**GRAPH THEORY PROJECT FINAL PROGRESS**

**EXAM SCHEDULER**

**(Group-3)**

*Abstract*— Scheduling exams is one of the tedious tasks in the academic institutions. Due to the huge number of students and variety of courses available in the institution, designing a conflict-free exam schedule by satisfying all the requirements is difficult. By using graph coloring approach, we can obtain the desired result. Although vertex coloring can be used to serve the purpose, we are using Welsh Powell algorithm as it promises an effective way to obtain the solution.

***Keywords***— **Graph coloring, Timetable, Scheduling, Course matrix**

Introduction:

Exam schedule is required for every academic institution. In order to conduct the exams, all the universities and colleges must create an exam schedule for each semester. Creating an exam schedule with no conflicts is an important task for the administration team of educational institution.

To prepare a perfect exam scheduler, we take list of students and courses offered by the educational institution as the input. Our aim is to create a conflict-free exam schedule where no student has two exams in the same time slot. Generating this exam scheduler can be a tedious task if it is to be performed manually. It takes lot of time and might result in designing an inefficient timetable at times.

In our project, we are building an exam scheduler which takes the details of students and courses being offered in the educational institution as the input and generate a conflict-free exam schedule as an output.

# **PROBLEM FORMULATION**

A solution for this problem statement can be designed by using the concept of graph coloring.

## Graph Coloring

It is a technique where certain elements of graph are assigned colors based on some constraints. Here we use vertex coloring which is one of the most common graph coloring methods. There are few techniques in this graph coloring concept like vertex coloring, edge coloring and face coloring. In the edge coloring, no vertex is incident to two edges of same color. In the case of face coloring, we color a planar graph and same color is not assigned to any vertices that have a common boundary. We are using the vertex coloring technique to build this project.

## Vertex Coloring

Vertex coloring is a technique where we consider some ‘x’ colors and try to color the vertices present in the graph by following a constraint which says that same color is not assigned to any of the adjacent vertices in the graph. Vertex coloring is an NP complete problem. If we use the general vertex coloring technique, the maximum number of colors used for the vertex coloring technique never exceeds the d+1 colors, here d is the highest degree of the vertex present in the given graph.

## Np Complete Problem

Np Complete Problems are the hardest problems in the NP set. We can give a solution to the NP complete problem, but we cannot determine an efficient solution to the problem.

# **IMPLEMENTATION OF VERTEX COLORING TECHNIQUE**

1. Color the first vertex of the graph with any desired color (No constraints are followed for coloring the first vertex).
2. Consider the course matrix and now try to assign different colors for the vertices which are not adjacent to each other.
3. If you find non-adjacent vertices (vertex which is not adjacent to the previous colored vertex), you can color them with the same color used for the previous vertex, else we will have to choose another color for vertex coloring.

## Code for vertex coloring :

class GraphColoring:

def \_\_init\_\_(self, e, n):

self.adjList = [[] for \_ in range(n)]

for (source, destination) in e:

self.adjList[source].append(destination)

self.adjList[destination].append(source)

def coloringGraph(graph, n):

res = {}

for x in range(n):

assigned = set([res.get(i) for i in graph.adjList[x] if i in res])

color = 1

for clr in assigned:

if color != clr:

break

color = color + 1

res[x] = color

for v in range(n):

print(f'Color assigned to vertex {v} is {colors[res[v]]}')

if \_\_name\_\_ == '\_\_main\_\_':

colors = ['' 'RED','BLACK','YELLOW', 'BROWN', 'WHITE','BLUE','PURPLE','VOILET', 'GREEN', 'ORANGE', 'PINK']

e = [(0, 2), (0, 4), (0, 5), (2, 4),(2, 5), (3, 4), (1, 5)]

n = 6

graph = GraphColoring(e, n)

coloringGraph(graph, n)

## Explanation of the vertex coloring code:

The class **GraphColoring** in the above code represents an undirected graph which is using an adjacency list to perform the vertex coloring.

* N = Number of vertices
* adjList[] = It is the adjacency list used in order to note down the adjacent vertices.
* GraphColoring()= It is a class used in the program to perform graph coloring.
* coloringGraph()= It is a function which assigns colors to the vertices in the graph.

## Code Flow:

* We used class Graph Coloring to represent an undirected graph.
* Declared number of vertices as N
* result variable is used to track the colors given to the vertices of the graph.
* Created an adjacency list adjList[] to list the adjacent vertices.
* At first all the vertices remain unassigned, and first color is assigned to first vertex.
* All the available colors are stored in colors array. When we assign a color to the vertex, we mark that the color is unavailable to its adjacent vertices. Select the next available color for the adjacent vertices. Continue this process until we color all the vertices of the graph.
* In order to formulate the graph, we give the various colors which are planning to use for vertex coloring in the colors array.
* The edges of the graph can be given in the array called edges. The edges between the vertices of the graph are drawn according to the edges given in this array.
* Then the graph is constructed by considering the n, which is the number of vertices given by us and the edges list given in the array called edges.
* The graph coloring is performed by calling the coloringGraph() function.

## Challenge/Risk while using the vertex coloring algorithm

vertex coloring is used to label each individual vertex hence any of the adjacent vertices will not have the same color. The important thing to consider is the number of colors we require to fulfil the given condition. Having too many colors is not a good condition.

## Creative solution for the above challenege

This is where the Welsh Powell Graph coloring algorithm comes into place for finding the minimum number of colors required to fulfil the conditions. We will also be able to identify the chromatic number of a graph using this algorithm. This algorithm follows the greedy approach.

# **IMPLEMENTATION OF WELSH POWELL GRAPH COLORING ALGORITHM**

A graph will need K distinct colors for its proper coloring, this is called a chromatic graph. The number of colors used for coloring the graph is called the chromatic number of that graph.

The working of Welsh Powell algorithm:

1. To find the degree of vertex.

2. The vertices should be listed in order of descending.

3. Take the first vertex and color it with the first one.

4. Go through the list and all the vertices should be colored which are not connected to the colored vertex with the same color that was used before.

5. The previous step is repeated on the uncolored vertices with a new color until all the vertices are colored.

Example of the wells Powell algorithm:

Chart

Description automatically generatedChart, radar chart

Description automatically generated

1)Welsh Powell algorithm is a coloring algorithm

2)Here in the above image the vertices are A, B, C, D, E, F, G, H, I, J, K. So, we need to color the vertices accordingly. The adjacent vertices should not be in the same color.

3)From the above graph we required to write the vertices in the descending order according to the degree of the vertices.

4.The descending order is: H, K, D, G, I, J, A, B, E, F, C

5.1st:I colored h with red. So we should look at the order and color of the vertices. I cannot color k because it is adjacent. And next we can color D because it is not adjacent. And next we cannot color G, I,J,A,B because they are adjacent to H and D. Next, we can color E.

6.Next we are taking color green, We should check the vertices which are not adjacent and should color it. According to the above step I analyzed and the green color vertices are A,I,K,F,C.

7.Last the left out vertices are G,J,B.I colored G with blue. So J and B are not adjacent. So I colored J and B with blue.

6.We can see the above example, We have used the three colors in the graph. So, The chromatic number of this graph is 3.

7.Not only these we can use the different types of colors and shade it on the vertex.

This is about the wells Powell algorithm.

# **EVALUATION OF THE SOLUTION USING WELSH POWELL ALGORITHM**

## Why welsh powell algorithm better than vertex coloring algorithm?

Vertex coloring in graph theory refers to the process of assigning a unique color to each vertex so that no two neighboring vertices share the same color. However, we must determine how many colors are necessary to meet the stated requirement. Having a wide range of colors or labels is not ideal. We therefore have an algorithm known as the Welsh Powell algorithm that provides the minimal colors we want. Finding a graph's chromatic number is another purpose for this approach. This strategy is greedy and iterative.

## Pseudo code for the Welsh Powell algorithm

* Find out each vertex's degree.
* Firstly, we need to list the vertices with decreasing valence, so that valence degree(A(i)) >= degree(A(i+1)).
* Next, we should draw a color on each vertex and should check the vertices and color it accordingly.
* Color each vertex in the sorted list that isn't connected to the colored vertices above the same color as you go down the list, then cross off every vertex that is colored.
* Use a fresh color and carry out the same procedure on the uncolored vertices, always working in descending order of degree to color all of the vertices.

## Code for the Welsh Powell algorithm

import java.nio.file.Path;

import java.util.Map;

import java.util.Collections;

import java.util.ArrayList;

import java.nio.charset.Charset;

import java.nio.file.FileSystems;

import java.nio.file.Files;

import java.util.Comparator;

import java.util.HashMap;

import java.util.List;

import java.io.IOException;

public class Graph {

private List<Vertex> vertices;

public Graph(String \_path){

vertices = new ArrayList<Vertex>();

ArrayList<String> lines = (ArrayList<String>) readGraphData(\_path);

if (lines != null){

for (int k = 0; i < lines.size(); k++){

String node = lines.get(k).split(":")[0];

String[] adj = lines.get(k).split(":")[1].split(" ");

List<String> neighbors = new ArrayList<String>();

for (int l = 0; l < adj.length; l++){

neighbors.add(adj[l]);

}

vertices.add(new Vertex(node, new ArrayList<String>(neighbors)));

}

}

}

public String toString() {

String result = "";

for (Vertex v: vertices){

result += v.node + ":" + v.neighbors.toString() + "\n";

}

return result;

}

public void colourVertices(){

Collections.sort(vertices, new VertexComparator());

Map<String, String> vertex\_color\_index = new HashMap<String, String>();

for (int i = 0; i < vertices.size(); i++){

if ((vertex\_color\_index.containsKey(vertices.get(i).node))){

continue;

}

else{

vertex\_color\_index.put(vertices.get(i).node, "Colour " + i);

for (int j = i+1; j < vertices.size(); j++){

if (!(vertices.get(i).neighbors.contains(vertices.get(j).node)) && !(vertex\_color\_index.containsKey(vertices.get(j).node))){

vertex\_color\_index.put(vertices.get(j).node, "Colour " + i);

}

else{

continue;

}

}

}

}

System.out.println(vertex\_color\_index);

}

private List<String> readGraphData(String \_path){

Path path = FileSystems.getDefault().getPath(\_path, "");

try {

return Files.readAllLines(path, Charset.defaultCharset());

} catch (IOException e) {

System.err.println("I/O Error");

return null;

}

}

class VertexComparator implements Comparator<Vertex>{

public int compare(Vertex a, Vertex b) {

return a.neighbors.size() < b.neighbors.size() ? 1 : a.neighbors.size() == b.neighbors.size() ? 0: -1;

}

}

public static void main(String[] args){

Graph graph = new Graph("data1.txt");

graph.colourVertices();

}

}

# **PROJECT EXECUTION**

We have python as a programming language in order to build our exam scheduler project.

In order to run this project we have imported few packages of the python: Pandas,numpy,networkx,itertools,matplotlib.pyplot.

## Python code used for the project implementation:

**1)Importing the required packages:**

**import** numpy **as** np

**import** itertools

**import** matplotlib.pyplot **as** plt

**import** networkx **as** nx

**import** pandas **as** pd

**2) creating a csv file in order to give student and course details as inputs**

**%%file** coursesheet.csv

Rno,Department,Year,Course:1,Course:2,Course:3,Course:4,Course:5,Course:6

1,CS,1,GT,CA,DS,BEE,DE,NaN

2,CS,1,GT,CA,DS,BEE,DE,NaN

3,CS,1,GT,CA,DS,BEE,DE,NaN

4,CS,1,GT,CA,DS,BEE,DE,NaN

5,CS,1,GT,CA,DS,BEE,DE,NaN

6,CS,2,CD,FDB,CAA,CAE,NaN,NaN

7,CS,2,CD,CA,CAA,CAE,DE,NaN

8,CS,2,DE,CA,CAA,CAE,CD,FDB

9,CS,2,CD,CA,CAA,CAE,ITE,NaN

10,CS,2,CD,SME,CAA,CAE,FDB,NaN

1,IT,1,GT,CA,DS,BEE,NaN,NaN

2,IT,1,GT,CA,DS,BEE,NaN,NaN

3,IT,1,GT,CA,DS,BEE,NaN,NaN

4,IT,1,GT,CA,DS,BEE,NaN,NaN

5,IT,1,GT,CA,DS,BEE,NaN,NaN

6,IT,2,SME,ITG,ITF,ITU,ITH,NaN

7,IT,2,ITG,DS,ITF,ITH,GT,ITU

8,IT,2,ITF,ITG,ITU,ITH,NaN,NaN

9,IT,2,ITG,ITF,ITU,SME,ITH,NaN

10,IT,2,ITF,ITU,ITH,NaN,ITG,GT

1,EEE,1,GT,CA,DS,BEE,NaN,NaN

2,EEE,1,GT,CA,DS,BEE,NaN,NaN

3,EEE,1,GT,CA,DS,BEE,NaN,NaN

4,EEE,1,GT,CA,DS,BEE,NaN,NaN

5,EEE,1,GT,CA,DS,BEE,NaN,NaN

6,EEE,2,EER,EET,DS,EEJ,ITE,NaN

7,EEE,2,EER,CA,DS,EEJ,ITE,EET

8,EEE,2,EER,CA,EEJ,SME,ITE,EET

9,EEE,2,EER,SME,EEJ,ITE,EET,NaN

10,EEE,2,EER,CA,EEJ,ITE,EET,NaN

**3) Creating a dataframe to display the course details**

df**=**pd**.**read\_csv("coursesheet.csv")

df

# **4)** **Reading the input cvs file and listing the courses and years:**

Data=pd**.**read\_csv("coursesheet.csv", delimiter **=** ',')

df **=** pd**.**DataFrame(data)

**5)Displaying the departments in the data:**

Dept = df['Department'].unique()

DeptList= len(Dept)

print(f"Department:{Dept}")

**6)Displaying the Years in the data:**

year = df['Year'].unique()

yearlist= len(year)

print(f"\nYear:{year}")

**7)We created a dictionary called courses and intially it is empty variable, var is created and its value is given as 0**

courses={}

var=0

**8)We are using the keys and value pairs in the dictionary to update the courses.**

for c1,c2,c3,c4,c5,c6,c in zip(df['Course:1'],df['Course:2'],df['Course:3'],df['Course:4'],df['Course:5'],df['Course:6'],df['Department']):

if c1 not in courses and c1 == c1:

courses.update({c1:var})

var=var+1

if c2 not in courses and c2 == c2:

courses.update({c2:var})

var=var+1

if c3 not in courses and c3 == c3:

courses.update({c3:var})

var=var+1

if c4 not in courses and c4 == c4:

courses.update({c4:var})

var=var+1

if c5 not in courses and c5 == c5:

courses.update({c5:var})

var=var+1

if c6 not in courses and c6 == c6:

courses.update({c6:var})

var=var+1

print("\n Number of courses :",len(courses))

print("\n List of courses:",courses)

**9)Creating a matrix for the courses**

matrix = [[0 for i in range(len(courses))] for j in range(yearlist\*DeptList)]

i=0

tmp=Dept[0]

for c1,c2,c3,c4,c5,c6,year,course in zip(df['Course:1'],df['Course:2'],df['Course:3'],df['Course:4'],df['Course:5'],df['Course:6'],df['Year'],df['Department']):

if tmp==course:

year=year+i\*yearlist

else:

i=i+1

tmp=Dept[i]

year=year+i\*yearlist

if c1 == c1:

matrix[year-1][courses[c1]]=1

if c2 == c2:

matrix[year-1][courses[c2]]=1

if c3 == c3:

matrix[year-1][courses[c3]]=1

if c4 == c4:

matrix[year-1][courses[c4]]=1

if c5 == c5:

matrix[year-1][courses[c5]]=1

if c6 == c6:

matrix[year-1][courses[c6]]=1

SMatrix=pd.DataFrame(matrix, columns=courses.keys())

print("\n\n Year wise list of courses:")

SMatrix

**10)Reversing the dictionary:**

rvs = dict(zip(courses.values(),courses.keys()))

print(rvs)

**11)Displaying graph for each year separtely:**

Max=2\*Max

CoursesColor={}

DataSheet=[]

chromatic=[]

TempChrom=[]

CGraph = nx.Graph()

x1=1

for j1 in range(0,Max):

TempChrom.append(j1)

for i1 in range(0,DeptList\*yearlist):

Subject=[]

G = nx.Graph()

for j1 in range(0,len(courses)):

if matrix[i1][j1]==1:

Subject.append(rvs[j1])

DataSheet.append(Subject)

chromatic=TempChrom

for y1 in range(0,i1):

for z1 in range(0,len(courses)):

if matrix[y1][z1] == 1 and rvs[z1] in Subject and CoursesColor.get(rvs[z1]) in chromatic:

chromatic.remove(CoursesColor[rvs[z1]])

for y1 in range(i1+1,yearlist\*DeptList):

for z1 in range(0,len(courses)):

if matrix[y1][z1] == 1 and rvs[z1] in CoursesColor.keys() and CoursesColor.get(rvs[z1]) in chromatic:

chromatic.remove(CoursesColor[rvs[z1]])

index=0

for Subjectject in range(0,len(Subject)):

if Subject[Subjectject] not in CoursesColor.keys():

CoursesColor.update({Subject[Subjectject]:chromatic[index]})

index=index+1

print("Graph for year",i1+1,":")

print(Subject)

x1+=1

G.add\_nodes\_from(Subject)

G.add\_edges\_from(itertools.combinations(Subject, 2))

val = [CoursesColor.get(node,0.25) for node in Subject]

CGraph.add\_nodes\_from(Subject)

CGraph.add\_edges\_from(itertools.combinations(Subject, 2), weight =8)

nx.draw(G, node\_size=1600,cmap=plt.get\_cmap('plasma'), node\_color=val, with\_labels=True, font\_color='white')

plt.show()

**12)Displaying graph for all the courses:**

print("All courses Graph:")

val = [CoursesColor.get(node,0.35) for node in CGraph.nodes()]

nx.draw(CGraph, node\_size=1600, cmap=plt.get\_cmap('plasma'), node\_color=values, with\_labels=True, font\_color='white')

nx.draw

plt.show()

**13) Displaying colors assigned to each course:**

print("Color of the Course:")

for x,y in zip(CoursesColor.keys(),CoursesColor.values()):

print(x,"-",y)

**14)Displaying the final exam scheduler developed:**

temp=Max+2

data=[['']\*temp for i in range(DeptList\*yearlist)]

column=['Department','Year']

for i in range(0,DeptList\*yearlist):

for j in range(0,len(courses)):

if matrix[i][j] is 1:

data[i][2+CoursesColor[rvs[j]]]=str(rvs[j])

for i in range(1,Max+1):

day='Day'+str(i)

column.append(day)

finalschedule = pd.DataFrame(data, columns=column)

j=1

for i in range(0,DeptList\*yearlist):

if i < j\*yearlist:

finalschedule.at[i,'Department']=(df['Department'].unique()[j-1])

else:

j=j+1

finalschedule.at[i,'Department']=(df['Department'].unique()[j-1])

finalschedule.at[i,'Year']=(df['Year'].unique()[(i)%yearlist])

print("\nFinal exam schedule generated for the given data is:")

finalschedule

# **Results**

Calendar

Description automatically generated with medium confidence

Diagram

Description automatically generated

Diagram

Description automatically generated with medium confidence

Diagram

Description automatically generated with medium confidence

Chart, bubble chart

Description automatically generated

Diagram

Description automatically generated

Diagram, schematic

Description automatically generated

Calendar

Description automatically generated with low confidence

##### References

1. Handayani, S., Ely, R., Paramita., M R.A. (2016). Penerapan Algorithm Welch Powell Dengan Pewarnaan Graph Pada Penjadwalan Mata Pelajaran SMA. National Seminar System Information Indonesia, 333- 338.
2. Maryana. (2015). Pewarnaan Graf Pada Penyusunan Jadwal Perkuliahan di Politeknik AKAMIGAS Palembang. Jurnal Teknik PATRA Akademika, 30-33.