International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 4, Issue 5(2), September - October 2015 ISSN 2278-6856

Study of Brute Force and Heuristic Approach to Solve Sudoku

Taruna Kumari[1], Preeti yadav[2], Lavina[3]

[1] YMCA University of Science and technology, Faridabad

[2] BRCM College of Engineering, Bahal(MDU)

[3] Govt. Girls College Sector-14, Gurgaon

Abstract

Couple of decades back, there was a tremendous development in the field of algorithms, which were aimed at finding efficient solutions for widespread applications. The benefits of these algorithms were observed in their optimality and simplicity with speed. In the last decade, solving the Sudoku puzzle has become every one's passion. The simplicity of puzzle's structure and the low requirement of mathematical skills caused people to have enormous interest in accepting challenges to solve the puzzle. Therefore, developers have tried to find algorithms in order to generate the variety of puzzles for human players so that they could be even solved by computer programming. Many of the algorithms were readdressed to solve the problem of finding solution of Sudoku. In this paper, we have applied backtracking, a brute force approach and rule based, a heuristic approach to solve Sudoku.

Keyword: Backtracking, Brute-Force, Heuristic, Rule-Based.

1. INTRODUCTION

Currently, Sudoku puzzles are becoming increasingly popular among the people all over the world. The game has become popular now in a large number of countries and many developers have tried to generate even more complicated and more interesting puzzles. Today, the game appears in almost every newspaper, in books and in many websites. It is therefore of interest to study how to solve, generate and rate such
puzzles by the help of computer algorithms. In this paper two very popular approaches brute force and heuristic are studied to solve Sudoku.

2. SUDOKU FUNDAMENTALS

Sudoku is a popular logic-based puzzle where the objective is to fill a 9×9 grid with numbers, with a subset of the solution already given, so that each column, each row, and each of the nine 3×3 sub-grids contains all of the numbers from 1 to9.

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

Figure 1 a Basic instance of a 9×9 Sudoku

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

Figure 2 a solution of the instance given in figure1

A Sudoku can vary in difficulty depending of various aspect including number of given numbers. Figure 1 gives an example of a basic Sudoku instance with 30 given numbers and the solution is presented in Figure 2 with filled in numbers displayed in red.

3. SOLVING TECHNIQUES AND ALGORITHMS

Sudoku solving is a research area in computer science and mathematics, with areas such as solving, puzzle difficulty rating and puzzle generation. Two major classes of algorithm to solve Sudoku are brute force and heuristic. Some of the existing solving algorithms are backtrack [6], rule-based [6], cultural genetic with variations [8], and Boltzmann machines [9].

3.1. Brute-Force Approach

One of the better-known algorithms that are used when solving Sudoku on a computer is called brute-force. The brute-force algorithm is sometimes called an exhaustive search algorithm and it has become famous in computer science because of its striking characteristics:" It always finds a solution because it will try all possible solutions hence it is not considered to be very effective [1]".

A Sudoku solver that uses brute-force will visit all the empty cells and try to fill it with a number from the available choices. If no violations are found after an insertion it will continue to the next empty cell and start over. As soon as a violation to the Sudoku rules is found the algorithm will backtrack and increment the previous cell. [2].In below example we have taken a sample 4 × 4

Sudoku and try to solve it using above approach. The search tree is shown in figure 3.

Algorithm: The basic algorithm of backtracking approach is:

```
if(no more choices) // BASE CASE
    return (conf is goal state);
for(all available choices){
        try one choice c;
if(solve(conf with choice c made)) return true;
unmake choice c;
}
return false; //tried all choices, no solution found
```

This basic backtracking algorithm is applied to solve Sudoku as:

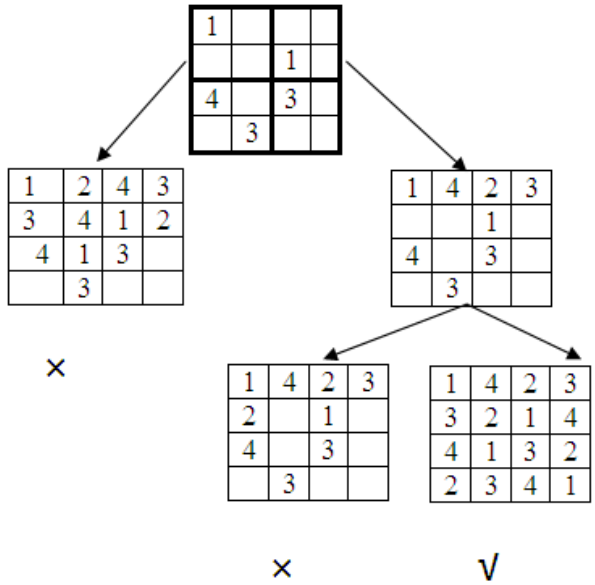

Figure 3 Comprised version of a search tree of a backtracking algorithm on a 4 ×4 Sudoku puzzle

```
boolean solveSudokuBacktrack(Grid grid)
{
 int row, col;
 if(!FindUnassignedField(grid, row, col)) // store
coordinates in row, col
return true; // all fields successfully assigned
 for(int num = 1; num <= 9; num++) // possible choices 1-
9
{
 if(isValidChoice(grid, row, col, num)) // if num looks
good
 {
 grid(row, col) = num; // try assign num
 if(solveSudoku(grid))
   return true; // try solving rest recursive
 grid(row, col) = UNASSIGNED; // undo and try with
next num
 } }
 return false; // no valid choice found, trigger backtracking
}
```

First of all this algorithm will try to find an unassigned position: if not found means all are filled and will return true and if found then try to fill with available choices from 1to 9 for that position. All numbers from 1 to 9 are checked in that particular row and column .If a number is found that does not exist already in that particular row and column , the number is filled in that position. After assigning, it will try to solve rest problem. If solved the algorithm will return true otherwise **backtracking** is applied. The previous position is unassigned and another number is searched for that position. The algorithm continues in this way.

The brute-force algorithm fills in the missing numbers without any sense of logic and check afterwards if it was a valid placement or not. These algorithms use exhaustive search to solve Sudoku, which means a large amount of backtracking and guessing. **An advantage with this method is that it will always find a solution.**

However this also means that it will try all different solutions and since there are 6.67 * 10^21 possible Sudoku it could take a very long time in the worst-case scenario. [3][4].

3.2. Rule based approach

This algorithm builds on a heuristic for solving Sudoku puzzles. The algorithm consists of testing a puzzle for certain rules that fills in squares or eliminates candidate numbers. Those rules are listed below:

- **Naked Single**: This means that a square only have one candidate number.
- **Hidden Single**: If a region contains only one square which can hold a specific number then that number must go into that square.
- **Naked pair**: If a region contains two squares which each only have two specific candidates. If one such pair exists, then all occurrences of these two candidates may be removed from all other squares in that region. This concept can also be extended to three or more squares.
- **Hidden pair**: If a region contains only two squares which can hold two specific candidates, then those squares are a hidden pair. It is hidden because those squares might also include several other candidates. Since these squares must contain those two numbers, it follows that all other candidates in these two squares may be removed. Similar to naked pairs, this concept may also be extended to three or more squares.
- **Guessing (Nishio)**: The solver finds an empty square and fills in one of the candidates for that square. It then continues from there and sees if the guess leads to a solution or an invalid puzzle. If an invalid puzzle comes up the solver return to the point where it made its guess and makes another guess. The reader might recognize this approach from the backtrack algorithm and it is indeed the same method. The same method for choosing which square to begin with is also used.

The human uses this technique to solve a Sudoku. This algorithm can also be described as rules that each consists of a pattern followed by an action [5]. If a pattern is matched when performing a technique during a Sudoku session, the technique's corresponding action will be executed before proceeding to the next technique. If an action is performed it will either place a number in a specific cell or remove one or more candidates from one or more cells. The above rules with their pattern and action are summarized in below table:

Table1: Rules with their pattern and actions

Technique	Pattern	Action
Naked Single	A cell only has single candidate left	Places that single candidate number in that cell
Hidden Single	A region only has a single cell in which it can place a remaining number.	Place that number in that single cell.
Naked Pair/Triple	Two/three cells in the same region have a union of two/three candidates in common.	The candidates in this union are removed from the same unit.
Hidden Pair/ Triple	Two or three cells in the same region have the last remaining two/three candidates for that region in error.	Any candidates that are not a member of the pair / triple in the found cells are removed.
Naked Quad	Four cells in the same region have a union of four candidates in common.	The candidates in this union are removed from the same unit.
Box Line Reduction	The only occurrences of a candidate cells in a row or column in the same box.	Removal all other occurrences in this candidate in the rest of this box.
Pointing Pairs /Triples	The only occurrences of a candidate in a box are aligned in cells on a row or column.	Remove all other occurrences of this candidate in the rest of this row or column.

Algorithm: In rule based approach first we will apply simple rules. If simple rules failed then advanced rules are applied. In this way the algorithm will return the solution puzzle if succeed. If no rule works backtracking is applied.

The psuedo code for this algorithm is presented below:

```
Puzzle Solve Sudoku Rulebased(puzzle)
while(true){
//Apply the rules and restart the loop if the rule
//was applicable. Meaning that the advanced rules
//are only applied when the simple rules failes.
//Note also that applyNakedSingle/Tuple takes a reference
//to the puzzle and therefore changes the puzzle directly
if(applyNakedSingle(puzzle))
continue
if(applyNakedTuple(puzzle))
continue
break
}
//Resort to backtrack as no rules worked
```

A human rule based algorithm uses a series of techniques that are based upon the given restrictions of Sudoku. The techniques are then applied by the solver in order (from easiest to hardest) to find a possible number placement. One rule might be to check for any naked singles in a row, meaning searching for locations where one and only one number can be placed according to the Sudoku restrictions. A problem with this algorithm is that it does not guarantee that a solution will be found because the rules are not exhaustive, i.e. there are Sudoku that cannot be solved using only rules [3].

4. CONCLUSION

This study has shown that the rule based algorithm is a feasible method to solve any Sudoku puzzles. The algorithm is also an appropriate method to find a solution faster and more efficient compared to the brute force algorithm. The proposed algorithm is able to solve such puzzles with any level of difficulties in a short period of time (less than one second).

The brute force algorithm seems to be a useful method to solve any Sudoku puzzles and it guarantee to find at least one solution. The algorithm does not adopt intelligent strategies to solve the puzzles. This algorithm checks all possible solutions to the puzzle until a valid solution is found which is a time consuming procedure resulting an inefficient solver. As it has already stated the main advantage of using the algorithm is the ability to solve any puzzles and a solution is certainly guaranteed.

Further research needs to be carried out in order to optimize the rule based algorithm.

References

[1]. Kann V. Föreläsning 7, Metod 2: Totalsökning. [Internet]. c2012 [cited 2013 Apr 10].Available from: http://www.csc.kth.se/utbildning/kth/kurser/DD1352/adk12/schema/ADK12-F7.pdf

[2]. Moler C. Cleve's Corner: Solving Sudoku with MATHLAB. [Internet]. 2009 [cited 2013 Apr 9].Available from:http://www.mathworks.se/company/newsletters/news_notes/2009/clevescorner.html

[3]. Felgenhauer B, Jarvis F. Enumerating possible Sudoku grids. [Internet]. 2005 [cited 2013Mar 21]. Available from: http://www.afjarvis.staff.shef.ac.uk/sudoku/

[4]. Felgenhauer B, Jarvis F. Mathematics of Sudoku I. [Internet]. 2006 [cited 2013 Mar 21].Available from: http://www.afjarvis.staff.shef.ac.uk/sudoku/felgenhauer_jarvis_spec1.pdf

[5]. Crook J. F. A Pencil-and-Paper Algorithm for Solving Sudoku Puzzles. Notices of the

AMS.[Internet]. 2009 [cited 2013 Feb 1]; 56(4):460-468. Available from:http://www.ams.org/notices/200904/200904-full-issue.pdf

[6]. Astraware Limited. Techniques For Solving Sudoku. [homepage on the Internet]. 2008 [cited 2012 Mar 8]. Available from:, Web site:http://www.sudokuoftheday.com/pages/techniques-overview.php

[7]. Ekeberg. Boltzmann Machines. [homepage on the Internet].2012 [cited 2012 Mar 8]. Available from:, Web site: http://www.csc.kth.se/utbildning/kth/kurser/DD2432/ann12/forelasningsanteckningar/07-boltzmann.pdf

[8]. Marwala T. Stochastic Optimization Approaches for Solving Sudoku. [homepage on the Internet]. 2008 [cited 2012 Mar 8]. Available from:, Web site:http://arxiv.org/abs/0805.0697

[9]. Cazenave Cazenave T. A search based Sudoku solver. [homepage on the Internet]. No date [cited 2012 Mar 13]. Available from: Université Paris, Dept. Informatique Website. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.459&rep=rep1&type=pdf