DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF ENGINEERING
DAYANANDA SAGAR UNIVERSITY
KUDLU GATE
BANGALORE - 560068

MINI PROJECT REPORT

ON

"Performance Analysis of Greedy, Backtracking and Welsh Powell algorithms"

SUBMITTED TO THE 4th SEMESTER DESIGN AND ANALYSIS OF ALGORITHMS LABORATORY-2020

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING

Submitted by

Siva Prakash Anupam – ENG18CS0277
Sujith T L– ENG18CS0290
Vaishaali Kondapalli– ENG18CS0305
Vaishakh Anil– ENG18CS0306
Vaishnavi A Punagin-ENG18CS0307

Under the supervision of
Prof. Kavyashree Dalawai

DAYANANDA SAGAR UNIVERSITY

School of Engineering, Kudlu Gate, Bangalore-560068

CERTIFICATE

This is to certify that Mr. Siva Prakash Aupam, Mr.Sujit T L, Ms. Vaishaali Kondapalli, Mr. Vaishakh Anil and Ms. Vaishnavi A Punagin bearing USN ENG18CS0277, ENG18CS0290, ENG18CS0305, ENG18CS0306 and ENG18CS0307 respectively have satisfactorily completed their Mini Project as prescribed by the University for the Fourth semester B.Tech. Programme in Computer Science & Engineering during the year 2020 at the School of Engineering, Dayananda Sagar University, Bangalore.

Date: 29/05/2020

Signature of the faculty in-charge

Max Marks	Marks Obtained

Signature of Chairman
Department of Computer Science & Engineering

DECLARATION

We hereby declare that the work presented in this mini project entitled "**Performance Analysis of Greedy, Backtracking and Welsh Powell algorithms**", has been carried out by us and it has not been submitted for the award of any degree, diploma or the mini project of any other college or university.

Siva Prakash Anupam – ENG18CS0277
Sujith T L– ENG18CS0290
Vaishaali Kondapalli– ENG18CS0305
Vaishakh Anil– ENG18CS0306
Vaishnavi A Punagin-ENG18CS0307

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of a task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts with success.

We are especially thankful to our **Chairman, Dr. M K Banga**, for providing us with necessary departmental facilities, moral support and encouragement.

We are very much thankful to **Prof.Kavyashree Dalawai**, for providing help and suggestions in completion of this mini project successfully.

We have received a great deal of guidance and co-operation from our friends and we wish to thank all that have directly or indirectly helped us in the successful completion of this project work.

Siva Prakash Anupam – ENG18CS0277
Sujith T L– ENG18CS0290
Vaishaali Kondapalli– ENG18CS0305
Vaishakh Anil– ENG18CS0306
Vaishnavi A Punagin-ENG18CS0307

TABLE OF CONTENTS

SL.NO	TOPIC	PAGE NO.
1	**INTRODUCTION**	1
2	**LITERATURE REVIEW** 2.1 HISTORY OF GRAPH COLORING 2.2 VERTEX COLORING	2
3	**PROBLEM STATEMENT**	5
4	**PROPOSED SOLUTION**	6
5	**SYSTEM DESIGN** 5.1 THE ANALYSIS FUNCTION 5.2 GREEDY ALGORITHM 5.3 BACKTRACKING ALGORITHM 5.4 WELSH POWELL ALGORITHM	7
6	**IMPLEMENTATION** 6.1 THE ANALYSIS FUNCTION 6.2 GENERATING A GRAPH 6.3 THE PICTORIAL REPRESENTATION 6.4 THE ALGORITHMS	11
7	**TESTING**	14
8	**RESULTS AND SNAPSHOTS**	20
9	**ADVANTAGES AND LIMITATIONS** 9.1 GREEDY ALGORITHM 9.2 BACKTRACKING ALGORITHM 9.3 WELSH POWELL ALGORITHM	37
10	**CONCLUSION AND FUTURE ENHANCEMENTS** 10.1 CONCLUSION 10.2 FUTURE ENHANCEMENTS	39
	REFERENCES	42

FIGURES:

FIG.NO	CONTENT	PAGE NO.
5.1	Flowchart- Analysis function	7
5.2	Flowchart- Greedy Algorithm	8
5.3	Flowchart- Backtracking Algorithm	9
5.4	Flowchart- Welsh Powell Algorithm	10
7.1	Test case 1	14
7.2	Test case 2	15
7.3	Test case 3	15
7.4	Test case 4	16
7.5	Test case 5	16
7.6	Test case 6	17
7.7	Test case 7	17
7.8	Test case 8	18

7.9	**Test case 9**	**18**
7.10	**Test case 10**	**19**
8.1	**Output of Test case 1**	**20**
8.2	**Output of Test case 2**	**22-23**
8.3	**Output of Test case 3**	**24-25**
8.4	**Output of Test case 4**	**25-26**
8.5	**Output of Test case 5**	**27-28**
8.6	**Output of Test case 6**	**29-30**
8.7	**Output of Test case 7**	**31-32**
8.8	**Output of Test case 8**	**32-33**
8.9	**Output of Test case 9**	**34-35**
8.10	**Output of Test case 10**	**35-36**
10.1	**Output given by network x module for the input graph given**	**41**

TABLES:

TABLE NO.	CONTENT	PAGE NO.
8.1	Runtimes for test case 1	21
8.2	Runtimes for test case 2	23
8.3	Runtimes for test case 3	25
8.4	Runtimes for test case 4	26
8.5	Runtimes for test case 5	28
8.6	Runtimes for test case 6	30
8.7	Runtimes for test case 7	32
8.8	Runtimes for test case 8	33
8.9	Runtimes for test case 9	35
8.10	Runtimes for test case 10	36
10.1	Table depicting runtimes for ten cases, under three different algorithms	39

ABSTRACT

The objective of the Graph Colouring problem is to colour vertices of a graph in such a way that no two vertices that share an edge are assigned the same colour. It is used to identify independent objects in a set and has applications in a wide variety of scientific and engineering problems. Optimal colouring of graphs is an NP-complete problem.

Deterministic graph colouring algorithms of contraction and sequential type are investigated. Sequential algorithms can be extended by backtracking to relatively effective algorithms for the chromatic number of a graph. Incomplete backtracking leads to new heuristics for graph colouring.

In this project we've worked on getting the performance analysis of three different approximate algorithms which can be used to solve the graph colouring problem. Namely, the Backtracking, Greedy and Welsh Powell algorithm.

CHAPTER 1

INTRODUCTION

A graph colouring is an assignment of labels, called colours, to the vertices of a graph such that no two adjacent vertices share the same colour. The chromatic number \chi(G)χ(G) of a graph GG is the minimal number of colours for which such an assignment is possible. Other types of colourings on graphs also exist, most notably edge colourings that may be subject to various constraints.

VERTEX COLORING

Vertex coloring is an assignment of colours to the vertices of a graph ‘G’ such that no two adjacent vertices have the same colour. Simply put, no two vertices of an edge should be of the same colour.

CHROMATIC NUMBER

The minimum number of colors required for vertex coloring of graph ‘G’ is called the chromatic number of G, denoted by X(G).

χ(G) = 1 if and only if 'G' is a null graph. If 'G' is not a null graph, then χ(G) ≥ 2

There are various real life applications of graph coloring, which include allocating radio frequencies in cellular networks, exam scheduling, air traffic scheduling and register allocation. These systems can be modelled as graphs, where we have a set of limited number of resources (colors) assigned to a set of variables (nodes) under certain incompatibility constraints (edges).

CHAPTER 2

LITERATURE REVIEW

In graph theory, graph colouring is a special case of graph labelling; it is an assignment of labels traditionally called “colours” to elements of a graph subject to certain constraints. In its simplest form, it is a way of colouring the vertices of a graph such that no two adjacent vertices share the same colour; this is called a vertex colouring. Similarly, an edge colouring assigns a colour to each edge so that no two adjacent edges share the same colour, and a face colouring of a planar graph assigns a colour to each face or region so that no two faces that share a boundary have the same colour. Vertex colouring is the starting point of the subject, and other colouring problems can be transformed into a vertex version. For example, an edge colouring of a graph is just a vertex colouring of its line graph, and a face colouring of a plane graph is just a vertex colouring of its dual. However, non-vertex colouring problems are often stated and studied as is. That is partly for perspective, and partly because some problems are best studied in non-vertex form, as for instance edge colouring.

The convention of using colours originates from colouring the countries of a map, where each face is literally coloured. This was generalized to colouring the faces of a graph embedded in the plane. By planar duality it became colouring the vertices, and in this form it generalizes to all graphs. In mathematical and computer representations, it is typical to use the first few positive or nonnegative integers as the “colours”. In general, one can use any finite set as the “colour set”. The nature of the colouring problem depends on the number of colours but not on what they are.

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.

2.1 HISTORY OF GRAPH COLORINGS

The first results about graph colouring deal almost exclusively with the planar graphs in the form of the colouring of maps. While trying to colour a map of the counties of England, Francis Guthrie postulated the four-colour conjecture, noting that four colours were sufficient to colour the map so that no regions sharing a common border received the same colour. Guthrie's brother

passed on the question to his mathematics teacher Augustus de Morgan at University College, who mentioned it in a letter to William Hamilton in 1852. Arthur Cayley raised the problem at a meeting of the London Mathematical Society in 1879. The same year, Alfred Kempes published a paper that claimed to establish the result, and for a decade The four colour problem was considered solved. For his accomplishment Kempes was elected a Fellow of the Royal Society and later President of the London Mathematical Society.

In 1890, Heawood pointed out that Kempes argument was wrong. However, in that paper he proved the five-colour theorem, saying that every planar map can be coloured with no more than five colours, using ideas of Kempe. In the following century, a vast amount of work and theories were developed to reduce the number of colours to four, until the four-colour theorem was finally proved in 1976 by Kenneth Appel and Wolfgang Haken. Perhaps surprisingly, the proof went back to the ideas of Heawood and Kempe and largely disregarded the intervening developments. The proof of the four colour theorem is also noteworthy for being the first major computer aided proof.

In 1912, George David Birkhoff introduced the chromatic polynomial to study the colouring problems, which was generalised to the Tutte polynomial by Tutte, important structures in algebraic graph theory. Kempe had already drawn attention to the general, non-planar case in 1879 and many results on generalisations of planar graph colouring to surfaces of higher order followed in the early 20th century.

In 1960, Claude Berge formulated another conjecture about graph colouring, the strong perfect graph conjecture, originally motivated by an information-theoretic concept called the zero-error capacity of a graph introduced by Shannon. The conjecture remained unresolved for 40 years, until it was established as the celebrated strong perfect graph theorem in 2002 by Chudnovsky, Robertson, Seymour, Thomas.

Graph colouring has been studied as an algorithmic problem since the early 1970s: the chromatic number problem is one of Karp's 21 NP-complete problems from 1972, and at approximately the same time various exponential-time algorithms were developed based on backtracking and on the deletion-contraction recurrence of Zykov(1949). One of the major applications of graph colouring, register allocation in compilers, was introduced in 1981.

2.2 VERTEX COLORING

When used without any qualification, a colouring of a graph is almost always a proper vertex colouring, namely a labelling of the graphs vertices with colours such that no two vertices sharing the same edge have the same colour. Since a vertex with a loop could never be properly coloured, it is understood that graphs in this context are loop less. The terminology of using colours for vertex labels goes back to map colouring. Labels like red and blue are only used when the number of colours is small, and normally it is understood that the labels are drawn from the integers $\{1, 2, 3, \cdots\}$.

A colouring that uses at most k colours is called a (proper) k-colouring. The smallest number of colours needed to colour a graph G is called its chromatic number, and is often denoted $\chi(G)$. A graph that admits a (proper) k-colouring is k-colourable, and it is k-chromatic if its chromatic number is exactly k. A subset of vertices assigned to the same colour is called a colour class, every such class forms an independent set. Thus, a k-colouring is the same as a partition of the vertex set into k independent sets, and the terms k-partite and k-colourable have the same meaning.

CHAPTER 3

PROBLEM STATEMENT

Graph colouring is an assignment of labels, called colours, to the vertices of a graph such that no two adjacent vertices share the same colour, thereby, known as vertex colouring.

The study of graph colourings has historically been linked closely to that of planar graphs and the four colour theorem [The four-color theorem states that any map in a plane can be coloured using four-colours in such a way that regions sharing a common boundary (other than a single point) do not share the same colour].

Through this project we are going to implement the graph coloring problem using the Greedy algorithm, The Backtracking algorithm and The Welsh Powell algorithm to determine which of these provides us with the best performance by comparing the efficiencies based on runtime.

CHAPTER 4

PROPOSED SOLUTION

Step 1:

We first implemented the graph coloring problem using all 3 algorithms(greedy, backtracking and welsh powell).

Step 2:

Copy the runtimes of the three algorithms, save the runtimes alongside the algorithms on an excel sheet. Save it as a .csv file.

Step 3:

Feed this saved .csv file to seaborn to get the performance analysis of the three algorithms on a graph.

The code we've written for getting the analysis using seaborn is as follows:

```
import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt



df=pd.read_csv('filename.csv')

sns.set(style='darkgrid')

ax = sns.catplot(x="alg", y="runtime",data =
df,kind='bar/box')

ax.set(ylim=(1,None))

plt.xticks(rotation=-15)

plt.show()
```

Here, we've imported numpy, pandas and matplotlib and Seaborn is dependent on all of them. 'df' is the data frame and we've used pandas to read the .csv file and on the x - axis of the graph we have th

CHAPTER 5

SYSTEM DESIGN

Basic system overview :

5.1 THE ANALYSIS FUNCTION:

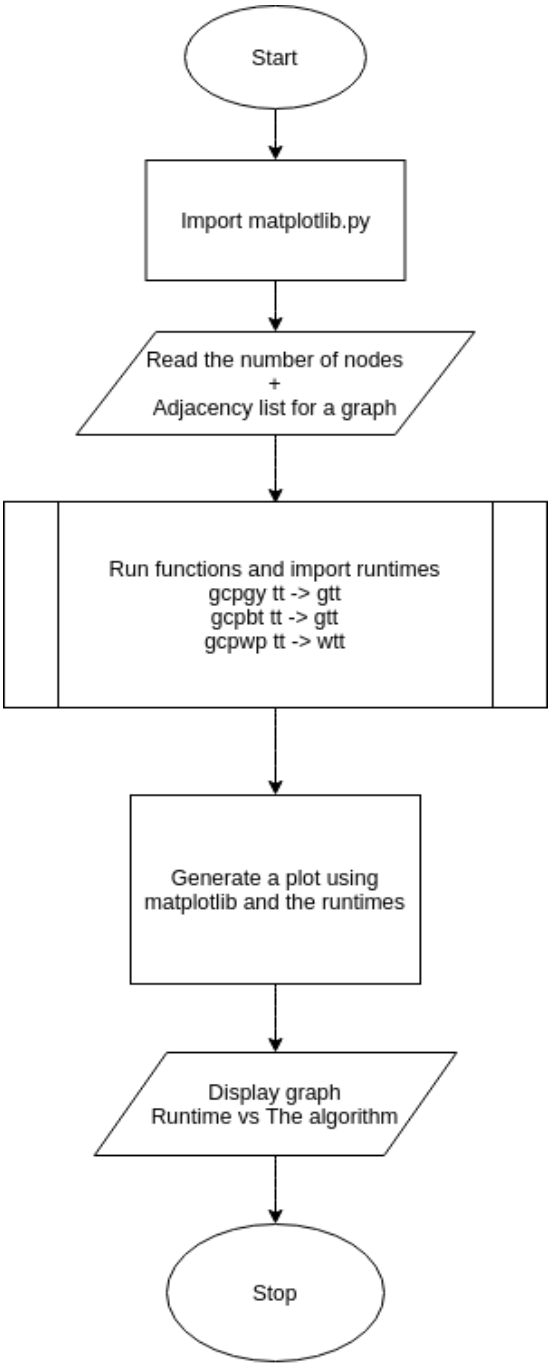


Fig 5.1 Flowchart - Analysis function

5.2 GRAPH COLORING PROBLEM USING GREEDY ALGORITHM - gcpgy.py function

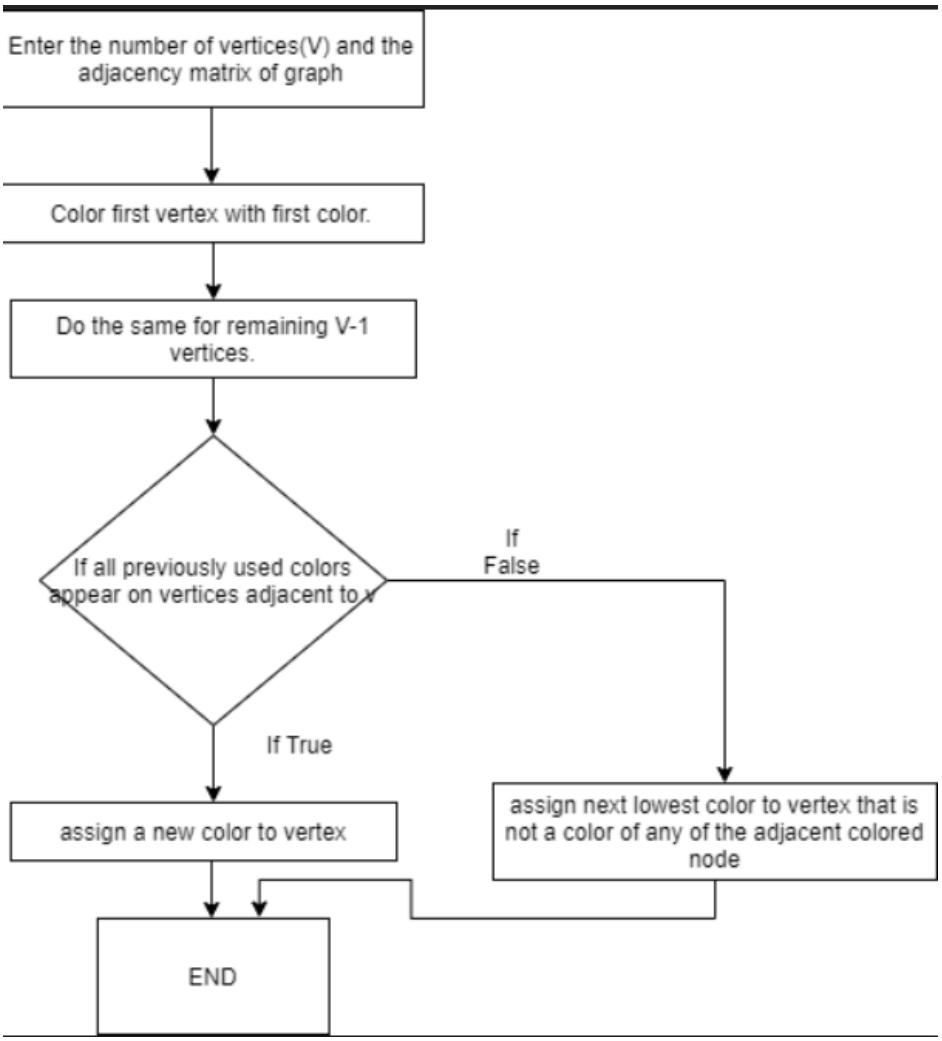


Fig 5.2 Flowchart - Greedy Algorithm

This function returns the solution, ie, a colored graph in the form of a python dictionary, where every vertex is mapped to a color along with the total runtime of this algorithm as tt, which is imported as gtt and used in the analysis.py function to generate a plot for pictorial comparison.

5.3 GRAPH COLORING PROBLEM USING BACKTRACKING ALGORITHM - gcpbt.py function

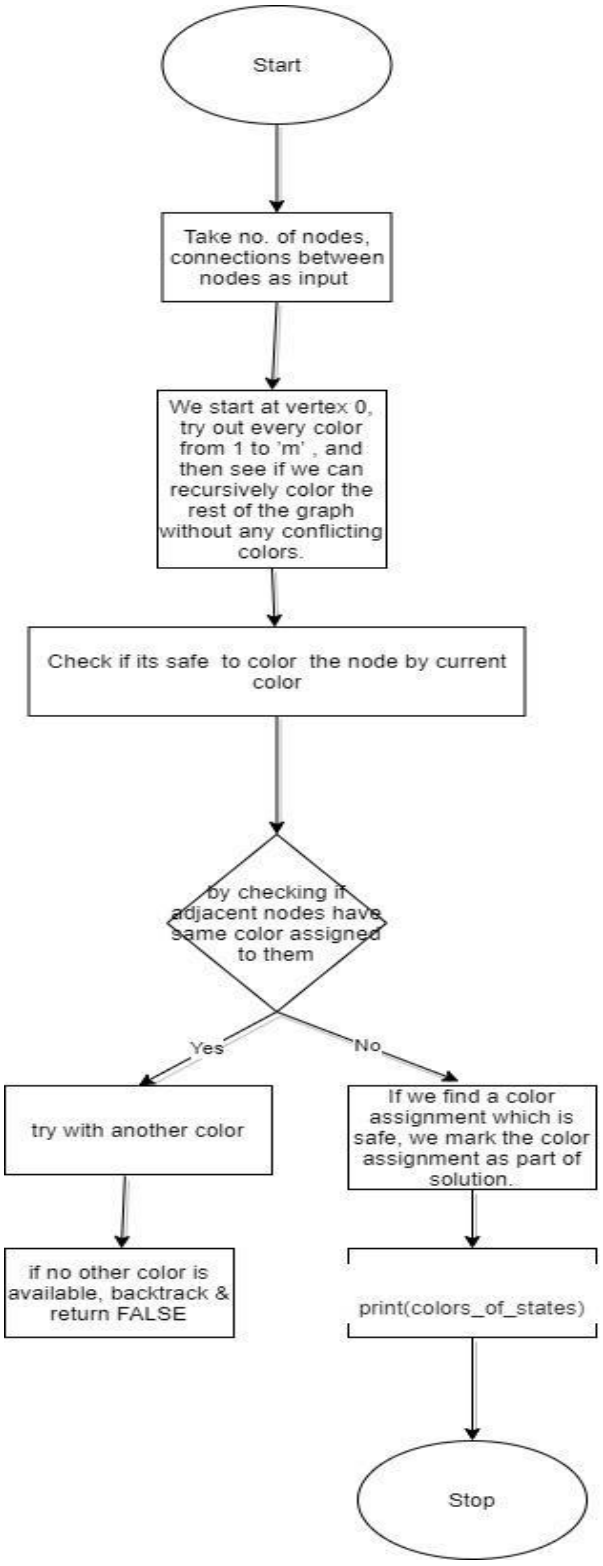


Fig 5.3 Flowchart - Backtracking Algorithm

5.4 GRAPH COLORING PROBLEM USING WELSH POWELL ALGORITHM - gcpwp.py function

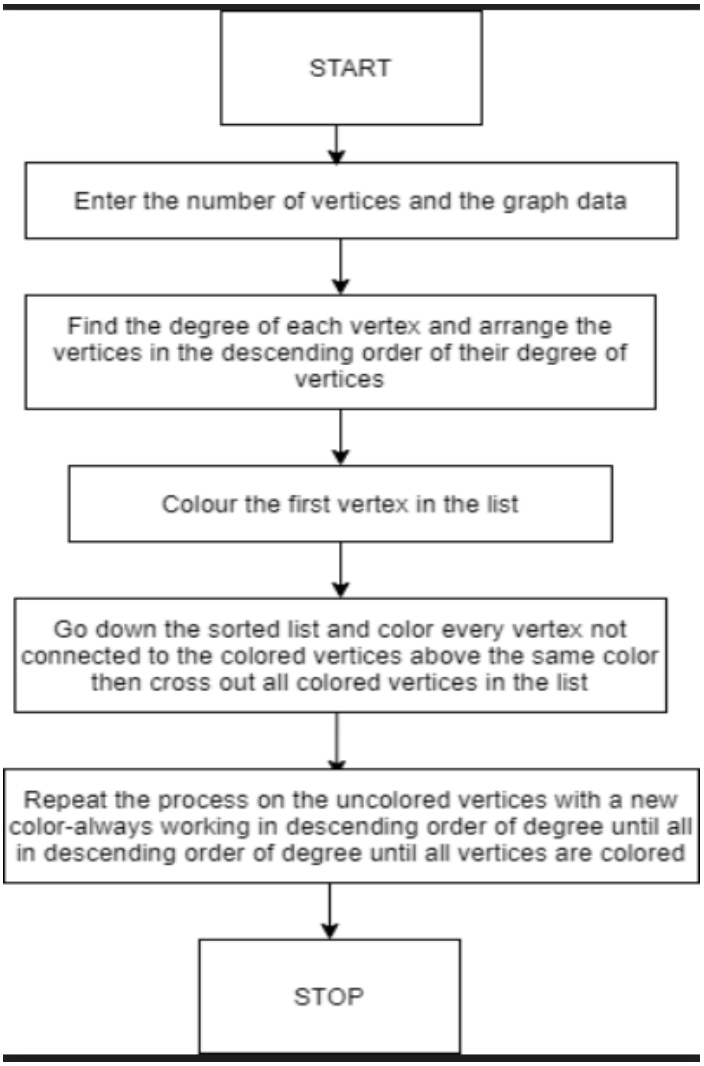


Fig 5.4 Flowchart - Welsh Powell Algorithm

This function returns the solution, ie, a colored graph in the form of a python dictionary, where every vertex is mapped to a color along with the total runtime of this algorithm as tt, which is imported as gtt and used in the analysis.py function to generate a plot for pictorial comparison.

CHAPTER 6

IMPLEMENTATION

6.1 THE ANALYSIS.PY FUNCTION

```
import matplotlib.pyplot as plt print("------------------
------------------------")
print("Analysis section of the graph coloring problem")

print("We use 3 different algorithms!")

print("Now, come, give us the data!")

print("-----------------------------------------")

from gcpgy import tt

gtt=tt



from gcpwp import tt

wtt=tt



from gcpbt import tt

btt=tt



import numpy as np



alg=['greedy', 'backtracking','welshpowell']

tt=[gtt,btt,wtt]
```

```
ypos=np.arange(len(alg))

plt.xticks(ypos,alg)

plt.ylabel("Time(s)")

plt.xlabel("greedy vs backtracking vs welsh powell")

plt.title ("Effeciencies Analysis of greedy vs
backtracking vs welshpowell algorithms")
l=plt.bar(ypos,tt)

l[0].set_color('r')

l[1].set_color('g')

plt.legend()

plt.show()
```

First the gcpgy.py(greedy algorithm) code is executed, we enter the number of nodes and enter the number of connections every node has.

Then the tt (time taken) here is imported and saved as gtt. Then gcpwp.py(welsh powell) code is executed,with the input values given earlier. Then the tt here also is imported and saved as wtt.Then gcpbt.py (backtracking) code is executed. Then the tt here also is imported and saved as btt.

6.2 GENERATING A GRAPH

As shown in the above code, we make a list containing the time taken by all 3 algorithms.

Now we feed this list of data to matplotlib for getting a performance analysis where on the Y - axis we have the runtime and on the X - axis we have the algorithms.

6.3 THE PICTORIAL REPRESENTATION

We initially used the Python module Matplotlib, that works in a way similar to MATLAB, to generate a graph to depict the time taken by each algorithm to produce an Output of every test case.

As you can see in the first test case in Chapter 7, the difference in the run times by each algorithm is minute to the human, but is of great essence to a system. Pictorial representation of these runtimes using matplotlib.pyplot was not satisfactory as a change in scale to actually view a difference.

Hence we pick an alternative method for the pictorial representation, ie, data visualization using Seaborn. Seaborn requires the used to store the data (the run times here) in a file with a Comma Separated file (a .csv extension) and use a set of codes to run that .csv file and generate the plots as shown and depicted in the Results and Snapshot section of this project report.

6.4 THE ALGORITHMS

6.4.1 GREEDY ALGORITHM

1. Color the first vertex with the first color.
2. Do the following for remaining V-1 vertices.
    a. Consider the currently picked vertex and color it with the lowest numbered color that has not been used on any previously colored vertices adjacent to it. If all previously used colors appear on vertices adjacent to v, assign a new color to it.

6.4.2 BACKTRACKING ALGORITHM

1. We start at vertex 0, try out every color from 1 to 'm' , and then see if we can recursively color the rest of the graph without any conflicting colors. We created a helper function isSafe (self, v, colour, c) that looks at the last colored vertex and all its neighbours to see if it conflicts with any of its neighbours (i.e. has the same color).
2. To represent the colors, we can just keep a separate colors list that maps 1-to-1 with the vertices. If we find a color assignment which is safe, we mark the color assignment as part of the solution. If we do not find colour due to clashes then we backtrack and return false.

6.4.3 WELSH POWELL ALGORITHM

1. Find the degree of each vertex .
2. List the vertices in order of descending valence i.e, valence degree(v(i)) >= degree(v(i+1)).
3. Colour the first vertex in the list.
4. Go down the sorted list and colour every vertex not connected to the colored vertices above the same colour then cross out all colored vertices in the list.
5. Repeat the process on the uncoloured vertices with a new colour -always working in descending order of degree until all in descending order of degree until all vertices are coloured.

CHAPTER 7

TESTING

The various test cases that were fed as input to our project's implementation were all Planar graphs, ie, a two dimensional graph without any self looks, or directed edges, entered after prompted by the program.

These graphs were entered in the form of an adjacency list, where based on degree (number of connections to nodes), all the vertices connected to the current node are entered one by one.

Test case 1 :

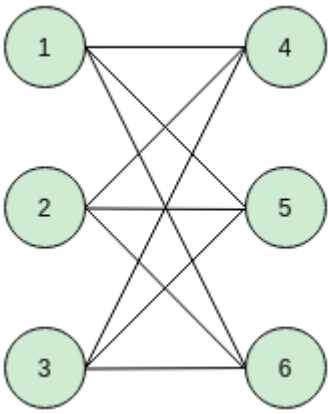


Fig 7.1 Test case 1 input

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 2 :

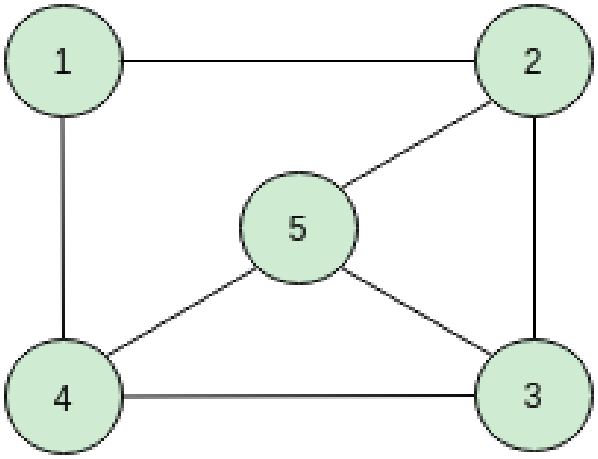


Fig 7.2 Test case 2

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 3 :

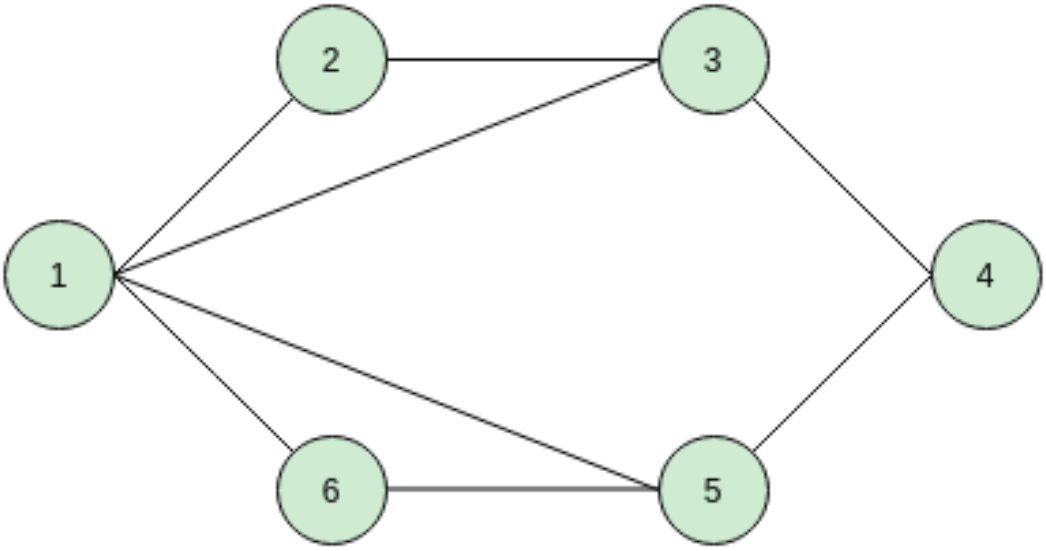


Fig 7.3 Test case 3

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 4 :

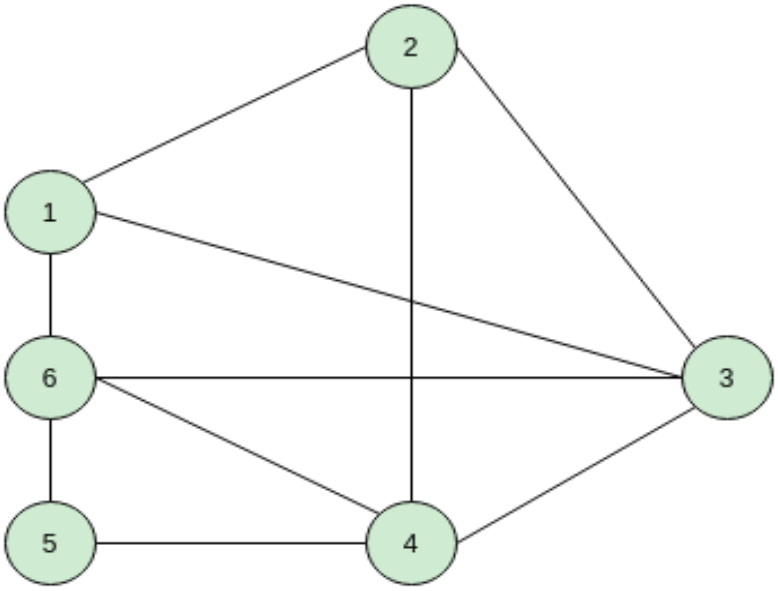


Fig 7.4 Test case 4

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 5 :

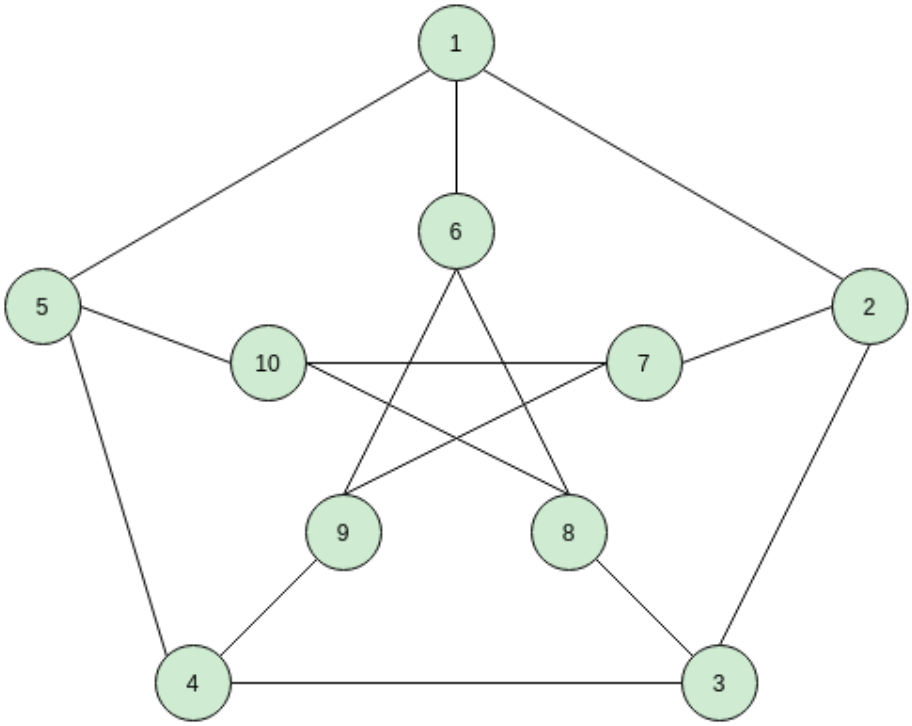


Fig 7.5 Test case 5

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 6 :

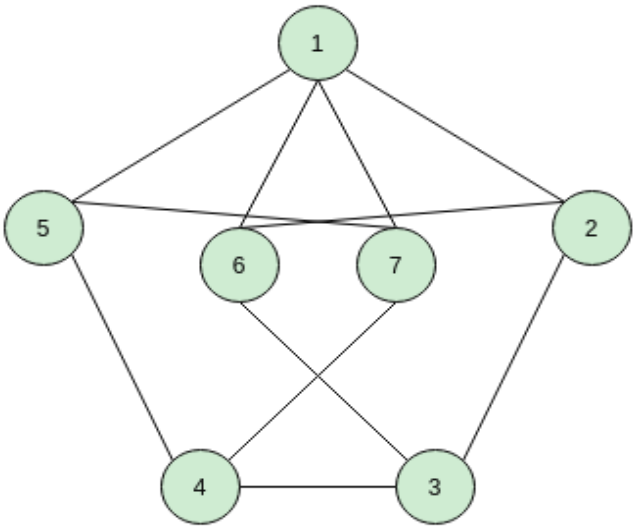


Fig 7.6 Test case 6

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 7 :

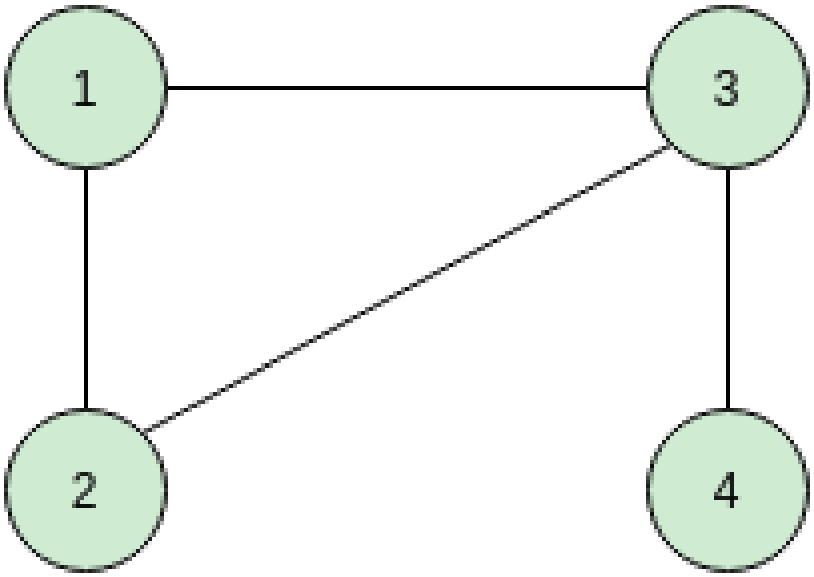


Fig 7.7 Test case 7

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 8 :

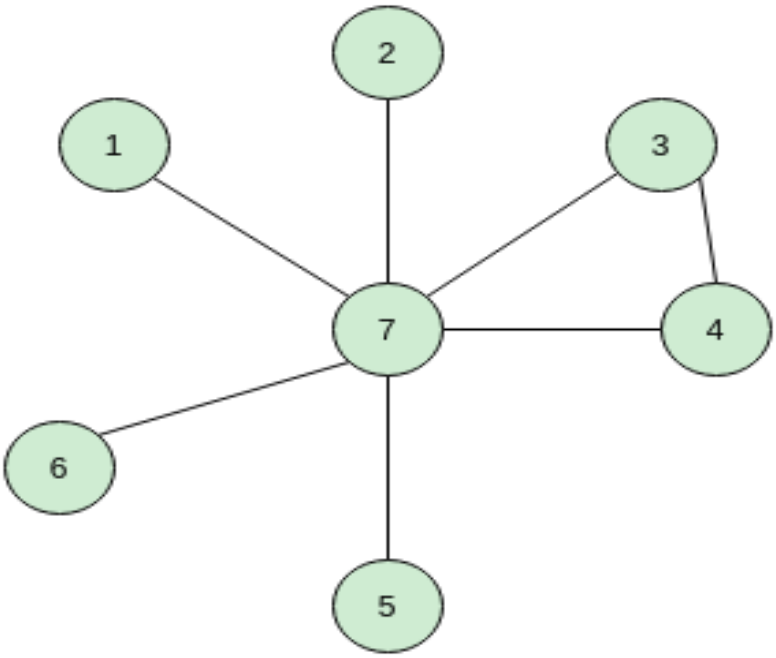


Fig 7.8 Test case 8

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 9 :

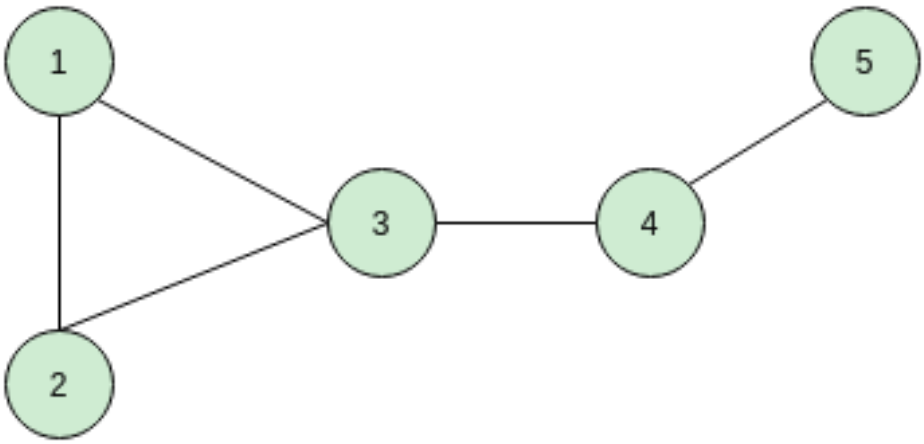


Fig 7.9 Test case 9

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

Test case 10 :

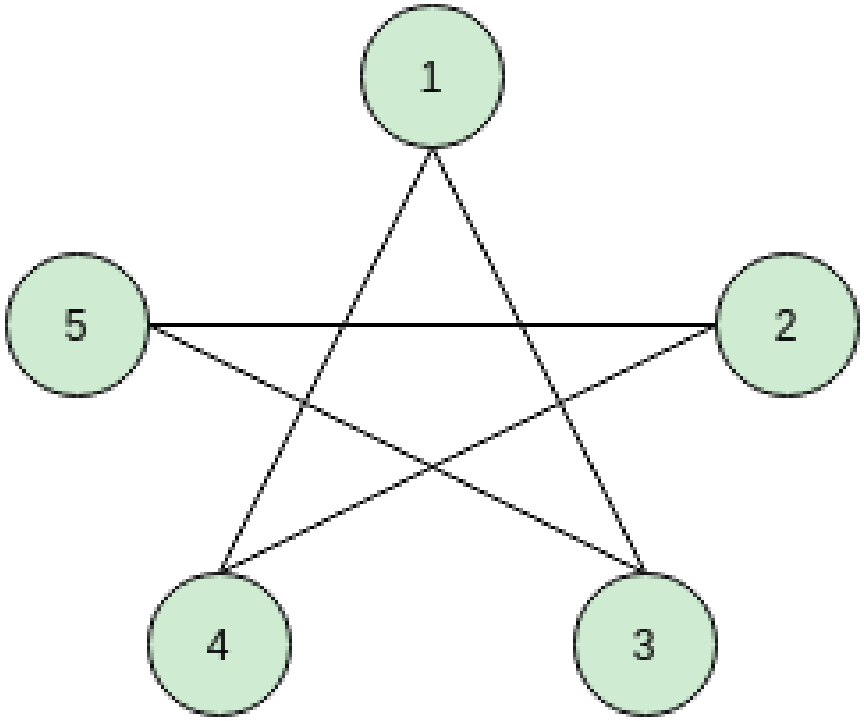


Fig 7.10 Test case 10

Expected output: Result should be colours assigned to nodes with runtimes of greedy, backtracking and welsh powell algorithms and the analysis of runtimes on a graph.

Test result: Success.

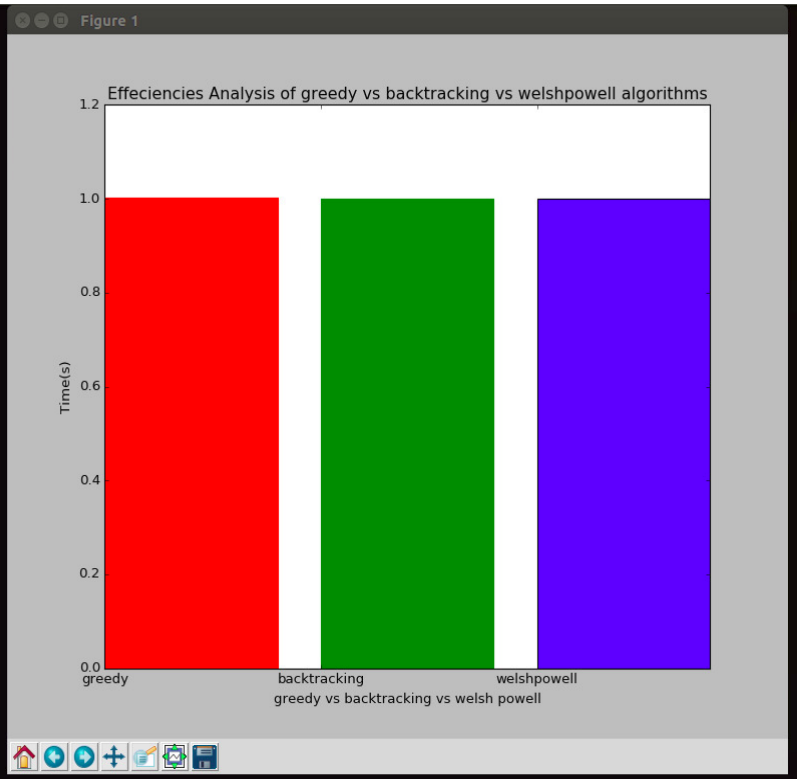
So, all the test cases shown above are planar graphs. The unsuccessful cases would basically be any graph that is not a planar graph. For example, in graphs with self - loops we cannot assign colours to nodes as the whole concept of having self - loops in our graph goes against the idea of graph colouring. As we've seen in the earlier chapters, we assign colours to nodes based on a constraint that no two adjacent nodes have the same colour given to them.

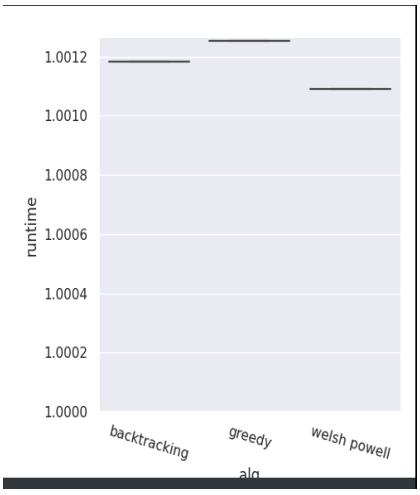
CHAPTER 8

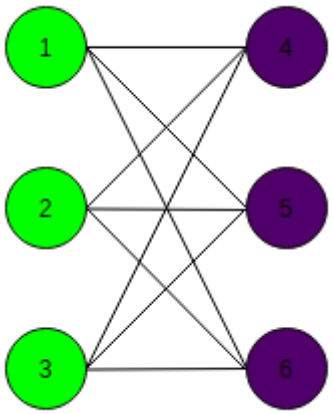
RESULTS AND SNAPSHOTS

Output of test case 1:

```
----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 6
[1, 2, 3, 4, 5, 6]
Enter the number of connections for {1}:
3
4
5
6
Enter the number of connections for {2}:
3
4
5
6
Enter the number of connections for {3}:
3
4
5
6
Enter the number of connections for {4}:
3
1
2
3
Enter the number of connections for {5}:
3
1
2
3
Enter the number of connections for {6}:
3
1
2
3

----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'r', 3: 'r', 4: 'b', 5: 'b', 6: 'b'}

------------------------------------------------

Greedy Algorithm : 1.001254

Welsh powell : 1.001091

Backtracking Algorithm : 1.001182
```







Algorithm	Time taken(in seconds)
Greedy	1.001254
Backtracking	1.001182
Welsh Powell	1.001091

Table 8.1: Runtimes for test case 1

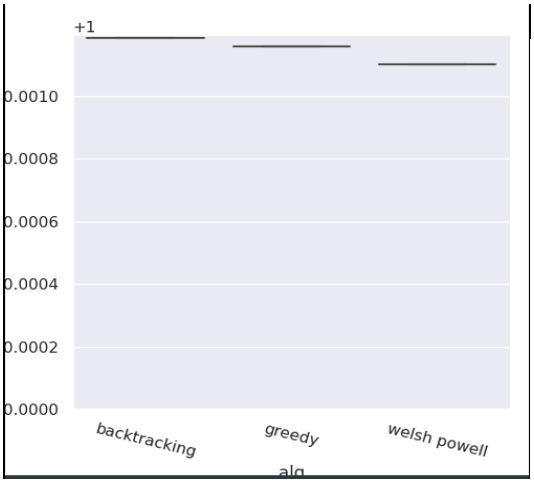
We initially used the Python module Matplotlib, that works in a way similar to MATLAB, to generate a graph to depict the time taken by each algorithm to produce an Output of every test case.

As you can see in the outputs above, the difference in the run times by each algorithm is minute to the human, but is of great essence to a system. Pictorial representation of these runtimes using matplotlib.pyplot was not satisfactory as a change in scale to actually view a difference.

Hence, we pick an alternative method for the pictorial representation, ie, data visualization using Seaborn. Seaborn requires the used to store the data (the run times here) in a file with a Comma Separated file (a .csv extension) and use a set of codes to run that .csv file and generate the plots as shown and depicted in the Results and Snapshot section of this project report.

Output of test case 2:

```
----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 5
[1, 2, 3, 4, 5]
Enter the number of connections for {1}:
2
2
4
Enter the number of connections for {2}:
3
1
3
5
Enter the number of connections for {3}:
3
2
4
5
Enter the number of connections for {4}:
3
1
3
5
Enter the number of connections for {5}:
3
2
3
4

----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'r', 4: 'b', 5: 'g'}

------------------------------------------------
Total time taken to solve the entered graph is :
------------------------------------------------

Greedy Algorithm : 1.001160

Welsh powell : 1.001102

Backtracking Algorithm : 1.001188
```



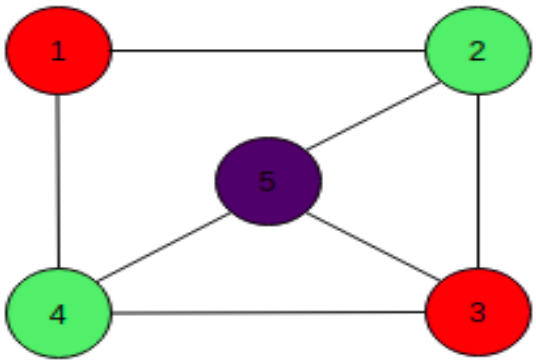


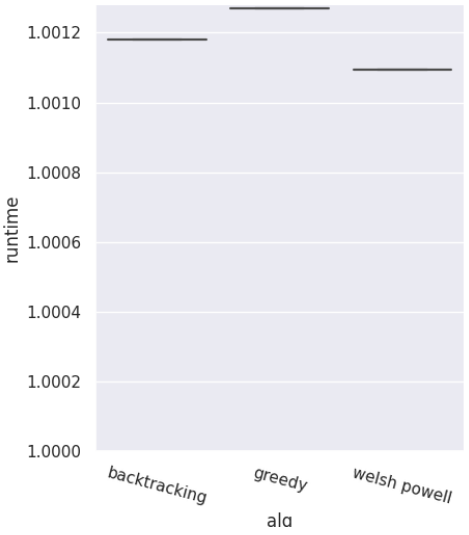
Fig 8.2 Output of test case 2

Algorithm	Time taken(in seconds)
Greedy	1.00116
Backtracking	1.001188
Welsh Powell	0.001102

Table 8.2: Runtimes for test case 2

Output of test case 3:

```
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 6
[1, 2, 3, 4, 5, 6]
Enter the number of connections for {1}:
4
2
3
5
6
Enter the number of connections for {2}:
2
1
3
Enter the number of connections for {3}:
3
1
2
4
Enter the number of connections for {4}:
2
3
5
Enter the number of connections for {5}:
3
1
4
6
Enter the number of connections for {6}:
2
1
5

----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'g', 4: 'r', 5: 'b', 6: 'g'}

------------------------------------------------
Total time taken to solve the entered graph is :
------------------------------------------------

Greedy Algorithm : 1.001270

Welsh powell : 1.001095

Backtracking Algorithm : 1.001180
```



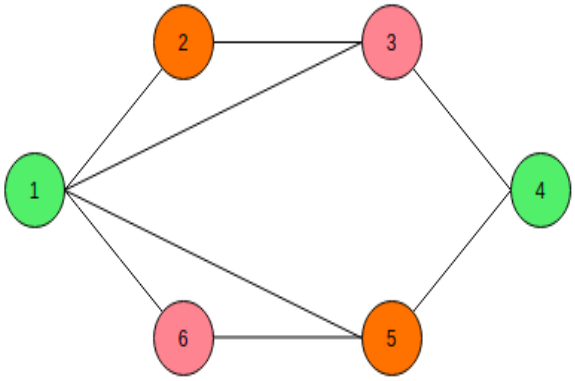


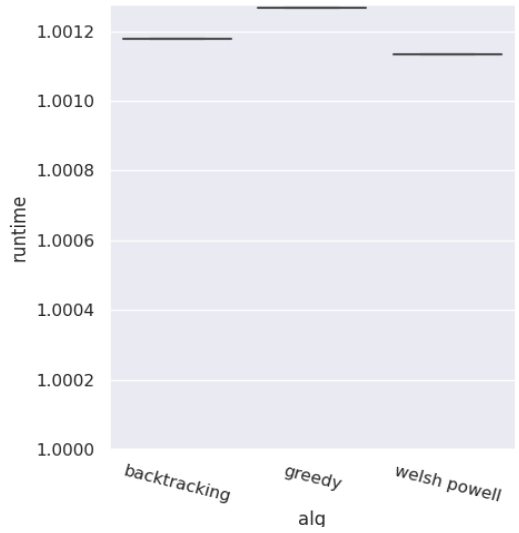
Fig 8.3 Output of test case 3

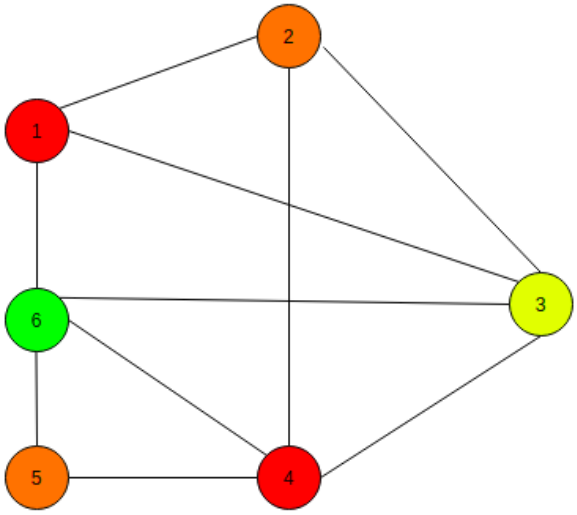
Algorithm	Time taken(in seconds)
Greedy	1.00127
Backtracking	1.00118
Welsh Powell	1.001095

Table 8.3: Runtimes for test case 3

Output of test case 4:

```
percyjackson@Percy-Vostro-3478:~/GraphColoring$ python3 analysis.py
-----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
-----------------------------------------
Enter number of nodes : 6
[1, 2, 3, 4, 5, 6]
Enter the number of connections for {1}:
3
2
3
6
Enter the number of connections for {2}:
3
1
3
4
Enter the number of connections for {3}:
4
1
2
4
6
Enter the number of connections for {4}:
4
2
3
5
6
Enter the number of connections for {5}:
2
4
6
Enter the number of connections for {6}:
4
1
3
4
5
-----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'g', 4: 'r', 5: 'b', 6: 'y'}

-------------------------------------------------
Total time taken to solve the entered graph is :
-------------------------------------------------

Greedy Algorithm : 1.001266

Welsh powell : 1.001133

Backtracking Algorithm : 1.001178
```





Algorithm	Time taken(in seconds)
Greedy	1.001266
Backtracking	1.001178
Welsh Powell	1.001133

Table 8.4: Runtimes for test case 4

Output of test case 5:

```
----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 10
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Enter the number of connections for {1}:
3
2
5
6
Enter the number of connections for {2}:
3
1
3
7
Enter the number of connections for {3}:
3
2
4
8
Enter the number of connections for {4}:
3
3
5
9
Enter the number of connections for {5}:
3
1
4
10
Enter the number of connections for {6}:
3
1
8
9
Enter the number of connections for {7}:
3
2
9
10
Enter the number of connections for {8}:
3
3
6
10
Enter the number of connections for {9}:
3
4
6
7
Enter the number of connections for {10}:
3
5
7
8

----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'r', 4: 'b', 5: 'g', 6: 'b', 7: 'r', 8: 'g', 9: 'g', 10: 'b'}

------------------------------------------------
Total time taken to solve the entered graph is :
------------------------------------------------

Greedy Algorithm : 1.001255

Welsh powell : 1.001091

Backtracking Algorithm : 1.001234
```

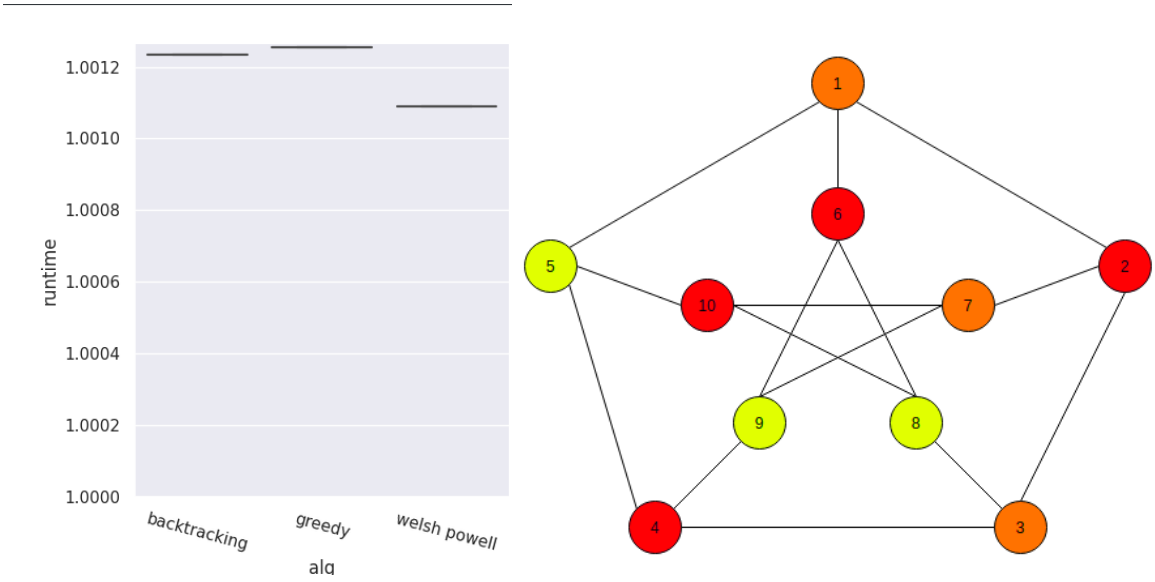


Fig 8.5 Output of test case 5

Algorithm	Time taken(in seconds)
Greedy	1.001255
Backtracking	1.001234
Welsh Powell	1.001091

Table 8.5: Runtimes for test case 5

Output of test case 6:

```
----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 7
[1, 2, 3, 4, 5, 6, 7]
Enter the number of connections for {1}:
4
2
5
6
7
Enter the number of connections for {2}:
2
3
6
Enter the number of connections for {3}:
3
2
4
7
Enter the number of connections for {4}:
3
3
5
7
Enter the number of connections for {5}:
2
4
7
Enter the number of connections for {6}:
3
1
2
3
Enter the number of connections for {7}:
3
1
4
5

----------------------------------------
----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'r', 3: 'b', 4: 'r', 5: 'b', 6: 'g', 7: 'g'}

------------------------------------------------
Total time taken to solve the entered graph is :
------------------------------------------------

Greedy Algorithm : 1.001239

Welsh powell : 1.001102

Backtracking Algorithm : 1.001193
```

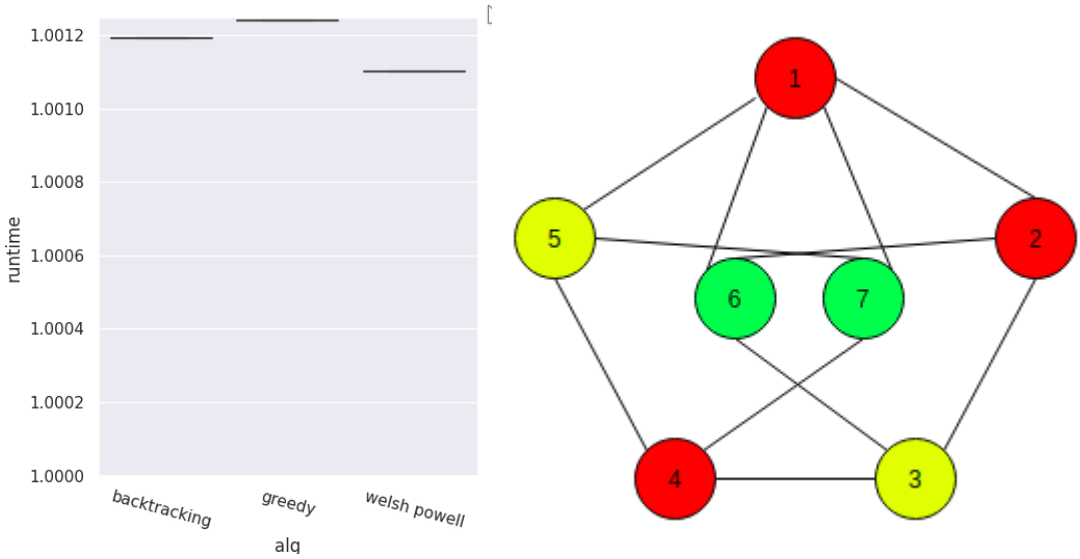


Fig 8.6 Output of test case 6

Algorithm	Time taken(in seconds)
Greedy	1.001239
Backtracking	1.001193
Welsh Powell	1.001102

Table 8.6: Runtimes for test case 6

Output of test case 7:

```
percyjackson@Percy-Vostro-3478:~/GraphColoring$ python3 analysis.py
----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
----------------------------------------
Enter number of nodes : 4
[1, 2, 3, 4]
Enter the number of connections for {1}:
2
2
4
Enter the number of connections for {2}:
2
1
4
Enter the number of connections for {3}:
1
4
Enter the number of connections for {4}:
3
1
2
3

----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'r', 4: 'g'}

------------------------------------------------
Total time taken to solve the entered graph is :
------------------------------------------------

Greedy Algorithm : 1.001208

Welsh powell : 1.001098
```

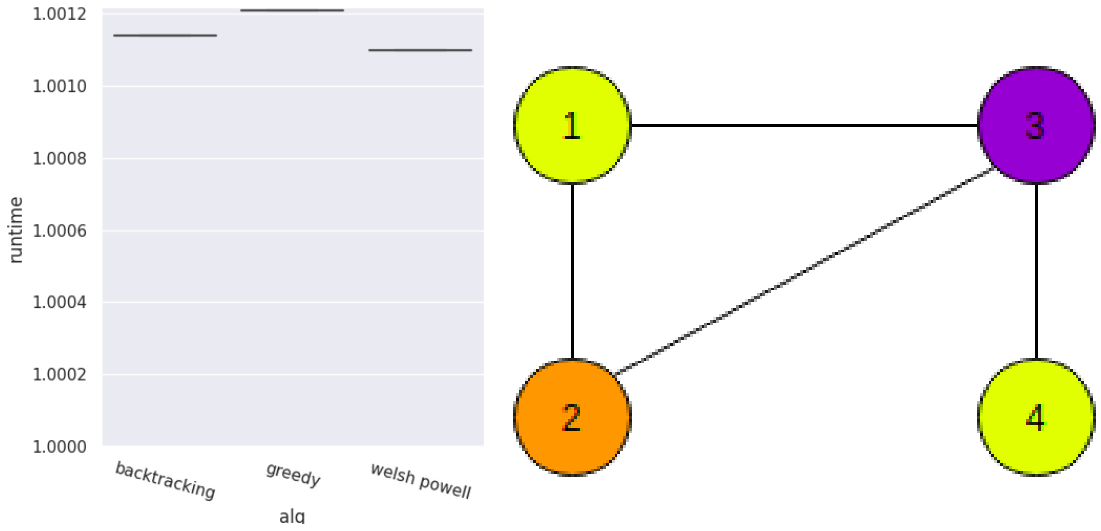
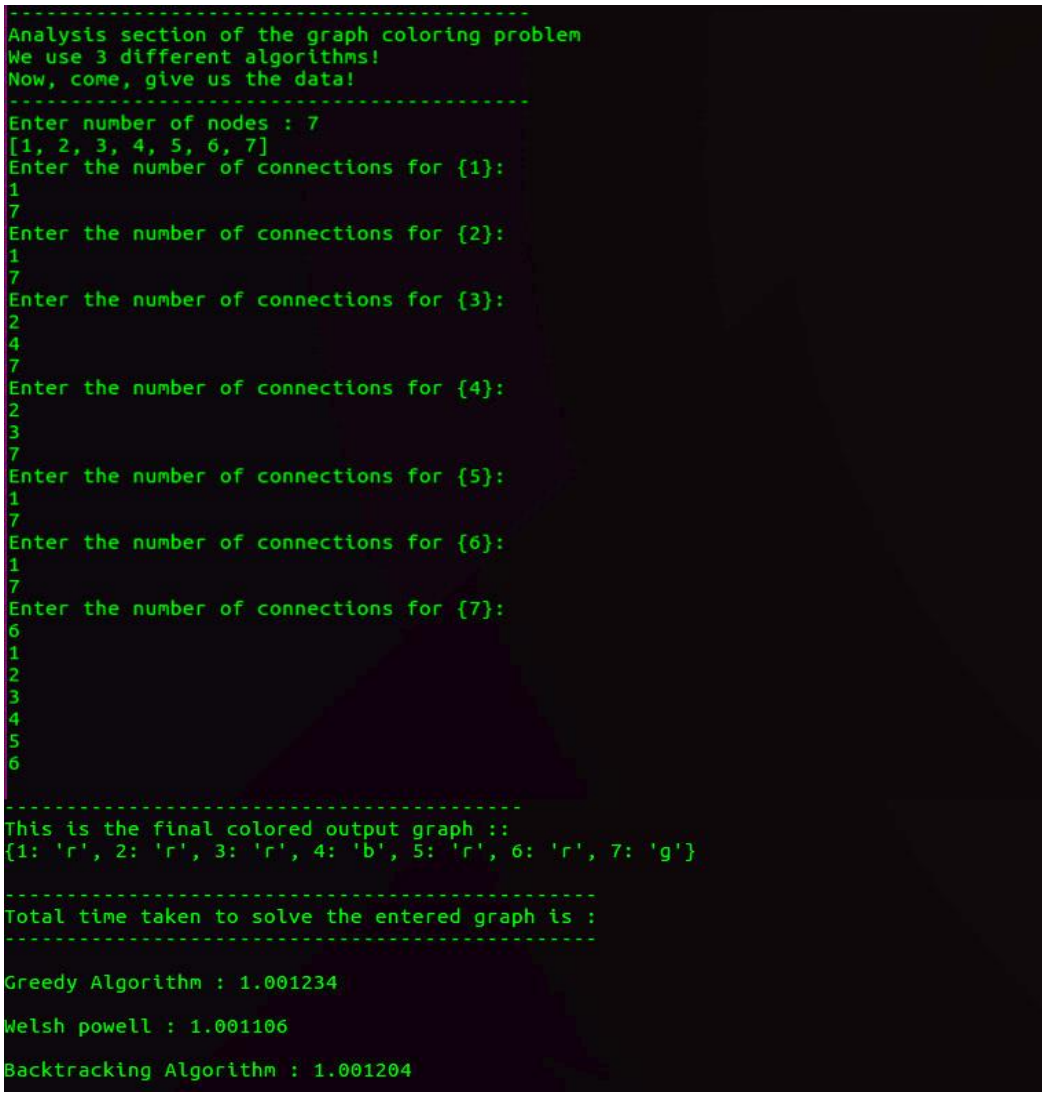


Fig 8.7 Output of test case 7

Algorithm	Time taken(in seconds)
Greedy	1.001208
Backtracking	1.001137
Welsh Powell	1.001098

Table 8.7: Runtimes for test case 7

Output of test case 8:



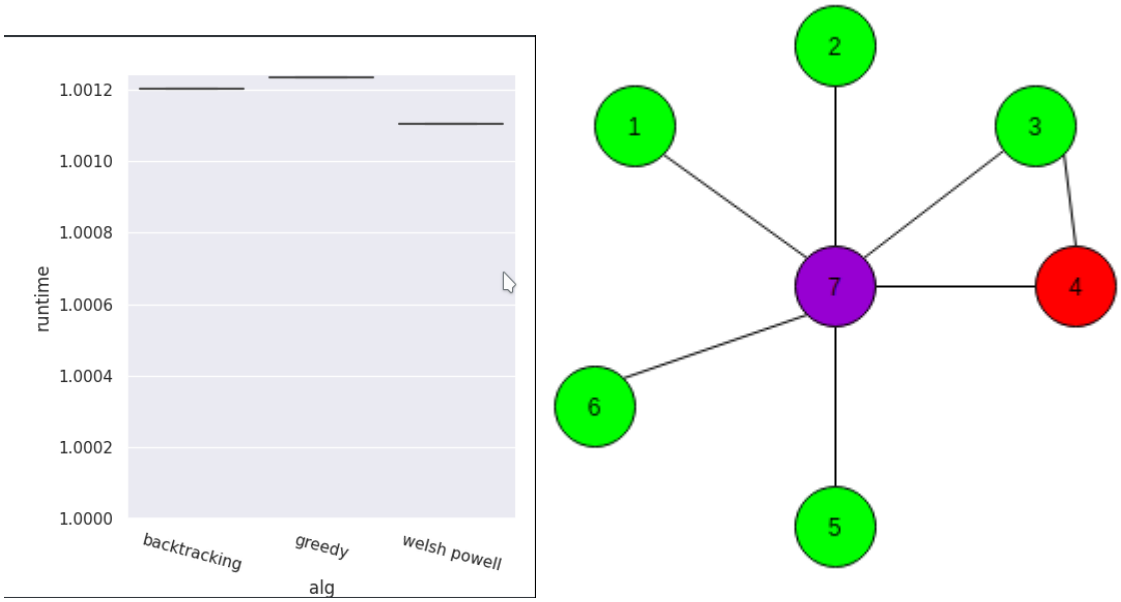


Fig 8.8 Output of test case 8

Algorithm	Time taken(in seconds)
Greedy	1.001234
Backtracking	1.001204
Welsh Powell	1.001106

Table 8.8: Runtimes for test case 8

Output of test case 9:

```
-----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
-----------------------------------------
Enter number of nodes : 5
[1, 2, 3, 4, 5]
Enter the number of connections for {1}:
2
2
3
Enter the number of connections for {2}:
2
1
3
Enter the number of connections for {3}:
3
1
2
4
Enter the number of connections for {4}:
2
3
5
Enter the number of connections for {5}:
1
4

-----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'b', 3: 'g', 4: 'r', 5: 'b'}

-------------------------------------------------
Total time taken to solve the entered graph is :
-------------------------------------------------

Greedy Algorithm : 1.001267

Welsh powell : 1.001105

Backtracking Algorithm : 1.001191
```

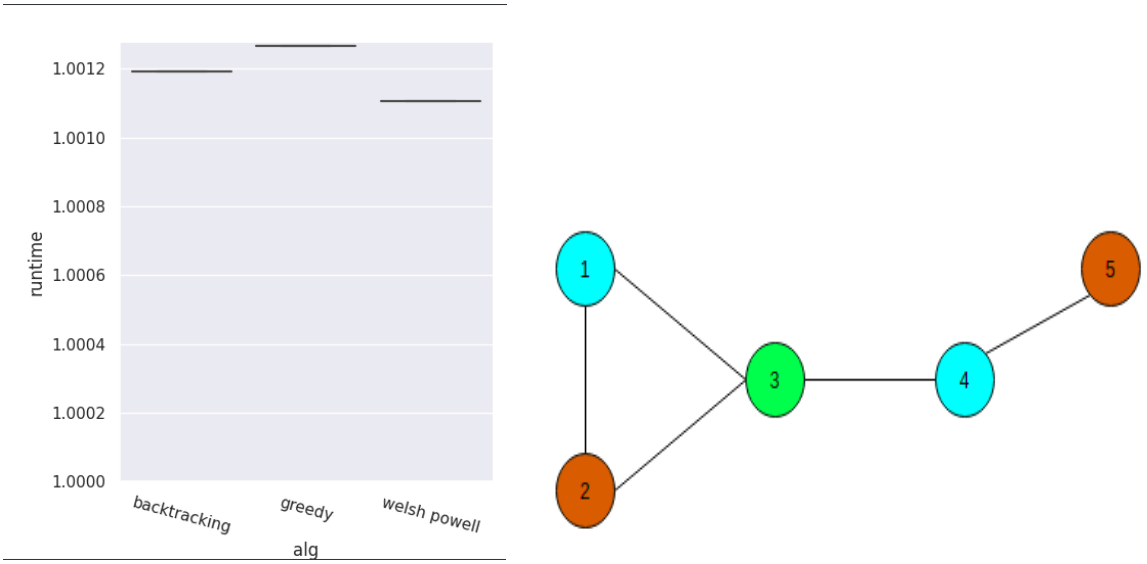


Fig 8.9 Output of test case 9

Algorithm	Time taken(in seconds)
Greedy	1.001267
Backtracking	1.001191
Welsh Powell	1.001105

Table 8.9: Runtimes for test case 9

Output of test case 10:

```
percyjackson@Percy-Vostro-3478:~/GraphColoring$ python3 analysis.py
-----------------------------------------
Analysis section of the graph coloring problem
We use 3 different algorithms!
Now, come, give us the data!
-----------------------------------------
Enter number of nodes : 5
[1, 2, 3, 4, 5]
Enter the number of connections for {1}:
2
3
4
Enter the number of connections for {2}:
2
4
5
Enter the number of connections for {3}:
2
1
5
Enter the number of connections for {4}:
2
1
2
Enter the number of connections for {5}:
2
2
3

-----------------------------------------
This is the final colored output graph ::
{1: 'r', 2: 'r', 3: 'b', 4: 'b', 5: 'g'}

-----------------------------------------------
Total time taken to solve the entered graph is :
-----------------------------------------------

Greedy Algorithm : 1.001284

Welsh powell : 1.001109

Backtracking Algorithm : 1.001237
```

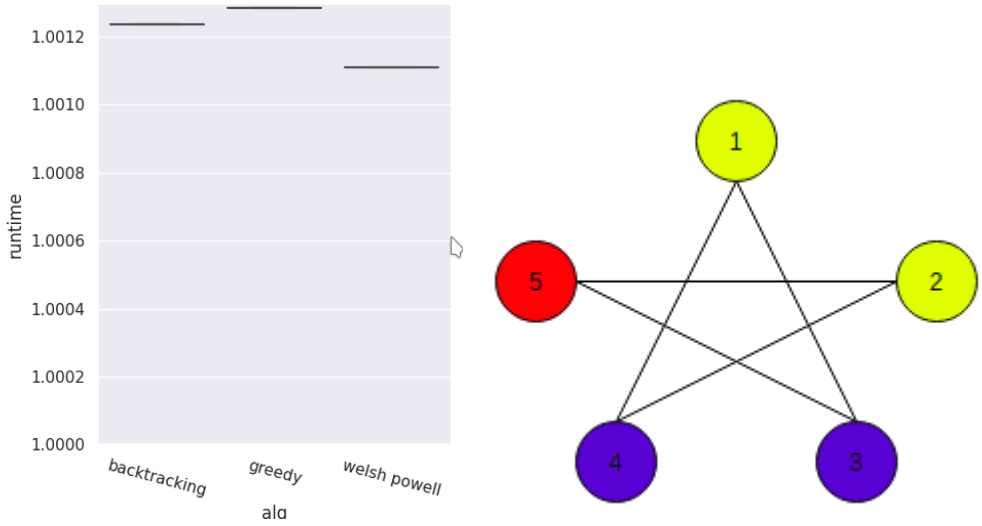


Fig 8.10 Output of test case 10

Algorithm	Time taken(in seconds)
Greedy	1.001267
Backtracking	1.001191
Welsh Powell	1.001105

Table 8.10: Runtimes for test case 10

So, as we can see in every test case, Welsh Powell algorithm turned out to be the most efficient to solve the graph coloring problems, since it has the least runtime amongst all the three. In the ten test cases, the Greedy algorithm has the highest runtime; Thus, it is the least efficient algorithm we can use to solve the graph coloring problem amongst the three algorithms.

CHAPTER 9

ADVANTAGES AND LIMITATIONS

9.1 GREEDY ALGORITHM

Advantages :

1. Finding a solution is quite easy with a greedy algorithm for a problem.
2. Analysing the run time for greedy algorithms will generally be much easier than for other techniques (like Divide and conquer).

Disadvantages:

Greedy Algorithm basically builds a solution part by part and chooses the next part in such a way that it yields the best solution to the present problem at hand immediately. As a result, there is no regard or worry about the consequences of the current decision taken. Never reconsidering the choices taken previously, Greedy Algorithm fails to produce an optimal solution, though it gives a near optimal solution. Knapsack Problem and Travelling Salesman Problem are examples of problems where the Greedy Algorithm fails to produce an optimal solution.

9.2 BACKTRACKING ALGORITHM

Advantages:

1) Simple to implement.
2) State changes are stored in stack, meaning we do not need to concern ourselves about them.
3) Intuitive approach of trial and error.
4) Code size is usually small.

Disadvantages:

1) Multiple function calls are expensive.
2) Inefficient when there is lots of branching from one state.
3) Requires a large amount of space as each function state needs to be stored on the stack.

9.3 WELSH POWELL ALGORITHM

Advantages :

1) This algorithm is the most efficient when compared to the other two.
2) exhaustive search is avoided in this algorithm.
3) this algorithm traverses through a minimum number of nodes in order to provide the solution.

Disadvantages :

1) In the case of a graph consisting of many vertices, it takes a lot of time to find the degree of all vertices and use them for solving.

CHAPTER 10

CONCLUSION AND FUTURE ENHANCEMENTS

10.1 CONCLUSION

Through this project we were able to completely understand the concepts and working of Graph Colouring using backtracking, greedy method approach and welsh powell algorithm.

Graph colouring is an assignment of labels, called colours, to the vertices of a graph such that no two adjacent vertices share the same colour, thereby, known as vertex colouring. This graph colouring problem is also known as M-colourability decision problem. Other types of colourings on graphs also exist, most notably edge colourings that may be subject to various constraints.

Through this project we also learned a lot about graph colouring and the working of the Greedy algorithm, The Backtracking algorithm and the Welsh Powell algorithm.

Also,the implementation of backtracking, greedy method approach and welsh powell algorithm to solve graph colouring.

From the table below, we can clearly observe that the Welsh Powell algorithm takes the least amount of time for execution for every test case and provides us with a solution, in comparison with the Greedy and the Backtracking algorithms.

Graph Number	Greedy Algorithm	Backtracking Algorithm	Welsh Powell Algorithm
1	1.001254	1.001182	1.001091
2	1.001160	1.001188	1.001102
3	1.001270	1.001180	1.001095
4	1.001266	1.001178	1.001133
5	1.001255	1.001234	1.001091
6	1.001239	1.001193	1.001102
7	1.001207	1.001137	1.001098
8	1.001234	1.001204	1.001106
9	1.001267	1.001191	1.001105
10	1.001284	1.001237	1.001109

Table 10.1: Table depicting runtimes for ten cases, under three different algorithms

10.2 FUTURE ENHANCEMENTS

As of now these existing solutions for graph coloring mainly just focused on computing a good coloring for a static graph. But, as seen earlier in the applications many of which have been stated in the introduction within the first chapter of this report, many real-world graphs are highly dynamic, so we look forward to trying to incrementally maintain the graph coloring when the graph is dynamically updated.

We shall next focus mainly on two goals : high effectiveness and high efficiency.

1. To achieve high effectiveness, we try maintaining the graph coloring in a way such that the coloring result is consistent with one of the best static graph coloring algorithms for large graphs.
2. To achieve high efficiency, we look into more efficient incremental algorithms to update the graph coloring by exploring a smaller number of vertices.

Also, the other improvements that can be made for a better visualisation of the user entered graph would be , using the networkx module in python. NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

NetworkX provides:

1. Tools for the study of the structure and dynamics of social, biological, and infrastructure networks
2. A standard programming interface and graph implementation that is suitable for many applications

With NetworkX you can load and store networks in standard and nonstandard data formats, generate many types of random and classic networks, analyze network structure, build network models, design new network algorithms, draw networks, and much more.

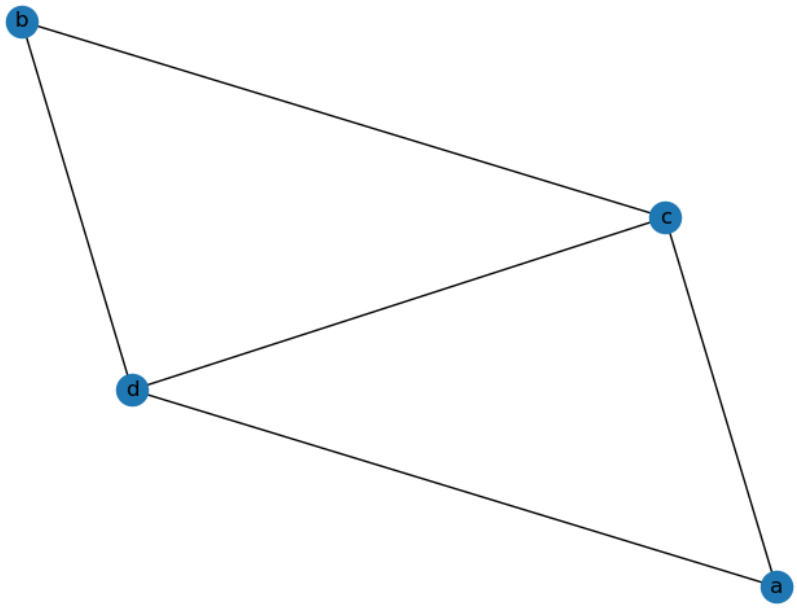


Fig 10.1: Output given by network x module for the input graph given

The figure above shows the use of the networkx module in python to implement the visualisation of the user entered graph data.

REFERENCES

- https://www.tutorialspoint.com/graph_theory/graph_theory_coloring.htm
- https://www.educba.com/what-is-a-greedy-algorithm/
- https://www.quora.com/What-are-the-advantages-and-disadvantages-of-a-backtracking-algorithm
- https://en.wikipedia.org/wiki/Graph_coloring
- https://www.geeksforgeeks.org/graph-coloring-applications/
- https://www.geeksforgeeks.org/graph-coloring-set-2-greedy-algorithm/
- https://iq.opengenus.org/welsh-powell-algorithm/
- https://app.diagrams.net/