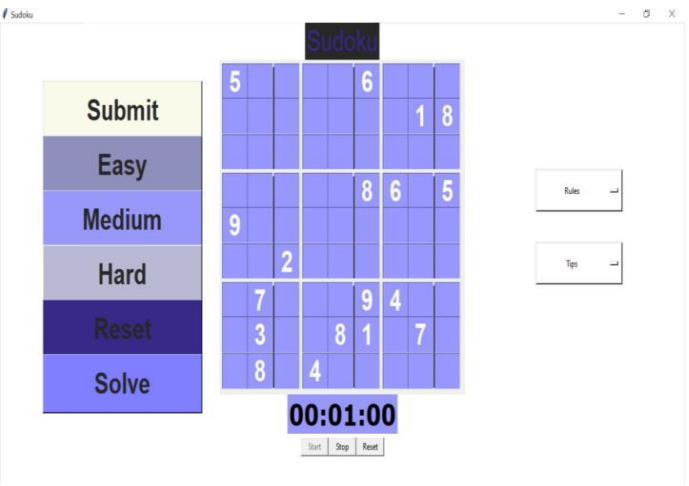


Figure 2: The Sudoku generated is of Medium level

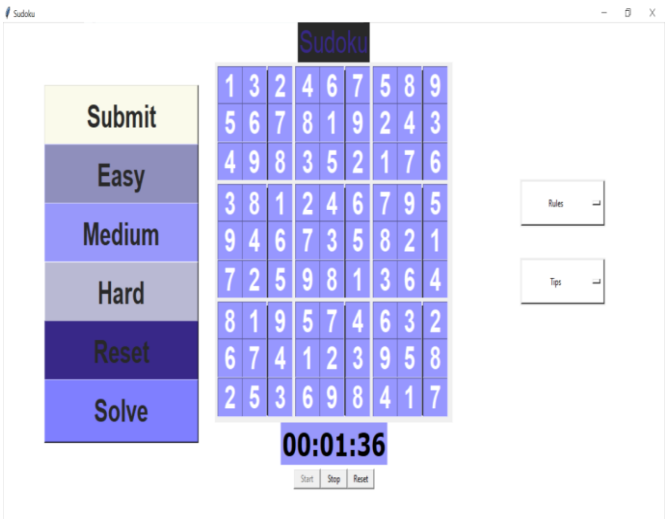


Figure 3: The Sudoku generated is of Hard level

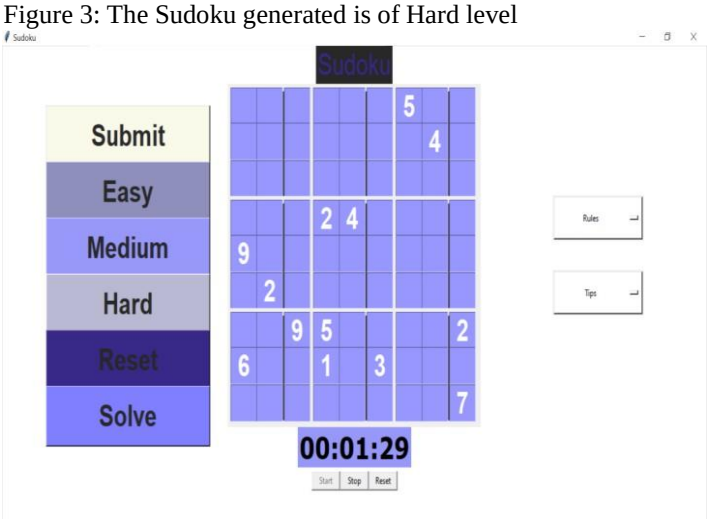
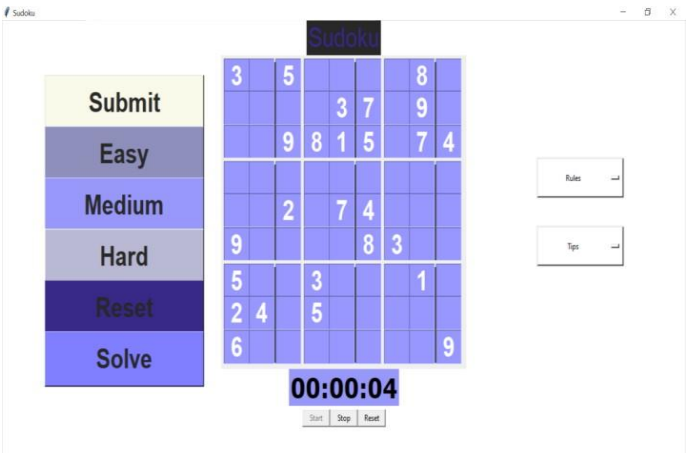


Figure 4,5 and 6: Backtracking Algorithm and Solve Puzzle With Manually



VII. CONCLUSION

When comparing sudoku working algorithms written in Java, the countermanding algorithm has been proven to be superior to both the constraint algorithm as well as the rule grounded algorithm. The constraint result was comparably hamstrung to such a degree that it should noway be considered other than for studying constraint programming in Java. The rule grounded algorithm was slower than countermanding indeed when only counting the inauguration and memory allocation, and as similar can noway be bettered to be faster than the backtracking result by adding further rules. This of course assuming the program is written in Java. It's possible there are ways to ameliorate the memory operation to such an extent it can outperform the countermanding algorithm, but we consider it doubtful. Since the countermanding algorithm is so effective as well as easy to apply, we consider chancing such a result gratuitous. The conclusion we've reached is that the countermanding algorithm is the stylish system for computationally working sudoku with the Java programming language.

VIII. REFERENCES

[1] [Bar08]] A. Bartlett, T. Chartier, A. Langville, T. Rankin. An Integer Programming Model for the Sudoku Problem, J. Online Math. & Its Appl., 8(May2008), May 2008

[2] [Bre79] Brelaz, D., New methods to color the vertices of a graph, Communications of the Assoc. of Comput. Machinery 22 (1979), 251-256.

[3] [Fel06] B. Felgenhauer, F. Jarvis. Enumerating possible Sudoku grids Online resource 2006 http://www.afjarvis.staff.shef.ac.uk/sudoku/

[4] [Kul10] O. Kullmann, Green-Tao numbers and SAT in LNCS (Springer), "Theory and Applications of Satisfiability Testing - SAT 2010", editors O.
Strichman and S. Szeider

[5] [Lew05] R. Lewis. Metaheuristics can solve Sudokupuzzles, Journal of Heuristics (2007) 13: 387-401

[6] [Lyn06] Lynce, I. and Ouaknine. Sudoku as a SAT problem, Proceedings of the 9th Symposium on Artificial Intelligence and Mathematics, 2006.

[7] [Sim05] H. Simonis. Sudoku as a Constraint Problem, Proceedings of the 4th International Workshop on Modelling and Reformuulating Constraint Satisfaction Problems. pp.13-27 (2005)

[8] [Var05] J. Gago-Vargas, I. Hartillo-Hermosa, J. Martin-Morales, J. M. UchaEnriquez, Sudokus and Groebner Bases: not only a Divertimento, In: Lecture Notes in Computer Science, vol. 4194. pp. 155-165. 200