"Vizing's theorem and Edge Coloring" Project Report

COMP7750, University of Manitoba, Winter 2018

Muhammad Arsal Asif
asifma@myumanitoba.ca

Abstract

The edge-coloring problem is one of the fundamental problems in graph theory and it is linked to various other domains. Vizing's Theorem states that graphs can be colored in Δ or $\Delta + 1$ colors. In this report, various graphs were tested and their edge chromatic number X'(G) noted. Graphs are recolored to achieve a Δ coloring and to determine the class of graph. Some graphs were colored in Δ colors without need of recoloring. Some $\Delta + 1$ graphs were successfully recolored to Δ colors while rest maintained $\Delta + 1$ coloring.

1 Introduction

Edge-coloring problem is to color graph G such that no two adjacent edges share the same color. Edge coloring problem's origin dates back to Peter Guthrie Tait's attempts to prove the Four-Color Theorem [1]. Edge coloring is linked to many other areas such as map coloring, matching theory, factorization theory, Latin squares and scheduling theory [1].

Formally, we can define m-edge-coloring of a graph G as a mapping from E(G) to m colors {1, 2, ..., m}, such that if e and f are adjacent, then m(e) $\neq$ m(f). Edge-chromatic number X'(G) is called the chromatic index of G. It is the minimum value of m such that G has a proper m-edge-coloring [2]. If we can color G in m colors, we can find an m-edge coloring of G and we can say that G is m-edge-colorable.

We can define a color set C(v) to represent colors used to color edges at v. If |C(v)| equals degree of v, then the coloring is *proper*. If C(v) is less than degree of v then a color is repeated at v [2].

1.1 Literature Review

Book [3] by Wallis has detailed explanation on edge coloring, including class 1 and class 2 graphs. Paper [4] details an algorithm to prove Vizing's theorem. Book [1] contains information on recent advances in edge coloring and gives details about using Vizing fans to obtain both known and new results. Paper [5] also proves Vizing's theorem giving an alternative algorithm. The paper that is referenced for algorithm contains an improved method of proving Vizing's theorem over previous method outlined by E.W. Dijkstra and J.R. Rao [6].

2 Problem Statement

2.1 Vizing's Theorem

Theorem 1. If G is simple, then $\Delta \leq$ X'(G) $\leq \Delta+1$ [2].

Δ is maximum degree in graph G [2]. Any graph can be colored in either Δ or $\Delta + 1$ colors. Graphs can be partitioned into two different classes: Class 1 and Class 2. Graphs that use Δ colors and satisfy X'(G) = Δ are called class 1. Graphs with X'(G) = $\Delta + 1$ are called class 2. For regular graphs, having one factorization equals being class 1. The class 2 graphs include the odd-order complete graphs and the odd cycles [3].

2.2 Background for Edge Coloring

Certain lemmas and theorems exist for edge coloring which can define base cases of graph edge coloring. Proofs of these lemmas and theorems are given in [3]. These theorems help understand the types of graphs and number of colors that can be used to color them.

One of the base cases is where G is a connected graph, with no odd cycles.

Lemma 1. If G is a connected graph other than an odd cycle, then there is a 2-edge-coloring of G in which C(v) ≥ 2 whenever DEG(v) ≥ 2 [3].

Next, for any bipartite graph G.

Lemma 2. If G is bipartite, then X'(G) = Δ(G) [3].

It is clear that graphs satisfying Lemma 1 have a 2-edge coloring and graphs satisfying Lemma 2 have edge chromatic number X'(G) = Δ(G).

2.3 Class 2 Graphs

More background information is needed to understand class 2 graphs. Properties of class 2 graphs are given in [3] and stated as theorems and corollaries.

Theorem 2. If G has v vertices and e edges and $e >$ [v/2] Δ(G) then G is class 2 [3].

The edge e is called edge-critical if X'(G - e) < X'(G). An edge-critical graph is defined to be a connected class 2 graph in which every edge is edge-critical. An edge-critical graph of maximum degree Δ is often called Δ-edge-critical. Every class 2 graph contains a Δ-edge-critical subgraph [3].

Theorem 3. If G is a graph of class 2, then G contains a k-edge-critical subgraph for each k such that $2 \leq k \leq \Delta$(G) [3].

Two types of vertices are defined for class 2 graphs. Vertices with degree Δ are called major. Vertices with degree $< \Delta$ are called minor. Major vertices are important in understanding if a graph is class 2.

Theorem 4. Suppose uv is an edge-critical edge in a class 2 graph G. Then u is adjacent to at least Δ - DEG(v) + 1 major vertices other than v [3].

Corollary 1. In edge-critical graph, every vertex is adjacent to at least two major vertices [3].

Corollary 2. Every class two graph contains at least three major vertices [3].

Corollary 3. An edge-critical graph G has at least Δ(G) - Δ(G) + 2 major vertices [3].

Corollary 4. A Δ-edge-critical graph on v vertices has at least 2v/Δ major vertices [3].

Corollary 5. Suppose w is a vertex of a graph G that is adjacent to at most one major vertex, and e an edge containing w. Then $\Delta(G - e) = \Delta(G) \equiv$ X'$(G - e)$ = X'(G) and $\Delta(G - w) = \Delta(G) \equiv$ X'$(G - w)$ = X'(G) [3].

Above corollaries and theorems define the properties of all class 2 graphs. Major vertices and edge-critical edges are most important features of class 2 graphs. Finding graphs that satisfy these conditions can lead us to conclude that they are class 2. Determining the class of any given graph G is a NP-Complete problem.

The scope of this project is to implement edge coloring algorithm given in [2]. Furthermore, various graphs are tested on this algorithm to find if they are colored in Δ or $\Delta + 1$ colors. If a graph can be colored in Δ colors, then it is class 1. If a graph is colored in $\Delta + 1$ colors, then it may or may not be class 2. It is more likely that it is class 2, but it can be recolored to see if it coloring changes to Δ. If coloring changes to Δ, then graph is of class 1.

3 Solution Strategy and Implementation

3.1 Edge Coloring Algorithm Overview

Edge coloring algorithm is given in [2]. The algorithm is based on Vizing's theorem and it guarantees coloring graphs in Δ or $\Delta + 1$ colors. This algorithm was implemented and use cases tested using given algorithm for edge coloring.

The following is a pseudocode of how it works:

Algorithm 1 Edge Coloring Algorithm

```
1: procedure EdgeColor
2:     Start by initializing all edges to color 0
3:     for each vertex u do
4:         Count color frequencies at u
5:         Find first repeated color k at u
6:         while there is still some repeated color k do
7:             Find first missing color j at u
8:             Try to recolor graph through BFS using u, k, j
9:             if fails then
10:                Find second missing color i at u
11:                Try to recolor graph through BFS using u, k, i
12:                if fails then
13:                    Do a color rotation
14:        for each edge uv do
15:            color uv with the first available color
```

Start by assigning color 0 to all edges. Color 0 means an edge is not colored. Algorithm works by processing vertices in order, from vertex 1 to n. Color frequencies at u are counted and if there is a repeated color, it is eliminated by recoloring with a missing color at u. Recoloring is done by color switching through BFS. If BFS fails in recoloring using the first missing color, then it attempts to recolor using second missing color. If both fail, then a color rotation is needed. Color rotation is done until color repetition is eliminated at u. Detailed explanation for algorithm is given in Edge Coloring chapter in [2]. This algorithm is based on Vizing's original algorithm which was used in proof of Vizing's theorem. Similar algorithm to Vizing's original algorithm is also demonstrated in [4].

3.2 Code Overview

The program is implemented in Java. There are six classes in total, each class is briefly defined below.

`AdjacencyList` class stores all edges of vertices as adjacency lists in form of linked lists. Index 0 is left unused and adjacency list is traversed from 1 to n. This is due to files containing vertices starting from 1 and not 0. Instead of converting vertices from 0 to n – 1, they were just stored as 1 to n and respective arrays initialized to n + 1 length.

`Queue` class is used to maintain a queue for edge coloring algorithm. It has basic enqueue(), dequeue() and isEmpty() methods.

`GraphColoring` class contains the edge coloring algorithm code in edgeColor() method. colorBFS() method is used for color switching in edgeColor(). A 2D array called clr[][] maintains coloring of edges. An array called degree[] is used to keep degree of each vertex. This array is used to find Δ and to check class 2 conditions. Graph is stored as an array of AdjacencyList called adjacencyLists[]. This class also has many supplementary methods needed in testing and edge coloring algorithm. Comments in code briefly define all these methods and their names are self explanatory. These include methods such as firstRepeatedColor() (To find first repeated color at vertex u) and firstMissingColor() (To find first missing color at vertex u). One important supplementary method is isProperColoring() which is used after coloring graph to check if given coloring is proper. This method calls anyRepetitions(u) method for each vertex to check if a color is repeated at any vertex, and allEdgesColored() method to make sure all edges in graph are colored.

`Util` class contains utility functions required in other classes. These include methods such as findMax() (To find maximum in an array), findMin() (To find minimum in an array), methods to initialize and print 1D and 2D arrays. One method of particular importance is nonEmptyIndices() which is used to find number of non-zero values in an array. This method is particularly used to find number of colors used after edge coloring. Another method Petersen(n, k) was used to generate generalized Petersen graphs for analysis and testing.

`GraphVisualization` class is used to visualize output graph. GraphStream API [7] is used for visualization. GraphStream is distributed under both licenses CeCILL-C (French version) [8] and LGPL v3 [9].

`Main` class functions for file reading, graph initialization and running tests and colorings. Graph is initially scanned from file and loaded into an object of GraphColoring class. Graph is colored 10 times (without changes) initially. Number of colors used (whether Δ or $\Delta + 1$) are outputted along with number of edges and vertices. Next, we try to recolor graph 10 times, each time maintaining previous coloring but removing the color which occurred minimum number of times. For some graphs, as shown in results section, this resulted in reducing X'(G) from $\Delta + 1$ to Δ. Afterwards, each edge in graph is removed and graph is recolored to see if it changes X'(G). If removal of an edge changes X'(G) from $\Delta + 1$ to Δ then it is an edge-critical edge.

Note that isProperColoring() method from GraphColoring class is used after every recoloring to make sure the new coloring is proper.

Program performs fairly fast overall, however, for graphs with very large number of edges, recoloring graph after removal of each edge took a long time.

4 Experiments and Results

4.1 Graphs for Experiment

Various graphs with different properties were used for testing and finding their coloring. Graphs including complete graphs, Petersen graph, generalized Petersen graph [10], tree, path, wheels, mid9, line graphs and antipodal graphs were used for testing. All these graph files were from our course. Generalized Petersen graphs and complete graphs were generated by code. Graphs shown in Table 1, were colored in Δ colors after initial coloring, clearly determining them to be of Class 1. Other graphs, as shown in Table 2, were colored in $\Delta + 1$ colors initially.

Graph
Path of 5 vertices
Tree from Assignment 1
Mid 7
Mid 9
Thomassen
ThreeKFour
Wheel 6
Wheel 6 Square Spoke
Non Ham 24
K4
K3,3
Petersen G(3, 1)
Petersen G(5, 1)
Petersen G(100, 1)
Petersen G(100, 2)
Petersen G(100, 3)
Petersen G(100, 5)
Petersen G(100, 6)

Table 1: Graphs colored in Δ colors.

Graph	**X'(G)**	**X'(G) changed by Recoloring**	**X'(G) after Recoloring**	**Number of Major Vertices**	**X'(G) change by Edge Removal**
Triangle	$\Delta + 1$	False	$\Delta + 1$	3	True
K5	$\Delta + 1$	False	$\Delta + 1$	5	False
K50	$\Delta + 1$	True	Δ	50	False
K100	$\Delta + 1$	False	$\Delta + 1$	100	False
Petersen	$\Delta + 1$	False	$\Delta + 1$	10	False
Petersen G(9,2)	$\Delta + 1$	False	$\Delta + 1$	18	False
Petersen G(9,4)	$\Delta + 1$	False	$\Delta + 1$	18	True
Petersen G(100,4)	$\Delta + 1$	False	$\Delta + 1$	200	True
L(L(K(6)))	$\Delta + 1$	False	$\Delta + 1$	8	True
L(L(L(L(K(6)))))	$\Delta + 1$	True	Δ	168	True
L(K(8))	$\Delta + 1$	True	Δ	28	True
L(L(K(8)))	$\Delta + 1$	True	Δ	168	True
L(Antipod(L(L(K(8)))))	$\Delta + 1$	True	Δ	2520	True
Antipod(L(L(K(8))))	$\Delta + 1$	True	Δ	168	True

Table 2: Graphs colored in $\Delta + 1$ colors.

4.2 Results

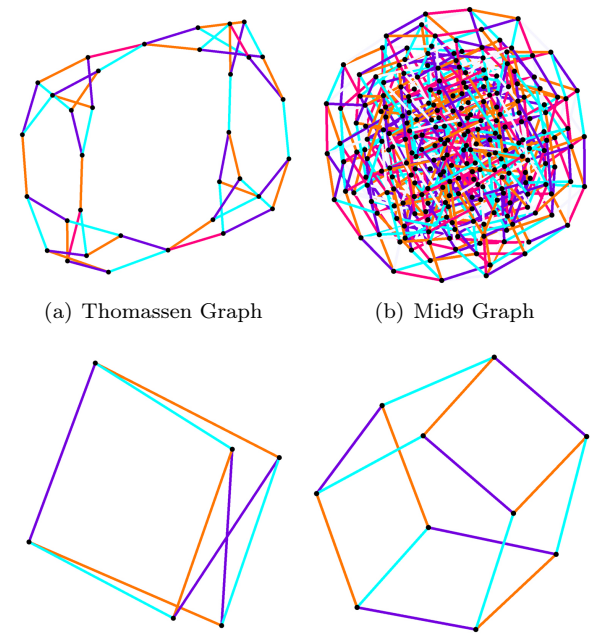
For graphs with $\Delta + 1$ colors, they were recolored 10 times in loop, keeping the previous coloring but removing the color which occurred minimum number of times. Secondly, each edge in these graphs was removed, one at a time, and graph was recolored to see if it changes coloring. If removing an edge changed coloring of a graph, then it satisfies X'(G - e) $<$ X'(G), meaning it is an edge-critical edge. X'(G) was reported after initial coloring and after recoloring. Graphs with $\Delta +$ 1 colors were also tested for corollary 2, 3, 4 and theorem 2. These are some properties of Class 2 graphs. These properties were computationally inexpensive to test and offered insight on whether

given graph contains some class 2 properties. None of the graphs satisfied theorem 2, whereas all of these graphs satisfied corollary 2, 3 and 4.

Graphs such as triangle, K5, K100, Petersen could not be recolored to Δ. It is very likely that these graphs are class 2 but results of these experiments alone cannot classify them as class 2. For concrete evidence, we need to test graph against all properties of class 2 graphs, which was outside of this project's scope. Project's focus was on trying to determine class of graph by recoloring it. Most interesting finding of this project was that graphs K50, L(K(8)), L(L(L(L(K(6))))), L(L(K(8))), L(Antipod(L(L(K(8))))) and Antipod(L(L(K(8)))) were successfully recolored in Δ. After every recoloring, it was also checked if the coloring was proper. These graphs had a proper Δ-edge-coloring. Hence, we can say that these graphs are of class 1.

4.3 Visualization of Edge Coloring

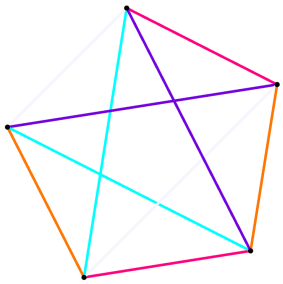
Below are some of the graphs visualized after they were colored in Δ or $\Delta + 1$ colors.



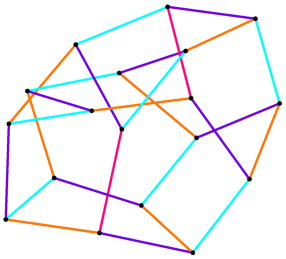
(a) Thomassen Graph (b) Mid9 Graph

(c) Complete Bipartite Graph K3,3 (d) Generalized Petersen Graph G(5,1)

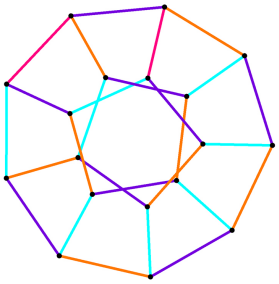
Figure 1: Graphs colored in Δ colors.



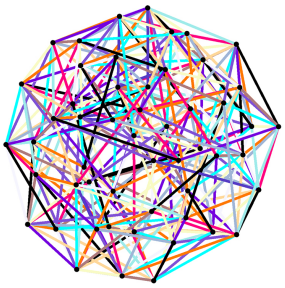
(a) Complete Graph K5



(b) Generalized Petersen Graph G(9,2)



(c) Generalized Petersen Graph G(9,4)

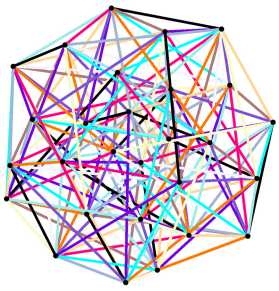


(d) L(L(K(6))) Graph

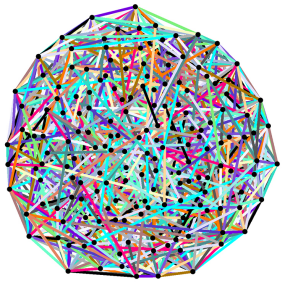
Figure 2: Graphs colored in $\Delta + 1$ colors.

Finally, graphs which were initially colored in $\Delta + 1$ are recolored in Δ colors and shown below. Visualization of L(L(L(L(K(6))))) and L(Antipod(L(L(K(8))))) are not included as they were too large.

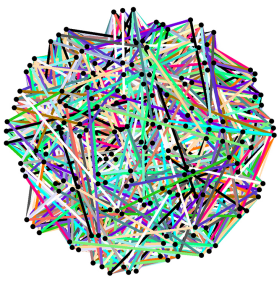
(a) Complete Graph K50



(b) L(K(8))



(c) L(L(K(8)))



(d) Antipod(L(L(K(8)))) Graph

Figure 3: Graphs recolored from $\Delta + 1$ to Δ colors.

5 Conclusion and Future Work

In conclusion, some graphs are very straightforward class 1 graphs and they can be colored in Δ colors. Graphs using $\Delta + 1$ are harder to determine and recoloring might color them to Δ colors. It was interesting to find some graphs which have a reduced X'(G) after recoloring. However, unless a graph can be colored in Δ colors, it is impossible to determine if it is class 2 based on recoloring alone. Further proof is needed in form of verifying properties of class 2 graphs with the respective graph. There are many more types of graphs other than the ones in this report and the algorithm, including recoloring, can be tested on them as well. Furthermore, a program can be coded to test them against class 2 properties.

6 References

[1] Stiebitz M., Scheide D., Toft B. and Favrholdt L. M., *Graph Edge Coloring*, Wiley Series in Discrete Mathematics and Optimization.
[2] Kocay, W., Kreher, *Graphs, Algorithms, and Optimization.* Chapman & Hall/CRC Press (2005)
[3] Walter Wallis, *A Beginner's Guide to Graph Theory*, Birkha user Boston, Boston, MA, 2007.
[4] J. Misra, D. Gries, *A constructive proof of Vizing's theorem*,Inf. Process. Lett., vol. 41, no. 3, pp. 131-133, 1992.
[5] C. Berge, J.C. Fournier, *A short proof for a generalization of Vizing's theorem* J. Graph Theory, 15 (1991), pp. 333-336.
[6] Dijkstra, E.W., and J.R. Rao. *Designing the proof of Vizing's algorithm.* EWD1082a, Computer Sciences, University of Texas at Austin, September. 1990. Vizing, V. G. (1965), *Critical graphs with given chromatic class*, Metody Diskret. Analiz., 5: 9–17.
[7] http://graphstream-project.org/
[8] http://www.cecill.info/licences/Licence_CeCILL-C_V1-en.html
[9] http://www.gnu.org/licenses/lgpl.html
[10] http://mathworld.wolfram.com/GeneralizedPetersenGraph.html