Sudoku Using Graph Colouring Techniques

Sachin Prasanna

Department of Information Technology
National Institute of Technology Karnataka, Surathkal

***Abstract* – This paper gives a view on the different graph colouring problems and how solving a sudoku is basically just a m-Colouring decision problem.**

Key Terms – Graph Colouring, Sudoku, m Colouring, Connections

I. AN OVERVIEW

Graph coloring is a well-known problem in computer science and mathematics that involves assigning colors to the vertices of a graph such that no adjacent vertices share the same color. This problem has many practical applications in various fields such as scheduling, map coloring, and register allocation in compiler design.

The graph coloring problem is an NP-complete problem, meaning that there is no known polynomial time algorithm that can solve it for all cases. Therefore, researchers have proposed various methodologies to approximate solutions to the problem, with varying degrees of accuracy and efficiency.

The backtracking algorithm is a common approach to solving the graph coloring problem. It systematically searches through all possible colorings of the graph, backtracking whenever it reaches a dead end or discovers an illegal coloring. While this approach is guaranteed to find a solution, it can be very inefficient for large graphs, as the number of possible colorings grows exponentially with the number of vertices. This is the algorithmic approach we take in solving the 16x16 sudoku.

II. DIFFERENT PROBLEMS IN GRAPH COLOURING

1. **m Colouring Problem** : The graph and a set of colours are given. We are to find all ways in which graph can be coloured using the given colours.

2. **m-Colouring Decision Problem** : Graph and set of colours are given. We are to find out if graph can be coloured using the set of colours.

3. **m-Colouring Optimization Problem** : Graph is given, we are the find out the minimum of colours required to colour the graph.

III. M COLOURING DECISION PROBLEM

Solving the sudoku is the m-Colouring Decision problem. The numbers that can be filled in are the “colours” and each spot in the sudoku form the “nodes” of the graph. The problem is how are we going to connect the nodes of the graph such that this converts to a m Colouring Decision Problem. Once done, the sudoku can be solved by recursion and backtracking.

IV. ALGORITHM TO SOLVE THE SUDOKU

In a sudoku, the same number should not appear twice in the column, row and its block it is part of. Hence, the crux of the algorithm lies in how to connect each node. Here, each node is connected to all other nodes in its column, row and its block. This way you cannot have the same number (colour) in these places and thus the sudoku will be solved by simply running the Graph colouring algorithm on the sudoku graph.

The pseudo code is as follows:

```
color_graph (G, curr_vertex, num_vertices, color_array):
   if curr_vertex == num_vertices:
      print(color_array)
      return true

   for color in range(1, m+1):
      if is_color_safe(G, curr_vertex, color, color_array):
         color_array[curr_vertex] = color
         if color_graph (G, curr_vertex+1, num_vertices,
color_array):
            return true

         color_array[curr_vertex] = 0

   return false
```

Where, the **is_color_safe** function checks if the current number (colour) assigned to that particular place is safe and the same number does not exist in the same column, row or block.

The **color_graph** function colours each vertex in the graph according to the constraint above. Once all the vertices are coloured with a safe colour, the algorithm ends and displays the solved sudoku.