A Review of Sudoku Solving using Patterns

Rohit Iyer*, Amrish Jhaveri*, Krutika Parab*

*B.E (Computers), Vidyalankar Institute of Technology
Mumbai, India

Abstract- A novel technique for very fast Sudoku solving using recognition of various patterns like Naked Singles, Hidden Singles, Locked Candidates, etc. is reviewed by conducting experiments and plotting the observations. Evaluation of the technique in solving random set of Sudoku puzzles collection show that the rate of solving can be greatly improved. However, only selected patterns are used for Sudoku solving in this review while even further improvement in solving rate may be possible if some more patterns could be detected and solved**.**

Index Terms—Sudoku, Naked Singles, Hidden Singles, Naked Pair, Hidden Pair, Locked Candidates

INTRODUCTION

Sudoku was developed by an American architect, Howard Garnes, in 1979, as a numerical combinatorial puzzle. The puzzle gained popularity in 2004, when Wayne Gould convinced The Times in London to publish it.[1] There are 6,670,903,752,021,072,936,960 possible combinations for completing a 9-by-9 Sudoku grid, but only 5,472,730,538 of them really count for different solutions and hence one needs a handful of lifetimes to solve all of them.[5]

Various Sudoku solving guides have explained the presence of a variety of patterns in Sudoku. These patterns are mostly aimed at, and also used by, humans to solve Sudoku by deducing hints. A machine or a processor is expected to solve the puzzle faster by rapid guessing and backtracking, rather than understanding every puzzle and solving it step by step.

This paper attempts to feed the tips meant for Human Sudoku solving by detecting different patterns, to a machine and also using its power to guess rapidly and backtrack, to gain a remarkable improvement in Sudoku solving than a simple Backtracking approach. Section II explains the basic rules of solving a Sudoku puzzle. Section III demonstrates the patterns used in our approach to solve a Sudoku. Section IV presents an experimental review to the theory of Sudoku solving and presents observations in a graphical and tabular manner.

THE RULES OF SUDOKU

The Sudoku rules [2] are explained in Figure 1. General Sudoku puzzles consist of a 9 x 9 matrix of square cells, some of which already contain a numeral from 1 to 9. The arrangement of given numerals when the puzzle is presented is called the starting point. In Figure 1, it contains 24 non-symmetrical given numbers, and the correct number for the other 57 points should be solved. The degree of difficulty varies with the number of given numerals and their placement. Basically, fewer given numerals means a higher number of combinations among which the solution must be found, and so raises the degree of difficulty. But, there are about 15 to 20 factors that have an effect on difficulty rating. A Sudoku puzzle is completed by filling in all of the empty cells with numerals 1 to 9, but no row or column and no 3 x 3 sub-block (the sub-blocks are bound by heavy lines in Figure 1 may contain more than one of any numeral. An example solution to the example Sudoku puzzle given in Figure 1 is shown in Figure 2. In this figure, the given numbers marked in bold-face.

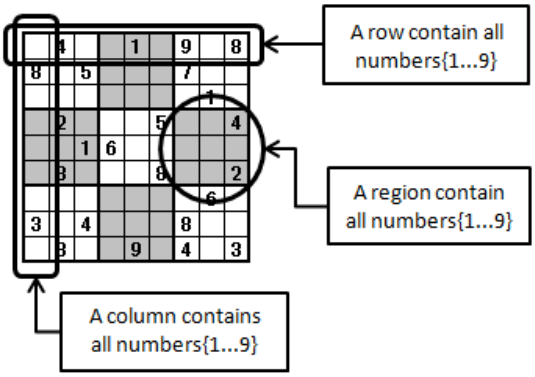

Figure 1: An example of Sudoku puzzles, 24 positions contain a given number, the other position should be solved.

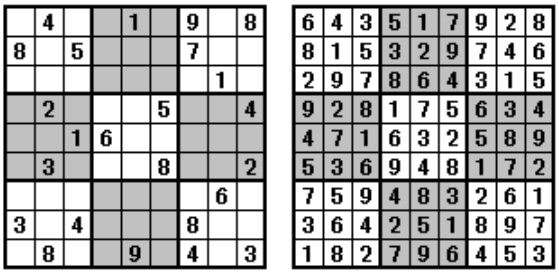

Figure 2: A solution for the Sudoku puzzles given in Figure 1. The given numbers marked in bold-face

PATTERNS IN SUDOKU

Our approach to fast Sudoku Solving employs detecting the patterns like Naked Singles, Hidden Singles, Naked Pairs, Hidden Pairs and Locked Candidates.

A. The Naked Singles Pattern

For any given Sudoku position, imagine listing all the possible candidates from 1 to 9 in each unfilled square. Next, for every square S whose value is v, erase v as a possible candidate in every square that is a buddy of S. The remaining values in each square are candidates for that square. When this is done, if only a single candidate v remains in square S, we can assign the value v to S. This situation is referred to as a "naked single".

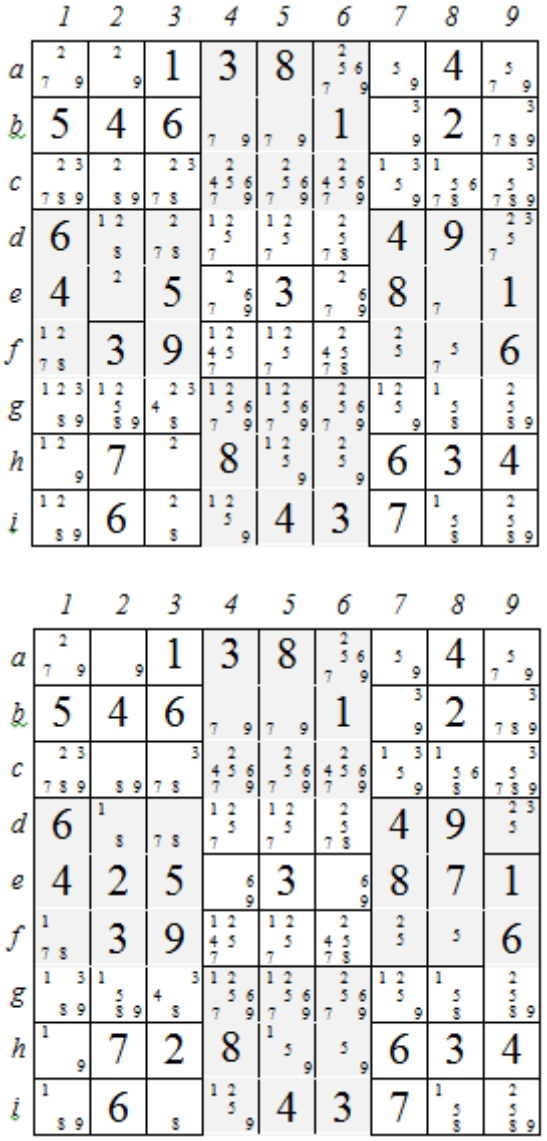

Figure 3: Candidate Elimination and Naked Singles

In the example on the first Sudoku in Figure 3 the larger numbers in the squares represent determined values. All other squares contain a list of possible candidates, where the elimination in the previous paragraph has been performed. In this example, the puzzle contains three naked singles at e2 and h3 (where a 2 must be inserted), and at e8 (where a 7 must be inserted).

Notice that once you have assigned these values to the three squares, other naked singles will appear. For example, as soon as the 2 is inserted at h3, you can eliminate the 2's as candidates in h3's buddies, and when this is done, i3 will become a naked single that must be filled with 8. The second Sudoku of Figure 3 shows the same puzzle after the three squares have been assigned values and the obvious candidates have been eliminated from the buddies of those squares.

B. The Hidden Singles Pattern

Sometimes there are cells whose values are easily assigned, but a simple elimination of candidates as described in the last section does not make it obvious. If you re-examine the situation on the left side of Figure 2, there is a hidden single in square g2 whose value must be 5. Although at first glance there are five possible candidates for g2 (1, 2, 5, 8 and 9), if you look in column 2 it is the unique square that can contain a 5. (The square g2 is also a hidden single in the block ghi123) Thus 5 can be placed in square g2. The 5 in square g2 is "hidden" in the sense that without further examination, it appears that there are 5 possible candidates for that square. To find hidden singles look in every virtual line for a candidate that appears in only one of the squares making up that virtual line. When that occurs, you've found a hidden single, and you can immediately assign that candidate to the square. To check your understanding, make sure you see why there is another hidden single in square d9 in Figure 3.The techniques in this section immediately assign a value to a square. Most puzzles that are ranked "easy" and many that are ranked "intermediate" can be completely solved using only these methods. The remainders of the methods that we will consider usually do not directly allow you to fill in a square. Instead, they allow you to eliminate candidates from certain squares. When all but one of the candidates have been eliminated, the square's value is determined.

C. The Locked Candidates Pattern

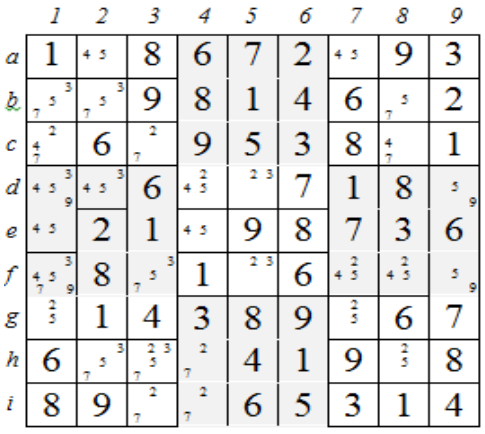

Figure 4: Locked Candidates Pattern Example

Locked candidates are forced to be within a certain part of a row, column or block. Sometimes you can find a block where

the only possible positions for a candidate are in one row or column within that block. Since the block must contain the candidate, the candidate must appear in that row or column within the block. This means that you can eliminate the candidate as a possibility in the intersection of that row or column with other blocks.

A similar situation can occur when a number missing from a row or column can occur only within one of the blocks that intersect that row or column. Thus the candidate must lie on the intersection of the row/column and block and hence cannot be a candidate in any of the other squares that make up the block.

Both of these situations are illustrated in Figure 4. The block def 789 must contain a 2, and the only places this can occur are in squares f 7 and f 8: both in row f. Therefore 2 cannot be a candidate in any other squares in row f, including square f 5 (so f 5 must contain a 3). Similarly, the 2 in block ghi456 must lie in column 4 so 2 cannot be a candidate in any other squares of that column, including d4.

Finally, the 5 that must occur in column 9 has to fall within the block def 789 so 5 cannot be a candidate in any of the other squares in block def 789, including f 7 and f 8.

D. The Naked and Hidden Pairs Pattern

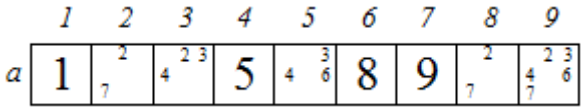

Figure 5: A Naked Pair Example

These are similar to naked singles, discussed in Section III.C, except that instead of having only one candidate in a cell, you have the same two candidates in two cells

Figure 5 shows how to use a naked pair. In squares a2 and a8 the only candidates that appear are a 2 and a 7. That means that 7 must be in one, and 2 in the other. But then the 2 and 7 cannot appear in any of the other squares in that row, so 2 can be eliminated as a candidate in a3 and both 2 and 7 can be eliminated as candidates in a9.

Hidden pairs are related to Naked Pairs in the same way that hidden singles are related to naked singles.

EXPERIMENT

We use the above explained patterns to solve bulk Sudoku puzzles using a Java program and plot the various observations using JFreeChart API [3].

E. Experiment Details

File used for solving Sudoku consists of 1000 Sudoku puzzles randomly selected by repeating 20 very hard Sudokus 50 times.

The puzzles in the file are arranged in the following format

1086720930098146020609538010060071800210987360801060000143890676000419088900653 14 (Sudoku in Figure 4)

i.e. 81 integers in every line and 0s indicating unsolved cells

Every single line denotes a separate puzzle to be solved
The file we use contain a variety of Sudokus which can be found here →
https://docs.google.com/file/d/0B3NNpon4i9lfb3NraG54VkM4V1k/edit?usp=sharing

We solve this file using a Java program that detects patterns like Naked Singles, followed by Hidden Singles, Naked Pair, Hidden Pairs, Locked Candidates and answers are obtained in the same format in a separate file.

F. Experiment Result

Our approach solved the 1000 puzzles in 6810 milliseconds on a machine with following configuration:

Processor Intel Core i3 RAM: 4 GB OS: Windows 7 Home Edition

Graphical plot of our results are presented below

Plot of Execution Time vs. Puzzle Number (Figure 6)
The plot below, contains only 25 puzzles for clarity
1 Unit= 10 milliseconds on Y axis

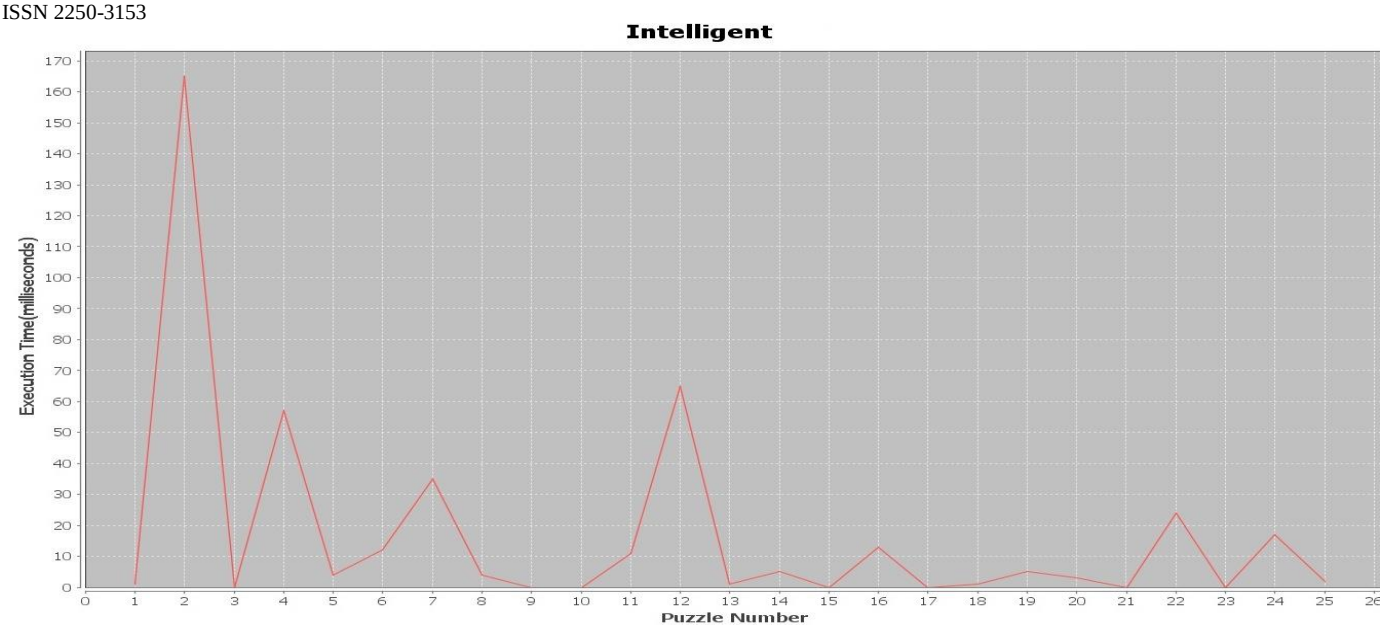

Figure 6: Graph : Execution Time vs. Puzzle Number
(1 Unit= 10 milliseconds steps on Y axis)

- Our approach required 6810 milliseconds time to solve 1000 Sudokus. (485 milliseconds to solve the 25 puzzles shown on the graph).
- On an average it required 40 milliseconds to solve a puzzle(evident from the graph)

Figure 7 shows another graph - the number of steps required in solving these 25 Sudokus using patterns.

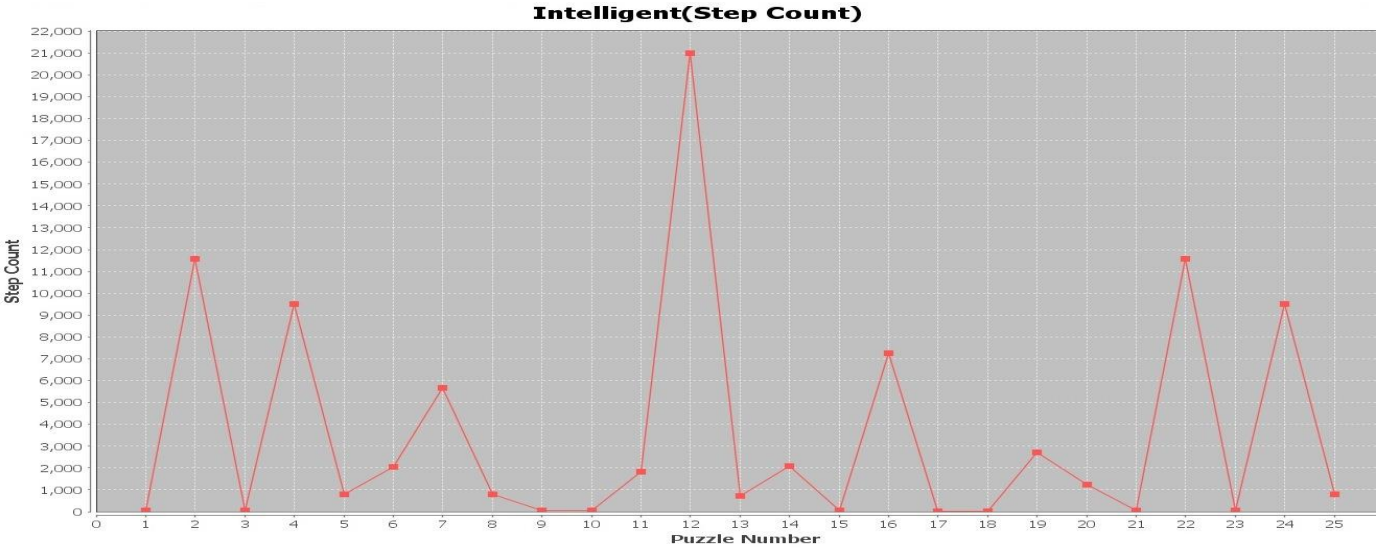

Figure 7: Graph : Step Count vs. Puzzle Number
(1 Unit= 1000 steps on Y axis)

We can observe that number of steps required is around 13000 on an average.

G. Comparison with Top Sudoku Solvers

An existing review of the performance of many Sudoku Solvers on the web [4] shows the following result.

Solver	Algorithm	Test Unique?	Language	Time (sec)
JSolve	backtracking	Yes	C	0.25
kudoku	backtracking	Yes	C	1.1
fast_solv_9r2	DLX	Yes	C	1.4
kudoku	backtracking	Yes	Java	2.0
kudoku	backtracking	Yes	JavaScript	6.3
kudoku	backtracking	Yes	Lua	7.5
sudoku-bb	Backtrack	No	Python	33.5
sudoku-gh	DLX	Yes	JavaScript	41.2
Peter Norvig's	backtracking	No	Python	147.4
kudoku	backtracking	Yes	Python	190.5
sudoku-pk	backtracking	No	Javascript	278.4
sudoku-aa	DLX	Yes	Python	514.0
sudoku_db	backtracking	No	Python	1915
Sudoku_bc	backtracking	Yes	Java	2180
Sudoku_dl	unknown	No	Java	2975
Sudoku_6l	brute-force	No	Java	25385

Figure 8: Image showing tabular comparison of Sudoku Solvers[4]

The above table shows performance of various solvers on the same set of 1000 puzzles collection.

CONCLUSION

Sudoku solving using patterns surely lowers the execution time required to solve huge number of Sudokus. The graphs and tables presented in this paper effectively prove the same.

FUTURE WORK

The motive of this paper was to review the performance of a brute-forced based Sudoku Solver using pattern matching.

This review used limited number of patterns that were comparatively easier to detect than some complex patterns. However, detecting more patterns may give considerably better results.

ACKNOWLEDGMENT

We would sincerely like to thank Dr. Sunil Patekar and Prof. Mahesh Bhave, for their constant support and inspiration in this project.

REFERENCES

[1] http://www.sudoku-tips.com/about_sudoku.php

[2] Sato, Yuji "Solving Sudoku with Genetic Operations that Preserve Building Blocks," *Computational Intelligence and Games (CIG), 2010 IEEE Symposium.*

[3] "JFreeChart-API" http://www.jfree.org/jfreechart/download.html

[4] "WordPress-WebBlog" http://attractivechaos.wordpress.com/2011/06/19/an-incomplete-review-of-sudoku-solver-implementations/

[5] http://www.afjarvis.staff.shef.ac.uk/sudoku/

AUTHORS

First Author – Rohit Iyer, B.E (Computers), Vidyalankar Institute of Technology, Mumbai.
Email: rohit.iyer@vit.edu.in
Second Author – Amrish Jhaveri, B.E (Computers), Vidyalankar Institute of Technology, Mumbai.
Email: amrish.jhaveri@vit.edu.in
Third Author – Krutika Parab, B.E (Computers), Vidyalankar Institute of Technology, Mumbai.
Email: krutika.parab@vit.edu.in

Correspondence Author – Rohit Iyer
rohitiyer2109@gmail.com, rohit.iyer@vit.edu.in,
+919022462764