

Project Report

On

APPLICATIONS OF VERTEX COLOURING

Submitted

in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

MATHEMATICS

by

ANN MARY JOSE

(Register No. SM20MAT002)

(2020-2022)

Under the Supervision of

NISHA OOMMEN



DEPARTMENT OF MATHEMATICS
ST. TERESA'S COLLEGE (AUTONOMOUS)

ERNAKULAM, KOCHI - 682011

APRIL 2022

ST. TERESA'S COLLEGE (AUTONOMOUS), ERNAKULAM

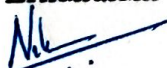


CERTIFICATE


This is to certify that the dissertation entitled, **APPLICATIONS OF VERTEX COLOURING** is a bonafide record of the work done by Ms. **ANN MARY JOSE** under my guidance as partial fulfillment of the award of the degree of **Master of Science in Mathematics** at St. Teresa's College (Autonomous), Ernakulam affiliated to Mahatma Gandhi University, Kottayam. No part of this work has been submitted for any other degree elsewhere.

Date: 26/5/2022

Place: Ernakulam


Nisha Oommen
Assistant Professor,
Department of Mathematics,
St. Teresa's College(Autonomous),
Ernakulam.

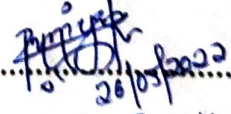



Dr. Ursala Paul
Assistant Professor & HOD,
Department of Mathematics,
St. Teresa's College(Autonomous),
Ernakulam.

External Examiners

1: DIVYA.....MARY.....DAISE-S


26/05/2022

2:

26/05/2022
MARY RUNIYA

DECLARATION

I hereby declare that the work presented in this project is based on the original work done by me under the guidance of NISHA OOMMEN, Assistant Professor, Department of Mathematics, St. Teresa's College(Autonomous), Ernakulam and has not been included in any other project submitted previously for the award of any degree.

Ernakulam.

Date: 26/05/2022



ANN MARY JOSE

SM20MAT002

ACKNOWLEDGEMENTS

I would like to express my gratitude to my project guide Nisha Oommen, Assistant Professor, Department of Mathematics, St. Teresa's College (Autonomous), Ernakulam for the valuable guidance, patience and encouragement during the entire period of this work.


I would like to express my gratitude to our HOD Dr. Ursula Paul for the continuous support and guidance. I would like to extend my gratitude to all the teaching and non-teaching staff of the department, the energy and constant encouragement of the Department had helped me a lot in completing this project.

Also I am highly thankful to my family for supporting me during the compilation of this project.

Above all, I thank God Almighty for the successful completion of this project.

Ernakulam.

Date: 26/05/2022


ANN MARY JOSE
SM20MAT002

Contents

<i>CERTIFICATE</i>	ii
<i>DECLARATION</i>	iii
<i>ACKNOWLEDGEMENTS</i>	iv
<i>CONTENT</i>	v
1 Literature Review	1
1.1 Introduction	1
1.2 History	2
1.3 Preliminaries	3
1.4 Graph Colouring	7
1.4.1 Graph Colouring	7
1.4.2 Vertex Colouring	7
1.4.3 Edge Colouring	7
1.4.4 Region Colouring	7
1.4.5 Graph Colouring Problem	7
2 Chromatic Partitioning	9
2.1 Independent Set	9
2.1.1 Maximal Independent Set	9
2.1.2 Maximum Independent set	9
2.1.3 Independence Number	9
2.2 Chromatic Partitioning	9
2.2.1 Colour classes	10
2.2.2 Upper and Lower bound of Chromatic Number	10
2.3 Clique	10
2.3.1 Clique Number	10
2.3.2 Critical Graph	11

2.4	Brook's Theorem	12
3	Chromatic Polynomials	13
4	Applications	16
4.1	In Traffic Light Signals	16
4.2	Sudoku	18
4.2.1	Algorithm	20
4.2.2	Python Program	20
4.3	Frequency Assignment	36
4.4	Register Allocation	37
5	Conclusion	39
	<i>REFERENCES</i>	40

Chapter 1

Literature Review

1.1 Introduction

Graph Colouring is a relatively new area of Mathematics. It is also a field in which much advancements are taking place. Graph Colouring is one of the sub-branch of Graph theory. Graph theory can be considered as a branch of Mathematics that depends very little on other branches of Mathematics and is independent in itself [1]. Graph theory is having a lot of applications. These are extended to other branches like Chemistry, Computer Science, etc. It has real life applications in many other fields too.

Origin of graph theory can be traced back to Leonhard Euler's paper on Königsberg bridge problem and it is considered as the first paper in the history of Graph theory. It was published in 1736 and Leonhard Euler is known as the father of graph theory. The problem is about two islands C and D formed by Pregel river in Königsberg. They were connected with each other and the banks A and B by means of seven bridges. The problem is to start from any of the four areas A, B, C and D and return back to the starting point by travelling through the seven bridges exactly once. Euler presented this problem by means of a graph such that land masses are represented by vertices and edges represents the bridges connecting them. This marked the beginning of graph theory. After that a lot of advancements have taken place and still now it is an active area of research.

Graph colouring is one of the sub-field of graph theory in which a lot of researches are going on. It is the assignment of colours to elements of a graph under some constraints. Graph theory started as the colouring of maps such that coun-

tries sharing the same boundary receives different colours by a question raised by Francis Guthrie. He postulated this as the four colour conjecture which states that it is possible to colour the map with no more than four colours. A lot of discussions had take place. And famous scientists like Augustus De Morgan, William Hamilton studied this problem. In this way a simple question leads to the birth of graph colouring. Kenneth Appel and Wolfgang Haken finally proved Four colour theorem in 1976 with the help of computer assistance. Their proof on Four colour theorem was the first major proof done with computer assistance. Meanwhile many other concepts like chromatic polynomials got its growth. In this way graph colouring had been always and still an active field of research.

Through this project we try to discuss about various applications of vertex colouring and to show how they are being related to our lives.

1.2 History

Graph colouring got its beginning by an interesting question put forwarded by Francis Guthrie. He while trying to colour the map of England noticed that four colours are sufficient to colour the map in such a way that no two regions with a common border is coloured with the same colour. Francis Guthrie's brother Frederick Guthrie passed this question to his professor Augustus De Morgan and he enquire about this observation in a letter to Hamilton on 23rd October 1852. De Morgan had continued his discussions about the Four Colour Conjecture to many of his friends. And it had remained as an unsolved problem for a long time.

On 17 June 1879 Cayley brought out this problem before the London Mathematical Association during a meeting. In the same year he sends a paper 'On the colouring of maps' to Royal Geographical Society [6]. It discusses about what all are the difficulties that are prevailing to solve this problem. Later it was published on 1879. But in the same year Alfred Bray Kempe one of Cayley's student announced that he has proof and on Cayley's advice Kempe published his work on American Journal of Mathematics in 1879 itself. His proof also contained Kempe chain argument, one of his famous work. For a long time Kempe's proof remain true.

But in 1890 Percy John Heawood points out the errors in Kempe's proof in his paper 'Map Colouring Theorem'. It was published in the Quarterly Journal

of Pure and Applied Mathematics. So again four colour theorem turn out to be four colour conjecture. But Heawood's paper give birth to another theorem called Five Colour theorem and Heawood proved this theorem in the same paper. Five colour theorem states that every planar graph can be coloured with five colours. A proof for four colour theorem was finally accepted in 1976. The proof was done by Kenneth Appel and Wolfgang Haken with the help of J. Koch and it was a computer based proof. Because of that many mathematicians did not consider it as a valid one for many years. But their proof becomes much more acceptable when Neil Robertson, Daniel Sanders, Paul Seymour and Robin Thomas developed a more systematic proof around 1994.

1.3 Preliminaries

Graph:

A graph is an ordered triplet $G = (V(G), E(G), I_G)$ where $V(G)$ is a non empty set, $E(G)$ is a set disjoint from $V(G)$ and I_G is an 'incidence' map that associates with each elements of $E(G)$, an unordered pair of elements (same or distinct) of $V(G)$.

Elements of $V(G)$ are called *vertices* and the elements of $E(G)$ are called the *edges* of G [2].

End Vertices:

A vertex v is incident with an edge e if there is an edge at v . Two vertices are said to be end vertices if they are incident with the same edge [3].

Self Loop:

An edge with same end vertex as end vertices is called a self-loop.

Adjacent Edges:

Two edges are said to be adjacent edges if they are incident on the same vertex.

Adjacent Vertices:

Two vertices that are the end vertices of the same edge is said to be adjacent

vertices.

Parallel Edges:

Two or more edges are said to be parallel if they are incident with same pair of vertices.

Simple Graph:

A graph is simple if it does not have any self-loop or parallel edges.

Degree of a vertex:

Degree of a vertex is defined as the number of edges incident on a vertex.

Maximum degree:

It is the maximum of degrees of the vertices of a graph G . It is denoted by $\Delta(G)$ [4].

Minimum degree:

It is the minimum of degrees of the vertices of a graph G . Minimum degree is denoted by $\delta(G)$ [4].

Isolated vertex:

A vertex whose degree is zero is known as an isolated vertex.

Walk:

It is a finite alternating sequence of edges and vertices in such a way that it begins and end with a vertex and each edge have an end vertices. Vertices can be repeated in a walk.

If a walk begins and ends at the same vertex then it is called a closed walk. A walk that is not closed is called open.

Trail:

A walk in which no edge is repeated is called a trail.

Path:

A trail in which no vertices and no edges are repeated is a path. Since a path is a trail, it is also an open walk.

Length of a Walk:

Number of edges in a walk is known as its length. Similarly length of a path or trail is the number of edges in the path or trail respectively.

Cycle:

A closed walk in which no edges and no vertices except the initial and final vertex appears more than once is known as a cycle.

Circuit:

A closed trail is known as a circuit.

Connected Graph:

A graph in which if every pair of vertices has at least one path between them is known as connected.

Disconnected Graph:

A graph that is not connected is known as a disconnected graph.

Tree:

A connected graph without any cycle is known as a tree.

Forest:

A graph without cycles is known as a forest. So each component of a forest is a tree [4].

Complete Graph:

A graph in which every pair of vertices is connected by an edge is called a complete graph.

Vertex Index:

Usually a vertex has a name associated with it. Internally, within the implementation of the graph in computer programs, for the computer to recognise it may be more convenient to refer a vertex using an integer number [5]. This integer number is known as an Vertex Index.

Planar Graph:

A graph is said to be planar if it can be embedded in a plane or a graph is said to be planar if it can be drawn in a plane so that no edges cross over each other.

Proper Colouring

Colouring the vertices of a graph such that no two adjacent vertices have the same colour is called the proper colouring of a graph.

A graph in which every vertex is given a proper colour according to proper colouring is known as a properly coloured graph. There are a lot of ways in which a given graph can be properly coloured. But in graph colouring problems it will be of interest to produce a properly coloured graph which uses minimum number of colours.

K - Colourable Graph

A graph is k - colourable, if it has a proper k-colouring, i.e. it will use one of k colours to produce a proper colouring of the graph. Here we are considering a graph without self - loops.

K – Chromatic Graph

If a graph uses k different colours and no less than k colours to obtain its proper colouring then such a graph is known as k – chromatic graph or the graph has chromatic number k [9].

- A graph with only isolated vertices is 1-chromatic.
- Every tree is 2 - colourable.
- A complete graph on n vertices is n chromatic [2].

Chromatic Number

Chromatic number of a graph G is denoted by $\chi(G)$ and it is defined as the minimum number of colours needed for obtaining a proper colouring of G . So G is K -chromatic if $\chi(G) = k$ [4].

1.4 Graph Colouring

1.4.1 Graph Colouring

In Graph theory, graph colouring is the assignment of colours to elements of a graph subject to certain constraints. By elements of a graph it means edges, vertices, regions, etc. Graph colouring is mainly of three types Vertex colouring, Edge colouring and Region colouring. Besides these, there are many other types of graph colouring like List colouring, List-edge colouring, Acyclic colouring, Oriented colouring, etc. The specialty of vertex colouring is that any type of graph colouring can be converted into a vertex colouring.

1.4.2 Vertex Colouring

By the term graph colouring, in most cases it means vertex colouring. Vertex colouring is defined as the assignment of colours to the vertices of a graph such that no two adjacent vertices have the same colour.

1.4.3 Edge Colouring

An edge coloring is the assignment of colours to the edges of a graph such that no two incident edges share the same colour [1].

1.4.4 Region Colouring

In region colouring or face colouring of a planar graph a colour is assigned to each region or face so that no two faces that share a boundary share the same colour.

1.4.5 Graph Colouring Problem

Graph colouring problem is the labelling of certain elements of a graph, subject to certain constraints [7]. The labels are typically called colours. Graph colouring

problem has found applications in many fields like Sudoku, Geographical maps, etc. The most common graph colouring problem is Vertex colouring.

Here when a graph G with n vertices is given, our main aim is to find the minimum number of colours that can be used to paint its vertices such that no two adjacent vertices have the same colour. This is the colouring problem [8].

Chapter 2

Chromatic Partitioning

2.1 Independent Set

Let G be a graph. Then a set of vertices of G is said to be independent set if no two vertices in it are adjacent in G .

2.1.1 Maximal Independent Set

Maximal Independent Set can be defined as an independent set to which no other vertices can be added without destroying its independence property.

2.1.2 Maximum Independent set

A graph can have a number of maximal independent sets of varying sizes. Among them an maximal independent set is called maximum independent set if it's cardinality is maximum than the others [2].

2.1.3 Independence Number

It is the number of vertices in a maximum independent set and is denoted by $\beta(G)$.

2.2 Chromatic Partitioning

Let G be a simple connected graph. Partitioning the vertices of G into smallest possible number of disjoint independent set is known as chromatic partitioning. In other words chromatic partitioning is obtained by finding all maximal independent sets and selecting the smallest number of sets that contain all the vertices of G .

2.2.1 Colour classes

If G is a k -chromatic graph then its vertex set $V(G)$ can be partitioned into k independent sets V_1, V_2, \dots, V_k called colour classes and vertices in same colour class will be assigned same colour [10]. In other words a colour class is a set of vertices which are having the same colour when a graph is coloured [1].

2.2.2 Upper and Lower bound of Chromatic Number

Theorem:

Let H be a subgraph of a graph G . Then,

$$\chi(H) \leq \chi(G) \quad (2.1)$$

Proof:

Let H be a subgraph of a graph G . Then any colouring of G produces the colouring of H too. Since it will possibly uses $\chi(G)$ or even fewer colours for H than G to get a colouring [10],

$$\chi(H) \leq \chi(G)$$

2.3 Clique

A complete subgraph of G is known as its Clique.

2.3.1 Clique Number

It is the order of the largest clique in a graph G . It is denoted by $\omega(G)$.

Theorem:

For every graph G ,

$$\chi(G) \geq \omega(G) \quad (2.2)$$

and

$$\chi(G) \geq \frac{n}{\beta(G)} \quad (2.3)$$

where n is the order of G .

Proof:

Let H be a clique of G with order $\omega(G)$. Then since H is a clique of G , it is a complete subgraph of G . So chromatic number of H is the number of vertices in

it, which is $\omega(G)$.

i.e., $\chi(G) = \omega(G)$.

Also H is a subgraph of G , therefore $\chi(H) \leq \chi(G)$.

i.e, $\chi(H) \geq \omega(G)$

Let $k = \chi(G)$. Suppose that G is coloured with k colours. Let the vertex set $V(G)$ be partitioned into k independent sets V_1, V_2, \dots, V_k . These are the colour classes and since they are independent any of them will be of maximum size β [10] [11]. Then,

$$n = \sum_{i=1}^k |V_i| \leq k\beta \quad (2.4)$$

i.e,

$$\chi(G) \geq \frac{n}{\beta(G)}$$

2.3.2 Critical Graph

A graph G is called critical if $\chi(H) < \chi(G)$ for every proper subgraph H of G . Also a graph is called K - critical if it is both K - chromatic and critical.

Theorem:

If G is K - critical then,

$$\delta(G) \geq k - 1 \quad (2.5)$$

Proof:

The proof is by contradiction. Suppose $\delta(G) < k-1$. Let V be a vertex of minimum degree in G , i.e., degree of V is δ in G . $G-V$ is $k-1$ colourable as G is k - critical. Then V will have at most $k-2$ neighbours and they can be coloured using at most $k-2$ colours in the $k-1$ colouring of $G-V$. So if we colour V with the remaining one colour of $k-1$, then a proper $k-1$ colouring is obtained for G . This is a contradiction as G is k - critical. Hence the assumption is wrong. So, $\delta(G) \geq k-1$ [12].

Theorem:

For any graph G ,

$$\chi(G) \leq 1 + \delta(G) \quad (2.6)$$

Proof:

Let G be a k chromatic graph. So $\delta(G) = k$. Let H be a subgraph of G , which is k - critical. i.e., $\chi(G) = k$. Also by above theorem since H is k - critical $\delta(H) =$

k-1. Then,

$$\begin{aligned}k &\leq 1 + \delta(H) \\ &\leq 1 + \Delta(H) \\ &\leq 1 + \Delta(G)\end{aligned}$$

i.e., $k \leq 1 + \Delta(G)$ where k is $\chi(G)$. Hence the proof [4].

2.4 Brook's Theorem

Let G be a connected simple graph which is neither an odd cycle nor a complete graph then,

$$\chi(G) \leq \Delta(G) \tag{2.7}$$

Note:

So Brook's theorem tells that if G is connected then,

$$\chi(G) = 1 + \Delta(G) \tag{2.8}$$

iff either G is an odd cycle or a complete graph [13].

Chapter 3

Chromatic Polynomials

As said earlier, there are different ways in which a graph can be properly coloured. So a graph G with n vertices can be properly coloured in many different ways using a sufficiently large number of colours [2]. We can express this property of a graph using a polynomial called chromatic polynomial.

For a graph G with n vertices chromatic polynomial can be denoted by $P_n(\lambda)$ and its value is defined as the number of ways of properly colouring a graph with λ or fewer colours.

If for instance a graph G can be properly coloured using i colours in c_i different ways, then

$$P_n(\lambda) = \sum_{i=1}^n c_i \binom{\lambda}{i} \quad (3.1)$$

As there are $\binom{\lambda}{i}$ different ways in which i colours can be selected from λ colours and as a result there will be $c_i \binom{\lambda}{i}$ different ways of properly colouring a graph using i colours out of the λ colours. So its sum as i vary will be the chromatic polynomial.

Theorem:

The chromatic polynomial $P_n(\lambda)$ of a graph G is a polynomial in λ [14].

Proof:

While colouring the graph G , $V(G)$ will be partitioned into independent sets and a unique colour will be assigned to each set. From the given λ colours there are λ ways to choose a colour for the first set, $(\lambda-1)$ ways to choose colours for the second set and this process can be continued similarly for the other independent sets. As a result there will be $\lambda(\lambda-1)(\lambda-2)\dots$ different possible ways in which the given graph G can be properly coloured, which is its chromatic polynomial. Clearly, the

chromatic polynomial is a polynomial [15].

It is not so easy to calculate chromatic polynomials in this simple method of inspection. So in this context Deletion Contraction becomes useful.

Deletion-contraction:

Let G be a graph with an edge e incident on the vertices u and v . The deletion of e is denoted by $G - e$ and it is the graph obtained by deleting an edge e from G without causing any change to the vertices. The contraction of e is denoted by G/e . It is obtained from $G - e$ by fusing the end vertices of the edge e deleted in $G - e$ earlier.

Theorem:

Let G be a graph and e be its edge, then its chromatic polynomial is,

$$P_G(\lambda) = P_{(G-e)}(\lambda) - P_{(G/e)}(\lambda) \tag{3.2}$$

Proof:

Given G is a graph and e an edge in it. Let u and v be the end vertices of e . To get proper colouring u and v should be given different colours. But in $G - e$ the edge e is deleted, so no longer they are adjacent. So u and v can be given same or different colours. In G/e u and v are fused as a single vertex, so they have same colour.

\therefore Total number of colourings of $G - e$ is equal to the total number of colourings of G/e and G .

Thus,

$$P_{(G-e)}(\lambda) = P_G(\lambda) + P_{(G/e)}(\lambda)$$

On rearranging,

$$P_G(\lambda) = P_{(G-e)}(\lambda) - P_{(G/e)}(\lambda)$$

Hence proved [16].

Properties:

- For any graph G the degree of chromatic polynomial is its number of vertices in G .
- The leading coefficient in $P_G(\lambda)$ for any graph G is 1.
- The first coefficient will be positive and others will alternate in sign.

- All the coefficients are integers [16].
- The chromatic polynomial of a complete graph on n vertices is $\lambda(\lambda-1)(\lambda-2)\dots(\lambda-n+1)$.
- A graph with n vertices is a tree iff its chromatic polynomial is $\lambda(\lambda-1)^{n-1}$.

As discussed in the introduction, the beginning of Graph colouring is by the four colour conjecture. Next we discuss about the two important theorems in graph colouring, which ultimately leads to the growth of this branch.

Four colour theorem:

Initially it appeared as a question raised by Francis Guthrie. He while colouring the administrative map of England noticed that only four colours are needed to colour the map such that neighbouring countries are given different colours. It was first a conjecture. After a lot of proofs and disproofs finally it was proved in 1976. This was proved by Kenneth Appel and Wolfgang Haken with the help of computers.

Four Colour theorem can be stated as follows, given any separation of a plane into contiguous regions, producing a figure called map, no more than four colours are required to colour the regions of the map so that no two adjacent regions have the same colour. Or it can be stated in terms of vertex colouring as: the chromatic number of every planar graph is at most 4.

Five colour theorem:

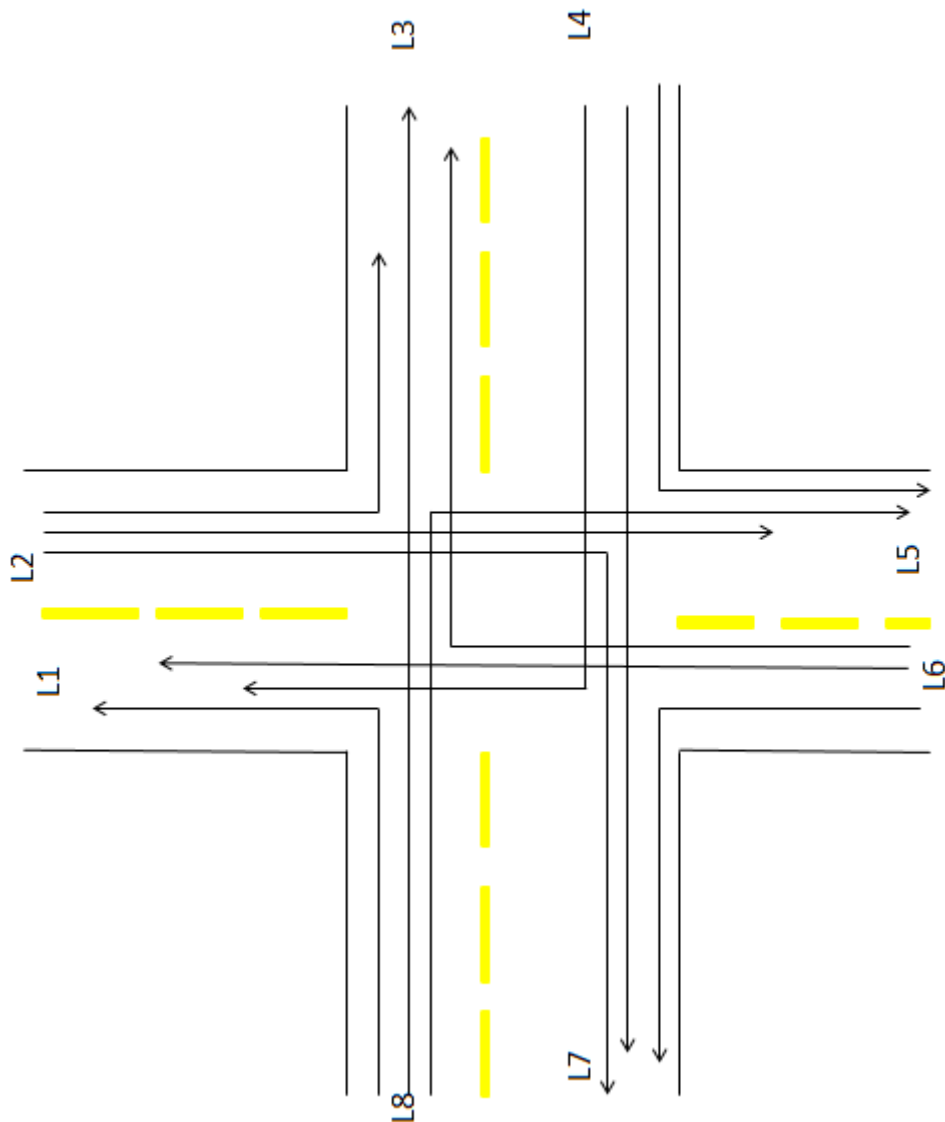
This is a theorem formed by Heawood in 1890 to show that Kempe's proof on Four colour theorem is wrong. It states that, the vertices of every planar graph can be properly coloured with five colours.

Chapter 4

Applications

4.1 In Traffic Light Signals

One of the applications of Vertex colouring is in the traffic light signals. It can be used to find the number of phases in which the signals should be operated so that all vehicles could pass the traffic intersection without causing accidents. Here I have considered an example of traffic intersection as given below and have used vertex colouring to find the number of phases in which signals should be operated.

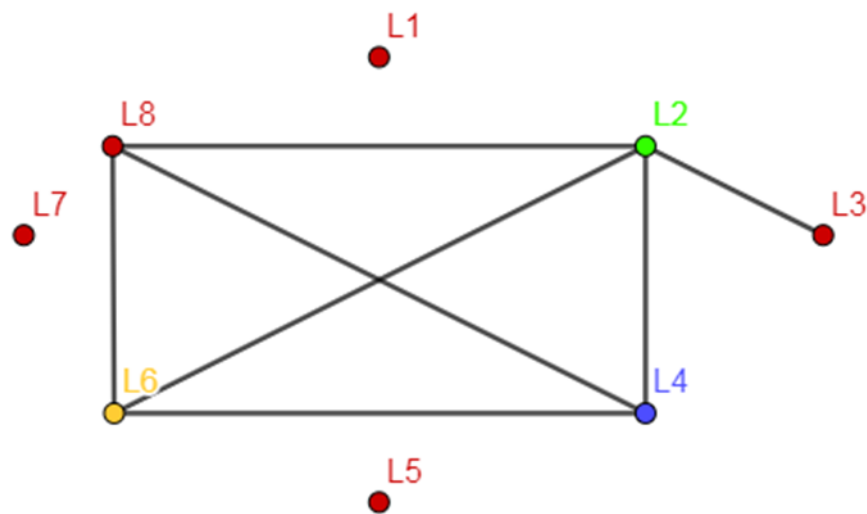


(an example of traffic lanes at a street intersection)

Here there are 8 traffic lanes which are named as L_1, L_2, \dots, L_8 at the intersection of two streets. The arrow marks shows the direction in which vehicles moves through each lane. Here there is a traffic signal situated at the intersection. During a particular phase the vehicles in the lane for which the signal is green will passes through the intersection. We can use vertex colouring in this context to find the minimum number of phases needed such that all vehicles passes the intersection smoothly.

The problem can be transformed into a graph colouring problem. Here a graph G is constructed whose vertices will represent each lanes L_1, L_2, \dots, L_8 . Let

$V(G)=\{L_1, L_2, \dots, L_8\}$. So there will be 8 vertices in this particular case. Now connect the vertices with edges if vehicles cannot run together without chances of accidents through the corresponding lanes represented by the vertices. For eg., in this case edges are connected between L_2 and L_4 as the vehicles in these lanes are moving in the same direction. Also an edge is connected between L_2 and L_8 as vehicles move across each other at intersection through these lanes. In a similar manner edges are drawn between vertices. And a proper vertex colouring is given to the graph so obtained, as given below.



(Graph G formed for the above problem)

In order to find the minimum number of phases in which the traffic signal should be operated so that all vehicles pass the intersection smoothly, we have to find the chromatic number of the graph G. From the proper colouring it is clear that it is 4. So our solution is obtained [10].

In this way vertex colouring can be used to model real life problems and to solve it.

4.2 Sudoku

Sudoku is a logic based, combinatorial, number placement puzzle. It is a single player game. A classic Sudoku puzzle consists of a 9x9 grid with numbers ranging from 1 to 9 such that no number is repeated in each row, each column and each of the nine 3x3 subgrid that constitute the whole grid. The player will be provided

with a partially filled Sudoku puzzle and his or her aim is to complete the puzzle using numbers from one to nine, satisfying the above said condition [17].

A Sudoku is solved by the classical pen and paper method. But it can also be solved simply with the help of various algorithms and computer programs. Solving of Sudoku can be considered as an application of vertex colouring too. ie., solving a Sudoku can be considered as an vertex colouring problem.

Even though a standard Sudoku is 9x9, here we considered the case of a 4x4 Sudoku puzzle which is partially filled with numbers from one to four. A 4x4 Sudoku have four rows, four columns and four 2x2 subgrids. There are 16 cells in such a Sudoku puzzle. Our aim is to fill the blank cells in it using numbers from 1 to 4. Inorder to do this we first convert it as a graph colouring problem.

For that a graph with 16 vertices is created. These 16 vertices corresponds to the 16 cells of the Sudoku and two vertices are connected by an edge if they cannot have the same value. Therefore vertices corresponding to cells in same row, same column or same 2x2 subgrid have edges between them, since no number can be repeated in the same row, same column and same 2x2 sub grid. Then a proper colouring of this graph will be the solution to the Sudoku puzzle. Here colours will be the numbers 1, 2, 3 and 4. In this way we can transform a Sudoku puzzle into a graph colouring problem. In this case the graph colouring problem can be solved using four colours since the chromatic number of graph created is four [18]. We can consider a graph colouring problem in different ways:-

1. A graph and a set of colours will be provided. Our aim will be to find the all possible ways in which the given graph can be coloured using the given colours.
2. To find whether the given graph can be coloured using the given set of colours. It is known as a m-colouring decision problem.
3. To find the minimum number of colours required to colour a given graph while the set of colours is not given. It can be called as a m-colouring optimization problem.

Here we will consider the solving of Sudoku as an m-colouring decision problem.

4.2.1 Algorithm

Our objective is to give colour to each of the vertices one by one. If the vertex 0 is given a particular colour, then while giving colour to the next vertex, check whether the colour is not the same. If there is a colour assignment that satisfies these conditions, keep it as part of solution. If not so, backtrack and return false [19]. So the algorithm will be as follows:

1. Create a recursive function whose arguments are current vertex index, number of vertices and output colour array.
2. Check whether the current vertex index and number of vertices are equal.
3. If step 2 satisfied, return true. Then print colour configuration as output array.
4. Give colour to a vertex. While doing so make sure the adjacent vertices are having different colours.
5. Recursively call the function with next index and number of vertices [19].
6. Break the loop and return true if recursive function also returns true and return false if there is no such recursive function.

There are many other algorithms and corresponding programs to solve Sudoku using vertex colouring. Here by using the above algorithm a python program will be produced.

4.2.2 Python Program

First of all a python program named **model.py** is created to generate graph. It mainly consists of two classes named as **Vertex** and **Graph**. In **Vertex** class different types of functions are created. First one is **__init__** which is similar to a constructor in C++ and it is used to initialize the class. In it the instance variables are id, data and **connectedTo**. The **connectedTo** is a dictionary which is used to store the ids of vertices connected to it along with its weight. Next in the function named **add_neighbour** the id and weight pair is stored to the **connectedTo** dictionary. To the last part of this particular class a setter is used to set the value of private attributes of the class and a set of getters are used to get access to the private attributes of the class.

The second class created was **Graph**. In it the total number of vertices in the graph is represented by a variable named **totalvertices**. At the beginning of this class **totalvertices** is initialized by the value zero. The program contains a dictionary called **allvertices** which stores the vertex id as both key and its value. This dictionary will help in accessing the vertices of the graphs with just its ids. Next a function named **addvertexdata** is defined to create a vertex and to store it to the **allvertices** dictionary and a function named **addedge** is created to add edges between two given vertices. Then a function is defined to check whether each vertex is a neighbour of the other, for all vertex pair combinations. This function is named as **isneighbour**. A function is defined to print the edges and it is named as **printedges**. Now similar to the last class a set of setters and getters are defined in this class too.

After the creation of two classes a test main function is created. In it the **Graph** class is called first and a graph object is created and to it six vertices with ids from zero to five is added [19]. Then the remaining code is builded by using previous functions in the classes to add edges between vertices and to print them. This module is saved as a file named **model.py**. In this way a blueprint of graph and vertices are created. Now in the next program by using this we will create our graph.

Python code of model.py (program 1):

```
class Vertex:
    def __init__(self, idx, data=0):
        self.id = idx
        self.data = data
        self.connectedTo = dict()

    def add_neighbour(self, neighbour, weight=0):
        if neighbour.id not in self.connectedTo.keys():
            self.connectedTo[neighbour.id] = weight

    def setdata(self, data):
        self.data = data
```

```
def getconnections(self):
    return self.connectedTo.keys()

def getid(self):
    return self.id

def getdata(self):
    return self.data

def getwght(self, neighbour):
    return self.connectedTo[neighbour.id]

def __str__(self):
    return str(self.data) + " Connected to : " + \
           str([x.data for x in self.connectedTo])

class Graph:
    totalvertices = 0

    def __init__(self):
        self.allvertices = dict()

    def addvertex(self, idx):
        if idx in self.allvertices:
            return None

        Graph.totalvertices += 1
        vertex = Vertex(idx=idx)
        self.allvertices[idx] = vertex
        return vertex
```



```
def addvertexdata(self, idx, data):

    if idx in self.allvertices:
        vertex = self.allvertices[idx]
        vertex.setData(data)
    else:
        print("No id to add the data.")

def addedge(self, src, dst, wt=0):

    self.allvertices[src].add_neighbour(
        self.allvertices[dst], wt)
    self.allvertices[dst].add_neighbour(
        self.allvertices[src], wt)

def isneighbour(self, u, v):

    if u >= 1 and u <= 16 and v >= 1 and v <= 16 and\
        u != v:
        if v in self.allvertices[u].getconnections():
            return True
    return False

def printedges(self):

    for idx in self.allvertices:
        vertex = self.allvertices[idx]
        for con in vertex.getconnections():
            print(vertex.getid(), " --> ",
                  self.allvertices[con].getid())

def getvertex(self, idx):
```

```
        if idx in self.allvertices:
            return self.allvertices[idx]
        return None

def getallverticesids(self):
    return self.allvertices.keys()

def main():
    g = Graph()
    for i in range(6):
        g.addvertex(i)

    print("Vertices : ", g.getallverticesids())

    g.addedge(src=0, dst=1, wt=5)
    g.addedge(0, 5, 2)
    g.addedge(1, 2, 4)
    g.addedge(2, 3, 9)
    g.addedge(3, 4, 7)
    g.addedge(3, 5, 3)
    g.addedge(4, 0, 1)
    g.addedge(5, 4, 8)
    g.addedge(5, 2, 1)

    g.printedges()

main()
```

Output:

```

C:\Users\sony\Desktop\python>python model.py
Vertices : dict_keys([0, 1, 2, 3, 4, 5])
0 --> 1
0 --> 5
0 --> 4
1 --> 0
1 --> 2
2 --> 1
2 --> 3
2 --> 5
3 --> 2
3 --> 4
3 --> 5
4 --> 3
4 --> 0
4 --> 5
5 --> 0
5 --> 3
5 --> 4
5 --> 2
C:\Users\sony\Desktop\python>_

```

In the next program first of all the **Graph** class is imported from the previous program and then created a class called **Sudoku**. In it inside a constructor the total value of rows, columns and blocks are initialized. Since we are using a 4x4 matrix values are given accordingly. Also a function to generate the vertices is called inside the constructor. This function is defined outside the constructor and the vertices created by the function are stored in the instance variable **self.graph**. Now as vertices are created our next aim is to connect edges between vertices in the same row, same column and same 2x2 block. For connecting the vertices the method called **connectededges(self)** is called.

Inorder to connect edges like as said above, one need to know the position of each vertex. For that first each vertex is given a position using **getgridmatrix(self)** function [19]. Then a particular element is named as head and we will need to find which all other vertices are connected to it. Similarly the process should be repeated for all other vertices by considering them as individual cases. For that a nested loop is used. The function **whattoconnect** will decide which all elements should be connected to head. This function has a dictionary named **connections** which stores all these connections. It will store **row** as the key and all the ids in the row as key value. Similarly **cols** will be a key with value as ids in the column and the next key is **block** with value as all ids in the 2x2 block.

Now to connect a vertex and every other vertices in the same row a **for** loop is used so that row will remain same and column value gets changed. Similarly to connect the vertex with all vertices in the same column a **for** loop is again used. Inorder to connect a vertex and other vertices in the same 2x2 block a set

of **if-else** conditions are used. After these **for** loops and **if-else** conditions we will get ids of the vertices in same row, same column and same block which are to be connected. Now using these ids the vertices can be connected as per the constraints.

Finally a function **test_connections** is defined in which the class **Sudoku** is called. Then vertices are generated and connected them as per the conditions made in the class.

This program is saved as a module named **connections.py**. The program code for **connections.py** (program 2) is as follows:

Python program for connections.py (program 2):

```
from model import Graph
class Sudoku:
    def __init__(self):

        self.graph = Graph()

        self.rows = 4
        self.cols = 4
        self.total_blocks = self.rows * self.cols

        self.generateGraph()
        self.connectededges()

        self.allids = self.graph.getallverticesids()

    def generateGraph(self):
        for idx in range(1, self.total_blocks + 1):
            _ = self.graph.addvertex(idx)

    def connectededges(self):
        matrix = self.getmatrix()
        head_connections = dict()
        for row in range(4):
```

```
    for col in range(4):
        head = matrix[row][col]
        connections = self.whattoconnect(
            matrix, row, col)
        head_connections[head] = connections
self.connectthose(head_connections=head_connections)

def connectthose(self, head_connections):
    for head in head_connections.keys():
        connections = head_connections[head]
        for key in connections:
            for v in connections[key]:
                self.graph.addedge(src=head, dst=v)

def whattoconnect(self, matrix, rows, cols):

    connections = dict()

    row = []
    col = []
    block = []

    for c in range(cols + 1, 4):
        row.append(matrix[rows][c])
    connections["rows"] = row

    for r in range(rows + 1, 4):
        col.append(matrix[r][cols])
    connections["cols"] = col

    if rows % 2 == 0:
        if cols % 2 == 0:
            block.append(matrix[rows + 1][cols + 1])
```

```

        elif cols % 2 == 1:
            block.append(matrix[rows + 1][cols - 1])
elif rows % 2 == 1:
    if cols % 2 == 0:
        block.append(matrix[rows - 1][cols + 1])
    elif cols % 2 == 1:
        block.append(matrix[rows - 1][cols - 1])
connections["blocks"] = block
return connections

def getmatrix(self):
    matrix = [[0 for cols in range(self.cols)]
              for rows in range(self.rows)]
    count = 1
    for rows in range(4):
        for cols in range(4):
            matrix[rows][cols] = count
            count += 1
    return matrix

def test_connections():
    sudoku = Sudoku()
    sudoku.connectededges()
    print("All vertex ids : ")
    print(sudoku.graph.getallverticesids())
    print()
    for idx in sudoku.graph.getallverticesids():
        print(idx, "Connected to->",
              sudoku.graph.allvertices[idx].getconnections())

test_connections()

```

Output:

```

C:\Users\sony\Desktop\python>python connections.py
Vertices : dict_keys([0, 1, 2, 3, 4, 5])
0 --> 1
0 --> 5
0 --> 4
1 --> 0
1 --> 2
2 --> 1
2 --> 3
2 --> 5
3 --> 2
3 --> 4
3 --> 5
4 --> 3
4 --> 0
4 --> 5
5 --> 0
5 --> 3
5 --> 4
5 --> 2
All vertex ids :
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
1 Connected to-> dict_keys([2, 3, 4, 5, 9, 13, 6])
2 Connected to-> dict_keys([1, 3, 4, 6, 10, 14, 5])
3 Connected to-> dict_keys([1, 2, 4, 7, 11, 15, 8])
4 Connected to-> dict_keys([1, 2, 3, 8, 12, 16, 7])
5 Connected to-> dict_keys([1, 2, 6, 7, 8, 9, 13])
6 Connected to-> dict_keys([1, 2, 5, 7, 8, 10, 14])
7 Connected to-> dict_keys([3, 4, 5, 6, 8, 11, 15])
8 Connected to-> dict_keys([3, 4, 5, 6, 7, 12, 16])
9 Connected to-> dict_keys([1, 5, 10, 11, 12, 13, 14])
10 Connected to-> dict_keys([2, 6, 9, 11, 12, 14, 13])
11 Connected to-> dict_keys([3, 7, 9, 10, 12, 15, 16])
12 Connected to-> dict_keys([4, 8, 9, 10, 11, 16, 15])
13 Connected to-> dict_keys([1, 5, 9, 10, 14, 15, 16])
14 Connected to-> dict_keys([2, 6, 9, 10, 13, 15, 16])
15 Connected to-> dict_keys([3, 7, 11, 12, 13, 14, 16])
16 Connected to-> dict_keys([4, 8, 11, 12, 13, 14, 15])
C:\Users\sony\Desktop\python>_

```

Now from the above program **connections.py** the class **Sudoku** is called to a new program. Here a new class **Sudoku_soln()** is created to print an incomplete Sudoku that we have chosen. Now to the next part of program the algorithm will be applied. For that first a function named **graphcoloring** is created which will assign colours to the index which are given in the puzzle. For that the variable **color** which is a one dimensional array is used in which each index is a vertex and value at vertex represents the colour given to that vertex. So in this case colours are numbers from one to four. Also the variable **partial_soln** is a list used to store the numbers provided in the puzzle. As per the algorithm a recursive function will be called to the last of this function definition.

Now in the next function **graphcolor** at first it is checked whether the vertex **u** is equal to the total number of vertices, if so it will return true. After that a **for** loop is used to colour **u** using numbers from one to four. Before doing so using the function it is verified whether the vertex can be coloured with that chosen colour or not. If it returns true then it can be safely coloured so and again the same function is called repeatedly for the next vertices till it returns false [19]. If

the function returns false then that vertex cannot be coloured with that colour, so program will skip to the next iteration, that is to the next colour. Actually in the function it is checked whether u is in the list given and if so it will check whether the colour given to u matches with the colour at that vertex then it will return true, otherwise it will return false. Also it checks whether the neighbours of u are already given the colour we are trying to assign to it, if so then also the function will return the value as false. In this way each index position is given a colour/number and that will ultimately becomes the solution of the Sudoku. This set of program is saved as a file named **solve_sudoku.py**

Python code of solve_sudoku.py(program 3):

```

from connections import Sudoku
class sudoku_soln:
    def __init__(self):

        self.puzzle = self.getpuzzle()

        self.sudokugraph = Sudoku()
        self.mappedgrid = self.getmatrix()

    def getmatrix(self):
        matrix = [[0 for cols in range(4)]
                  for rows in range(4)]

        count = 1
        for rows in range(4):
            for cols in range(4):
                matrix[rows][cols] = count
                count += 1
        return matrix

    def getpuzzle(self):

        puzzle = [

```



```

        [0, 0, 0, 4],
        [4, 0, 1, 0],
        [0, 1, 0, 0],
        [0, 0, 3, 0]
    ]
    return puzzle

def printpuzzle(self):
    print("  1 2    3 4 ")
    for i in range(len(self.puzzle)):
        if i % 2 == 0:
            print("  - - - - ")

        for j in range(len(self.puzzle[i])):
            if j % 2 == 0:
                print(" | ", end="")
            if j == 3:
                print(self.puzzle[i][j], " | ", i + 1)
            else:
                print(f"{self.puzzle[i][j]} ", end="")
    print("  - - - - ")

def is_blank(self):
    for row in range(len(self.puzzle)):
        for col in range(len(self.puzzle[row])):
            if self.puzzle[row][col] == 0:
                return (row, col)
    return None

def is_valid(self, num, pos):
    for col in range(len(self.puzzle[0])):
        if self.puzzle[pos[0]][col] == num and\

```

```
        pos[0] != col:
            return False

    for row in range(len(self.puzzle)):
        if self.puzzle[row][pos[1]] == num and \
            pos[1] != row:
            return False

    x = pos[1] // 2
    y = pos[0] // 2

    for row in range(y * 2, y * 2 + 2):
        for col in range(x * 2, x * 2 + 2):
            if self.puzzle[row][col] == num and (
                row, col) != pos:
                return False

    return True

def solve(self):

    blank = self.is_blank()

    if blank is None:
        return True
    else:
        row, col = blank
    for i in range(1, 5):
        if self.is_valid(i, (row, col)):
            self.puzzle[row][col] = i

        if self.solve():
            return True
```

```

        self.puzzle[row][col] = 0
    return False

def graphcoloring(self):

    color = [0] * (self.sudokugraph.graph.
                  totalvertices + 1)
    partial_soln = []
    for row in range(len(self.puzzle)):
        for col in range(len(self.puzzle[row])):
            if self.puzzle[row][col] != 0:
                idx = self.mappedgrid[row][col]

                color[idx] = self.puzzle[row][col]
                partial_soln.append(idx)
    return color, partial_soln

def color_graph(self, m=4):
    color, partial_soln = self.graphcoloring()
    if self.graphcolor(m=m, color=color,
                      u=1, partial_soln=partial_soln) is None:
        print(":(")
        return False
    count = 1
    for row in range(4):
        for col in range(4):
            self.puzzle[row][col] = color[count]
            count += 1
    return color

def graphcolor(self, m, color, u, partial_soln):
    if u == self.sudokugraph.graph.totalvertices + 1:
        return True

```

```

    for c in range(1, m + 1):
        if self.iscolorable(u, color, c, partial_soln):
            color[u] = c
            if self.graphcolor(
                m, color, u + 1, partial_soln):
                return True
        if u not in partial_soln:
            color[u] = 0

def iscolorable(self, u, color, c, partial_soln):
    if u in partial_soln and color[u] == c:
        return True
    elif u in partial_soln:
        return False

    for i in range(1, self.sudokugraph.graph.
                    totalvertices + 1):
        if color[i] == c and self.sudokugraph.\
            graph.isneighbour(u, i):
            return False
    return True

def test():
    s = sudoku_soln()
    print("BEFORE SOLVING ...")
    print("\n\n")
    s.printpuzzle()
    print("\nSolving ...")
    print("\n\n\nAFTER SOLVING ...")
    print("\n\n")
    s.color_graph(m=4)
    s.printpuzzle()

```

test()

Output:

```

C:\Users\sony\Desktop\python>python solve_sudoku.py
Vertices : dict_keys([0, 1, 2, 3, 4, 5])
0 --> 1
0 --> 5
0 --> 4
1 --> 0
1 --> 2
2 --> 1
2 --> 3
2 --> 5
3 --> 2
3 --> 4
3 --> 5
4 --> 3
4 --> 0
4 --> 5
5 --> 0
5 --> 3
5 --> 4
5 --> 2
All vertex ids :
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16])
1 Connected to-> dict_keys([2, 3, 4, 5, 9, 13, 6])
2 Connected to-> dict_keys([1, 3, 4, 6, 10, 14, 5])
3 Connected to-> dict_keys([1, 2, 4, 7, 11, 15, 8])
4 Connected to-> dict_keys([1, 2, 3, 8, 12, 16, 7])
5 Connected to-> dict_keys([1, 2, 6, 7, 8, 9, 13])
6 Connected to-> dict_keys([1, 2, 5, 7, 8, 10, 14])
7 Connected to-> dict_keys([3, 4, 5, 6, 8, 11, 15])
8 Connected to-> dict_keys([3, 4, 5, 6, 7, 12, 16])
9 Connected to-> dict_keys([1, 5, 10, 11, 12, 13, 14])
10 Connected to-> dict_keys([2, 6, 9, 11, 12, 14, 13])
11 Connected to-> dict_keys([3, 7, 9, 10, 12, 15, 16])
12 Connected to-> dict_keys([4, 8, 9, 10, 11, 16, 15])
13 Connected to-> dict_keys([1, 5, 9, 10, 14, 15, 16])
14 Connected to-> dict_keys([2, 6, 9, 10, 13, 15, 16])
15 Connected to-> dict_keys([3, 7, 11, 12, 13, 14, 16])
16 Connected to-> dict_keys([4, 8, 11, 12, 13, 14, 15])
BEFORE SOLVING ...

  1 2   3 4
- - - - -
| 0 0 | 0 4 | 1
| 4 0 | 1 0 | 2
- - - - -
| 0 1 | 0 0 | 3
| 0 0 | 3 0 | 4
- - - - -

Solving ...

AFTER SOLVING ...

  1 2   3 4
- - - - -
| 1 3 | 2 4 | 1
| 4 2 | 1 3 | 2
- - - - -
| 3 1 | 4 2 | 3
| 2 4 | 3 1 | 4
- - - - -

C:\Users\sony\Desktop\python>_

```

4.3 Frequency Assignment

Radio waves are the ones in the electromagnetic spectrum with longest wavelength and lowest frequencies. The rate of oscillation of electromagnetic radio waves are ranging from 3 kHz to 300 GHz and these currents carrying radio signals is referred as Radio Frequency [20]. Mainly radio frequencies are used for communication and broadcasting purposes, like its use in Television, Mobile phones, Radio, etc. In this context, it is referred to as frequency band. Radio wave consists of different frequencies. Correspondingly frequency band is divided into different bands like VHF (Very High Frequency), FM (Frequency Modulation) Radio, SW (Short Wave) Radio, etc. Here each band is allocated for a certain purpose. In this way every wireless technology works in the range of a certain frequency band.

Likewise FM radio stations are also allocated by a particular frequency band between 88 MHz and 108 MHz. In this different FM stations operates at different frequencies. Then as we tune the radio to a particular frequency, we can listen to stations operating at that particular frequency. But we may have noticed when we travel from one place to another without even changing the frequency other stations get played in the radio. This is because there are more than one radio station operating at the same frequency. So if all those radio stations with same frequencies or even adjacent frequencies are operating in the same geographical area, the frequencies will interfere to each other and it would create unnecessary noise. This will leads to quality loss and creates disturbance to the user. So we have to allocate radio frequencies to the radio towers in such a way that there is no frequency interference. This is a general problem in assignment of frequencies in the case of Mobile Phone, FM stations, Television broadcasting, etc. And this is known as frequency assignment problem. In this paper we will be discussing about the frequency assignment problem in the case of FM stations.

In an area there will be a number of towers for the purpose of radio broadcasting. Each tower can be used to transmit more than one frequency. But care must be taken while allocating frequencies to the towers. Because if same frequency or even adjacent frequencies are allocated to the same tower or to towers which are near to each other it will cause overlapping of frequencies as said earlier. In order to avoid this we could make use of vertex colouring. Also the radio frequencies

available are limited, therefore there should be more than one station with the same frequency, but care should be taken about the distance between the transmitters with same frequency. In this regard too vertex colouring will be helpful to ideally allocate frequencies [21].

So frequency allocation can be considered as a vertex colouring problem. Here first a graph will be constructed with towers as vertices and two towers are connected by an edge if the areas at which they are constructed overlap with each other. Then we will try to properly colour this graph. Here different colours represent the different frequencies. So nearby towers will be coloured with different colours. Thus, frequency allocation can be done using vertex colouring.

4.4 Register Allocation

A compiler is used to translate a source code to machine code. A compiler does this with the help of an intermediate code [22]. Intermediate code uses a lot of temporary variables. Since these variables are used frequently, they are placed in registers. Because variables stored in registers can be easily accessed by the computer comparing to other memory locations. So storing the temporaries in registers will increase efficiency as it would reduce the program execution time. But for a computer the number of registers available is limited. But one should remember that not all temporaries are used at the same time, so they can be placed in the same register. But variables which are used at the same time should not be stored in the same register. So if there are more variables than the registers available then some will be moved to RAM and will be brought back to register as and when required [23]. This process is known as Register spilling. Accessing RAM is time consuming than registers. So there is a need to reduce spilling.

For that as much as possible variables should be allocated to all the registers available. But care should be taken not to allocate the variables used at same run time to the same register. Register allocation is the process by which the temporary variables are assigned to the available registers without allocating variables which are used simultaneously to the same register. It determines which variable at what time of program execution should be allocated to suitable registers.

Register allocation can be considered as a graph colouring problem. Even though John Cocke have studied about this relationship in 1971, the breakthrough

occurred when G. J Chaitin along with his colleagues in 1981 implemented it in a compiler [24]. Graph colouring is one of the most prominent method to do register allocation. To do register allocation with the help of vertex colouring first an undirected graph should be created with vertices representing each temporary variables used. Edges are added between vertices if they are simultaneously used in any part of the code. So it creates a register interference graph. Here the vertices are coloured such that no two adjacent vertices have the same colour. Each colour constitutes each registers. So in register allocation problem our aim is to minimize the number of colours used and it should not exceed the number of colours given. This problem can be considered as a k -colouring too, i.e. to properly colour the interference graph using k given colours.

Chapter 5

Conclusion

Throughout this paper we discussed about vertex colouring and its applications. From this study it is clear that many complicated problems can be easily solved by vertex colouring. Problems that need minimum number of solutions and maximum result mainly utilize this method. Also through this project paper from the discussion about the application of vertex colouring in traffic light signals, frequency allocation, solving of Sudoku and register allocation we tried to show how vertex colouring is used in activities that are part of our life in one way or another. Vertex colouring has many other applications too. It can be used to schedule aircrafts, exams, classes etc. It is used in storage problems that come from the branch of Chemistry. In this way vertex colouring has a lot of applications in many disciplines. Also it is a branch of mathematics in which many researches are still taking place.

REFERENCES

- [1] Bharathi S N, “A Study on Graph Coloring”, International Journal of Scientific & Engineering Research Volume 8, Issue 5, May 2017
- [2] Narsingh Deo, “Graph Theory with Applications to engineering and Computer Science(2’nd ed.)”, Prentice-Hall India Learning Private Limited 1979
- [3] Reinhard Diestel, “Graph theory”, Springer 2006
- [4] R. Balakrishnan, & K. Ranganathan, “A Textbook of Graph Theory ”Springer 2012
- [5] “Vertices, vertex names, and vertex indices,”retrieved from, <http://cseweb.ucsd.edu/~kubec/cls/100/Lectures/lec12.graphimpl/lec12-12.html>
- [6] “The Four Colour Theorem”, retrieved from, https://mathshistory.st-andrews.ac.uk/HistTopics/The_four_colour_theorem/
- [7] “Graph coloring”, retrieved from, https://en.m.wikipedia.org/wiki/Graph_coloring
- [8] “Graph coloring algorithm using backtracking”, retrieved from, <https://www.interviewbit.com/tutorial/graph-coloring-algorithm-using-backtracking/>
- [9] Robin J. Wilson, “Introduction to graph theory”, Prentice Hall 1986
- [10] Gary Chartrand, Ping Zhang, “Intoduction to graph theory ”, Tata McGraw-Hill Education 2006
- [11] “Graph Coloring”,retrieved from,https://www.whitman.edu/mathematics/cgt_online/book/section05.08.html
- [12] J.A. Bondy, U.S.R. Murthy, “Graph Theory with Applications”, Springer 2008
- [13] Tero Harju, “Lecture Notes on Graph Theory”, retrieved from, <http://www.cs.bme.hu/~sali/fcs/graphtheory.pdf>
- [14] B.R. Srinivas, A. Sri Krishna Chaitanya, “The chromatic polynomial and its algebraic properties”, International Journal of Scientific and Innovative Mathematical Research vol 2 Issue 11 November 2014

- [15] Julie Zhang, “An Introduction to Chromatic Polynomials”, retrieved from, https://klein.mit.edu/~apost/courses/18.204_2018/Julie_Zhang_paper.pdf, 2018
- [16] Amanda Aydelotte, “An exploration of the chromatic polynomial”, Boise State University, Mathematics undergraduate thesis, 2017
- [17] “Sudoku”, retrieved from, <https://en.wikipedia.org/wiki/Sudoku>
- [18] “A Sudoku solver using Graph coloring”, retrieved from, <https://www.codeproject.com/Articles/801268/A-Sudoku-Solver-using-Graph-Coloring>
- [19] Ishaan Gupta, “Sudoku Solver-Graph Coloring”, retrieved from, <https://medium.com/code-science/sudoku-solver-graph-coloring-8f1b4df47072>
- [20] “Radio Frequency(RF)”, retrieved from, <https://www.techopedia.com/definition/5083/radio-frequency-rf>
- [21] “What are radio waves?”, retrieved from, <https://www.livescience.com/50399-radio-waves.html>
- [22] A.K.Bincy, B.Jeba Presitha, “Graph Coloring and its Real Time Applications an Overview”, International Journal of Mathematics And Its Applications 2017, retrieved from, <http://ijmaa.in/v5n4-f/845-849.pdf>
- [23] “Register allocation”, retrieved from, https://en.wikipedia.org/wiki/Register_allocation
- [24] “Register allocation by graph coloring”, retrieved from, <https://www.lighterra.com/papers/graphcoloring/>