THE LINEAR ARBORICITY CONJECTURE FOR 3-DEGENERATE GRAPHS

Thesis *Submitted in partial fulfillment of the requirements for the degree of*

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING - INFORMATION SECURITY

by

Aman Sonkar

(Reg. No. 192IS004)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL

MANGALORE - 575025

JULY 2021

The Linear Arboricity Conjecture for 3-degenerate graphs

by

Aman Sonkar
(Reg. No. 192IS004)

Under the Guidance of

Dr. Manu Basavaraju
Associate Professor
Department of CSE

THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING - INFORMATION SECURITY

Department of Computer Science and Engineering
National Institute of Technology Karnataka, Surathkal
Mangalore - 575025

July 2021

DECLARATION

I hereby *declare* that the Thesis entitled **The Linear Arboricity Conjecture for 3-degenerate graphs** which is being submitted to **National Institute of Technology Karnataka, Surathkal**, in partial fulfilment for the requirements of the award of degree of **Master of Technology** in **Computer Science and Engineering (Information Security)** in the department of **Computer Science and Engineering**, is a *bonafide report of the work carried out by me*. The material contained in this report has not been submitted at any other University or Institution for the award of any degree.

192671 IS 004, Aman Sonkar

Aman Sonkar

.....

(Register Number, Name and Signature of the student)
Department of Computer Science and Engineering

Place : NITK, Surathkal
Date : 15 July 2021

CERTIFICATE

This is to *certify* that the Thesis entitled **The Linear Arboricity Conjecture for 3-degenerate graphs** submitted by **Aman Sonkar** (Registration number: 192671 IS 004) as the record of the work carried out by him, is *accepted as the P.G. Major Project Thesis submission* in partial fulfilment for the requirements of the award of degree of **Master of Technology** in **Computer Science and Engineering (Information Security)** in the department of **Computer Science and Engineering** at **National Institute of Technology Karnataka, Surathkal** during the academic year 2020-21.

.....

Dr. Manu Basavaraju

Project Guide

Department of CSE,

NITK Surathkal

.....

Dr. Shashidhar G Koolagudi

Chairman-DPGC

Department of CSE,

NITK Surathkal

ACKNOWLEDGEMENT

I take this opportunity to express my deepest gratitude and appreciation to all those who have helped me directly or indirectly towards the successful completion of this project.

I would like to express my gratitude to my guide Dr. Manu Basavaraju, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal for their continuous encouragement and support in carrying out my project work. They have continuously mentored my work and corrected my mistakes, without which it would have been difficult to carry out this project. I would like to thank all of them for all the help and necessary suggestions that they have always provided me with.

I would also like to thank all the supporting staff members of the department of Computer Science and Engineering. Without their timely help and support, it would not have been possible to carry out my project.

Place: Surathkal Aman Sonkar

Date: 15 July 2021

Abstract

A *k-linear coloring* of a graph G is an edge coloring of G with k colors so that each color class forms a linear forest−a forest whose each connected component is a path. The *linear arboricity* $\chi_l'(G)$ of G is the minimum integer k such that there exists a k-linear coloring of G. Akiyama, Exoo and Harary conjectured in 1980 that for every graph G, $\chi_l'(G) \leq \lceil(\Delta(G)+1)/2\rceil$ where $\Delta(G)$ is the maximum degree of G. In this thesis we implement an O(n)-time algorithm that partitions the edge set of any 3-degenerate graph G on n vertices into at most $\lceil(\Delta(G)+1)/2\rceil$ linear forests. Since $\chi_l'(G) \geq \lceil\Delta(G)/2\rceil$ for any graph G, the partition produced by the algorithm differs in size from the optimum by at most an additive factor of 1.

Keywords: forest , graph, linear arboricity, k-linear coloring, 3-degenerate graph

Contents

List of Figures

Chapter 1

Introduction

All graphs considered here are finite, simple and undirected, i.e., have no loops and no multiple edges, unless otherwise specified. For a graph G, we let $V(G)$ and $E(G)$ denote its vertex set and edge set, respectively. The *neighborhood* of a vertex u in G, denoted by $N_G(u)$, is the set $\{v : uv \in E(G)\}$. We abbreviate it to just N(u) when the graph G is clear from the context. Given a graph G, the degree of a vertex $u \in V(G)$ is |N(u)| and is denoted by $d_G(u)$. The maximum degree of a graph G, denoted by $\Delta(G)$, is defined to be $\max\{d_G(u) : u \in V(G)\}$. When the graph G under consideration is clear, we sometimes abbreviate $\Delta(G)$ to just Δ.

Given a graph G on n vertices and an integer t, an ordering $v_1, v_2, ..., v_n$ of the vertices of G such that for each $1 \leq i \leq n$, $|\{v_j : v_j \in N(v_i) \text{ and } j > i\}| \leq t$ is called a t-degeneracy ordering of G. A graph G is said to be t-degenerate if it has a t-degeneracy ordering. Equivalently, a graph G is t-degenerate if every subgraph of G has minimum degree at most t.

An edge coloring of a graph G using the colors $\{1, 2, ..., k\}$ is a mapping $c : E(G) \rightarrow \{1, 2, ..., k\}$. Given an edge coloring using colors {1, 2, . . . , k}, the color class i, for some $i \in 1, 2, ..., k$, is the set of edges $c^{-1}(i) = \{e \in E(G) : c(e) = i\}$.

The concept of decomposing a graph into the minimum number of trees or forests is introduced by Nash-Williams and Tutte. In 1961, Tutte and Nash-Williams gave a condition for when a graph could be decomposed into k forests. The arboricity of a graph G is defined to be the least number of disjoint forests whose union covers the edge set of G. Nash-Williams [15] showed this number to be:

$$k = max \left\lceil \frac{E(H)}{V(H) - 1} \right\rceil$$

where the maximum is taken over all subgraphs H on at least two vertices.

A linear forest is a forest in which every connected component is a path. Given a graph G, we define its linear arboricity, denoted by la(G), to be the minimum number of edge-disjoint linear forests in G whose union is the set of all edges of G, denoted by E(G). This notion was introduced by Harary [13] in 1970 as one of the covering invariants of graphs, and has been studied quite extensively since then.

Graphs without cycles are known as forests. The arboricity of a graph is the minimum integer k such that its edge set can be partitioned into k forests. A linear forest is a forest whose each connected component is a path. The linear arboricity of a graph is the minimum number of linear forests into which its edge set can be partitioned. A covering by linear forests can be viewed as an edge coloring where each color class induces a linear forest. Thus viewed as a edge coloring problem.

A k-linear coloring of a graph G is an edge coloring of G such that each color class is a linear forest. Or in other words, it is an edge coloring in which every vertex has at most two edges of the same color incident with it and there is no cycle in the graph whose edges all receive the same color. The linear arboricity of a graph G is clearly the smallest integer k such that it has a k-linear coloring and is denoted by χ'(G). The parameter χ'(G) was introduced by Harary [13].

It is immediate that $la(G) \leq |E(G)|$ as every edge uv (along with the isolated vertices V(G)/ u, v) forms a linear forest. A less trivial upper bound can be obtained as follows: by a classical theorem due to Vizing, E(G) can be partitioned into at most $\Delta + 1$ matchings, where $\Delta := \Delta$(G) denotes the maximum degree of G; observe that each matching is a linear forest, and therefore we get that $la(G) \leq \Delta + 1$. For a lower bound, note that every Δ-regular graph G on n vertices has $n\Delta/2$ edges, and every linear forest has at most $n-1$ edges (and equality holds if and only if the linear forest is a Hamiltonian path). Therefore, if G is a $\Delta-$regular graph, then

$$la(G) \geq \frac{|E(G)|}{(n-1)} \geq \frac{n\Delta}{2(n-1)} > \frac{\Delta}{2},$$

which implies (recall that la(G) is an integer) that $la(G) \geq \lceil (\Delta+1)/2 \rceil$.

Akiyama, Exoo and Harary [1] proposed a conjecture, known as the Linear Arboricity Conjecture, stating that for every Δ-regular graph G, la(G) = $\left\lceil \frac{\Delta+1}{2} \right\rceil$. It is easy to see as shown in [3] that this conjecture is equivalent to the following:

The Linear Arboricity Conjecture (LAC) *For every graph G with maximum degree* Δ ,

$$\left\lceil \frac{\Delta}{2} \right\rceil \leq la(G) \leq \left\lceil \frac{\Delta+1}{2} \right\rceil$$

The lower bound is easily obtained since at least $\lceil \frac{\Delta}{2} \rceil$ linear forests are needed to cover all edges incident with a vertex with degree Δ. However, despite much effort, the conjecture for the upper bound is still open. Here, The linear arboricity of every Δ-regular graph is $\lceil \frac{\Delta+1}{2} \rceil$. Note also that since every graph G with maximum degree Δ is a subgraph of a Δ-regular graph (which may have more vertices, as well as more edges than G), the linear arboricity conjecture is equivalent to the statement that the linear arboricity of every graph G with maximum degree Δ is at most $\lceil \frac{\Delta+1}{2} \rceil$.

Chapter 2

Literature Review

2.1 Related works

Although the linear arboricity conjecture received a considerable amount of attention, it has been proved only for several special cases: complete graphs [16], complete bipartite graphs [1], series parallel graphs [18] and planar graphs [17, 19]. It is also proved that the LAC is true when Δ = 3, 4, 5, 6, 8, 10 (see [1, 2, 10, 11]). For general graphs G, the best known upper bound of $la(G)$, due to Guldan [12], is $\lceil \frac{3\Delta}{5} \rceil$ for even Δ and $\lceil \frac{3\Delta+2}{5} \rceil$ for odd Δ.

Although arboricity can be computed in polynomial time, computing linear arboricity is NP-hard. Even recognizing the graphs of linear arboricity two is NP-complete [6]. However, for cubic graphs and other graphs of maximum degree three, the linear arboricity is always two [2], and a decomposition into two linear forests can be found in linear time using an algorithm based on depth-first search [9].

As $la(G) \geq \lceil \Delta/2 \rceil$ for any graph G, a 2-factor approximation algorithm for computing linear arboricity can be obtained using Vizing's Theorem. Cygan et al. [8] showed an O(n log n) algorithm that produces a linear coloring of every planar graph on n vertices with the optimum number of colors when Δ(G) ≥ 9. Linear arboricity has applications in File Retrieval Systems [14].

It is known that the conjecture is true for 2-degenerate graphs from the fact that the acyclic chromatic index of 2-degenerate graphs is at most $\Delta + 1$ [5]. (The acyclic chromatic index $\chi_a^{'}$(G) of a graph G is the minimum number of colors required to properly color the edges of G — i.e. no two incident edges get the same color — such that the union of any two color classes is a forest. Since the union of any two color classes in such a coloring will always be a linear forest, we get that χ_a^{1}(G) $\leq \lceil \chi_a^{'}(G)/2 \rceil$

We prove the following theorem which shows that the linear arboricity conjecture is true for 3-degenerate graphs[4].

Theorem 1.[4] *Let G be a 3-degenerate graph having* $\Delta(G) \leq 2k-1$*, where k is a positive integer. Then* $\chi_l^{'}(G) \leq k$.

Our proof also serves as an alternative (we believe, simpler) proof for the the result of Akiyama, Exoo and Harary [1] that every cubic graph has a 2-linear coloring. Graphs having treewidth at most 3, also called partial 3-trees, are 3-degenerate graphs, and hence our result establishes the linear arboricity conjecture for this class of graphs. Our result provides an alternative proof for the conjecture for some other classes of 3-degenerate graphs like triangle-free planar graphs and Halin graphs.

2.2 Notations and Preliminaries

Given a graph G and a set S $\subseteq$ V (G), we denote by G - S the graph obtained by removing the vertices in S from G, i.e. V (G - S) = V (G) \S and E(G - S) = E(G) $\setminus\{uv : u \in S\}$. When S $\subseteq$ E(G), we abuse notation to let G - S denote the graph obtained by removing the edges in S from G; i.e. V (G - S) = V (G) and E(G - S) = E(G) \S. In both cases, if S = {s}, we sometimes denote G - S by just G - s.

Let G be a t-degenerate graph. A pivot in G is a vertex that has at most t neighbors of degree more than t. A pivot edge in G is an edge between a pivot and a vertex with degree at most t.

Observation 1 *Every t-degenerate graph G has at least one pivot edge.*

Proof. If $\Delta(G) \leq t$, then every vertex of G is a pivot, and every edge of G is a pivot edge. If $\Delta(G) > t$, then the graph $G' = G - \{u : d_{\mathrm{G}}(u) \leq t\}$ contains at least one vertex. Since G' is also t-degenerate (as every subgraph of a t-degenerate graph is also t-degenerate), there exists a vertex v $\in$ V (G′) such that d'_{G} (v) $\leq$ t. It can be seen that $\{u \in N_{\mathrm{G}}(v) : d_{\mathrm{G}}(u) > t\}$ = N'_{G} (v). As $\left|\mathrm{N}'_{\mathrm{G}}(\mathrm{v})\right| = \mathrm{d}'_{\mathrm{G}}$ (v) $\leq$ t, we have that *v* is a pivot in G. Also, as $d_{\mathrm{G}}(v) > t$, there exists $u \in N_{\mathrm{G}}(v)$ having $d_{\mathrm{G}}(u) \leq t$. Then *uv* is a pivot edge in G.

Alternatively, given a t-degeneracy ordering of a graph G, consider the first vertex v such that it has a neighbor u before it in the ordering. It is not difficult to see that uv is a pivot edge of G.

The following observation about linear forests is easy to see.

Observation 2 *Let H be a linear forest and let P1 and P2 be two paths in H having end*

vertices u1, v1 and u2, v2 respectively such that u1 ≠ v1, u2 ≠ v2, u1, v1 ≠ u2, v2 and V (P1) ∩ V (P2) ≠ ϕ. Then at least one of u1, v1, u2, v2 has degree 2 in H.

Identification of vertices: Given a graph G and vertices $u, v \in V(G)$ such that $uv \notin E(G)$ and $N(u) \cap N(v) = \phi$, we let $G/(u,v)$ denote the graph obtained by "identifying" the vertex v with u. That is, $V(G/(u,v)) = V(G)\backslash\{v\}$ and $E(G/(u,v)) = E(G-v) \cup \{ux : x \in N(v)\}$. Note that given a k-linear coloring of $G/(u,v)$, the vertex u can be "split back" into the vertices u and v so as to obtain the graph G together with a k-linear coloring of it. The following observation states this fact.

Observation 3 *Let G be a graph and $u, v \in V(G)$ such that $uv \notin E(G)$ and $N(u) \cap N(v) = \phi$. If c is a k-linear coloring of G/(u, v), then*

$$c'(e) = \begin{cases} c(e) & \text{if } e \text{ is not incident with } v \\ c(ux) & \text{if } e = vx \end{cases}$$

is a k-linear coloring of G.

Definition 1. *Let c be a k-linear coloring of a graph G. For a vertex x ∈ V (G), we define Colors(x) to be the set of colors in {1, 2, . . . , k} that appear on the edges incident with x. Further, we define Missing(x) to be the colors in {1, 2, . . . , k} that do not appear on any edge incident with x, Twice(x) to be the set of colors that appear on two edges incident with x, and Once(x) to be the set of colors that appear on exactly one edge incident with x.*

Note that for any vertex $x \in V(G)$, $\big|\text{Missing(x)}\big| + \big|\text{Once(x)}\big| + \big|\text{Twice(x)}\big| = \text{k}$ and also that the degree of x in G is $\big|\text{Once(x)}\big| + 2\big|\text{Twice(x)}\big|$.

2.3 Proof of Theorem 1

We prove the theorem using induction on $\big|\text{E(G)}\big|$. We assume that all 3-degenerate graphs having maximum degree at most $2k-1$ and less than $\big|\text{E(G)}\big|$ edges can be linearly colored using k colors. We shall show that G can be linearly colored using k colors.

Let uv be a pivot edge of G such that $d_{\text{G}}(u) \leq 3$ and v is a pivot.

Lemma 1. *If $d_G(v) < 2k-1$, then G has a k-linear coloring.*

Proof. Let $H = G - uv$. By the induction hypothesis, there is a k-linear coloring c of H.

Claim. If either

1. $\left|\text{Once(v)}\right| \geq 3$, or

2. $\left|Missing(v)\right| \geq 1$ and $\left|Missing(v) \cup Once(v)\right| \geq 2$,

then G has a k-linear coloring.

First, suppose that $\left|Once(v)\right| \geq 3$. Since $d_{\text{H}}(u) \leq 2$, we have $\left|Colors(u)\right| \leq 2$, and therefore there exists a color $i \in Once(v) \backslash Colors(u)$. Then, by coloring uv with i, we can get a k-linear coloring of G, and we are done. Next, suppose that $\left|Missing(v)\right| \geq 1$ and $\left|Missing(v) \cup Once(v)\right| \geq 2$. Then there exist $i \in Missing(v)$ and $j \in (Missing(v) \cup Once(v)) \backslash \{i\}$. If $j \notin Colors(u)$, then we can color uv with j to obtain a k-linear coloring of G. Otherwise, since $\left|Colors(u)\right| \leq 2$, we have $i \notin Twice(u)$, implying that we can color uv with i to obtain a k-linear coloring of G. This proves the claim.

Observe that $d_H(v) = 2\left|Twice(v)\right| + \left|Once(v)\right|$. From the above claim, if $\left|Missing(v)\right| + \left|Once(v)\right| \geq 3$, then we are done. Therefore, we shall assume that $\left|Missing(v)\right| + \left|Once(v)\right| \leq 2$. As $\left|Twice(v)\right| + \left|Once(v)\right| + \left|Missing(v)\right| = k$, this means that $\left|Twice(v)\right| \geq k-2$. Since $d(v) \leq 2k-2$, we have $d_{\text{H}}(v) \leq 2k-3$, implying that $\left|Twice(v)\right| \leq k-2$. Thus, we have $\left|Twice(v)\right| = k-2$. Since $2k-3 \geq d_{\text{H}}(v) = 2\left|Twice(v)\right| + \left|Once(v)\right|$, we get $\left|Once(v)\right| \leq 1$. Since $\left|Twice(v)\right| + \left|Once(v)\right| + \left|Missing(v)\right| = k$, we have $\left|Missing(v)\right| \geq 1$ and $\left|Missing(v)\right| + \left|Once(v)\right| = 2$. We are now done by the above claim.

By the above lemma, we shall assume from here onwards that $d_{\text{G}}(v) = 2k-1$. Also, we can assume that $k \geq 2$, as the statement of the theorem can be easily seen to be true for the case $k = 1$. Since v has at most 3 neighbors having degree more than 3, it has at least $2k-4$ neighbors with degree at most 3. If $k = 2$, then $\Delta(G) \leq 3$, implying that every vertex in N(v) has degree at most 3. If $k \geq 3$, then v has at least $2k-4 \geq 2$ neighbors having degree at most 3. Thus, in any case, there exists $w \in N(v) \backslash \{u\}$ such that $d_{\text{G}}(w) \leq 3$.

Lemma 2. *If $uw \in E(G)$, then G has a k-linear coloring.*

Proof. Let $H = G - \{u, w\}$. Let x be a neighbor of u other than v, w. Note that x may not exist (i.e. if $\text{d}_{\text{G}}(\text{u}) = 2$). Similarly, let y be a neighbor of w other than u, v, if such a neighbor exists. We shall assume from here onwards that both x and y exist, since if one of them, say x, does not exist, then we can just add a new vertex x that is adjacent to only u and continue the proof. By the inductive hypothesis, there exists a k-linear

coloring c of H. Since $d_{\mathrm{H}}(x) \leq 2k-2$, there exists a color $i \in Missing(x) \cup Once(x)$. Color ux with i. Since $d_{\mathrm{H}}(v) = 2k-3$, we know that either:

(a) $\left|\text{Once(v)}\right| = 3$, or

(b) $\left|\text{Missing(v)}\right| = 1$ and $\left|\text{Once(v)}\right| = 1$.

First let us consider case (a). Color uv, uw with two different colors in $Once(v)\backslash\{i\}$, and then color vw with the remaining color j in $Once(v)$. Since $d_{\mathrm{H}}(y) \leq 2k-2$, we have either $\left|Missing(y)\right| \geq 1$ or $\left|Once(y)\right| \geq 2$. In the former case, we color wy with a color in Missing(y) and in the latter case, we color wy with a color in $Once(y)\backslash\{j\}$. We now have a k-linear coloring of G.

Now let us consider case (b). We color uv, vw with the color in Missing(v) and color uw with the color in $Once(v)$. As $d_{\mathrm{H}}(y) \leq 2k-2$, either $\left|Missing(y)\right| \geq 1$ or $\left|Once(y)\right| \geq 2$. If $\left|Missing(y)\right| \geq 1$, then color wy with a color in Missing(y). On the other hand, if $\left|Once(y)\right| \geq 2$, then color wy with a color in $Once(y)\backslash\{i\}$. This gives a k-linear coloring of G.

Lemma 3. *If u and w have a common neighbor other than v, then G has a k-linear coloring.*

Proof. Let z be a common neighbor of u and w other than v. Note that by Lemma 2, we can assume that $uw \notin E(G)$. As in the proof of Lemma 2, we assume that u has a neighbor x other than v, z and w has a neighbor y other than v, z. Define $H = G - \{u,w\}$. Clearly, $d_{\mathrm{H}}(v) = 2k-3$ and $d_{\mathrm{H}}(z) \leq 2k-3$. Thus we have either $\left|\text{Once(v)}\right| = 3$ or we have both $\left|\text{Missing(v)}\right| = 1$ and $\left|\text{Once(v)}\right| = 1$. We will use the weaker statement that either $\left|Once(v)\right| \geq 3$ or both $Missing(v) \neq \phi$ and $\left|Missing(v) \cup Once(v)\right| \geq 2$. Since we also have that either $\left|Once(z)\right| \geq 3$ or both $Missing(z) \neq \phi$ and $\left|Missing(z) \cup Once(z)\right| \geq 2$, we treat v and z symmetrically, so there are only three cases to consider. Note also that since $d_{\mathrm{H}}(x), d_{\mathrm{H}}(y) \leq 2k-2$, we have $Missing(x) \neq \phi$ or $\left|Once(x)\right| \geq 2$, and we have $Missing(y) \neq \phi$ or $\left|Once(y)\right| \geq 2$.

First, let us consider the case when Missing(v) $\neq \phi$, $\left|Missing(v) \cup Once(v)\right| \geq 2$, Missing(z) $\neq \phi$, and $\left|Missing(z) \cup Once(z)\right| \geq 2$. Choose $i \in$ Missing(v), $j \in$ (Missing(v) $\cup$ Once(v)) \{i}, $p \in$ Missing(z), and $q \in$ (Missing(z) $\cup Once(z))\backslash\{p\}$. If $i \neq p$, then color uv, vw with i, and wz, uz with p. If $i = p$, then color uv, wz with i, vw with j, uz with q. Let r denote the color so given to vw and l the color so given to uz. Now

color wy with a color in Missing(y) $\cup(Once(y)\backslash\{r\})$ and ux with a color in Missing(x) $\cup(Once(x)\backslash\{l\})$. We now have a k-linear coloring of G.

Next, suppose that Missing(v) $\neq \phi$, $\left|\text{Missing(v)}\cup Once(v)\right| \geq 2$, and $\left|Once(z)\right| \geq 3$. Choose $i \in$ Missing(v) and $j \in$ (Missing(v) $\cup$ Once(v)) \{i}. Color uv with i, vw with j, wy with a color t in Missing(y) $\cup$ (Once(y) \{j}), ux with a color $s \in$ Missing(x) $\cup$ Once(x), wz with a color in $l \in Once(z)\backslash\{t,j\}$, and uz with a color in $Once(z)\backslash\{s,l\}$. We now have a k-linear coloring of G.

Finally, suppose that $\left|\text{Once(v)}\right|$, $\left|\text{Once(z)}\right| \geq 3$. Choose distinct $i,j,l \in$ Once(v). Color vw with j, wy with a color t in Missing(y) $\cup(Once(y)\backslash\{j\})$, uv with a color h in {i, l} \t, and ux with a color s in Missing(x) $\cup(Once(x)\backslash\{h\})$. Then, if $h \in Once(z)$, let $f = h$, otherwise let f be a color in $Once(z)\backslash\{t,j\}$. Now color wz with f and uz with a color in $Once(z)\backslash\{f,s\}$. This gives a k-linear coloring of G.

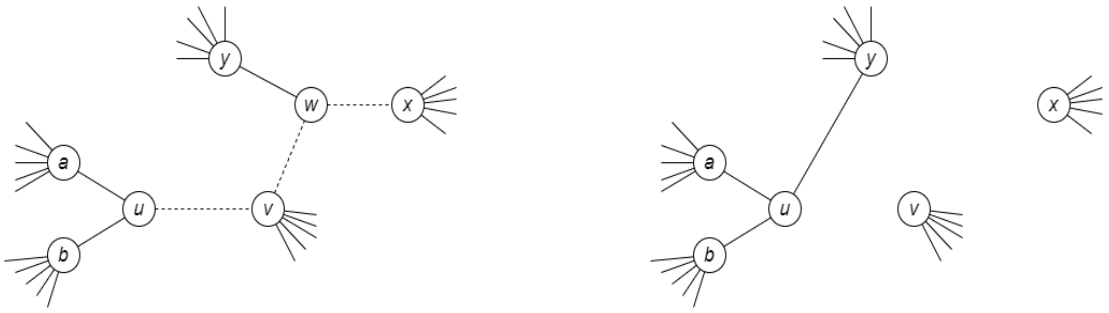
Now we are ready to complete the proof of Theorem 1. By Lemma 3, we can assume that u and w have no common neighbors other than v. assume that w has two neighbors x and y other than v (if not, new vertices of degree one adjacent to w can be added so as to ensure that x and y always exist). By Lemma 2, we can assume that $uw \notin E(G)$, which further implies that x and y are distinct from u.

Let $H = G - \{uv, vw, wx\}$. Let H' be the graph obtained by identifying the vertex w with u in H; i.e. H'= H/(u, w). Figure 1 shows the construction of the graph H'from G. Notice that $d_{H'}$ (u) ≤ 3 and that the graph H'$-u$ is nothing but $G - \{u,w\}$, which is a 3-degenerate graph as d_G(u), d_G(w) ≤ 3. Thus, H'is a 3-degenerate graph having $\left|\text{E(H)}\right|$ < $\left|\text{E(G)}\right|$. Then by the inductive hypothesis, there exists a k-linear coloring c'of H'. Let c be the coloring of H obtained by "splitting" the vertex u in H'to get back H. Formally, we define for all $e \in E(H)$

$$c(e) = \begin{cases} c'(e) & \text{if } e \neq wy \\ c'(uy) & \text{if } e = wy \end{cases}$$

It can be seen that c is a k-linear coloring of H, which is a subgraph of G. Note that since $d_H(x) \leq 2k-2$, either $\left|\text{Missing(x)}\right| \geq 1$ or $\left|\text{Once(x)}\right| \geq 2$. We first extend c to a coloring of G - {uv, vw} by coloring the edge wx by a color in Missing(x) $\cup$ (Once(x) $\backslash\{c(wy)\}$. We now describe how the edges uv and vw can be colored so that a k-linear coloring of G can be obtained.

Since $d_H(v) = 2k-3$, we have that either both $\left|\text{Missing(v)}\right|$ = 1 and $\left|\text{Once(v)}\right|$ = 1,



(a) Deleting edges uv, vw, wx to obtain the graph H,

(b) identifying w with u to get H'

Figure 2.1: Construction of the graph H'

or $\left|\text{Once(v)}\right| = 3$. Suppose first that $\left|\text{Missing(v)}\right| = 1$ and $\left|\text{Once(v)}\right| = 1$. Let Missing(v) = {i} and Once(v) = {j}. First, suppose that i belongs to either Twice(u) or Twice(w). We shall assume by the symmetry between u and w that $i \in$ Twice(u). Color uv with j and vw with i (note that $i \notin$ Twice(w) as if that were the case, u would have three edges of color i incident with it in c') to obtain a k-linear coloring of G. So let us assume that $i \notin$ Twice(u) $\cup$ Twice(w). If $j \in$ Twice(u) $\cup$ Twice(w), then we can color uv and vw with i to obtain a k-linear coloring of G. Thus, we can now assume that $i, j \notin$ Twice(u) $\cup$ Twice(v). If there is a path of color j having endvertices u and v, then color uv with i and vw with j (we know by Observation 2 that there is no path of color j having endvertices v and w). Otherwise, color uv with j and vw with i. We now have a k-linear coloring of G.

Next, suppose that $\left|\text{Once(v)}\right| = 3$. Let L be the set of all the colors for which there is a path of that color having endvertices u and v in c (such colors will all be in Once(u) $\cap$ Once(v)). The fact that $d_{\text{H}}(u) \leq 2$ implies that $0 \leq \left|L\right| \leq 2$ and also that if Twice(u) $\neq \phi$, then L = ϕ. If $\left|\text{L}\right| = 2$, then color vw with a color in L \Twice(w) (again, by Observation 2, there is no path having a color in L and having endvertices v and w) and uv with a color in Once(v) \L. Otherwise color vw with a color $r \in$ Once(v) \Colors(w) and uv with a color in Once(v) \({r} $\cup$ Twice(u) $\cup$ L). We now have a k-linear coloring of G.

This completes the proof of Theorem 1.

We convert this proof to a linear time algorithm that computes a $\lceil(\Delta(G)+1)/2\rceil$ -linear coloring for any input 3-degenerate graph G. For triangle-free planar graphs or partial 2-trees, our algorithm has better asymptotic runtime complexity than the algo-

rithm for planar graphs given in [8], with the caveat that our algorithm may produce a linear coloring using one more color than the optimum number of required colors.

2.4 A Linear Time Algorithm

We now describe how to compute a k-linear coloring of a 3-degenerate graph G having $\Delta(G) \leq 2k-1$[4]. Our algorithm will be a linear-time algorithm; i.e. having a running time of O(n + m), where n and m are the number of vertices and edges in G respectively. Since G is 3-degenerate, we have $m \leq 3n-6$, and therefore our algorithm will also be an O(n)-time algorithm. We assume that the input graph G is available in the form of an adjacency list representation.

Our general strategy will be to convert the inductive proof of Theorem 1 into a recursive algorithm, but there are some important differences, the main one being that the algorithm computes a more general kind of edge coloring using k colors. The algorithm follows the proof of Theorem 1 and removes some edges and if needed identifies two vertices to obtain a smaller graph G' for which an edge coloring of the desired kind is found by recursing on it. The graph G' is changed back into G by splitting back any identified vertices and adding the removed edges. The newly added edges are then colored to obtain an edge coloring of the desired kind for G. During this process, we never change the color of an edge that is already colored. We shall first discuss why our algorithm needs to compute a generalized version of k-linear coloring.

If the algorithm were to construct a k-linear coloring of G from a k-linear coloring of G' according to the proof of Theorem 1, and still have overall linear runtime, we would like to be able to decide the right color to be given to an uncolored edge uv of G in O(1) time. This means that we need data structures that allow us to determine in O(1) time a color i for uv such that:

(a). $i \notin$ Twice(u) $\cup$ Twice(v), and

(b). if $i \in$ Once(u) $\cap$ Once(v), there is no path colored i having end-vertices u and v.

The requirement (a) can be met by storing the sets Once(u) and Missing(u) for every vertex u as described in Section 2.4.3.

For (b), we could store a collection of "path objects" representing the monochromatic paths in the current coloring in such a way that by examining these objects, we can

determine in O(1) time whether there is a monochromatic path of color i having endvertices u and v. In particular, for a monochromatic path P having endvertices u and v, we could have a path object that stores the pointers to u and v. Further, we store the pointer to this object on the first and last edges of P. In this way, given a vertex u and an edge e colored i incident with u, where $i \in$ Once(u), we can examine the path object whose pointer is stored on e to determine in O(1) time the other end-vertex of the path colored i starting at u. Note that as an edge uv gets colored with color i, a path of color i can get extended (if $i \in Once(u) \cap Missing(v)$ or $i \in Missing(u) \cap Once(v)$) or two paths of color i can get fused into one path of color i (if $i \in Once(u) \cap Once(v)$). When two monochromatic paths get fused, we have to replace the two path objects corresponding to these paths with a single path object representing the new monochromatic path. If we store the pointer to a path object on each edge of the path it represents, then it becomes difficult to fuse a path with another path in O(1) time as we cannot afford to update the pointer stored on every edge of the path so as to point to the new path object. We can get around this difficulty by storing the pointer to a path object only on the first and last edges of the path represented by it. Since we do not need to know what the internal vertices or edges of a path are in order to fuse it with another path, this method could allow us to fuse two paths in O(1) time. As no edge that already has a color is ever recolored, a monochromatic path never gets split into two paths or gets shortened when an edge is colored. But a monochromatic path might need to get split into two monochromatic paths when a vertex is split into two vertices. Since we only split vertices of degree at most 3, at most one monochromatic path gets split during this operation. Suppose that a vertex v that needs to be split into two vertices is an internal vertex of a monochromatic path P. Since the internal vertices or edges of a path do not store the pointer to the corresponding path object, we cannot obtain the pointer to the path object representing P, given just the vertex v. We thus cannot update the collection of path objects so as to replace the monochromatic path P with two new monochromatic paths. We solve this problem by making sure that two paths that meet at a point that will be split later are never fused together into one path. This is explained in more detail below.

We say that a path having an endvertex u and containing the edge uv is "ending at u through uv". Suppose that a vertex w is identified with a vertex u when G' is constructed from G. It is clear from the proof of Theorem 1 that in G' , the vertex u has degree at

most 3, and there is possibly an edge uy that corresponds to an original edge wy in G. Before recursing to find the coloring for G' , we mark the vertex-edge pair (u, uy) as "special" (we call this a "special vertex-edge incidence"; more details given in Section 2.4.1). This mark, which can be stored inside the adjacency list of u, indicates that while computing the coloring for G' , a monochromatic path ending at u through uy should not be fused with another monochromatic path ending at u, even if they have the same color. Thus, while splitting the vertex u back into u and w, no path needs to be split. Note that this means that while the coloring for G' is being computed, we might have a path object for a path P colored i ending at u through uy and another path object for a path P' also colored i and ending at u, but through a different edge (as these paths will not be fused). Once this happens, if we denote the other endvertices of P and P' by x and x' respectively, then we can no longer detect that in the coloring constructed so far, there is a monochromatic path colored i starting at x and ending at x' , as there is no path object representing the monochromatic path colored i having endvertices x and x'. This means that the edge xx' , if it exists, could get colored i, and thus there may be a monochromatic cycle colored i in the coloring of G' . We will allow this to happen, since this monochromatic cycle will anyway get destroyed when the vertex u is split into u and w while recovering G back from G' . Thus, at any stage of the recursion, we compute a coloring for a graph in which certain vertex-edge pairs have been marked as special, and this coloring is not a k-linear coloring any more as it could contain monochromatic cycles. We call this kind of coloring a "pseudo-k-linear coloring". Since the path objects that we store do not correspond to maximal monochromatic paths anymore, we call them "segments" instead of paths. We now define these notions more rigorously.

2.4.1 Pseudo-k-linear colorings and segments

We define a $vertex-edgeincidence$ of a graph G to be a pair consisting of a vertex and an edge incident with it; i.e. it is a pair of the form (u, uv) where u, $v \in V(G)$ and $uv \in E(G)$.

Given a graph G and a set S of vertex-edge incidences in it, a mapping $c : E(G) \to \{1,2,...,k\}$ is said to be a pseudo-k-linear coloring of (G, S) if each color class is a disjoint union of paths and cycles, in which every cycle contains some edge uv such that $(u,uv) \in S$. Given a pair (G, S), we call the set S the special vertex-edge incidences

of G. Note that a pseudo-k-linear coloring of (G, ϕ) is a k-linear coloring of G and also that a k-linear coloring of G is a pseudo-k-linear coloring of (G, S) for any set S of vertex-edge incidences of G. Our algorithm computes a pseudo-k-linear coloring for an input (G, S), where G is a graph with $\Delta(G) \leq 2k-1$ and S is a set of vertex-edge incidences of G that are marked as special.

Definition 2. *Given a graph G and a set S of vertex-edge incidences in it, a segment of (G, S) is a sequence $\sigma = (u_1, u_2, ..., u_s)$ of vertices of G, where $s \geq 2$, such that:*

(i). for each $i \in \{1, 2, ..., s-1\}$, $u_i u_{i+1} \in E(G)$,

(ii). the edges u_1u_2, u_2u_3, . . . , $u_{s-1}u_s$ are pairwise distinct and their union gives a path or cycle in G, and

(iii). for each $i \in \{2, 3, ..., s-1\}$, $(u_i, u_{i-1}u_i), (u_i, u_iu_{i+1}) \notin S$.

Let $\sigma = (u_1, u_2, ..., u_s)$ be a segment of (G, S), We say that the edges u_1u_2, u_2u_3, . . . , $u_{s-1}u_s$ are the "edges in the segment σ". We say that u_1 and u_s are the terminal vertices of this segment. Note that the two terminal vertices of a segment can in fact be the same vertex (this happens when the edges in the segment form a cycle in G). Further, we call (u_1, u_1u_2) and $(u_s, u_{s-1}u_s)$ the terminal vertex-edge incidences of this segment. We also sometimes say that this segment is "ending at u_1 through the edge u_1u_2" and "ending at us through the edge $u_{s-1}u_s$". Note that every segment has exactly two terminal vertex-edge incidences and they can be in S.

Let c be a pseudo-k-linear coloring of (G, S). A segment of (G, S) is said to be a monochromatic segment of color i if every edge in it is colored i. A segment $\sigma = (u_1, u_2, ..., u_s)$ is said to be contained in a segment $\sigma' = (u'_1, u'_2, ..., u'_{s'})$ if u_1, u_2, . . . , u_s occur consecutively in that order in σ'. A monochromatic segment σ that is not contained in any other monochromatic segment is called a maximal monochromatic segment of (G, S) (the coloring c is assumed to be clear from the context). Clearly, every edge belongs to some maximal monochromatic segment, as the edge by itself is a monochromatic segment. In fact, it can be seen that each edge belongs to a unique maximal monochromatic segment—the reason being that the union of any two monochromatic segments having a common edge gives another monochromatic segment. This means that the edge set of the graph decomposes into a collection of pairwise edge-disjoint maximal monochromatic segments in a unique way. It can be seen that if u is

a vertex such that $i \in Colors(u)$, then u is a terminal vertex of a maximal monochromatic segment σ of color i if and only if either $i \in Once(u)$ or there exists an edge e of color i incident with u such that $(u,e) \in S$ (note that in the latter case, e may not belong to σ—trying to extend σ by including e in it will result in σ not being a segment). Observe that a monochromatic cycle C in the graph decomposes into a collection of $\left|\{u \in V(C) : \exists e \in E(C) \text{ such that } (u,e) \in S\}\right|$ maximal monochromatic segments.

At a given point of time, we maintain a set of segment objects, one corresponding to each maximal monochromatic segment of (G, S) under the current pseudo-k-linear coloring. The segment object corresponding to a maximal monochromatic segment stores just the terminal vertex-edge incidences of the segment, as will be explained in Section 2.4.3. The following observations are easy to see.

Observation 4 *Let c be a pseudo-k-linear coloring of (G, S) and let M be a collection of monochromatic segments of (G, S). The set M is the set of all maximal monochromatic segments of (G, S) if and only if:*

(i) every edge of G is contained in exactly one segment in M, and

(ii) there does not exist a vertex u and segments σ_1, $\sigma_2 \in M$ (possibly $\sigma_1 = \sigma_2$) ending at u through distinct edges e_1, e_2 respectively such that $c(e_1) = c(e_2)$ and $(u,e_1),(u,e_2) \notin S$.

Observation 5 *We can extend a pseudo-k-linear coloring c of $(G-uv,S)$ to a pseudo-k-linear coloring of (G,S), for some $uv \in E(G)$, by giving uv a color i if:*

(i) $i \notin Twice(u) \cup Twice(v)$, and

(ii) there is no maximal monochromatic segment of color i having u and v as terminal vertices.

Details of Implementation

We now present the details of our linear time algorithm that computes a pseudo-k-linear coloring given an input graph G having maximum degree at most $2k-1$ and a set S of vertex-edge incidences in it that are marked as special. We first describe the various data structures that we use. Note that in the following, a "list" refers to a doubly-linked list.

2.4.2 Encoding the graph

A representation of G as described below can be computed using the input adjacency list representation of G in linear time during the initialization phase. We maintain a list $Edges$ of the edges of the graph. For each vertex u, we maintain a list $Adj(u)$ of the edges incident with u. The node in $Adj(u)$ corresponding to an edge e incident with u stores the pointer to the node for e in the list Edges. For an edge uv, let N_u and N_v be the nodes corresponding to uv in the lists $Adj(u)$ and $Adj(v)$ respectively. The node for uv in the list Edges stores (u, N_u, v, N_v). Thus if we have the pointer to the node for an edge uv in the list Edges, the list $Adj(u)$, or the list $Adj(v)$, then the edge can be removed from the graph in O(1) time. It is easy to see that adding an edge, identifying a vertex of degree at most 1 with another of degree at most 2, and splitting a vertex of degree at most 3 into two vertices can all be done in O(1) time in this representation. For every vertex u, we store its degree $d_G(u)$ in the current graph G. The degree d(u) of a vertex u in the original input graph is assumed to be known at all times. If $(u, uv) \in S$, then this fact is stored by setting a binary flag in the node corresponding to the edge uv in $Adj(u)$ to true. We do not bother to record the vertex u in each node of $Adj(u)$, since using our representation, given just the pointer to a node in the $Adj(u)$, we can anyway find u in O(1) time using the $Edges$ list. The set of special vertex-edge incidences S is encoded by means of binary flags inside the nodes of the Adj lists: we set the binary flag inside the node for an edge uv in $Adj(u)$ to $true$ if and only if $(u, uv) \in S$.

2.4.3 Encoding the coloring

Let (G, S) be the graph and the set of special vertex-edge incidences at any stage of the algorithm. We encode the pseudo-k-linear coloring of (G, S) that we compute as follows.

We color the edges of the graph using the integers {1, 2, . . . , k}. Every node in the list Edges also contains a field in which the color assigned to the corresponding edge is stored. Every vertex u maintains two lists of colors $Onc(u)$ and $Miss(u)$, to store the sets $Once(u)$ and $Missing(u)$ respectively. We simply let $Onc(u)$ be a list that contains one node for each color in $Once(u)$. The node corresponding to a color i in $Onc(u)$ also stores the pointer to the node in $Adj(u)$ corresponding to the edge colored i incident with u. Note that we cannot store all the colors in Missing(u) in the list

$Miss(u)$, because if we do, then initializing these lists for all the vertices will take too long ($\Omega(nk)$ time). To overcome this, we will use a trick from [7]: we store in $Miss(u)$ only the colors in Missing$(u) \cap \{1,2,...,min\{d(u)+2,k\}\}$, where d(u) is the degree of u in the graph given as input to the algorithm. Note that at any stage of the algorithm, if G is the graph being colored by the recursive coloring procedure, for any vertex $u \in V(G)$, $d_G(u) \leq d(u)$. This way of storing the list $Miss(u)$ ensures that the total size of these lists $\Sigma_{u \in V(\hat{G})} \left|Miss(u)\right| \leq \Sigma_{u \in V(\hat{G})}(d(u)+2)$ which is $O(n+m) = O(n)$ (as 3-degenerate graphs have at most $3n-6$ edges), where $\hat{G}$ is the initial input graph. Thus we can initialize all these lists in linear time. This trick works even though $Miss(u)$ may not contain all the colors in Missing(u) because we never need to check if some particular color is in Missing(u) unless $d_G(u) \leq 3$, in which case we can do it in O(1) time by just checking the colors of all the edges incident with u. For every vertex u such that $d_G(u) > 3$, we will only need to find some two colors in Missing(u). The following observation shows we will never go wrong when trying to do this.

Observation 6 *If Miss(u) $\neq$ Missing(u), then $\left|Miss(u)\right| \geq 2$.*

Proof. If $d(u)+2 > k$, we store all the colors in Missing(u) in $Miss(u)$, so we have nothing to prove. So let us assume that $d(u)+2 \leq k$. Then both the sets {1, 2, . . . , d(u) + 2} and Missing(u) are subsets of {1, 2, . . . , k}, and therefore $\left|\{1,2,...,d(u)+2\} \cup Missing(u)\right| \leq k$. This implies that $\left|Miss(u)\right| = \left|Missing(u) \cap \{1,2,...,d(u)+2\}\right| \geq \left|Missing(u)\right| + d(u) + 2 - k$. Since $\left|Missing(u)\right| \geq k - d_G(u)$ and $d_G(u) \leq d(u)$, we get $\left|Miss(u)\right| \geq 2$.

The list $Miss(u)$ needs to be updated when an edge incident with u gets colored, say with a color i, that is in $Miss(u)$. Now that we have colored uv with i, we need to remove i from the list $Miss(u)$. We may not have the pointer to the node corresponding to i in $Miss(u)$ since we may have determined that $i \in$ Missing(u) by checking the adjacency list of u (as $d_G(u) \leq 3$). We cannot afford to search the list $Miss(u)$ to find and remove the node corresponding to the color i (note that the length of $Miss(u)$ could be $d(u)+2$ which can much larger than $d_G(u)$). For this purpose, we follow [7] and maintain an array $Ptrs(u)$ of size min{d(u) + 2, k} that contains pointers to the nodes in the list $Miss(u)$. For each $i \in Miss(u)$, the i-th element of the array $Ptrs(u)$ will store the pointer to the node in $Miss(u)$ corresponding to the color i. Thus, given a color i in the list $Miss(u)$, we can use the array $Ptrs(u)$ to remove the node corresponding to i from

the list $Miss(u)$ in O(1) time. It is clear that the array $Ptrs(u)$ can also be initialized in linear time during the initialization phase and updated in O(1) time whenever a node is removed from the list $Miss(u)$.

We maintain a "segment object" corresponding to each maximal monochromatic segment of (G,S) under the current pseudo-k-linear coloring of (G,S). For a maximal monochromatic segment having terminal vertex-edge incidences (u,uv) and (x,xy), we store the pointer to its segment object in the node for uv in $Adj(u)$ and the node for xy in $Adj(x)$. The segment object will store the pointers to these two nodes as well. Every time we add edges to G and then extend the current pseudo-k-linear coloring to the new graph by coloring the newly added as yet uncolored edges, we have to update the collection of segment objects so that they correspond exactly to the maximal monochromatic segments under the new coloring. We do this by ensuring that the family of segments represented by the new collection of segment objects satisfies Observation 4. Our strategy for doing this will be as follows. When an uncolored edge uv gets colored i, we first create a new segment object corresponding to the segment containing just uv. If there was already an edge e colored i incident on u (resp. v), then we have at this point two segment objects corresponding to two segments, each having color i, one ending at u (resp. v) through uv and the other ending at u (resp. v) through e. If $(u,uv),(u,e) \notin S$ (resp. $(v,uv),(v,e) \notin S$), then by Observation 4, these segments are not maximal monochromatic segments in the new coloring and hence the segment objects that we have at the moment do not correspond to the maximal monochromatic segments under the new coloring. In order to correct this, whenever some segment objects represent segments whose union gives a larger monochromatic segment under the new coloring, we "fuse" them into a single segment object that represents the new monochromatic segment formed by the union of these segments. This procedure shall be described in detail in the proof of Lemma 5. We first show that the operation of fusing two segment objects can be done in O(1) time.

Lemma 4. *A new segment object containing a single edge can be created in O(1) time. Two segment objects can be fused into a single segment object in O(1) time.*

Proof. If we want to create a segment object containing a single edge uv, the pointer to whose node in the list Edges is known, we first use this node to find the nodes N_u and N_v in the lists $Adj(u)$ and $Adj(v)$ respectively corresponding to uv. We create a

segment object σ and store in it the pointers to N_u and N_v and we store the pointer to σ in N_u and N_v. It is clear that this process takes just O(1) time. Suppose that we would like to fuse two segment objects σ_1 and σ_2 corresponding to segments ending at a vertex u through distinct edges e_1 and e_2 incident with u respectively. Let (u'_1, e'_1) be the terminal vertex-edge incidence other than (u, e_1) of the segment represented by σ_1 and (u'_2, e'_2) be the terminal vertex-edge incidence other than (u, e_2) of the segment represented by σ_2. From the segment objects σ_1 and σ_2, we can find the nodes N_1 in $Adj(u'_1)$ and N_2 in $Adj(u'_2)$ corresponding to e'_1 and e'_2 respectively. We now create a new segment object σ containing the pointers to N_1 and N_2; this segment object represents the segment having terminal vertex-edge incidences (u'_1, e'_1) and (u'_2, e'_2) that results from fusing the segments corresponding to σ_1 and σ_2. We replace the pointers to σ_1 and σ_2 in N_1 and N_2 respectively with pointers to σ. We can now destroy the objects σ_1 and σ_2 and remove their pointers from the nodes corresponding to e_1 and e_2 in $Adj(u)$. It is easy to see that all this can be done in O(1) time.

The following observation is easy to see.

Observation 7 *For a vertex $u \in V(G)$ such that $d_G(u) \leq 3$, we can compute Once(u), Twice(u) and Colors(u) in O(1) time.*

The next lemma shows that when we extend a pseudo-k-linear coloring by coloring an uncolored edge, the data structures that encode the coloring can all be updated in O(1) time.

Lemma 5. *Let G be a graph and S a set of vertex-edge incidences of G. Let c be a pseudo-k-linear coloring of $(G - uv, S \setminus \{(u,uv),(v,uv)\})$. Then c can be extended in O(1) time to a pseudo-k-linear coloring of (G, S) by coloring the edge uv with a color i provided that $d_G(u) \leq 3$, $i \notin$ Twice(u), the pointer to a node containing color i in either Miss(v) or Onc(v) is known, and there is no maximal monochromatic segment of color i having terminal vertices u and v.*

Proof. As noted in Observation 5, it is clear that by coloring the edge uv with the color i, we obtain a pseudo-k-linear coloring of (G,S). We only need to show that we can update our data structures encoding the coloring in O(1) time

We assume that we have the pointer to the node N_{uv} for uv in the list Edges and also the pointer to a node N'containing the color i in one of the lists $Miss(v)$ or $Onc(v)$. We first set the color field in N_{uv} to i and create a new segment object for a segment

σ colored i containing the single edge uv, and having terminal vertex-edge incidences (u,uv) and (v,uv) as described in the proof of Lemma 4. From the node $\mathrm{N_{uv}}$ we get the pointers to the nodes $\mathrm{N^u_{uv}}$, $\mathrm{N^v_{uv}}$ corresponding to the edge uv in the lists $Adj(u)$ and $Adj(v)$ respectively. Since $d_{\mathrm{G}}(u) \leq 3$, we can use Observation 7 to check if $i \in Once(u)$. If $i \notin Once(u)$, then we know that $i \notin Colors(u)$. In this case, we use the array $Ptrs(u)$ to find the node for i in $Miss(u)$ and remove it in O(1) time if $i \leq d(u)+2$, and we do nothing if $i > d(u)+2$. Add to $Onc(u)$ a node containing the color i and the pointer to the node corresponding to the edge uv in $Adj(u)$. Let us now suppose that $i \in Once(u)$; i.e. there is an edge $\mathrm{e_u}$ colored i incident with u. We can traverse $Onc(u)$ to find the node containing i and the pointer to the node $\mathrm{N_u}$ in $Adj(u)$ corresponding to $\mathrm{e_u}$, since $\big|Onc(u)\big| \leq d_{\mathrm{G}}(u) - 1 \leq 2$. We remove this node from $Onc(u)$. Furthermore, from $\mathrm{N_u}$, we find the pointer to the segment object representing the maximal monochromatic segment σ_{u} colored i ending at u through $\mathrm{e_u}$ in the coloring c. If neither (u,e_{u}) nor (u,uv) are in S (this can checked by inspecting the binary flags in the nodes $\mathrm{N^u_{uv}}$ and $\mathrm{N_u}$), then we fuse the segments σ_{u} and σ into a single segment. Note that this segment has (v,uv) as one of its terminal vertex-edge incidences. By Lemma 4, this operation can be done in O(1) time. Let σ' be the segment containing the edge uv after this step (if $i \notin Once(u)$ or if either (u,e_{u}) or (u,uv) is in S, then $\sigma' = \sigma$).

Now if N' is a node from the list $Miss(v)$, we can easily remove it from the list $Miss(v)$ in O(1) time (and update the array $Ptrs(v)$ as needed). We add to the list $Onc(v)$ a node containing i and a pointer to the node $\mathrm{N^v_{uv}}$. Note that the segments represented by the current collection of segment objects satisfy the conditions in Observation 4, and hence are the maximal monochromatic segments in the new coloring. Thus, we are done. So let us suppose that N' is a node from the list $Onc(v)$. Remove N' from $Onc(v)$. From N', we can find the node $\mathrm{N_v}$ in the list $Adj(v)$ corresponding to the edge $\mathrm{e_v}$ colored i incident with v in the coloring c. From the node $\mathrm{N_v}$, we can find the pointer to the segment object that represents the maximal monochromatic segment $sigma_{\mathrm{v}}$ colored i ending at v through $\mathrm{e_v}$ in the coloring c. Since there was no maximal monochromatic segment of color i having u and v as terminal vertices in the coloring c, we can conclude that $\sigma_{\mathrm{v}} \neq \sigma'$. Now if neither (v,uv) nor (v,e_{v}) are in S (i.e. if the binary flags in the nodes $\mathrm{N^v_{uv}}$ and $\mathrm{N_v}$ are both set to false), fuse σ' and σ_{v} into a single segment. Again this can be done in O(1) time by Lemma 4. The segments represented

by the collection of segment objects now satisfy the conditions in Observation 4 and hence are the maximal monochromatic segments of the new coloring.

2.4.4 Maintaining the Pivots list

In order to follow the proof of Theorem 1, at each recursive step, the algorithm needs to find a pivot edge so that some edges can be removed from its vicinity (followed by the identification of two vertices, if necessary). We maintain a list Pivots in order to accomplish this in O(1) time. The Pivots list stores the pivots that have at least one neighbor of degree at most 3. Each vertex stores a pointer that will point to its node in the list Pivots whenever it is in the list. Thus, we can check in O(1) time if a particular vertex is in the list Pivots and if needed, also remove it from the list in O(1) time. Recall that for each vertex u, we maintain a variable $d_G(u)$ that stores its current degree. In addition, we also store another variable $d'_G(u)$ that maintains the number of neighbors of u that have degree at most 3. Note that a vertex u is a pivot when $d_G(u) - d'_G(u) \leq 3$. We make sure that the list Pivots always contains exactly those vertices u for which $d_G(u) - d'_G(u) \leq 3$ and $d'_G(u) \geq 1$. Whenever our algorithm removes some edges or identifies two vertices in order to create a smaller graph, we update the Pivots list accordingly. It can be seen that for both these operations, we can update $d_G(u)$ and $d'_G(u)$ for every vertex u for which these parameters change in O(1) time and therefore, we can update the Pivots list also in O(1) time. For example, if an edge uv is removed, we update the Pivots list as follows. As will be seen in Section 2.4.5, our algorithm always removes an edge uv such that $d_G(u) \leq 3$. When this happens, we decrease $d_G(u)$ and $d_G(v)$ by one, and since u had degree at most 3 before, we decrease $d'_G(v)$ by one. If v also had degree at most 3 before, we decrease $d'_G(u)$ also by one. If now $d'_G(u) = 0$ or $d'_G(v) = 0$, we remove that vertex from the list Pivots if it is present in the list. Note that when the edge uv is removed, u or v will not become eligible to be in the Pivots list if it was not already in the list. If v has now become a vertex of degree exactly 3, for each neighbor w of v (there are 3 such neighbors), we increment $d'_G(w)$ by 1, and if for any of them this results in $d_G(w) - d'_G(w) \leq 3$ (clearly, $d'_G(w) \geq 1$), we add w to the Pivots list if it is not already present in the list. Observe that all of these operations associated with the removal of an edge take only O(1) time in total. Similarly, we can update the Pivots list in O(1) time after an identification operation too as follows. Note

that we always identify a vertex w of degree at most 1 with a vertex u of degree at most 2. We update $d_G(u)$ and $d'_G(u)$ and remove w from the list Pivots if it is present in the list. Further, if now $d'_G(u) \geq 1$, we add u to the list Pivots if it is not already in it.

2.4.5 The algorithm

The algorithm takes as input a graph G and a set S of vertex-edge incidences of G, and computes a pseudo-k-linear coloring of (G, S). If we want the algorithm to produce a k-linear coloring of G, we simply give the set $S = \Phi$ as input to the algorithm.

The initialization phase of the algorithm consists of doing some preprocessing in order to compute the degrees of each vertex, determine the pivots, and to construct the list Pivots of pivots that have at least one pivot edge incident with them. It is easy to see that this stage takes linear time. The data structures required to encode the graph and the k-linear coloring (see Sections 3.2 and 3.3) are also initialized during this phase. Note that after the initialization, no edges are colored, so for every vertex u, we will have Onc(u) = ϕ and Miss(u) will contain all the colors in $\{1, 2, \ldots, \min\{d(u) + 2, k\}\}$. Also, there will be no segment objects. Then we invoke a recursive procedure Color(G, S), setting $S = \phi$, which works as follows.

A The procedure COLOR(G,S) :

The procedure does not take G and S as parameters, but rather expects that they are encoded in the data structures: i.e. it assumes that the graph G along with the set S of special vertex-edge incidences is encoded in the lists Edges and the lists $\{\text{Adj(u)}\}_{u \in V(G)}$ as described in Section 2.4.2. The procedure returns a pseudo-k-linear coloring of (G, S) by storing the color of each edge in its node in the list Edges, by filling data in the lists Miss and Onc of each vertex, and also by constructing a collection of segment objects representing the maximal monochromatic segments of (G, S) under the coloring, as explained in Section 2.4.3.

Note that whenever we say "color e with i", where e is an edge and i a color, we mean that we use Lemma 5 to assign the color i to the edge e in O(1) time. Thus, whenever we color an edge, we make sure that the conditions of Lemma 5 are satisfied. Further, we make sure that the conditions of Observation 5 are also satisfied, and therefore we always get a pseudo-k-linear coloring of (G, S).

If the list Pivots is empty, then the graph G is empty, and therefore the procedure simply returns without constructing any coloring. Suppose that the list Pivots is not empty. Then let v be the first vertex in Pivots. Determine a neighbor u of v such that dG(u) ≤ 3. This will take only O(1) time as v, being a pivot, has at most three neighbors having degree more than 3 and being in the list Pivots, v has at least one neighbor of degree at most 3.

If $d_G(v) < 2k - 1$, then modify G to G' = G $-$ uv. As mentioned in Sections 3.2 and 3.4, we can do this in O(1) time. Note that even though we remove the nodes N_{uv}, N_u, and N_v corresponding to uv from Edges, Adj(u), and Adj(v) respectively, we retain them for later use. Observe that the set of vertex-edge incidences of G' that are marked as special is $S' = S \backslash \{(u, uv), (v, uv)\}$, as the nodes N_u and N_v have been removed from the lists Adj(u) and Adj(v) respectively. Update the Pivots list in O(1) time as described in Section 2.4.4. Construct a pseudo-k-linear coloring of (G' , S') by invoking Color(G' , S'). Modify G' back to G by adding the edge uv. While doing this, we add back the stored nodes N_{uv}, N_u, and N_v to the lists Edges, Adj(u), and Adj(v) respectively. Note that after this step, we get back S as the set of vertex-edge incidences marked as special in G. We check whether there is a color i in Onc(v) such that $i \notin$ Colors(u) (the set Colors(u) can be computed in O(1) time by Observation 7). Note that since $\left|\text{Colors(u)}\right| \leq 2$, we will never need to traverse beyond the third element in Onc(v) to find the color i and hence this check can be performed in O(1) time. If we find such a color i in the list Onc(v), since we then have the pointer to the node corresponding to i in Onc(v), we satisfy all the conditions of Lemma 5, and therefore we color uv with i in O(1) time. Now suppose that we cannot find a color in Onc(v) that is not in Colors(u). Then it must be the case that $\left|\text{Onc(v)}\right| \leq 2$, which implies that Missing(v) $\neq \phi$ and that $\left|\text{Missing(v)} \cup \text{Once(v)}\right| \geq 2$ (as argued in the proof of Lemma 1). By Observation 6, we have that Miss(v) $\neq \phi$. Let i be the first color in the list Miss(v) (note that we again have the pointer to the node containing i in the list Miss(v)). If $i \notin$ Twice(u) (which can be checked in O(1) time by Observation 7), we color the edge uv with i using Lemma 5 in O(1) time. Otherwise, if $\left|\text{Missing(v)}\right| \geq 2$, then we have from Observation 6 that $\left|\text{Miss(v)}\right| \geq 2$, so we take the second color j in the missing list and color uv with j. Lastly, if $\left|\text{Missing(v)}\right| = 1$, then $\left|\text{Once(v)}\right| \geq 1$, and therefore we take the first color j in Onc(v) and color uv with j. We thus obtain a pseudo-k-linear coloring of (G, S). Note

that all the steps of Color(G, S) outside the recursive call to Color(G' , S') together take only O(1) time.

So let us now assume that $d_{\mathrm{G}}(v) = 2k-1$. As explained in the proof of Theorem 1, then there exists a vertex $w \in N(v)\backslash\{u\}$ such that $d_{\mathrm{G}}(w) \leq 3$. Clearly, this vertex can be found in O(1) time as v is a pivot. We first check if $uw \in E(G)$. Note that this check can be done in O(1) time as $d_{\mathrm{G}}(u), d_{\mathrm{G}}(w) \leq 3$. If $uw \in E(G)$, then we can follow the steps in the proof of Lemma 2 to compute a pseudo-k-linear coloring for (G, S) as we did before. Similarly, if there exists a vertex $z \in (N(u) \cap N(w))\backslash\{v\}$ (again this can be checked in O(1) time as $d_{\mathrm{G}}(u), d_{\mathrm{G}}(w) \leq 3$), we follow the steps in the proof of Lemma 3 to compute a pseudo-k-linear coloring for (G, S). In the both the above cases, it can be easily seen that the steps outside the recursive call to the procedure Color can be done in O(1) time.

Now suppose that $uw \notin E(G)$ and that $N(u) \cap N(w) = \{v\}$. The algorithm in this case too follows the proof of Theorem 1 closely. We shall only describe the algorithm for the case when $\mathrm{d_G(w)} = 3$, as the procedure for other cases can be easily deduced (or we could add dummy vertices of degree 1 as neighbors of w to make $\mathrm{d_G(w)} = 3$). Let N(w) = {v, x, y}. Remove the edges uv, vw, wx from G (the pointers to the nodes for these edges in the list Edges can be obtained in O(1) time as $d_{\mathrm{G}}(u), d_{\mathrm{G}}(w) \leq 3$, and hence they can be removed in O(1) time) and update the Pivots list. Observe that as the nodes corresponding to these edges have disappeared from Adj(u), Adj(v), Adj(w) and Adj(x), the set of special vertex-edge incidences S has now changed to $S_1 = S\backslash\{(u,uv),(v,uv),(v,vw),(w,vw),(w,wx),(x,wx)\}$. We now identify the vertex w with u — in other words, the edge wy has to change one of its endpoints from w to u. We do this by modifying the node N corresponding to wy in the list Edges. Further, we add a node $\mathrm{N_u}$ containing a pointer to N to Adj(u) and remove the node $\mathrm{N_w}$ containing the pointer to N from Adj(w) (also decrease the degree of w to zero and update the Pivots list as mentioned in Section 4.4). Let us call the graph so obtained G'. Note that the set of special vertex-edge incidences $\mathrm{S_1}$ has now changed to $S_2 = S_1\backslash\{(w,wy)\}$ if $(y,wy) \notin S1$ or to $S_2 = S_1\backslash\{(w,wy),(y,wy)\}) \cup \{(y,uy)\}$ if $(y,wy) \in S1$.

We now set the binary flag in the node $\mathrm{N_u}$ in the list Adj(u) to true so that the set of special vertex-edge incidences is now $S' = S_2 \cup \{(u,uy)\}$. The procedure Color(G', S') is invoked to construct a pseudo-k-linear coloring of (G\, S\). In order to modify

this into a pseudo-k-linear coloring of (G, S), we first split the vertex u back into u and w. For this, we just change the endpoint u to w in the node N, remove the node N_u from Adj(u) and add N_w back to Adj(w), while increasing the degree of w to 1. Note that this step automatically changes the set of special vertex-edge incidences from S'to S_1. Add the edges uv, vw, wx. While doing this, we restore the nodes that we removed from Adj(u), Adj(v), Adj(w) and Adj(x), so that the set of special vertex-edge incidences changes from S_1 back to S. We now color these edges as explained in the proof of Theorem 1, with the only difference being that instead of checking whether there is a path of some color *i* having *u* and *v* as endvertices, we check whether there is a maximal monochromatic segment of color *i* having *u* and *v* as terminal vertices. We explain in detail below. First, we state an observation that we need.

Observation 8 *Given vertices u, v and an edge e incident with v, we can check in O(1) time if there exists a maximal monochromatic segment having (v, e) as a terminal vertex-edge incidence and u as a terminal vertex.*

Proof. We assume that we have the node N for e in Adj(v). We check if the pointer to some segment object is stored in N. If not, then we can immediately conclude that the maximal monochromatic segment that we seek does not exist. So let us suppose that N contains the pointer to a segment object σ. Let N'be the node other than N whose pointer is stored σ. If N'$\in$ Adj(u) (this can be checked in O(1) time as mentioned in Section 2.4.2), we conclude that σ represents a maximal monochromatic segment having (v, e) as a terminal vertex-edge incidence and u as a terminal vertex. Clearly, all this takes only O(1) time.

As noted in the proof of Theorem 1, we first color wx. If $Miss(x) \neq \phi$, then we color wx with a color in Miss(x) (note that as always, the conditions of Lemma 5 are satisfied). Otherwise, by Observation 6, we have that $Missing(x) = \phi$, which implies that $|Onc(x)| \geq 2$ (refer to proof of Theorem 1). By checking at most the first two nodes of Onc(x), we find a color i in Onc(x) that is different from the color of wy. Color wx with i. We shall now color uv and vw, again following the proof of Theorem 1.

We have two cases: either |Missing(v)| = 1 and |Once(v)| = 1, or |Once(v)| = 3. Let us first consider the case when |Missing(v)| = 1 and |Once(v)| = 1. In this case, we have |Onc(v)| = 1 and by Observation 6, we also have |Miss(v)| = 1. Let i be the color in Miss(v) and j the color in Onc(v). If $i \in Twice(u)$, then color uv with j and

vw with i. Otherwise, if $i \in Twice(w)$, then color vw with j and uv with i. So let us consider the case when $i \notin Twice(u) \cup Twice(w)$. Again following the proof of Theorem 1, if $j \in Twice(u) \cup Twice(w)$, we color both uv and vw with i. So we assume that $j \notin Twice(u) \cup Twice(w)$. From the node in Onc(v) containing the color j, we can find the node in Adj(v) corresponding to the edge e_j colored j incident with v. Using this node, we determine in O(1) time if there is a maximal monochromatic segment having (v, e_j) as a terminal vertex-edge incidence and u as a terminal vertex as described in Observation 8. If there is, we color uv with i and vw with j, and otherwise, we color uv with j and vw with i. To show that this works even though the coloring we have now is a pseudo-k-linear coloring (which may contain monochromatic cycles), we first observe that Observation 2 actually holds for this kind of colorings too, as stated below.

Observation 9 *Let H be a graph that is the disjoint union of paths and cycles. If P1 and P2 are two distinct non-zero-length paths in H such that V (P1)∩ V (P2) ≠ ϕ, then one of the endvertices of P1 or one of the endvertices of P2 must be a vertex of degree two in H.*

Thus since the edges colored j form a disjoint union of paths and cycles, $j \in Once(v)$, and $j \notin Twice(u) \cup Twice(w)$, if there is a path colored j from v to u, then by Observation 9, there can be no path colored j from v to w. Note that the union of the edges in any maximal monochromatic segment of color j having distinct terminal vertices form a path colored j that has as its endvertices the terminal vertices of the segment. Therefore we can conclude that there cannot be two maximal monochromatic segments colored j, one having terminal vertices u, v and the other having terminal vertices v, w. This shows that our strategy for coloring *uv* and *vw* satisfies the conditions in Observation 5, and therefore yields a pseudo-k-linear coloring of (G, S).

We shall now consider the case when |Once(v)| = 3. For this case, our algorithm and its proof of correctness differs slightly from the proof of Theorem 1. We traverse the nodes of Onc(v) to find the three colors i, j, l in Once(v). From these nodes, we can find the nodes in Adj(v) corresponding to the edges e_i, e_j , e_l incident with v having colors i, j, l respectively. Using Observation 8, we determine the set $L = \{p \in \{i, j, l\}$: there exists a maximal monochromatic segment having (v, e_p) as a terminal vertex-edge incidence and u as a terminal vertex$\}$. It is clear that $|L| \leq 2$ as u has at most two colored edges incident with it.

Claim. If Twice(u) $\neq \phi$, then L $\subseteq$ Twice(u) and $|L| \leq 1$.

If there exists $p \in$ Twice(u), then both the colored edges incident with u have color p, i.e. Colors(u) = Twice(u) = {p}. Then any maximal monochromatic segment having u as a terminal vertex must have color p, which implies that $L \subseteq \{p\}$ = Twice(u). Clearly, this also means that $|L| \leq 1$. This proves the claim.

Suppose that $|L| = 2$. Then by the above claim, $Twice(u) = \phi$. Let $p \in L \backslash Twice(w)$ (p exists as $|Twice(w)| \leq 1$). We now have that there is at most one edge colored p incident with each of u, v, w. Since there is a maximal monochromatic segment of color p having terminal vertices u, v (as $p \in L$), there is a path colored p having endvertices u, v. We can now conclude by Observation 9 that there is no path having color p between v and w, and therefore there is no maximal monochromatic segment having color p and terminal vertices v, w. Therefore, in this case, we color vw with p and uv with the color in $\{i, j, l\} \backslash L$. If $|L| \leq 1$, color vw with a color $r \in \{i, j, l\} \backslash Colors(w)$ and uv with a color in $\{i, j, l\} \backslash (r \cup Twice(u) \cup L)$ (note that this set is nonempty by our claim above). This completes the description of the algorithm.

Chapter 3

Implementation and Results

3.1 Software Requirements:

- Programming Language: C++
- IDE: VScode
- Operating System: Windows

3.2 Algorithm for k-linear coloring of 3-degenerate graph

3.2.1 Input

The algorithm takes as input a graph G and a set S of vertex-edge incidences of G,

3.2.2 Output

The algorithm computes a pseudo-k-linear coloring of (G, S) by populating thedata structures that encode the coloring. If we want the algorithm to produce a k-linear coloring of G, we simply give the set S = ϕ as input to the algorithm.

3.2.3 Algorithm

A Initialization Phase

First of all, Perform the Initialization that consists of doing some preprocessing :

1. Initialize the data structures required to encode the graph and the pseudo-k-linear coloring (see Sections 2.4.2 and 2.4.1 respectively).

```
struct vNode {
  struct vNode *prev, *next;
```

```
  int adj_v;
  struct eNode *edge;
  bool special;
  struct segmentNode *segment;
};
struct eNode {
  struct eNode *prev, *next;
  int u, v, color;
  struct vNode *Nu, *Nv;
};
struct Pivot {
  struct Pivot *prev, *next;
  int v;
};
struct clrNode {
  struct clrNode *prev, *next;
  int color;
  struct vNode *Nu;
};

struct segmentNode {
  struct vNode *Nu, *Nv;
  int u, v;
};
vector<int> d, dG, ddG;
vector<pair<struct vNode *, struct vNode *>> adjList;
pair<struct eNode *, struct eNode *> Edges;
pair<struct Pivot *, struct Pivot *> Pivots;
vector<struct Pivot *> VtoPivot;
vector<pair<struct clrNode *, struct clrNode *>> Once, Miss;
vector<vector<struct clrNode *>> Ptrs;
vector<set<int>> Colors;
```

2. Construct the Graph and compute the degrees of each vertex,

3. Encoding the coloring (see Sections 2.4.3),

```
void Encoding_the_coloring(int N){
  Once.resize(N, {NULL, NULL});
  Miss.resize(N, {NULL, NULL});
  Colors.resize(N, {});
  Ptrs.resize(N);
  for (int u = 0; u < N; u++){
    for (int i = 0; i < min(d[u] + 2, K); i++){
      add_Miss_node(u, i);
    }
  }
}
```

4. Construct the list Pivots of pivots that have at least one pivot edge incident with them (see Sections 2.4.4).

```
void The_Pivots_list(int N){
  Pivots = {NULL, NULL};
  VtoPivot.resize(N, NULL);
  for (int u = 0; u < N; u++){
    for (int v : adj[u]){
      if (dG[v] <= 3) ddG[u]++;
    }
  }
  for (int u = 0; u < N; u++){
    if (ddG[u] > 0
        && (dG[u]-ddG[u] <= 3) && VtoPivot[u] == NULL){
        VtoPivot[u] = add_Pivot(u);
    }
  }
}
```

Note that after the initialization, no edges are colored, so for every vertex u, we will have Onc(u) = Φ and Miss(u) will contain all the colors in $\{$ 1, 2, . . . , min$\{$ d(u) + 2, k$\}\}$. Also, there will be no segment objects. Then we invoke a recursive procedure Color(G, S) which constructs a pseudo-k-linear coloring of (G, S) by populating the data structures that encode the coloring.

B The Procedure COLOR(G,S)

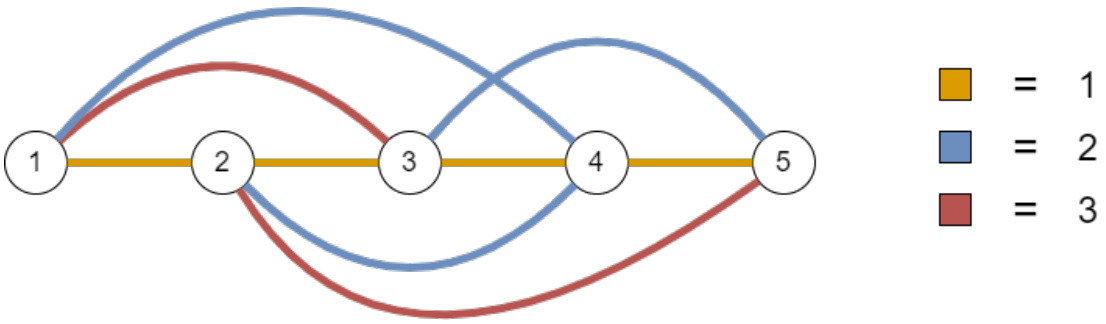
The procedure COLOR(G,S) is discussed in detail in Chapter 3 (section 2.4.5.A).

In this algorithm, we computes a k-linear coloring of a 3-degenerate graph G having $\Delta(G) \leq 2k-1$ that also shows that $\chi_l'(G) \leq \lceil(\Delta(G)+1)/2\rceil$.

This linear-time algorithm provides the efficient way to partition the edge set of any 3-degenerate graph G on n vertices into at most $\lceil(\Delta(G)+1)/2\rceil$. Also, the output produced by an algorithm for a random test input graph, shown in figure, justify that algorithm works correctly.

```
Input Format :
 First line of the input contains N and M where N = |V(G)| and M = |E(G)|
 Next M lines contains u and v where uv is an element of E(G)
5 9
1 2
1 3
1 4
2 3
2 4
2 5
3 4
3 5
4 5

---------------- End of Input ----------------

K = 3

Output :

Edge (4,5) colored by Color=1
Edge (3,4) colored by Color=1
Edge (2,4) colored by Color=2
Edge (1,4) colored by Color=2
Edge (3,5) colored by Color=2
Edge (2,3) colored by Color=1
Edge (1,3) colored by Color=3
Edge (2,5) colored by Color=3
Edge (1,2) colored by Color=1
```

(a) Output for a Given Input



(a) Graphical Representaion of Output

Figure 3.0: k-linear coloring of input graph

Chapter 4

Conclusions and Future Work

We have implemented our proposed algorithm to experimentally verify their correctness. We have tested the algorithm on some diverse input programs with several executions. We have observed that the results computed for the proposed approaches are correct for all input experimental programs. This experimentally validated the correctness of our linear-time algorithm and also shows that for every 3-degenerate graph G,$\chi_1^{'}(G) \leq \lceil(\Delta(G)+1)/2\rceil$.

We have identified several opportunities for future research. Since the proposed algorithm used to partitions the edge set of any 3-degenerate graph, as our future work, we plan to modify this method to apply on 4-degenerate graph. And we also try to advance our algorithm to be available for the low-degeneracy Graph.

References

AKIYAMA, J., EXOO, G., AND HARARY, F. Covering and packing in graphs. iii: Cyclic and acyclic invariants. *Mathematica Slovaca 30* (01 1980).

AKIYAMA, J., EXOO, G., AND HARARY, F. Covering and packing in graphs iv: Linear arboricity. *Networks 11*, 1 (1981), 69–72.

ALON, N. The linear arboricity of graphs, 1988.

BASAVARAJU, M., BISHNU, A., FRANCIS, M., AND PATTANAYAK, D. The linear arboricity conjecture for 3-degenerate graphs. *Lecture Notes in Computer Science* (2020), 376–387.

BASAVARAJU, M., AND CHANDRAN, L. S. Acyclic edge coloring of 2-degenerate graphs. *Journal of Graph Theory 69*, 1 (2012), 1–27.

CIOABÙA, S. M. The np-completeness of some edge-partitioning problems.

COLE, R., AND KOWALIK, Ł. New linear-time algorithms for edge-coloring planar graphs. *Algorithmica 50*, 3 (Feb. 2008), 351–368.

CYGAN, M., HOU, J.-F., KOWALIK, , LUŽAR, B., AND WU, J.-L. A planar linear arboricity conjecture. *Journal of Graph Theory 69*, 4 (2012), 403–425.

DUNCAN, C. A., EPPSTEIN, D., AND KOBOUROV, S. G. The geometric thickness of low degree graphs. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (New York, NY, USA, 2004), SCG '04, Association for Computing Machinery, p. 340–346.

ENOMOTO, H., AND PÉROCHE, B. The linear arboricity of some regular graphs. *J. Graph Theory 8* (1984), 309–324.

GULDAN, F. The linear arboricity of 10-regular graphs. *Mathematica Slovaca 36* (1986), 225–228.

GULDAN, F. Some results on linear arboricity. *Journal of Graph Theory 10* (1986), 505–509.

HARARY, F. Covering and packing in graphs, i. *Annals of the New York Academy of Sciences 175* (1970).

HSIAO, D., AND HARARY, F. A formal system for information retrieval from files. *Commun. ACM 13*, 2 (Feb. 1970), 67–73.

NASH-WILLIAMS, C. S. A. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society s1-39*, 1 (1964), 12–12.

STANTON, R. G., COWAN, D., AND JAMES, L. O. Some results on path numbers. In *Proc. Louisiana Conf. on Combinatorics, Graph Theory and computing* (1970), vol. 112, p. 135.

WU, J. On the linear arboricity of planar graphs. *J. Graph Theory 31* (1999), 129–134.

WU, J. The linear arboricity of series-parallel graphs. *Graphs and Combinatorics 16* (2000), 367–372.

WU, J., AND WU, Y. The linear arboricity of planar graphs of maximum degree seven is four. *Journal of Graph Theory 58* (2008), 210–220.