# CO4352 Advanced Algorithms

# Mini Project

# Sudoku Solving Algorithm

**NAME : WEERASINGHE W.M.K.N.B**

**INDEX NO: 17/ENG/113**

## INTRODUCTION

Sudoku is a puzzle based game which was originated in Japan. It is a game that is logic based . Standard Sudoku contains 81 cells, in a 9×9 grid, and has 9 boxes, each box being the intersection of the first, middle, or last 3 rows, and the first, middle, or last 3 columns. Each cell may contain a number from one to nine, and each number can only occur once in each row, column, and box. A Sudoku starts with some cells containing numbers , and the goal is to solve the remaining cells. Proper Sudoku's have only one solution. But there can be instances a same Sudoku can have multiple solutions. And there are variants of Sudoku like 4x4, 5x5, 16x16, 25x25, diagonal Sudoku, sandwich Sudoku and etc. The main goal of this project is implementing an algorithm that can solve 9x9 and 16x16 Sudoku puzzles in a reasonable time with high efficiency. Here I have used **exact cover method with Knuth's Algorithm X** as the solving method. The average time taken for **9x9 puzzles is 0.0066 seconds** and the average time for **16x16 puzzles is 0.035 seconds**.

## BACKGROUND AND CONSTRAINTS

Sudoku is a 9x9 grid that comprises 81 cells, some of which have values since the beginning. The aim is to allocate the digits (1 to 9) to the empty cells, so that every row, column, and sub-grid of size 3×3 contains exactly one instance of the digits from 1 to 9. These constraints are used to determine the missing numbers. There are several computer algorithms that will solve 9×9 puzzles ($n$=9) in fractions of a second. Let's look at them one by one.

1. **Backtracking -** Backtracking is a *depth-first search* algorithm, because it will completely explore one branch to a possible solution before moving to another branch. The grid is represented by a two-dimensional array called grid and a value of 0 means that the location is empty. Grid locations are filled in through a process of a systematic ordered search for empty locations, guessing values for each location, and backtracking and undoing guesses if they are incorrect. Here advantages are a solution is guaranteed, simpler than other algorithms. Disadvantage of backtracking is it consumes considerable time to solve the puzzle, when the puzzle is more complex and large.

2. **Stochastic Search -** In this method random based algorithms are used. In this method firstly numbers are assigned to the blank cells in the grid randomly.Then calculate the number of errors. Finally shuffle the inserted numbers until the number of mistakes is reduced to zero. Then a solution to the puzzle found. Approaches for shuffling the numbers include simulated annealing, genetic algorithm and tabu search. This method is fast , but some times they are not fast as deductive techniques.

3. **Constraint Programming -** A Sudoku can be also modelled as a constraint satisfaction problem. many *reasoning algorithms* based on constraints can be applied to model and solve Sudoku puzzles. An algorithm combining a constraint-model-based algorithm with backtracking would have the advantage of fast solving time, and the ability to solve all Sudokus.

4. **Exact Cover -** This method allows for an elegant description of the problem and an efficient solution. Modelling Sudoku as an exact cover problem and using an algorithm such as Knuth's Algorithm X will typically solve a Sudoku in a few milliseconds. An alternative approach is the use of Gauss elimination in combination with column and row striking.

## IMPLEMENTATION OF THE ALGORITHM

In this project , as I have used exact cover method has been used along with Knuth's algorithm X, firstly, the puzzle should be explained and organized as the exact cover problem. Then Knuth's algorithm was used to solve the problem.

**1. Explain Sudoku as exact cover problem**

According to the Sudoku rules, for a n x n puzzle, n numbers must be in every row, column and box, and numbers can not be repeated in each row, column or box.

Puzzle has $n^2$ cells and each cell has a number --> n x n = $n^2$

Puzzle has $n^2$ rows and each row has n numbers --> n x n = $n^2$

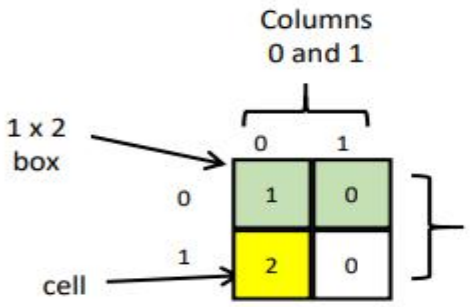
Puzzle has $n^2$ columns and each column has n numbers --> n x n = $n^2$

Puzzle has $n^2$ boxes and each box has n numbers --> n x n = $n^2$

A regular Sudoku board of size 9X9 has the following constraints.

1. **Row-Column:** Each intersection of a row and column, each cell, must contain exactly one number. (9*9=81 total constraints)
2. **Row-Number:** Each row must contain each number exactly once. (9*9=81 total constraints)
3. .**Column-Number:** Each column must contain each number exactly once. (9*9=81 constraints)
4. **Box-Number:** Each box must contain each number exactly once. (9*9=81 total constraints)

Because of that total number of constraints will be equal to $n^2$ by 4. So we get 324 constraints for 9 **x** 9 puzzle and 1024 constraints for 16 **x** 16 puzzle. And also the number of rows in exact cover matrix is equals to,( number of rows **x** number of columns **x** n). So we get 729 for 9**x**9 puzzle as 9**x**9**x**9 = 729 and 4096 for 16**x**16 puzzle as 16**x**16**x**16 = 4096. Below example explains the exact cover for a simple puzzle.




Number of constraint = (2*2) + (2*2) + (2*2) + (2*2) = 16

Number of rows in exact cover matrix = 2 * 2 * 2 = 8

We need to find (0,1) and (1,1)

RC : Row-Column, RN: Row-Number, CN : Column-Number, BN : Box-Number

**Constraints** = [(RC', (0, 0)), (RC', (0, 1)), ('RC', (1, 0)), ('RC', (1, 1)), ('RN', (0, 1)), ('RN', (0, 2)), ('RN', (1, 1)), ('RN', (1, 2)), ('CN', (0, 1)), ('CN', (0, 2)), ('CN', (1, 1)), ('CN', (1, 2)), ('BN', (0, 1)), ('BN', (0, 2)), ('BN', (1, 1)), ('BN', (1, 2))]

**Matrix** = {(0, 0, 1): [('RC', (0, 0)), ('RN', (0, 1)), ('cn', (0, 1)), ('BN', (0, 1))], (0, 0, 2): [('RC', (0, 0)), ('RN', (0, 2)), ('CN', (0, 2)), ('BN', (0, 2))], (0, 1, 1): [('RC', (0, 1)), ('RN', (0, 1)), ('CN', (1, 1)), ('BN', (0, 1))], (0, 1, 2): [('RC', (0, 1)), ('RN', (0, 2)), ('CN', (1, 2)), ('BN', (0, 2))], (1, 0, 1): [('RC', (1, 0)), ('RN', (1, 1)), ('CN', (0, 1)), ('BN', (1, 1))], (1, 0, 2): [('RC', (1, 0)), ('RN', (1, 2)), ('CN', (0, 2)), ('BN', (1, 2))], (1, 1, 1): [('RC', (1, 1)), ('RN', (1, 1)), ('CN', (1, 1)), ('BN', (1, 1))], (1, 1, 2): [('RC', (1, 1)), ('RN', (1, 2)), ('CN', (1, 2)), ('BN', (1, 2))]}

**Arranged matrix using above two** : {('RC', (0, 0)): {(0, 0, 1), (0, 0, 2)}, ('RC', (0, 1)): {(0, 1, 2), (0, 1, 1)}, ('RC', (1, 0)): {(1, 0, 2), (1, 0, 1)}, ('RC', (1, 1)): {(1, 1, 1), (1, 1, 2)}, ('RN', (0, 1)): {(0, 1, 1), (0, 0, 1)}, ('RN', (0, 2)): {(0, 1, 2), (0, 0, 2)}, ('RN', (1, 1)): {(1, 1, 1), (1, 0, 1)}, ('RN', (1, 2)): {(1, 0, 2), (1, 1, 2)}, ('CN', (0, 1)): {(0, 0, 1), (1, 0, 1)}, ('CN', (0, 2)): {(1, 0, 2), (0, 0, 2)}, ('CN', (1, 1)): {(0, 1, 1), (1, 1, 1)}, ('CN', (1, 2)): {(0, 1, 2), (1, 1, 2)}, ('BN', (0, 1)): {(0, 1, 1), (0, 0, 1)}, ('BN', (0, 2)): {(0, 1, 2), (0, 0, 2)}, ('BN', (1, 1)): {(1, 1, 1), (1, 0, 1)}, ('BN', (1, 2)): {(1, 0, 2), (1, 1, 2)}}

According to the Sudoku rules, same number cannot be repeated within same row, column or box. So the values related to the empty cells can be removed from arranged matrix. Therefore, only the high-lighted rows in the above matrix will be remained and the others will be removed.

2. **Apply Knuth's algorithm X to solve the Sudoku puzzle**

After creating the arranged exact cover matrix, Knuth's algorithm X should be applied to the matrix. Following are the steps of algorithm.

1) If the columns of the matrix are zero, the current solution is the valid solution for the problem.

2) Else, select the column c that has the least number of 1s.

3) Select the row r such that Matrix[r] = 1.

4) Add row r into the solution.

5) For each column j such that Matrix[r][j] = 1,

for each row i such that Matrix[i][j] = 1,

delete row i from Matrix. delete column j from Matrix.

Repeat these steps of the algorithm recursively on the reduced Matrix.

Algorithm X is the name Donald Knuth gave for "the most obvious trial-and-error approach" for finding all solutions to the exact cover problem. Technically, Algorithm X is a recursive, nondeterministic, depth-first, backtracking algorithm.When Algorithm X is implemented efficiently using Donald Knuth's Dancing Links technique on a computer, Knuth calls it DLX. DLX uses the matrix representation of the problem, implemented as a series of doubly linked lists of the 1s of the matrix: each 1 element has a link to the next 1 above, below, to the left, and to the right of itself.

## OPTIMIZATION TECHNIQUES

First of all whether the given puzzle has repeating values in same columns, rows and boxes was checked. And whether there are invalid numbers was checked. Because of that the solving process will go smoothly. As we used exact cover problem with Knuth's algorithm X to solve the puzzle it gives the solution within milliseconds. So there are no new optimization techniques added to this solution.

## CHALLENGES FACED

The main challenge of the project was to optimize the time consumption to solve the puzzle. Here 1st used MIT open-courseware's recursive Sudoku solving algorithm. At 1st the backtracking count was around 600 for a puzzle. After that I used implications during recursive search to optimize the backtracking count. Then when I was using optimized version backtracking count was down to 2 for some solutions. But still it took a very significant time. So after doing some searching found the exact cover method and checked results with using it. Finally the method of using exact cover with Knuth's algorithm X gave some very efficient results.

## LIMITATIONS

The major limitation of using exact cover method for solving the Sudoku puzzle is , it does not give answers for the puzzles which have a fully empty column or row. Here it creates an infinite loop for this type of puzzles. This loop can be avoided by checking whether there are any fully empty rows or columns before starting the solving the process.

## FUTURE IMPROVEMENTS

Design a Graphical user interface to simulate how the process is done.

Further improving the efficiency of the algorithm.

Expanding algorithms for working with different sizes of Sudoku puzzles.

Expanding algorithms to work on different types of Sudoku puzzles.