MATHEUS ARMANI MALDONADO

AN IMPLEMENTATION OF THE MISRA & GRIES EDGE COLORING ALGORITHM AND ITS INTEGRATION ON SAGEMATH

(*pre-defense version, compiled at March 31, 2023*)

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Renato Carmo.

CURITIBA PR

2023

RESUMO

Uma coloração das arestas de um grafo simples G consiste em atribuir uma cor a cada aresta em G de tal maneira que nenhum par de arestas que compartilhe um vértice tenha a mesma cor. O menor número de cores necessárias para colorir um grafo G é chamado de índice cromático de G ($\chi'(G)$), e o Teorema de Vizing postula que $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$. Determinar o índice cromático e computar uma coloração ótima para G são problemas $\mathcal{NP}$-difíceis. Neste trabalho, apresentamos uma implementação do algoritmo de Misra & Gries, um algoritmo polinomial para computar uma coloração de arestas usando até $\Delta + 1$ cores. O algoritmo é implementado em Python e integrado no ambiente de computação matemática SageMath, uma alternativa open-source para plataformas como Matlab, Maple e Magma. Fazemos uma descrição detalhada do algoritmo e estrutura de dados utilizados, uma análise assintótica do algoritmo e uma breve comparação de tempo de execução com a rotina utilizada para coloração de arestas previamente implementada no SageMath.

Palavras-chave: Teoria dos grafos, coloração de arestas, Teorema de Vizing

ABSTRACT

An edge coloring of a simple graph G consists of assigning a color to every edge in G in a way that no pair of edges that shares a vertex has the same color. The smallest amount of colors needed to color a graph G is called the chromatic index of G ($\chi'(G)$), and Vizing's theorem states that $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$. Computing the chromatic index and an optimal coloring for G are $\mathcal{NP}$-complete problems. We present an implementation of the Misra & Gries algorithm, a polynomial time algorithm that computes an edge coloring using at most $\Delta + 1$ colors. The algorithm is implemented in Python and integrated in SageMath, an open-source alternative to platforms such as Matlab, Maple and Magma. A detailed explanation of the algorithm and used data structure is presented, as well as an asymptotic analysis of the algorithm and a simple comparison of running time with the previously implemented routine used for edge coloring in SageMath.

Keywords: Graph theory, edge coloring, Vizing's Theorem

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

We discuss our implementation of the Misra & Gries (Misra and Gries, 1992) edge coloring algorithm, a polynomial algorithm capable of computing a $\Delta + 1$ coloring for any graph. This algorithm is implemented using Python and is integrated with mathematical computing software SageMath (The Sage Developers, 2023). We also provide a preliminary experimental run time comparison of the Misra & Gries implementation and the routine available for edge coloring previosly available on SageMath.

In this chapter, we introduce definitions related to graph theory and edge coloring, as well as an example of the edge coloring problem applied to file transfer scheduling. Chapter 2 provides an in-depth explanation of the Misra & Gries edge coloring algorithm. Chapter 3 presents an asymptotic analysis of the algorithm, as well as remarks on time complexity of the SageMath routine. Chapter 4 presents a run time speed comparison of both algorithms. Finally, Chapter 5 presents a conclusion and future improvement ideas.

1.1 DEFINITIONS

In order to understand the edge coloring problem, it is first necessary to be acquainted with some definitions related to graph theory and edge coloring, which are discussed in this section. The focus of this work is in simple graphs, which are collections of vertices and edges, an edge being an undirected connection between a pair of vertices (in simple graphs, a pair of vertices cannot be connected by more than one edge, a vertex cannot be connected to itself, and edges do not have weights). If there is an edge connecting a pair of vertices, these vertices are *neighbors*, and the number of neighbors of a vertex v is its *degree* ($\delta(v)$). The maximum degree of a vertex in graph G is denoted $\Delta(G)$.

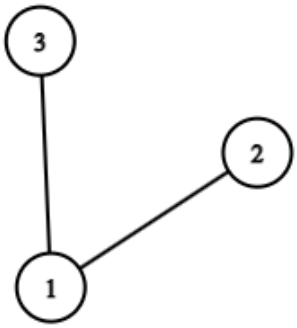

Figure 1.1: A simple graph with vertices 1, 2 and 3 and edges $\{1, 2\}$ and $\{1, 3\}$. $\delta(2) = 1$ and $\delta(1) = \Delta(G) = 2$

An edge coloring of a graph $G = (V, E)$ is the assignment of a color to each edge in E. If, for every vertex x in V, no two edges sharing x as an endpoint have the same color, G has a *proper coloring*. Note that, because of this constraint, it is possible to infer that the minimum amount of colors needed for a proper coloring of all the edges of G is $\Delta(G)$. Vizing's theorem (Vizing, 1964) states that every graph G can be colored using at most $\Delta(G) + 1$ colors, and proves this by describing an algorithm to color graphs using $\Delta + 1$ colors in polynomial time. The Misra & Gries edge coloring algorithm was devised with the intention of simplifying existing proofs of Vizing's theorem, and is also a procedure that shows how to color an uncolored edge of a graph, maintaining a proper coloring at every step, in polynomial time. Vizing's theorem naturally leads to a classification of graphs as Δ-colorable (Class 1) or $\Delta + 1$-colorable (Class 2). Deciding if a graph is Class 1 or Class 2 is an $\mathcal{NP}$-complete problem, as demonstrated by Holyer (1981).

1.2 EDGE COLORING APPLIED TO FILE TRANSFER SCHEDULING

The edge coloring problem is a widely studied subject in graph theory, and it has several applications, such as tournament scheduling (Januario et al., 2016), link scheduling in sensor networks (Gandham et al., 2008) and the open shop scheduling problem (Williamson et al., 1997). Another such application is the problem of scheduling file transfers in computer networks, introduced by Chung et al. (1987). This problem consists of, given a network of computers and a list of files to be transferred, calculating the shortest possible schedule for transferring all files, while respecting the maximum amount of simultaneous connections of each computer, as well as the time taken to transfer each file individually. We will use a simplified instance of this problem as an example, modeled as a graph $G = (V, E)$ where each vertex represents a computer v in the network and edges represent files to be transferred. The example instance is depicted in Figure 1.2 below:

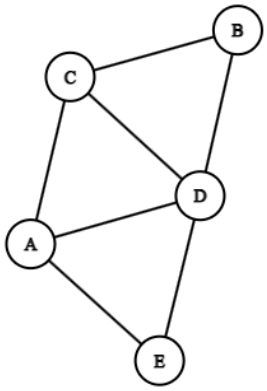

Figure 1.2: A graph model of a network with computers A to E. Edges represent files to be transferred

By finding a proper edge coloring to the graph, it is possible to compute the shortest time span needed to transfer all files. Every color represents a time slot, and the total number of different colors is the amount of time slots needed to transfer all files.

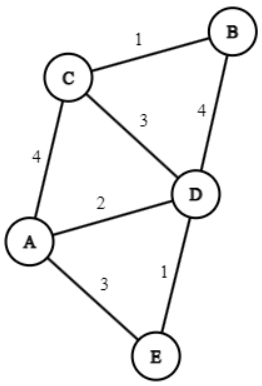

Figure 1.3: A graph model of a network with computers A to E and edges with colors 1 to 4

In the example we can color the graph with 4 different colors, and since $\Delta(G) = 4$, this coloring is optimal. This means that the smallest time window to transfer all files if the time slot were, for example, 1 second, would be 4 seconds.

With the Misra & Gries algorithm, it is possible to calculate a proper edge coloring with at most $\Delta + 1$ colors, which is optimal to some graphs and just 1 color beyond optimal to others, and we are able to do so in $O(|E||V|)$ time, as will be discussed in Chapter 3.

2 MISRA AND GRIES EDGE COLORING ALGORITHM

This chapter provides a detailed explanation on the algorithm implemented, the data structure used and every operation applied to the graph being colored. The algorithm serves as a proof of Vizing's theorem, and demonstrates how to color an uncolored edge, sometimes recoloring other edges to maintain a proper coloring. By repeating this process for all edges, it's possible to compute a proper edge coloring with at most $\Delta + 1$ colors.

2.1 THE FAN

The data structured used is called a 'fan'. It consists of a nonempty sequence $F[0..k]$ of distinct neighbors of a vertex v, where the edge $\{v, F[0]\}$ is uncolored, and the color of every edge $\{v, F[i+1]\}$ is free at vertex $F[i]$ (a color being free at a vertex means no edge adjacent to that vertex has that color). We number these constraints for future reference:

C1: edge $\{v, F[0]\}$ is uncolored and other edges of the fan must be colored;

C2: color of edge $\{v, F[i+1]\}$ is free at vertex $F[i]$, for every i in $[0..k-1]$.

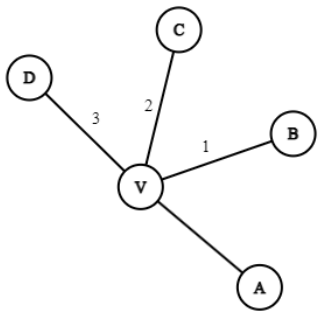

Figure 2.1: A fan F = [A,B,C,D]. 1 must be free at A, 2 at B and 3 at C.

The following pseudocode demonstrates the process of creating a fan with a central vertex v that is maximal, in that no remaining neighbor of v outside the fan is able to extend it.

```
fan = [] fan.append(f) # fan starts with uncolored edge {v, f}
last vertex = f # most recent addition to fan
neighbors # neighbor vertices of v
maximal = False
while not maximal:
   maximal = True
   for n in neighbors:
      if color of edge{v, n} not in colors adjacent to last vertex:
         fan.append(n)
         last vertex = n
         neighbors.remove(n)
         maximal = False
return fan
```

2.2 INVERTING THE CD-PATH OF V

The cd-path of v is a maximal path starting in v, of edges colored c or d in succession. Inverting the cd-path consists of recoloring edges in the path from c to d and d to c. This operation is done in order to make the color d free at vertex v, so that d can be attributed to some edge adjacent to v.

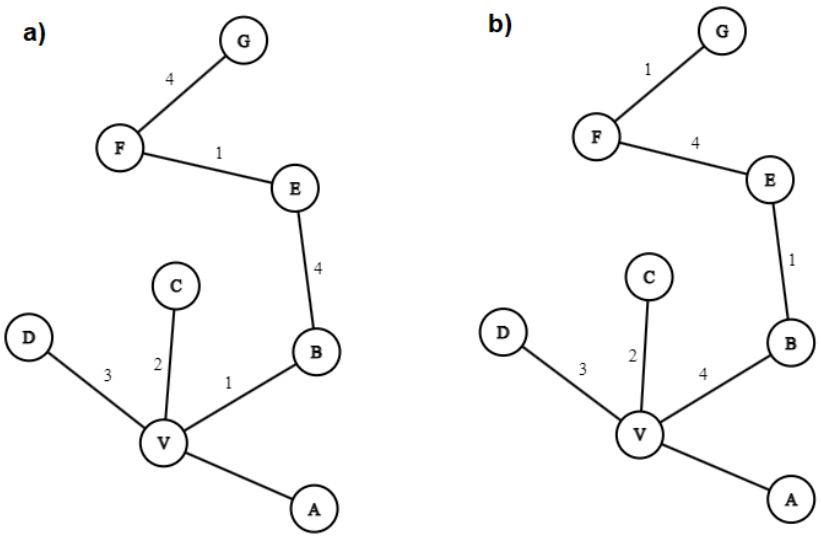

Figure 2.2: A fan with a cd-path where $c = 4$ and $d = 1$, a) before and b) after the inversion.

This operation maintains the coloring proper, since colors outside of the cd-path are unchanged, colors adjacent to vertices inside the cd-path also remain the same, since only the colors of c-edges and d-edges are swapped, and the new colors at the endpoints of the path were previously free, since the path is maximal. After the inversion, there is a vertex w in F[0..k] such that F[0..w] is a fan and the color d is free at w. We can prove this by analyzing two possible cases prior to the inversion:

- **No fan edge has color d:** if the fan $F[0..k]$ is maximal and d is free at k, there is no edge with color d incident on v (if there was such an edge, it would be possible to extend the fan with said edge, and since the fan is maximal, no edge with color d incident to v must exist). Since c is also free at v, this means that the cd-path is empty and the inversion has no effect in the graph. In this case, w = k
- **A fan edge $u + 1$ has color d:** by C2, the color d is free at some vertex u. C2 still holds after the inversion for vertices other than $u + 1$, since their colors are neither c nor d and inversion only alters colors c and d. If u does not belong to the cd-path, colors incident to u are not changed by inversion and d remains free at u. In this case w = u and the fan becomes $F[0..u]$. If u is in the cd-path, d was free at u before the inversion, which means u is an endpoint of the cd-path, and the inversion changes the color of $\{v, u + 1\}$ from d to c and makes c free at u. This, coupled with the previous statement about C2, means that C2 still holds after the inversion, thus $F[0..k]$ remains a fan, and d is still free at k (k could only be an endpoint of the cd-path since d is free at k, but the endpoints of the path are v and u, therefore k does not belong to the cd-path and its' colors are unchanged by the inversion). In this case, w = k.

2.3 ROTATING THE FAN

The rotate fan operation, for a fan $F[0..k]$ with central vertex v, consists of assigning the color of edge $\{v, i + 1\}$ to edge $\{v, i\}$ in parallel, for every i in $[0...k - 1]$ and uncoloring $\{v, k\}$. Note that the coloring remains proper after the rotation because the colors outside the fan remain unchanged, and by C2, every new color adjacent to $F[i]$ was free beforehand. After rotating the fan, edge $\{v, F[0]\}$ is now colored and $\{v, F[k]\}$ is uncolored, therefore the number of colored edges is unchanged, and the previous color of $\{v, k\}$ is now free at k.

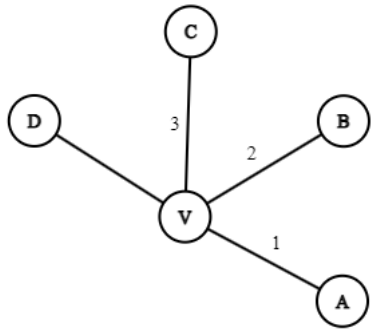

Figure 2.3: The fan from Figure 2.1 after rotation.

2.4 THE ALGORITHM

The following pseudo-code illustrates how the algorithm functions, and is very similar to the code implemented in SageMath:

```
for every uncolored edge {v, y} in G:
      f = maximal fan f[y..k] with center v
      c = a free color on vertex v
      d = a free color on vertex k
      invert cd-path of v
      f = f[y..w]
      rotate fan f
      colors {v, w} with color d
```

Note that the edge coloring is proper, as the graph starts with no colored edges, for every iteration a new graph edge is colored, and every step maintains a proper coloring. The algorithm also requires at most $\Delta + 1$ colors: at each iteration only colors c and d are used. At least one color in $[1..\Delta]$ will be free at v to be used as c, as it has at least one uncolored edge and $\delta(v) \leq \Delta$. Vertex k may have degree Δ and every adjacent edge colored, therefore an additional color $\Delta + 1$ may be used.

3 ASYMPTOTIC ANALYSIS

It is important to note that the Misra & Gries algorithm was developed with the sole intention of being easier to understand and more thorough in its explanation than another existing proof of Vizing's Theorem (Bollobás, 1979) and thus provides no asymptotic improvement over Vizing's proof. The differences between Misra & Gries' and Bollobás' proofs are the addition of nomenclature (fan, rotation and cd-path), the inclusion of arguments that are omitted in Bollobás' proof, the usage of cd-paths instead of subgraphs and the order of rotation and inversion of the fan.

3.1 MISRA AND GRIES ALGORITHM

The process of coloring an edge $\{x, y\}$ in the algorithm can be broken down into the following steps:

- Finding a maximal fan $F[0..k]$ of x
- Finding colors c and d, free at x and $F[k]$ respectively
- Inverting the cd-path
- Finding w in $F[0..k]$ where d is free on w
- Rotating the fan $F[0..w]$ and coloring $\{x, w\}$ with d

This process is executed for every edge in $G = (V, E)$, which means this is a loop executed $O(|E|)$ times. Finding a maximal fan of x takes $O(\Delta(G))$ time in the worst case, since a fan consists of neighbors of a vertex v and in the worst case traversing all neigbors of v would be needed. The same idea applies to finding colors c and d, as well as finding w in $F[0..k]$, as the maximum amount of incident colors on a vertex is at most Δ . A cd-path could span every vertex of the graph, therefore the worst case time of inverting the cd-path is $O(|V|)$. The time taken rotating the fan is bound by the degree of the center vertex of the fan $\delta(x) \leq \Delta(G)$ and coloring the edge $\{x, w\}$ can be done in constant time. Thus, the time complexity of the algorithm is $O(|E||V|)$.

3.2 INTEGRATION OF THE ALGORITHM WITH SAGEMATH

Even though the time complexity of Misra & Gries' algorithm is $O(|E||V|)$, the same time complexity was not achieved in the integration with SageMath. This is due to the fact that, in order to achieve $O(\Delta(G))$ time complexity while computing a maximal fan of x, it is necessary to have some data structure that, for every edge $\{u, v\}$ in G, tracks every other edge $\{u, y\}$ that

can extend a fan F of u if $\{u, v\}$ is the last edge in F, for every possible central vertex u, while being able to fetch $\{u, y\}$ in $O(1)$ time. This increases the space complexity of the algorithm, but reduces the time complexity to $O(\Delta(G))$. This data structure was not created in the integration with SageMath; instead, the process by which maximal fans are created is very similar to the pseudo-code in Section 2.1, and its time complexity is $O(|V|^2)$, thus increasing the overall time complexity of the integration to $O(|E||V|^2)$. Time complexities for other steps of the algorithm is achieved by using standard Python data structures.

3.3 SAGEMATH SOLUTION TO THE EDGE COLORING PROBLEM

SageMath is a mathematics software created as an open-source alternative to Magma, Maple, Mathematica and Matlab[1]. It is built on top of many open-source packages, and has interfaces for a wide range of mathematical applications. The previously implemented edge coloring routine used Mixed Integer Linear Programming (MILP). MILP is a generalization of Linear Programming (LP) which is a mathematical technique used to solve problems that involve optimizing a linear objective function that is subject to linear constraints. The difference between LP and MILP is that it is possible to restrict some of the variables in MILP to integers.

The function responsible for coloring edges has the signature `edge_coloring(g, value_only=False, vizing=False, hex_colors=False, solver=None, verbose=0, integrality_tolerance=0.001)` where g is the graph to be colored; `value_only` is a flag that, if set to true, returns only the chromatic index of g instead of the edge coloring; `vizing` is a flag that, if set to true, would not try to calculate a Δ coloring of g, only a $\Delta + 1$ coloring. `hex_colors` defines if the dictionary's keys output by the function are hexadecimal colors; `solver` specifies which MILP solver the function should use (defaults to GLPK); `verbose` defines the level of verbosity of the output; and finally `integrality_tolerance` is a setting for the MILP solver, if the variable values differ from the nearest integer by more than this value, the solver raises an exception.

The routine models an edge coloring problem into a MILP problem by using colors as variables, and adding the constraints that two edges adjacent to the same vertex cannot have the same color, and that every edge must have a color. If the user opts for solving using Δ colors (`vizing=False`) and this is not possible, a $\Delta + 1$ coloring is output instead. This was done for every connected component H of G, except if H was a clique, in this case an $O(|V|^2)$ time algorithm is used.

MILP is an $\mathcal{NP}$-hard problem (Conforti et al., 2014), but there is no exact expression for the time complexity of edge coloring modeled as an MILP problem, as it can vary greatly depending on constraints, structure and size of the problem, as well as the solving algorithm being used. Moreover, SageMath provides several different MILP solvers, each with customizable

[1] Available at https://sagemath.org

parameters. For these reasons, a comparison of running time will be made in the following chapter.

4 RUN TIME COMPARISON

For the comparison, the function RandomGNP of SageMath was used to generate two sets of graphs: Set 1 contains graphs with 15 vertices, where every edge starts with a 5% probability of existing, increasing by 5% for each graph, and ending with 95% probability. Set 2 contains graphs with 5 to 29 vertices, where every edge has a 50% probability of existing. Different versions of SageMath had to be used for testing: 9.8 for Misra & Gries and 9.7 for MILP, since the integration of the Misra & Gries algorithm handles every $\Delta + 1$ coloring from 9.8 onwards, and it would not be possible to test the performance of MILP on $\Delta + 1$ colorings at version 9.8. Running times were measured using the `timeit` function of Python. The results are displayed on Tables 4.1 and 4.2, as well as Figures 4.1 and 4.2.

The time growth rate is not linear for the MILP algorithm because of the auxiliary algorithm used for coloring cliques inside the graph in polynomial time, as well as the general structure of the graph changing the constraints of the MILP problem, which can affect the running time for graphs individually. The Misra & Gries algorithm outperforms MILP in every instance, more noticeably for graphs with greater number of edges, as expected.

Edge %	Misra & Gries (ms)	MILP (ms)
0.05	0.1	1.3
0.10	0.2	1.6
0.15	0.3	3.0
0.20	0.3	4.8
0.25	0.4	3.7
0.30	0.9	4.6
0.35	0.9	6.2
0.40	0.9	7.6
0.45	1.5	8.5
0.50	1.6	15.0
0.55	1.3	11.0
0.60	2.8	13.8
0.65	3.0	17.7
0.70	2.6	148.0
0.75	3.0	19.7
0.80	2.6	235.0
0.85	3.7	22.6
0.90	4.2	695.0
0.95	4.5	700.0

Table 4.1: Time taken to find a $\Delta + 1$ coloring for graphs in Set 1

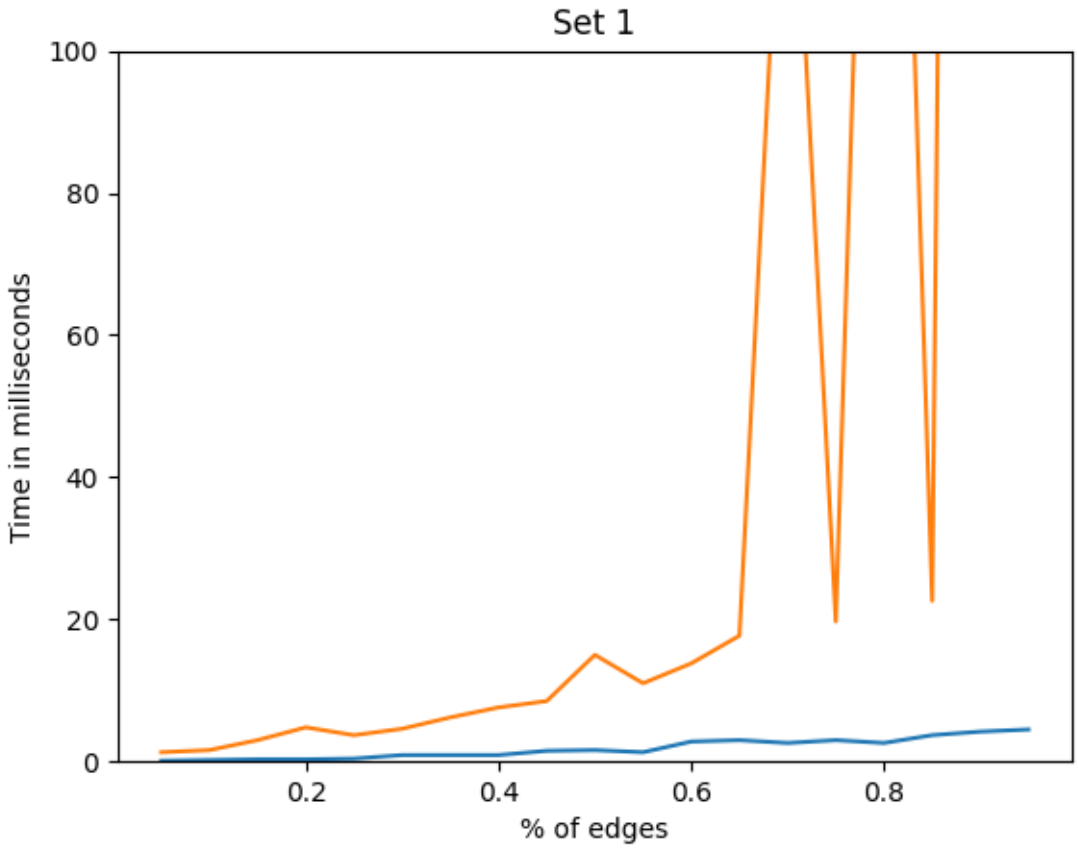

Figure 4.1: Time taken to find a $\Delta + 1$ coloring for graphs in Set 1

Vertices	Misra & Gries (ms)	MILP (ms)
5	0.1	0.7
6	0.1	1.0
7	0.1	2.1
8	0.4	2.3
9	0.3	2.7
10	0.4	3.8
11	0.6	5.1
12	0.9	5.7
13	0.6	7.4
14	1.4	7.2
15	1.7	11.3
16	1.8	15.6
17	2.5	14.6
18	2.0	15.4
19	2.2	25.1
20	3.0	26.4
21	4.2	32.7
22	3.8	42.1
23	5.8	37.3
24	5.2	47.6
25	6.1	58.6
26	7.4	79.1
27	7.2	72.1
28	7.4	89.2
29	9.5	7140.0

Table 4.2: Time taken to find a $\Delta + 1$ coloring for graphs in Set 2

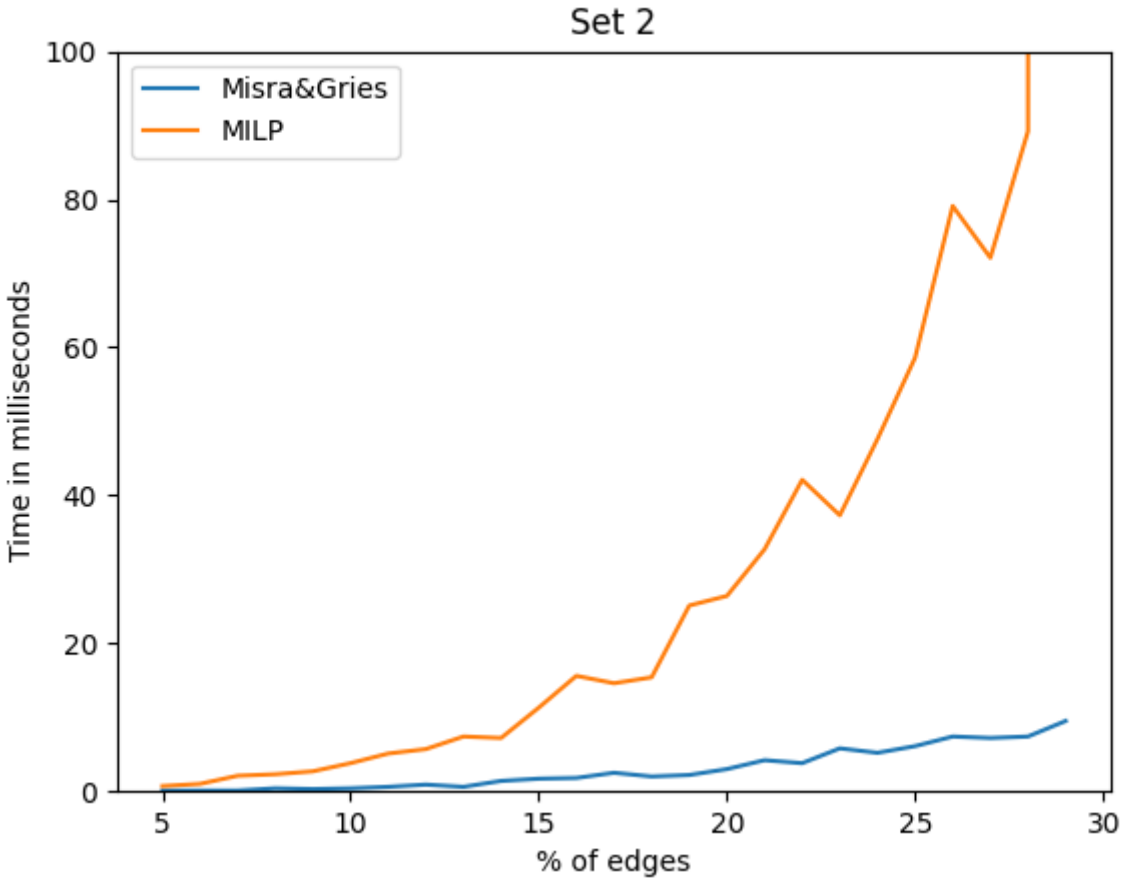

Figure 4.2: Time taken to find a $\Delta + 1$ coloring for graphs in Set 2

5 CONCLUSION

It was shown that polynomial time algorithms for edge coloring simple graphs are important in order to calculate colorings of large simple graphs in a feasible time, if the tradeoff of possibly using an extra color can be tolerated. The Misra & Gries edge coloring algorithm was implemented in SageMath, an open-source mathematics software, and provided a significant increase in performance, compared to the previously implemented MILP algorithm for coloring graphs with $\Delta + 1$ colors. As of version 9.8 of SageMath, the MILP algorithm handles colorings with Δ colors and the Misra & Gries algorithm deals with $\Delta + 1$ colorings.

5.1 FURTHER IMPROVEMENTS

Further improvements can be made related to the edge coloring algorithm in SageMath, particularly with respect to classes of graphs that are known to be Class 1 and have polynomial time algorithms capable of calculating an optimal edge coloring. Such classes and algorithms are discussed in Zatesko (2018) and would provide an even greater improvement in performance if implemented in SageMath.

REFERENCES

Bollobás, B. (1979). *Graph theory: an introductory course*, volume 63. Springer Science & Business Media.

Chung, F., Coffman, E., Reiman, M., and Simon, B. (1987). The forwarding index of communication networks. *IEEE Transactions on Information theory*, 33(2):224–232.

Conforti, M., Cornuéjols, G., Zambelli, G., et al. (2014). *Integer programming*, volume 271. Springer.

Gandham, S., Dawande, M., and Prakash, R. (2008). Link scheduling in wireless sensor networks: Distributed edge-coloring revisited. *Journal of Parallel and Distributed Computing*, 68(8):1122–1134.

Holyer, I. (1981). The np-completeness of edge-coloring. *SIAM Journal on computing*, 10(4):718–720.

Januario, T., Urrutia, S., Ribeiro, C. C., and De Werra, D. (2016). Edge coloring: A natural model for sports scheduling. *European Journal of Operational Research*, 254(1):1–8.

Misra, J. and Gries, D. (1992). A constructive proof of vizing's theorem. *Information Processing Letters*, 41(3):131–133.

The Sage Developers (2023). *SageMath, the Sage Mathematics Software System (Version 9.8)*. `https://www.sagemath.org`.

Vizing, V. G. (1964). On an estimate of the chromatic class of a p-graph. *Diskret analiz*, 3:25–30.

Williamson, D. P., Hall, L. A., Hoogeveen, J. A., Hurkens, C. A., Lenstra, J. K., Sevast'janov, S. V., and Shmoys, D. B. (1997). Short shop schedules. *Operations Research*, 45(2):288–294.

Zatesko, L. M. (2018). Novel procedures for graph edge-colouring.