Scientific Journal Impact Factor (SJIF) : 4.364
International Journal of Modern Trends in Engineering and Research
ISSN (ONLINE) : 2349 - 9745 ISSN (PRINT) : 2393 - 8161

SOLVING SUDOKU USING BACKTRACKING ALGORITHM

Charu Gupta[1]
[1]*ME-Digital Communication, Department of Electronics & Communication Engineering, MBM Engineering College, JNV University, Jodhpur-342011*

Abstract: Sudoku puzzles appear in magazines, newspapers, web pages and even books on a daily basis. In fact, millions of people around the world know how to play Sudoku. However, the science behind this game is much more complex than it looks. That is why many re- searchers have put a notable amount of effort to generate efficient algorithms to solve these puzzles. Some researchers might even suggest that an algorithm to solve Sudoku games without trying a large amount of permutations does not exist. This paper describes the development and implementation of a Sudoku solver using MATLAB.
Keywords -Backtracking algorithm, Matlab .

I. INTRODUCTION

Games and puzzles have been a platform for application of mathematics, artificial intelligence and other various techniques. The past years have brought into attention a very popular puzzle called Sudoku. The typical Sudoku puzzle grid is a 9-by-9 cells into nine 3-by-3 squares. The rules of the game are: Every row, column, and square (of 3-by-3) must be filled with each of the numbers 1 till 9 and that number cannot appear more than once in any of the row, column, or square. The initial grid is populated with a few digits, known as *clues*. In contrast to magic squares and other numeric puzzles, no arithmetic is involved; the elements in a Sudoku grid could just as well be letters of the alphabet or any other symbols.

Sudoku has led other researchers to some advances in algorithm design and implementation. This work was largely motivated by the interesting mathematical concepts behind it. This paper describes the development of a Sudoku solver using MATLAB. some authors are proposing a search based solution by using some heuristic factors in a modified steepest hill ascent. Some researchers, are suggesting the design of a genetic algorithm by representing the puzzle as a block of chromosomes, more precise as an array of 81 integers. Any crossover appears between the 3x3 grids and any mutations occur only inside the 3x3 grids. Geem is proposing a Sudoku solver model based on harmony search that mimics the characteristics of a musician. Santos-Garcia and Palomino are suggesting a method for solving Sudoku puzzles using simple logic with rewriting rules to mimic human intelligence.

Others are suggesting neural networks by modeling an energy driven quantum (Q'tron) neural network to solve the Sudoku puzzles. Barlett and Langville are proposing a solution based on binary integer linear programming (BILP). To formulate in a simple way the method, we can say that it uses binary variables to pick a digit for any cell in a Sudoku puzzle. Our MATLAB program uses only one pattern—singletons—together with a basic computer science technique, recursive backtracking.

II. BACTRACKING ALGORITHM

Backtracking is a *depth-first* search (in contrast to a *breadth-first* search), because it will completely explore one branch to a possible solution before moving to another branch. Although it has been established that approximately 6.67 x 10^{21} final grids exist, a brute force algorithm can be a practical method to solve Sudoku puzzles. A brute force algorithm visits the empty cells in some order, filling in digits sequentially, or backtracking when the number is found to be not valid. Briefly, a

Volume 04, Issue 12, [December– 2017] ISSN (Online):2349–9745; ISSN (Print):2393-8161

program would solve a puzzle by placing the digit "1" in the first cell and checking if it is allowed to be there. If there are no violations (checking row, column, and box constraints) then the algorithm advances to the next cell, and places a "1" in that cell. When checking for violations, if it is discovered that the "1" is not allowed, the value is advanced to "2". If a cell is discovered where none of the 9 digits is allowed, then the algorithm leaves that cell blank and moves back to the previous cell. The value in that cell is then incremented by one. This is repeated until the allowed value in the last (81st) cell is discovered.

Advantages of this method are:

- A solution is guaranteed (as long as the puzzle is valid).
- Solving time is mostly unrelated to degree of difficulty.
- The algorithm (and therefore the program code) is simpler than other algorithms, especially compared to strong algorithms that ensure a solution to the most difficult puzzles.

The disadvantage of this method is that the solving time may be comparatively slow compared to algorithms modeled after deductive methods

.

III. SOLVING SUDOKU WITH RECURSIVE BACKTRACKING

An easy puzzle could be defined as one that can be solved by just inserting the singletons. The input array for the puzzle shown in Figure1 is generated by the MATLAB statement

```
X = diag(1:4)
```

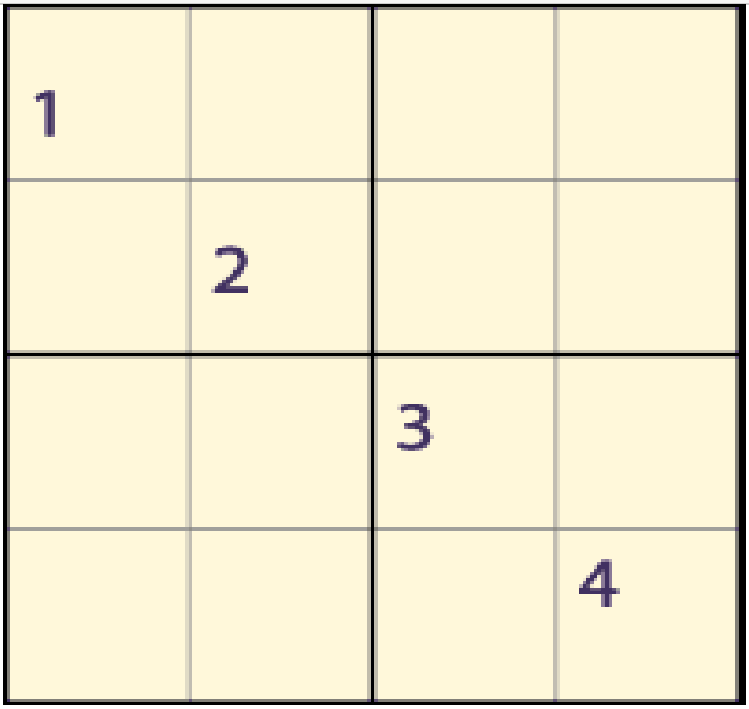

Figure1: shidoku(diag(1:4))

Because there are no singletons in this puzzle (Figure 2), we will use recursive backtracking. We select one of the empty cells and tentatively insert one of its candidates. We have chosen to consider the cells in the order implied by MATLAB one-dimensional subscripting, X(:), and try the candidates in numerical order. So we insert a “3” in cell (2,1), creating a new puzzle (Figure 3). We then call the program recursively.

Volume 04, Issue 12, [December– 2017] ISSN (Online):2349–9745; ISSN (Print):2393-8161

1	3 4	2 4	2 3
3 4	2	1 4	1 3
2 4	1 4	3	1 2
2 3	1 3	1 2	4

Figure 2: The candidates. There are no singletons.

1			
3	2		
		3	
			4

Figure 3:Tentatively insert a "3" to create a new puzzle. Then backtrack.

This puzzle is easy; the result is shown in Figure 4. However, this solution depends upon the choices that we made before the recursive call. Other choices could produce different solutions. For this simple diagonal initial condition, there are two possible solutions, which happen to be matrix transposes of each other. Since the solution is not unique, the grid in Figure 1 is not a valid puzzle.

1	4	2	3
3	2	4	1
4	1	3	2
2	3	1	4

Figure 4. The resulting solution. This solution is not unique; its transpose is another solution.

Volume 04, Issue 12, [December– 2017] ISSN (Online):2349–9745; ISSN (Print):2393-8161

IV. THE SUDOKU-SOLVING ALGORITHM

Our MATLAB program involves just four steps:

1. Fill in all singletons.
2. Exit if a cell has no candidates.
3. Fill in a tentative value for an empty cell.
4. Call the program recursively.

The key internal function is candidates. Each empty cell starts with z = 1:9 and uses the numeric values in the associated row, column, and block to zero elements in z. The non zeros that remain are the candidates.

For example, consider the (1,1) cell in Figure 5. We start with

```
z = 1 2 3 4 5 6 7 8 9
```

The values in the first row change z to

```
z = 1 0 0 0 5 6 7 8 9
```

Then the first column changes z to

```
z = 1 0 0 0 5 0 7 0 9
```

The (1,1) block does not make any further changes, so the candidates for this cell are

```
C{1,1} = [1 5 7 9]
```

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

Figure 5. An example puzzle

V. CONCLUSION

Implementing a Sudoku solver in MATLAB allowed us to use many of the tools and built-in functions. Nowadays, there are extensive studies regarding the mathematics of Sudoku as well as many different algorithms to solve the puzzles. Some algorithms are designed to solve the puzzles as quick as possible while some others are designed to solve them as efficiently in terms of computational power

Volume 04, Issue 12, [December– 2017] ISSN (Online):2349–9745; ISSN (Print):2393-8161

and memory. However, finding a suitable algorithm to solve any particular Sudoku game proved to be difficult. The combination of a simple, yet effective algorithm with a graphical user interface allowed us to generate games, solve them and verify the given solutions in a simple and quick way.

REFERENCES

[1] A.C. Bartlett and A.N. Langville. An integer programming model for the Sudoku problem. Preprint, available at http://www. cofc. edu/~ langvillea/Sudoku/sudoku2. pdf, 2006.
[2] JF Crook. A pencil-and-paper algorithm for solving Sudoku puzzles. Notices of the AMS, 56(4):460{468, 2009.
[3] Z. Geem. Harmony search applications in industry. Soft Computing Applications in Industry, pages 117{134, 2008.
[4] R.C. Green II. Survey of the Applications of Arti_cial Intelligence Techniques to the Sudoku Puzzle. 2009.
[5] SK Jones, PA Roach, and S. Perkins. Construction of heuristics for a search-based approach to solving Sudoku. Research and Development in Intelligent Systems XXIV, pages 37{49, 2008.
[6] T. Mantere and J. Koljonen. Solving, rating and generating Sudoku puzzles with GA. In Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, pages 1382{1389. Ieee.
[7] Mathworks. Creating graphical user interfaceshttp://www.mathworks.com/ help/techdoc/creating guis/bqz79mu.html, March 2011.

