A PROJECT REPORT

ON

SUDOKO

Submitted in partial fulfillment of the requirment for the IV semester

Of

BACHELOR OF TECHNOLOGY

IN

ANALYSIS AND DESIGN OF ALGORITHM

Submitted By:

SRIJAN MISHRA(RA2111047010108)

M.GOWSHIGAN(RA2111047010086)

S.YUVARAJ(RA2111047010085)

Under the Supervision of

Dr.SUDHA RAJESH

DEPARTMENT OF COMPUTATIONAL INTELLIGENCE

SRM INSTITUTE OF TECHNOLOGY

SRM

INSTITUTE OF SCIENCE & TECHNOLOGY

INSTITUrE OF SCIENCE & TECHNOLOGY

(Deemed to be University u/s 3 ofUGC Act, 1956)

Declaration:

l.hereby declare that the work presented in this dissertation entitled "SUDOKO" has been done by me and my team,and this dissertation embodies my own work

Contents:

CONTRIBUTION TABLE

Name	Reg No	Contribution
SRIJAN MISHRA	RA2111047010108	Algorithm,Code,Time Complexity
M.GOWSHIGAN	RA2111047010086	Problem Statement, Problem, Explanation
S.YUVIRAJ	RA2111047010085	Backtracking,Time Complexity Analysis, Result

1. Problem Definition:

Solving Sudoku has been a challenging problem in the last decade. The purpose has been to develop more effective algorithm in order to reduce the computing time and utilise lower memory space. This essay develops an algorithm for solving Sudoku puzzle by using a method, called pencil-and-paper algorithm. This algorithm resembles human methods, i.e. it describes how a person tries to solve the puzzle by using certain techniques. Our ambition is to implement the pencil-and-paper algorithm by using these techniques.

There are currently different variants of Sudoku such as 4X4 grids, 9X9 grids and 16X16 grids. This work is focused on classic and regular Sudoku of 9X9 board, and then a comparison is performed between the paper-and-pencil method and Brute force algorithm. Hopefully, by doing this work we might be able to answer the following questions: How does the pencil-and-paper algorithm differ from the Brute force algorithm? Which one of them is more effective? Is it possible to make these algorithms more efficient?

2. Problem Explanation with diagram and example:

Algorithm stems from the name of a Latin translation of a book written by al-Khwarizem, A Persian mathematician astronomer and geographer .The word "Sudoku" is short for Su-ji wa dokushin ni kagiru (in Japanese),which means "the numbers must be single shown in figure 1.This game is popular in Japan since the mid 1980s.It has become extremely popular throughout the world in the last decade, triggered primarily by the publishing of Sudoku puzzles in British newspapers starting November 2004. Software programmers now supply the different programs to fulfil the growing demand for unique Sudoku puzzles. The aim of the puzzle is to enter a numerical digit from 1 through 9 in each cell of a grid made up of 3×3sub squares or sub grids, starting with various digits given in some cells; each row, column, and sub squares region must contain each of the numbers 1 to 9 exactly once. Players may use a wide range of strategies which deals the solutions either from zones or from the whole grid.. A sample Sudoku game and solution is depicted in figure 1.

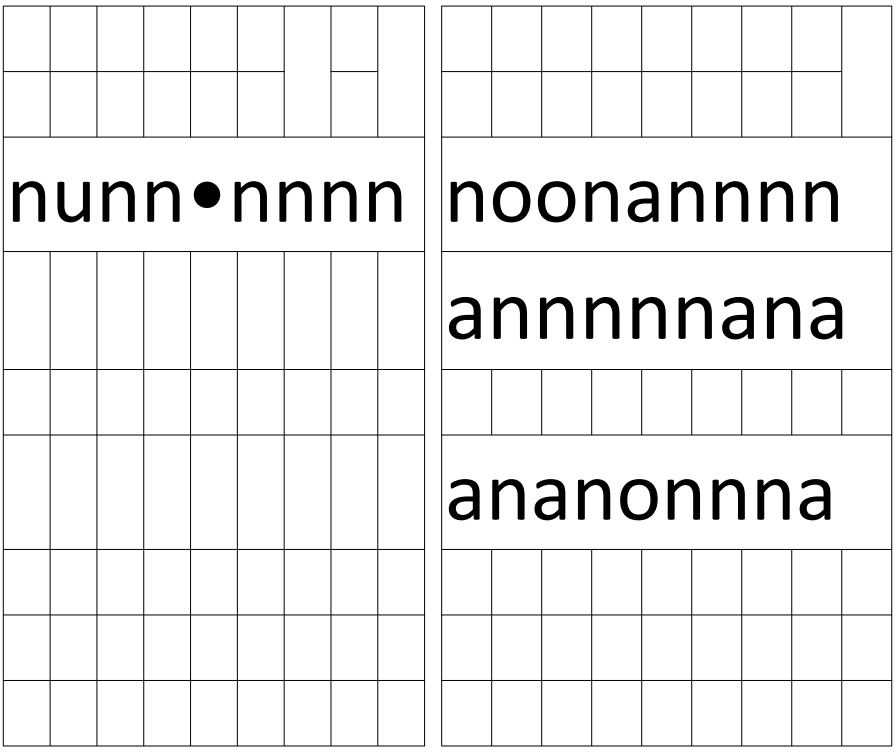


Figure I-Sample Sudoku Game and Solution

3. Design Techniques used:

Backtracking is a general algorithmic technique that considers searching every possible combination in order to solve an optimization problem. The basic principle of a backtracking algorithm, in regards to Sudoku, is to work forwards, one square at a time to produce a working Sudoku grid. When a problem occurs, the algorithm takes itself back one step and tries a different path. It's nearly impossible to produce a valid Sudoku by randomly plotting numbers and trying to make them fit. Likewise, backtracking with a random placement method is equally ineffective. Backtracking best works in a linear method. It is fast, effective if done correctly.

4. Algorithm for the problem: Find row, column of an unassigned cell

If there is none, return true

For digits from 1 to 9 a) If there is no conflict for digit at row, column assign digit to row, column and recursively try fill in rest of grid

b) If recursion successful, return true\

c) Else, remove digit and try another

If all digits have been tried and nothing worked, return false.

5. Explanation of algorithm with example:

Create a function that checks after assigning the current index the grid becomes unsafe or not. Keep Hashmap for a row, column and boxes. If any number has a frequency greater than 1 in the hashMap return false else return true; hashMap can be avoided by using loops.

Create a recursive function that takes a grid.

Check for any unassigned location. If present then assign a number from 1 to 9, check if assigning the number to current index makes the grid unsafe or not, if safe then recursively call the function for all safe cases from O to 9. if any recursive call returns true, end the loop and return true. If no recursive call returns true then return false.

If there is no unassigned location then return true.

6. Code and Output: // A Backtracking program in

// C++ to solve Sudoku

problem #include

```
<bits/stdc++.h>                using
namespace std;

// UNASSIGNED is used for empty
// cells in sudoku grid
#define UNASSIGNED O

// N is used for the size of Sudoku grid.
// Size will be NxN
#define N 9

// This function finds an entry in grid
// that is still unassigned bool FindUnassignedLocation(int grid[N][N],
int& row, int& col);

// Checks whether it will be legal // to
assign num to the given row, col bool
isSafe(int grid[N][N], int row, int col,
int num);

/ * Takes a partially filled-in grid and
attempts to assign values to all unassigned
locations in such a way to meet the
requirements for Sudoku solution (non-
duplication across rows, columns, and boxes)
* /
bool SolveSudoku(int grid[N][N])

        int row, col;
```

```
// If there is no unassigned location,
// we are done if
(!FindUnassignedLocation(grid, row, col))
        // success!
        return true;

// Consider digits 1 to 9 for (int
num = 1; num <= 9; num++)


        // Check if looks promising if
        (isSafe(grid, row, col, num))


                // Make tentative assignment
                grid[row][col] = num;

                // Return, if success
                if
                (SolveSudoku(grid))
                return true;

                // Failure, unmake & try again
                        ow][col]= UNASSIGNED;
// This triggers backtracking
return false;
```

/ * Searches the grid to find an entry that is still unassigned. If found, the reference parameters row, col will be set the location that is unassigned, and true is returned. If no unassigned entries remain, false is returned. * / bool FindUnassignedLocation(int grid[N][N], int& row, int& col)

for (row = O; row < N; row++) for (col = O; col < N; col++) if (grid[row][col] UNASSIGNED) return true; return false;

/ * Returns a boolean which indicates whether an assigned entry in the specified row matches the given number. * / bool UsedlnRow(int grid[N][N], int row, int num)

for (int col = 0; col < N; col++) if (grid[row][col] == num) return true; return false;

/ * Returns a boolean which indicates whether an assigned entry in the specified column matches the given number. */ bool UsedlnCol(int grid[N][N], int col, int num)

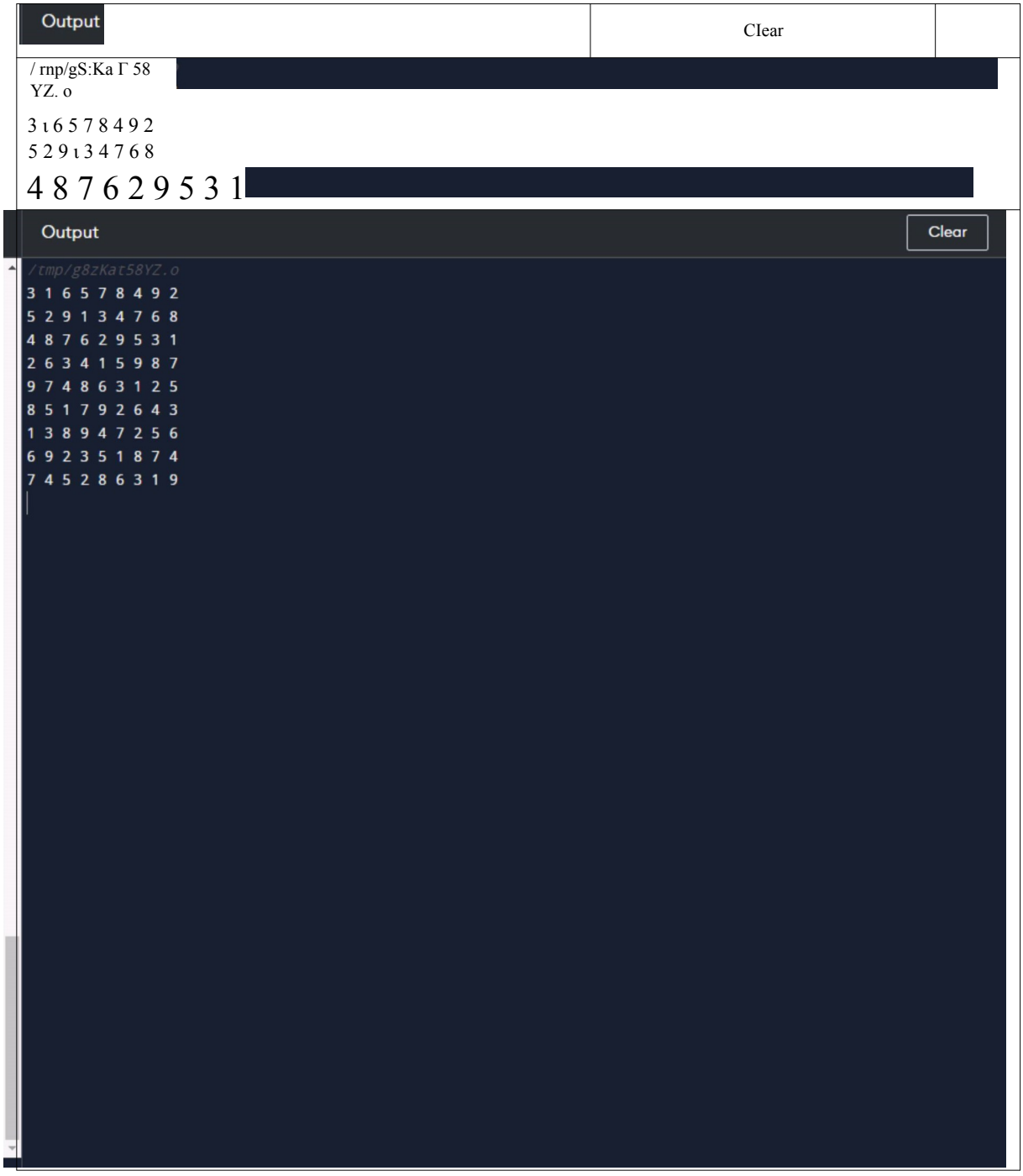
for (int row = 0; row < N; row++) if (grid[row][col] num) return true; return false;

```
/ * Returns a boolean which indicates whether
an assigned entry within the specified 3x3 box
matches the given number. */ bool
UsedlnBox(int grid[N][N], int boxStartRow, int
boxStartCol, int num)

        for (int row = O; row < 3; row++) for (int
                col = 0; col < 3; col++) if (grid[row
                + boxStartRow]
                                        [col + boxStartCol]==
                                                                num)
                                return true;
        return false;


/ * Returns a boolean which indicates whether
it will be legal to assign num to the
given row, col location. * / bool
isSafe(int grid[N][N], int row, int col, int
num)

        / * Check if 'num' is not already placed
        in current row, current column and
        current 3x3 box */ return
        !UsedlnRow(grid, row, num) &&
        !UsedInCol(grid, col, num)
                && !UsedlnBox(grid, row - row % 3, col - col %
                                                3, num)
                && grid[row][col] == UNASSIGNED;
```

```
/ * A utility function to print grid */
void printGrid(int grid[N][N])

    for (int row = O; row < N; row++)

        for (int col = 0; col < N; col++) cout
        << grid[row] [coll << 11 ';cout <<
        endl;



// Driver Code
int main()

    // 0 means unassigned cells int
    grid[N][N] = { { 3, O, 6, 5, O, 8, 4, 0, 0 },
                   { 5, 2, 0, 0, 0, 0, 0, 0, 0 },
                   { 0, 8, 7, 0, 0, 0, 0, 3, 1 },
                   { 0, 0, 3, 0, 1, 0, 0, 8, 0 },
                   { 9, 0, 0, 8, 6, 3, 0, 0, 5 },
                   { 0, 5, 0, 0, 9, 0, 6, 0, 0 },
                        , 0, 0, 0, 2, 5, 0 },
                   { 0, 0, 0, 0, 0, 0, 0, 7, 4 },
                     , 0, 5, 2, 0, 6, 3, 0, 0

    if (SolveSudoku(grid)          true)
    printGrid(grid); else cout << "No
    solution exists";
```

return O;

ουτρυτ:



7. Complexity analysis:

Time complexity: O(9 ᴬ(n*n)).

For every unassigned index, there are 9 possible options so the time complexity is

O(9 ^A (n*n)). The time complexity remains the same but there will be some early pruning so the time taken will be much less than the naive algorithm but the upper bound time complexity remains the same.

Space Complexity: O(n*n).

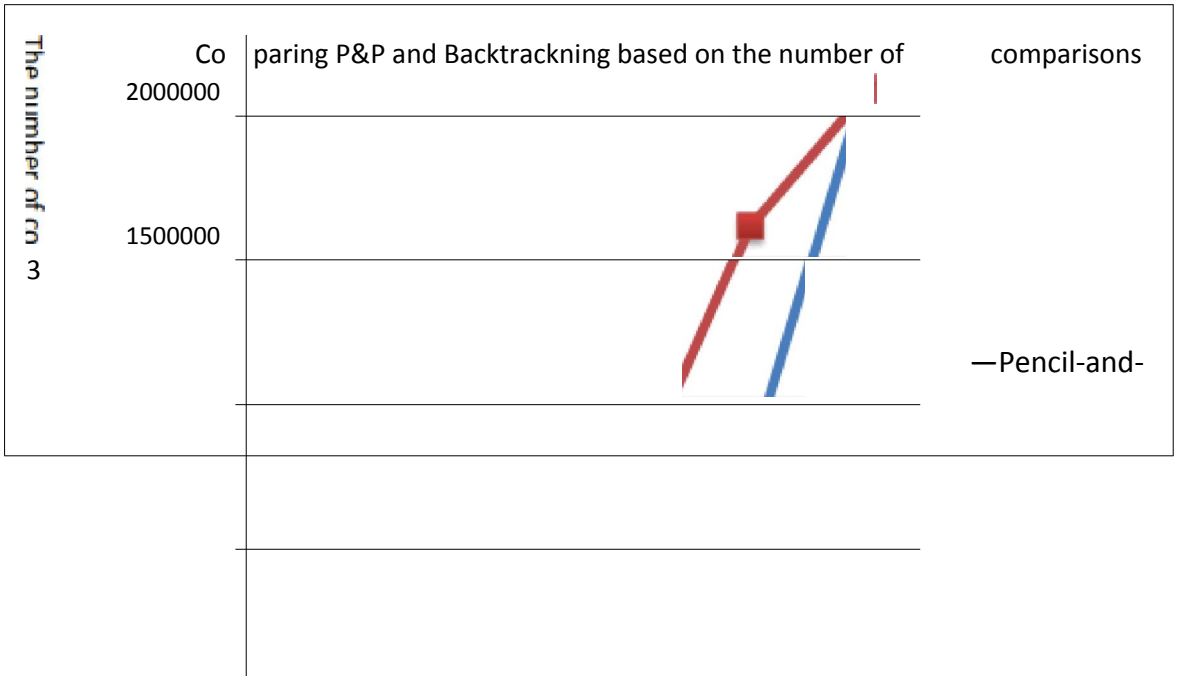
To store the output array a matrix is needed.

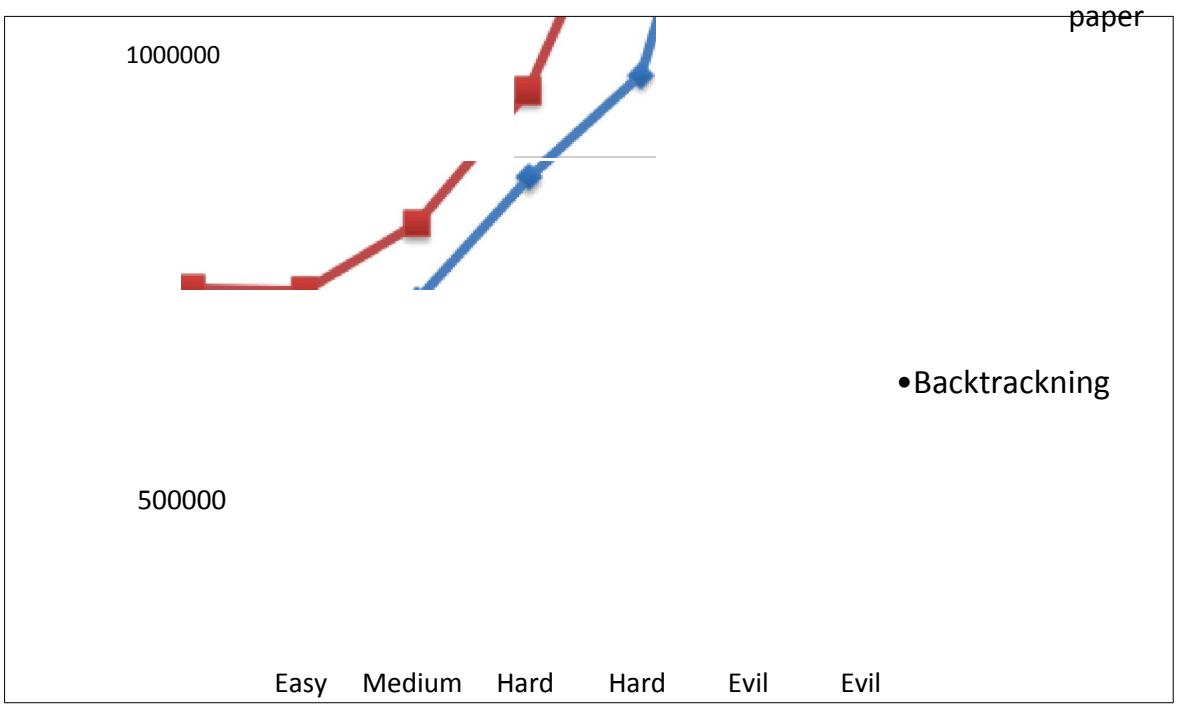
8. Conclusion:

This study has shown that the pencil-and-paper 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 testing results have revealed that the performance of the pencil-and-paper algorithm is better than the Backtracking algorithm with respect to the computing time to solve any puzzle.

The Backtracking algorithm seems to be a useful method to solve any Sudoku puzzles and it can guarantee to find at least one solution. However, this algorithm is not efficient because the level of difficulties is irrelevant to the algorithm. In other words, 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.





9. References

[1] Knuth, Donald (1997). Fundamental Algorithms, Third Edition,lSBN 0-201-89683-4. [2] Ch. X u , W. X u , 2009, The model and algorithm to Estimate The Difficulty Levels of Sudoku puzzles, Journals of Mathematicsv Research.. [3] Lewis,R,Meta heuristics, 2007, Can solve Sudoku puzzles .Journal of heuristics, 13(4),387-401. [4] Xu,J, 2009, Using backtracking method to solve Sudoku puzzle, computer programming skills & maintainence 5,pp 17-21. [5] Chakraborty ,r,Palidhi, S,Banerjee„ 2014, An optimized Algorithm for solving combinatorial probl;em using reference graph IOSR Journal of CS 16(3PP1-7 [6] Darwin , c, 1859, The origin of species Oxford University. [7] Mantere T Kolljonen solving rating & generating Sudoku with GA,Proc.of IEEE 1382-1389 [8] Geem,Zong WOO, 2007, Harmony search algorithmpp371-378. [9] Viksten ,Henrik, 2013, Performance and scalability of
Sudoku solvers,Bachelor's Theses at NADA,Sweden [10] T. Kovacs, Nov 2008., Artificial Intelligence through Search: Solving Sudoku Puzzles, Journal Papers, University of Bristol, Department of Computer Science, pp 1-14.