AAA Project

Solving Sudoku using the backtracking algorithm

800107 - Daniel Ng See Cheong
839521 - Uzair Moolla

September 22, 2016

1 Introduction

Sudoku is a numerical based logic puzzle game. The idea is to fill in an $n \times n$ grid with numbers in a specific manner. The most commonly known grid usually consists of square blocks with 3 rows and 3 columns. These individual blocks are then arranged in a similar manner again, with 3 blocks along rows and 3 along columns, producing a 9×9 matrix. An example of this described matrix can be seen in Figure 1. We need to fill each of the blocks with one of these numbers $1, 2, 3, 4, 5, 6, 7, 8, 9$. The rules for filling these blocks are as follows:

- Within each 3×3 block we can only have single occurrence of each number.
- Along each 9 block row we can only have a single occurrence of each number.
- Along each 9 block column we can only have a single occurrence of each number.

An example of a Sudoku Puzzle completed using the rules stated above can be seen in Figure 2.

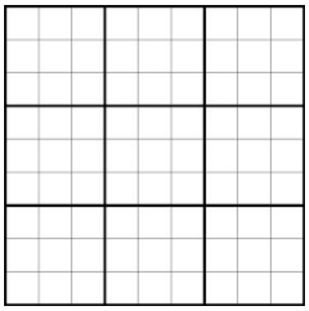
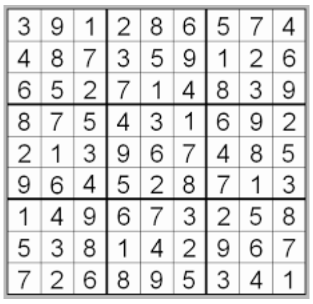


Figure 1: Sample of a 9×9 Blank Sudoku Board



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

Figure 2: Sample of a 9×9 Completed Sudoku Board

Solving sudoku puzzles are a fun exercise for many. However we would like to find a suitable algorithm which can be used to solve sudoku puzzles correctly as well as efficiently. There are various possible methods which can be used to solve sudoku puzzles. We will be focussing on the *Backtracking Algorithm* in our experiments.
#Fill in facts#

2 Aims

We are going to use the backtracking algorithm to solve sudoku puzzles of size $n \times n$. We will attempt to solve these sudoku puzzles accurately and in as little time as possible. Once we have the algorithm working correctly we will perform both empirical as well as theoretical analysis on it. We aim to find the best, average and worst case complexities of the algorithm being used. These results will be obtained through both methods of analyses (empirical as well as theoretical) and correlations between the results will be obtained.
We will create a database of unsolved puzzles and another with their corresponding solutions. These lists will be used for all analysis done as well as ensuring that the solutions found to each of the puzzles are correct.
#add some stuff#

3 Summary of Theory

3.1 Backtracking Algorithm

In general backtracking algorithms are mainly used to solve NP-complete computational problems. Simply put, the backtracking algorithm starts with the selection of one possible move out of all available moves and will try solving the problem with the selected move. If the selected move allows us to find a correct final solution to the problem, the solution is given as an output (it is printed out). However, if the selected move does not lead to a correct final solution to the problem, we backtrack. The backtracking is done to the point where the move was selected and choose an alternative one at this point. This process is repeated until the correct solution to the problem is found. If none of the available moves work in finding the correct solution to the problem it is sufficient to say there is no solution to the problem.
Recursion is the most appropriate method to implement this type of algorithm.
#add more stuff#

4 Pseudo code

Algorithm 1 Check if a Move is Valid

```
function validMove(number , row, column)
Where:
*number* is the number to be inserted
*row* is the row of current position
*col* is the column of the current position
*n* is the length of the square matrix representing the sudoku puzzle grid
*board* is the $n \times n$ square matrix representing the sudoku puzzle grid

    for i from 1 to n do
        if number is equal to board[row][i] then
            return false
    for i from 1 to n do
        if number is equal to board[i][col] then
            return false
initialise rowmin to $\frac{row}{blocksize} \times blocksize$
initialise colmin to $\frac{column}{blocksize} \times blocksize$
initialise rowmax to rowmin + blocksize
initialise colmax to colmin + blocksize

    for i from rowmin to rowmax  do
        for j from colmin to colmax do
            if num = board[i][j] then
                return false
    return true
```

Algorithm 2 Check if a Position is Open

```
function openPosition
*n* is the length of the square matrix representing the sudoku puzzle grid
*board* is the $n \times n$ square matrix representing the sudoku puzzle grid

    initialise temp to be an array of length 2
    for i from 1 to n do
        for j from 1 to n do
            if board[i][j] is empty then
                temp[0] = i
                temp[1] = j
                return temp
    return NULL
```

Algorithm 3 Backtracking

```
function SOLVE
n is the length of the square matrix representing the sudoku puzzle grid
board is the $n \times n$ square matrix representing the sudoku puzzle grid
openPosition(),validMove(i,row,col) are the functions described above in Algorithm 1
and Algorithm 2 respectively.

    initialise row and col;
    initialise temp to openPosition();
    if temp is not NULL then
        row = temp[0];
        col = temp[1];
    else
        return true
    for i from 1 to n do
        if validMove(i,row,col) is true then
            board[row][col]=i;
            if solve() is true then
                return true;
            set board[row][col] to be empty;
    return false;
```

5 Theoretical Analysis

#do the theoretical analysis#

6 Experimental Methodology

#Define methods of experimentation#

7 Presentation of results

#Show results of empirical analysis#

8 Interpretation of results

#interpret results from empirical analysis#

9 Conclusion

10 References

11 Acknowledgements