Recursive Backtracking for Solving 9*9 Sudoku Puzzle

Dhanya Job and Varghese Paul

Abstract— Nowadays Sudoku is a very popular game throughout the world and it appears in different medias, including websites, newspapers and books. There are numerous methods or algorithms to find Sudoku solutions and Sudoku generating algorithms. This paper explains possible number of valid grids in a 9*9 sudoku and developed a programming approach for solving a 9*9 sudoku puzzle and the results have been analysed in accordance with various number of clues for 9*9 sudoku.

Keywords-- Back Tracking, Enumeration, Pseudocode, Sudoku Puzzle.

I. INTRODUCTION

SUDOKU Puzzle was invented by American Howard Garns in 1979. [1] In the year 1984 Maki Kaji of Japan has published in the magazine of his puzzle company 'Nikoli' and gave the name Sudoku to the puzzle game, which means "Single Numbers". The puzzle became popular in Japan and New Zealander Wayne Gould, wrote a computer program that would generate Sudokus. It has become a regular puzzle game in many leading newspapers and magazines around the world and is enjoyed by the people globally. The name Sudo is derived from 'Sujiwadokushionikagiru', which is the Japanese word means the digits must remain single.

II. THEORETICAL BACKGROUND

A. Sudoku Definition

A Sudoku consists of a 9×9 square grid containing 81 cells.[2]The grid is subdivided into nine 3×3 blocks. Some of the 81 cells are filled in with numbers from {1,2,3,4,5,6,7,8,9}. These filled-in cells are called givens or clues. The goal of the player is to fill in the whole grid using the nine digits so that each row, column and block contains each number exactly once and this constraint on the rows, columns, and blocks is known as One Rule. The solution of a Sudoku Puzzle requires that every row, column and block contain all the numbers in the set[1,2,……9] and every cell will be occupied by only one number.

Most frequently a 9×9 grid made up of 3×3 sub grids, starting with several numerals given in some of the cells ("givens"). Each row, column, and region or Block must contain only one instance of each numeral. Fig 1 is an example for solved Sudoku. A unique solution exists for a Sudoku Puzzle which can be determined by solving for all possible solutions.

Fig. 1: Solution to Sample Sudoku Puzzle

\[
\begin{array}{cccc|cccc|ccc}
5 & 3 & 4 & 6 & 7 & 8 & 9 & 1 & 2 \\
6 & 7 & 2 & 1 & 9 & 5 & 3 & 4 & 8 \\
1 & 9 & 8 & 3 & 4 & 2 & 5 & 6 & 7 \\
8 & 5 & 9 & 7 & 6 & 1 & 4 & 2 & 3 \\
4 & 2 & 6 & 8 & 2 & 3 & 7 & 9 & 1 \\
7 & 1 & 3 & 9 & 2 & 4 & 8 & 5 & 6 \\
9 & 6 & 1 & 5 & 3 & 7 & 2 & 8 & 4 \\
2 & 8 & 7 & 4 & 1 & 9 & 6 & 3 & 5 \\
3 & 4 & 5 & 2 & 8 & 6 & 1 & 7 & 9 \\
\end{array}
\]

B. Sudoku grids

Felgenhauer and Jarvis [6] enumerated total number of valid Sudoku solutions. The mathematical derivation can be shown as follows. Consider a 9* 9 sudoku with 9 blocks B1,B2,…,B9. The Block representation is shown in Table 2.1.

Table 2.1: Block Representation

\[
\begin{array}{ccc}
B1 & B2 & B3 \\
B4 & B5 & B6 \\
B7 & B8 & B9 \\
\end{array}
\]
Take the top left Block which is B1 in the standard form. So we can arrange numbers from 1 to 9 in 9! ways. So N1 will be N9/9!. Possibilities for B2 and B3. Consider the top row of Sudoku grid other than the first block. The possible ways for arranging the numbers are given below.

Table 2.2: Solutions for First Row

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>(4,5,6)</td>
<td>(7,8,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5,7)</td>
<td>(6,8,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5,8)</td>
<td>(6,7,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,5,9)</td>
<td>(6,7,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,6,7)</td>
<td>(5,8,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,6,8)</td>
<td>(5,7,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4,6,9)</td>
<td>(5,7,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5,6,7)</td>
<td>(4,8,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5,6,8)</td>
<td>(4,7,9)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5,6,9)</td>
<td>(4,7,8)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since the first Block B1 is in standard form, as per the solution of table 2.1 the top row of Sudoku Blocks B2 and B3 can be filled as follows

Table 2.3: Solution for First Row when B1 is in Standard Form

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

In total we have 2*(3!)^6 + 18 *3 *(3!)^6
= 56 *(3!)^6
i.e. 2612736 possible combinations and Number of possibilities for top three rows of Sudoku grid= 9!* 2612736
= 948109639680. Same will be true for the Blocks B2 to B6 and B7 to B9. Number of Sudoku grids= (948109639680)^9 / (9!)^9
which is equal to 6.657*10^21

V. SOLVING SUDOKU

One of the popular solution for Sudoku game is based on backtracking and linking algorithms. Backtracking, is a common technique in artificial intelligence, is a brute force search technique that explores a constrained set of possibilities until solution is reached[7].

Backtracking is a systematic method to iterate through all the possible arrangements of a search space[8]. It is a general method or technique which must be customized for single application. In the general case, we will model our solution as a vector x = (x1; x2; ...; xn), where each element ai is selected from a finite ordered set Si. Such a vector might represent an arrangement where x contains the ith element of the permutation. [9] Or the vector might represent a given subset S, where xi is true if and only if the ith element of the set is in S.

Pseudo Code for BackTracking

Pseudo code for solving Sudoku using Backtracking is given below.

```c
bool Solve(configuration conf)
{
    if (no more choices) // BASE CASE
        return (conf is goal state);
    for (all available choices) {
        try one choice c;
        // solve from this point., if works out, you're done.
        if (Solve(conf with choice c made)) return true;
        unmake choice c;
    } return false; // tried all choices and no solution is found
}
```

VI. IMPLEMENTATION

The above defined Pseudo Code have been used to solve a 9*9 sudoku in JAVA.

```java
boolSolveSudoku(Grid<int>&grid)
{
    int row, col;
    if (!FindUnassignedLocn(grid, row, col))
        return true; // all locations successfully assigned.
    for (intnum = 1; num<= 9; num++) { // options are 1-9
        if (!Conflicts(grid, row, col, num)) { // if # looks ok
            grid(row, col) = num; // try assign #
            if (SolveSudoku(grid)) return true;
            if (SolveSudoku(grid)) return true;
        }
    }
    return false; // this triggers backtracking from earlier decisions
}
```

VII. ANALYSIS

To get an idea of how backtracking algorithm performs it is suitable to plot solving times. The backtracking algorithm is an efficient algorithm in its performance. The backtrack algorithm was tested on 30 puzzles with different number of clues in the time limit of 20 seconds.

In figure 2 the Number of clues is plotted against the Solving time. From the graph, it can be seen that time for solving seems to decrease at approximately exponential rate as number of clues increases. It can also be observed that the solving times is around time limit of 20s. This probably means that the time for solving would have continued to increase for the other unsolved puzzles.

Fig. 6.1
Table 6.1. A plot showing the results of the backtracking approach. Note that the y-axis represents solving times and the x-axis represents number of clues.

Table 6.1: Experimental Results

<table>
<thead>
<tr>
<th>Number of Puzzles</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Clues</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>Solving Time</td>
<td>22.2</td>
<td>21.1</td>
<td>20.1</td>
<td>19.98</td>
<td>19.85</td>
</tr>
</tbody>
</table>

Among the 30 puzzles we have taken for doing the experiment, it has been found out the average solving time is 20.365 s.

VIII. CONCLUSION AND FUTURE WORK

In this paper we used backtracking algorithm for solving Sudoku puzzles with different number of clues. We have implemented the algorithm using Java and also compared the results. The results reveals that the application program developed by us performs well for solving 30 puzzles of size 9*9 with various clues. Future work includes studying neural network and developing an algorithm using Neural network to solve a 9*9 sudoku puzzle. Overall there is space for larger studies with more algorithms for Solving Sudoku Puzzles of other sizes. Also implementation of Sudoku Puzzles in Data security remains as a future work.

REFERENCE