A Project Report

on

**SUDOKU – AN APPLICATION OF GRAPH COLOURING**

# DESIGN AND ANALYSIS OF ALGORITHMS

By

Shreya Chintawar (2010030155)

Harika Nethi (2010030119)

Tejaswi Kottakki (2010030088)

Amrutha Varshini (2010030009)

under the supervision of

# Ms. P.SREE LAKSHMI

# Assistant Professor



Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500075, Telangana, India.

April, 2022

**DECLARATION**

The Project Report entitled “**SUDOKU – AN APPLICATION OF GRAPH COLOURING**” is a record of bonafide work of **Shreya Chintawar (2010030155), Harika Nethi (2010030119), Tejaswi Kottakki (2010030088)** submitted in partial fulfillment for the award of B.Tech in the Department of Computer Science and Engineering to the K L University, Hyderabad. The results embodied in this report have not been copied from any other Departments/University/Institute.

Shreya Chintawar (2010030155)

Harika Nethi (2010030119)

Tejaswi Kottakki (2010030088)

Amrutha Varshini (2010030009)

**CERTIFICATE**

This is to certify that the Project Report entitled “**SUDOKU – AN APPLICATION OF GRAPH COLOURING** ” is being submitted by **Shreya Chintawar (2010030155), Harika Nethi (2010030119), Tejaswi Kottakki (2010030088)** submitted in partial fulfillment for the award of B.Tech in CSE to the K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/institutes.

## **Signature of the Supervisor**

Ms. P.SREE LAKSHMI

Assistant Professor

## **Signature of the HOD Signature of the External Examiner**

**ACKNOWLEDGEMENT**

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take a grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. L. Koteswara Rao** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgment & sincere thanks to our Head of the Department

**Dr. Chiranjeevi Manike** for her exemplary guidance, monitoring, and constant encouragement throughout the course of the project. We thank **Ms.P. Sree Lakshmi**  of our department who has supported us throughout this project holding the position of supervisor.

We wholeheartedly thank all the teaching and non-teaching staff of our department without whom we won’t have made this project a reality. We would like to extend our sincere thanks, especially to our parents, our family members, and our friends who have supported us to make this project a grand success.

**ABSTRACT**

Sudoku is a logic-based, combinatorial number-placement puzzle. It helps in boosting logical thinking.A Sudoku puzzle (9x9) can be thought of as a graph with 81 vertices, one for each cell, and two vertices are connected by an edge if they cannot be assigned the same value. For example, all cells in the same row, column, or block will have edges between their corresponding vertices.

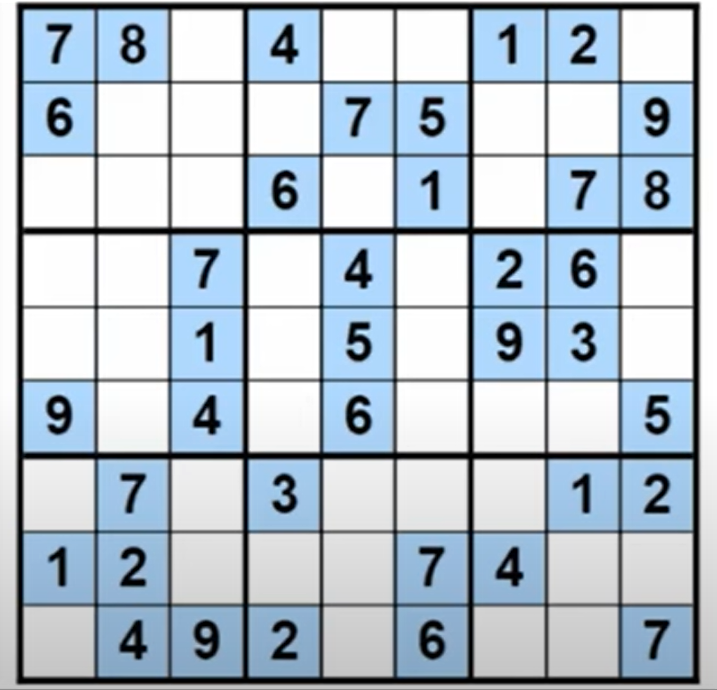


Fig 1: Sudoku Puzzle

The optimal solution in the case of the Sudoku puzzle is to find a coloring using only 9 colors (chromatic numbers).Given a Sudoku puzzle we can build the associated graph. The given number in the puzzle can be used to add additional edges to the graph we can then use graph coloring to find a 9-coloring of this graph (colors 1-9)

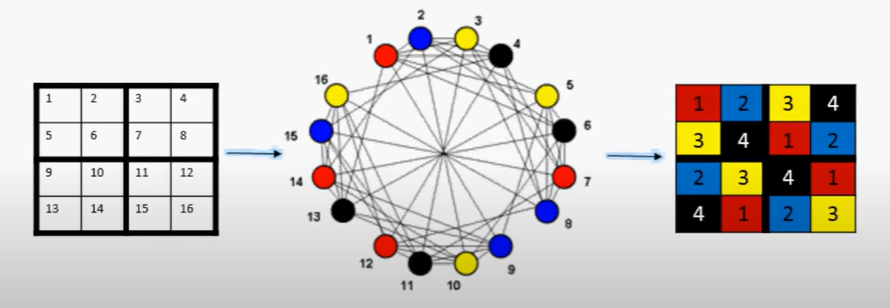


Fig 2: Graph Coloring

**TABLE OF CONTENTS**

1. **List Of Chapters**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Topics** | **Page. No** |
|  | Introduction | 1 |
|  | Literature Survey  2.1 Existing System | 2  2 |
|  | Hardware & Software requirements | 3 |
|  | Functional & Non-functional Requirements | 4 |
|  | Proposed System   * 1. Proposed Algorithm Design Technique   2. Data Structures Needed | 5 – 6  5  6 |
|  | Implementation   * 1. Work Flow   2. Console Code   3. GUI Code   4. Code Explanation | 7 – 13  7  8 – 9  10 – 12  13 |
|  | Results Discussion | 14 - 17 |
|  | Conclusion and Future Work  8.1 Conclusion  8.2 Future Scope | 18  18  18 |
|  | References | 19 |

1. **List Of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Figure Name** | **Page Number** |
| 5.1.1 | Chromatic Number | 5 |
| 6.1.1 | Flow Chart | 7 |
| 7.1.1 | Console Output | 14 |
| 7.2.1 | Same Row | 15 |
| 7.2.2 | Same Column | 15 |
| 7.2.3 | Same Grid | 16 |
| 7.2.4 | Final Sudoku | 16 |
| 7.2.5 | Output | 17 |

1. **INTRODUCTION**

The mathematics of sudoku has been a subject of inquiry and interest to many combinatorialists. Many mathematicians have also applied the notions of graph theory to sudoku.

Sudoku is a popular number-placement puzzle based on logic and combinatorics. The objective is to fill a 9 × 9 grid with digits such that each column, each row, and each of the nine 3 × 3 subgrids that compose the grid contain all of the digits from 1 to 9 (once and only once). Usually the puzzle is partially filled in a way that guarantees a unique solution, as of now from what we know atleast 17 cues are needed to create a puzzle with a unique solution.

Another way of looking at this puzzle is as follows:

* View the 81 cells as nodes of a graph
* Consider the connections(being in the same row, column or grid) as edges

This is the graph-theoretic framing of the problem, after this point we can treat the Sudoku as a vertex coloring problem, where we assign a color to each number(1-9) and ensure that no two nodes of the same color are connected by an edge (thus satisfying the constraints provided)

In the mathematics of Sudoku, the Sudoku graph is an undirected graph whose vertices represent the cells of a (blank) Sudoku puzzle and whose edges represent pairs of cells that belong to the same row, column, or block of the puzzle. The problem of solving a Sudoku puzzle can be represented as precoloring extension on this graph. It is an integral Cayley graph.

Here pre-coloring extension simply means translating the pre-existing cues into a graph with 81 nodes, coloring the nodes that are already given as clues, and then trying to color the rest of the vertices within the onstraints.

Cayley graph is simply a way of encoding information about group in a graph, as in we can define the sudoku puzzle completely in terms of a Graph, without missing any logical information or mathematical properties

1. **LITERATURE SURVEY**
   1. **EXISTING SYSTEM**

**Depth-First Search:**

* The simplest method is the **brute-force method** which randomly assigns numbers to the empty cells and checks whether the completed puzzle is a solution. If not, this process is repeated until a solution is found.
* Clearly, this method can be very time-consuming. A similar method is to generate all possible combinations for the empty cells. This can only be done for easy problems.
* Creating a search space for the puzzle and applying searches such as a **depth-first search** with backtracking is used as one of the algorithms for solving sudoku.
* **Time complexity:** O(V+E)

**Best First Search:**

* BFS is one of the most general and basic techniques for searching for possible solutions to a problem. The idea is that we want to generate every possible move within the game and then test to see whether it solves our problem.
* In Sudoku, the **BFS** algorithm visits the empty cells in some order, filling in digits sequentially (from 1 to 9). The program would solve a puzzle by placing the digit “1” in the first cell and checking if it is allowed to be there. If there are no violations (checking row, column, and box constraints) then the algorithm advances to the next cell and places a “1” in that cell. When checking for violations, if it is discovered that the “1” is not allowed, the value is advanced to “2”.

**Stochastic Optimization:**

* Stochastic optimization refers to a collection of methods for minimizing or maximizing an objective function when randomness is present**.**
* This method can work with any kind of optimization problems but they are of the weak capability of guaranteeing the global optimal solutions.

1. **HARDWARE AND SOFTWARE REQUIREMENTS**
   1. **HARDWARE REQUIREMENTS**

Device name : DESKTOP-9U38MCH

Processor : Intel® Core™ i7-8565U CPU @ 1.80GHz 1.99 GHz

Installed RAM : 16.0 GB (15.8 GB usable)

Device ID : 2F0DA166-C34F-44D9-80DC-1264CC873D6C

Product ID : 00331-10000-00001-AA182

System type **:** 64-bit operating system, x64-based processor

* 1. **SOFTWARE REQUIREMENTS**

Software : PyCharm Community Edition

Programming Language **:** Python

**4. FUNCTIONAL & NON-FUNCTIONAL REQUIREMENTS**

* 1. **FUNCTIONAL REQUIREMENTS**

The required python packages for GUI:

* pygame
* valid from solver
* time
  1. **NON-FUNCTIONAL REQUIREMENTS**
* Frame sizes must be set properly.
* Current cell must be highlighted.
* Timer should be displayed while playing the game
* Cross mark should be displayed in red for wrong entry and should display ‘wrong’ in the console.
* If entry is accepted it should display ‘Success’ in the GUI console.
* In the end after solving the puzzle it should display ‘game over’ in the console.

**5. PROPOSED SYSTEM**

**5.1 PROPOSED ALGORITHM DESIGN TECHNIQUE**

**GRAPH COLORING:**

Graph coloring problem involves assigning colors to certain elements of a graph subject to certain restrictions and constraints. In other words, the process of assigning colors to the vertices such that no two adjacent vertexes have the same color is caller Graph Coloring.

In G- Graph Coloring Problem, we have to find if a graph can be colored with a minimum of ‘G’ colors. This ‘G’ is also known as the Chromatic Number of a Graph and is denoted as χ(G)

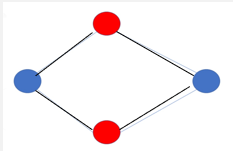


Fig 5.1.1 : Chromatic Number

For this Graph, Chromatic Number, G = 2 { χ(2) }

**BACKTRACKING:**

Backtracking is an algorithmic technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

For solving the Sudoku problem we try filling digits one by one. Whenever we find that the current digit cannot lead to a solution, we remove it (backtrack) and try the next digit. This is better than the naïve approach (generating all possible combinations of digits and then trying every combination one by one) as it drops a set of permutations whenever it backtracks.

**5.2 DATA STRUCTURES NEEDED**

Sudoku is represented in the form of a matrix; therefore a 2D array data structure is used. Sudoku also uses the concept of graph colouring; hence graph data structure is used.

**2D – Array data structure:**

A two-dimensional array is similar to a one-dimensional array, but it can be visualized as a grid (or table) with rows and columns. Positions in a two-dimensional array are referenced like a map using horizontal and vertical reference numbers.

**Graph data structure:**

A graph is a non-linear data structure that consists of a finite set of nodes (or vertices) and a set of edges connecting them. The nodes or vertices are used to store data and this data can be used further.

1. **IMPLEMENTATION**

**6.1 WORK FLOW**

Diagram

Description automatically generated

Fig 6.1.1: Flow Chart

* 1. **CONSOLE CODE**

board = [

[7,8,0,4,0,0,1,2,0],

[6,0,0,0,7,5,0,0,9],

[0,0,0,6,0,1,0,7,8],

[0,0,7,0,4,0,2,6,0],

[0,0,1,0,5,0,9,3,0],

[9,0,4,0,6,0,0,0,5],

[0,7,0,3,0,0,0,1,2],

[1,2,0,0,0,7,4,0,0],

[0,4,9,2,0,6,0,0,7]

]

print(“====SUDOKU SOLVER====”)

def solve(bo):

find = find\_empty(bo)

if not find:

return True # which means we found the solution

else:

row, col = find

for I in range(1,10):

if valid(bo, I, (row, col)): #i is a num from 1 to 9

bo[row][col] = i

if solve(bo):

return True

bo[row][col] = 0

return False

def valid(bo, num, pos):

for I in range(len(bo[0])):

if bo[pos[0]][i] == num and pos[1] != i:

return False

for I in range(len(bo)):

if bo[i][pos[1]] == num and pos[0] != i:

return False

box\_x = pos[1] // 3 #for column

box\_y = pos[0] // 3 #for row

for I in range(box\_y\*3, box\_y\*3 + 3):

for j in range(box\_x \* 3, box\_x\*3 + 3):

if bo[i][j] == num and (I,j) != pos:

return False

return True

def print\_board(bo):

for I in range(len(bo)):

if I % 3 == 0 and I != 0:

print(“- - - - - - - - - - - - - “)

for j in range(len(bo[0])):

if j % 3 == 0 and j != 0:

print(“ | “, end=””)

if j == 8:

print(bo[i][j])

else:

print(str(bo[i][j]) + “ “, end=””)

print\_board(board)

solve(board)

print(“\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”)

print(“====Solution for the SUDOKU SOLVER====”)

print\_board(board)

**6.3 GUI CODE**

import pygame

from solver import solve, valid

import time

pygame.font.init()

class Grid:

board = [

[7, 8, 0, 4, 0, 0, 1, 2, 0],

[6, 0, 0, 0, 7, 5, 0, 0, 9],

[0, 0, 0, 6, 0, 1, 0, 7, 8],

[0, 0, 7, 0, 4, 0, 2, 6, 0],

[0, 0, 1, 0, 5, 0, 9, 3, 0],

[9, 0, 4, 0, 6, 0, 0, 0, 5],

[0, 7, 0, 3, 0, 0, 0, 1, 2],

[1, 2, 0, 0, 0, 7, 4, 0, 0],

[0, 4, 9, 2, 0, 6, 0, 0, 7]

]

def \_\_init\_\_(self, rows, cols, width, height):

self.rows = rows

self.cols = cols

self.cubes = [[Cube(self.board[i][j], I, j, width, height) for j in range(cols)] for I in range(rows)]

self.width = width

self.height = height

self.model = None

self.selected = None

def update\_model(self):

self.model = [[self.cubes[i][j].value for j in range(self.cols)] for I in range(self.rows)]

def place(self, val):

row, col = self.selected

if self.cubes[row][col].value == 0:

self.cubes[row][col].set(val)

self.update\_model()

if valid(self.model, val, (row,col)) and solve(self.model):

return True

else:

self.cubes[row][col].set(0)

self.cubes[row][col].set\_temp(0)

self.update\_model()

return False

def draw(self, win):

# Draw Grid Lines

gap = self.width / 9

for I in range(self.rows+1):

if I % 3 == 0 and I != 0:

thick = 4

else:

thick = 1

pygame.draw.line(win, (0,0,0), (0, i\*gap), (self.width, i\*gap), thick)

pygame.draw.line(win, (0, 0, 0), (I \* gap, 0), (I \* gap, self.height), thick)

def select(self, row, col):

# Reset all other

for I in range(self.rows):

for j in range(self.cols):

self.cubes[i][j].selected = False

self.cubes[row][col].selected = True

self.selected = (row, col)

class Cube:

rows = 9

cols = 9

def \_\_init\_\_(self, value, row, col, width ,height):

self.value = value

self.temp = 0

self.row = row

self.col = col

self.width = width

self.height = height

self.selected = False

def draw(self, win):

fnt = pygame.font.SysFont(“comicsans”, 40)

gap = self.width / 9

x = self.col \* gap

y = self.row \* gap

if self.temp != 0 and self.value == 0:

text = fnt.render(str(self.temp), 1, (128,128,128))

win.blit(text, (x+5, y+5))

elif not(self.value == 0):

text = fnt.render(str(self.value), 1, (0, 0, 0))

win.blit(text, (x + (gap/2 – text.get\_width()/2), y + (gap/2 – text.get\_height()/2)))

if self.selected:

pygame.draw.rect(win, (255,0,0), (x,y, gap ,gap), 3)

main()

pygame.quit()

**6.4 CODE EXPLANATION**

* The board variable contains all of the values for each row and column, as well as their width and height.
* \_\_init\_\_() initializes this class by setting its rows, cols, width, and height variables.
* update\_model() updates the model of this grid so that it can be solved for using valid().
* place(), sketch(), draw(), select(), clear() are methods that allow us to interact with our grid object in order to change what happens when we call them on it.
* The code starts by creating an instance of Cube with value set to 0, row set to 1, and column set to 2.
* The code is the main function of the game.
* It sets up the board, draws it, and handles input.
* The code above is a class definition for a cube object that has 9 rows and 9 columns.
* Next, it creates a window with the title "Time: " and sets its size to be 640x480 pixels.
* Lastly, it starts running while looping through pygame events until either "quit" is pressed or all of them have been processed (in which case run will be set to false).

1. **RESULT DISCUSSION**

**7.1 CONSOLE OUTPUT**

A screenshot of a computer

Description automatically generated with medium confidence

Fig 7.1.1: Console Output

**7.2 GUI OUTPUT**

A picture containing text, crossword puzzle

Description automatically generated

Fig: 7.2.1: Same Row

The position of entered number is Incorrect. The number is present in the same row.

A picture containing text, crossword puzzle

Description automatically generated

Fig: 7.2.2: Same Column

The position of the entered number incorrect as number is present in the same column.

A picture containing text, crossword puzzle

Description automatically generated

Fig: 7.2.3: Same Grid

The position of the entered number incorrect as the number is present in the same sub-grid.

Text

Description automatically generated

Fig: 7.2.4 : Final Sudoku

Graphical user interface, text

Description automatically generated with medium confidence

Fig: 7.2.5**:**  Output

**8. CONCLUSION AND FUTURE WORK**

**8.1 CONCLUSION**

Graph colouring enjoys many practical applications as well as theoretical challenges. Beside the classical types of problems, different limitations can also be set on the graph or on the way a colour is assigned or even on the colour itself. It has even reached popularity with the general public in the form of the popular number puzzle Sudoku. Graph colouring is still a very active field of research.

It is amusing that the relationship between Sudoku and graph theory helps to find the number of possible Sudoku puzzles. In this project, the basic idea of Graph coloring and Sudoku are discussed. The main aim of this project is to understand how graph coloring is used for solving sudoku.

**8.2 FUTURE SCOPE**

Our project can be still further improved as **Steganography scheme using Sudoku Puzzle**.

Steganography is an art of concealing the existence of information within seemingly harmless carriers. It is a method similar to covert channels, spread spectrum communication and invisible inks which adds another step in security.

A message in cipher text may arouse suspicion while an invisible message will not. A digital image is a flexible medium used to carry a secret message because the slight modification of a cover image is hard to distinguish by human eyes.

**9. REFERENCES**

[1] A. Elumalai1. Malaya Journal of Matematik, Vol. S, No. 2, 1672-1674, 2020. Graph coloring and its implementation.

[2] Fonna, Muhammad Rizki. Makalah IF2211 Strategi Algoritma, Semester II Tahun 2018/2019. "Application of Backtracking Algorithm in Sudoku Solver." 13516001@std.stei.itb.ac.id <13516001@std.stei.itb.ac.id>;

[3] Gustavo Santos-Garcia. 28 July 2007. Solving Sudoku Puzzles with Rewriting Rules.

[4] Katelyn D. May, Murray State University. Colorings and Sudoku Puzzles. 2020.

[5] Kumar Ajay, Pruthi Manju. Year:2019, Volume: 11, Issue:1. Graph Coloring and its applications.

[6] Pilavakis, Nikolas. "Sudoku Solver." 2021.