UNIVERSIDADE FEDERAL FLUMINENSE

Matheus Souza D'Andrea Alves

Coloring, Recognition, and Obstructions of (r, ℓ)-Graphs

Niterói

2021

UNIVERSIDADE FEDERAL FLUMINENSE

Matheus Souza D'Andrea Alves

Coloring, Recognition, and Obstructions of (r, ℓ)-Graphs

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Orientador:
Uéverton dos Santos Souza

Niterói

2021

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

A474c Alves, Matheus Souza D'Andrea
Coloring, Recognition, and Obstructions of (r, l)-Graphs / Matheus Souza D'Andrea Alves ; Uéverton dos Santos Souza, orientador. Niterói, 2021.
53 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2021.

DOI: http://dx.doi.org/10.22409/PGC.2021.m.06103997720

1. Algoritmo. 2. Teoria dos Grafos. 3. Ciência da Computação. 4. Otimização (Computação). 5. Produção intelectual. I. Souza, Uéverton dos Santos, orientador. II. Universidade Federal Fluminense. Instituto de Computação. III. Título.

CDD -

Bibliotecário responsável: Debora do Nascimento - CRB7/6368

MATHEUS SOUZA D'ANDREA ALVES

Coloring, Recognition, and Obstructions of (r, ℓ)-Graphs

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Ciência da Computação

Aprovada em Maio de 2021.

BANCA EXAMINADORA

Prof. D.Sc Uéverton dos Santos Souza - Orientador, UFF (Presidente)

Prof. D.Sc. Fábio Protti, UFF

Prof. D.Sc. Julliano Rosa Nascimento, UFG

Niterói

2021

Agradecimentos

Agradeço primeiramente aos meus pais, Haroldo e Valéria, pela dedicação, sacrifício, esforço e carinho em me proporcionar sempre a melhor educação. Agradeço minha noiva Carolina, pelas palavras de ajuda e motivação, e pelo companheirismo durante toda essa jornada.

Agredeço a todos os membros do Instituto de Computação; Professores, pesquisadores, cordenadores, colaboradores e colegas pelo trabalho duro e competente, sempre provendo um ambiente ímpar para o desenvolvimento científico e acadêmico.

Por fim agradeço em especial ao meu orientador Dr. Uéverton dos Santos Souza, pois sem sua paciência, habilidade, orientação e motivação tal trabalho não seria possível.

Resumo

Esse trabalho apresenta um estudo sobre problemas de reconhecimento, partição e coloração em grafos. Mais especificamente, esta dissertação aborda a classe de grafos conhecida como grafos-(r, ℓ) (denotado também em alguns estudos como grafos-(k, ℓ)) que contém os grafos que podem ter seu conjunto de vértices particionado em r conjuntos independentes e ℓ cliques.

Neste trabalho, primeiramente investigamos a complexidade do problema de coloração e lista coloração para grafos-(r, ℓ). Demonstrando como o mesmo se comporta para todos os valores de r e ℓ, investigamos também como utilizar o ferramental de complexidade parametrizada para resolver esses problemas de forma viável computacionalmente em instâncias satisfazendo características específicas. Além disso, apresentamos a equivalência entre coloração de grafos-$(2, 1)$ e lista coloração de grafos bipartidos.

Em seguida, fornecemos uma análise sobre a complexidade do reconhecimento dos grafos-(r, ℓ), provendo um algoritmo de reconhecimento para a grafos-$(2, 2)$ com complexidade de pior caso inferior a do corrente estado da arte.

Por fim, fornecemos uma estratégia para gerar obstruções minimais de grafos-$(2, 1)$ mostrando as obstruções geradas pelo mesmo e seu tempo de execução.

Palavras-Chave— grafos-(r, ℓ), partição, coloração, obstruções, algoritmo FPT portuguese

Abstract

This work presents a study on graph partition, coloring and recognition problems. More specifically, this dissertation addresses the class of graphs known as (r, ℓ) graphs (also called (k, ℓ) graphs in some studies), which contains all graphs that can have their vertex set partitioned in r independent sets and ℓ cliques.

In this work, we first investigated the complexity of the problem of coloring and list-coloring (r, ℓ) graphs. Demonstrating how those problems behaves for all values of r and ℓ, we also investigate how to use the parameterized complexity framework to solve these problems in a computationally feasible for instances satisfying specific characteristics. Besides that, we present the equivalence between the problem of coloring $(2, 1)$ graphs and the problem of list coloring bipartite graphs.

Next, we provide an analysis on the complexity of recognizing (r, ℓ) graphs, providing a recognition algorithm for $(2, 2)$ graphs having worse case complexity better than the current state-of-the-art.

Finally, we provide a strategy to generate minimal obstructions for $(2, 1)$ graphs showing the generated obstructions by the method and its running time.

Keywords— (r, ℓ) graphs, partition, graph coloring, obstruction, FPT.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In the process of solving a graph problem, a common step is to find structural features that makes it easier to be solved. Several of these features can be seen as a kind of partition.

Partitioning in graphs received attention from several different perspectives. The main goal of a partitioning problem in graphs is to partition the set of vertices of a graph into subsets $V_1, V_2, \ldots, V_k$, where $V_1 \cup V_2 \cup \ldots \cup V_k = V$ and $V_i \cap V_j = \emptyset$, $i \neq j$, $1 \leq i \leq k$ and $1 \leq j \leq k$, however, some properties may be required on these subsets of vertices. Feder[1] describes some common properties, that may be *internal*: V_i may be required to be stable or complete, or *external*: V_i and V_j may be required to be completely nonadjacent, that is, no vertex of V_i is adjacent to any vertex of V_j, or completely adjacent, i.e., each vertex in V_i is adjacent to each vertex in V_j.

A well-known partitioning problem is the recognition of *split graphs*, that is, deciding if a given graph G admits a partition of $V(G)$ into a clique C and an independent set I. Among other studies, the recognition of a split graph can be derived from its characterization by finite forbidden subgraphs or its degree sequence, see [2] and references therein. Brandstädt [3] introduced a generalization of split graphs, defined here as an (r, ℓ)-*graph*, a graph whose set of vertices can be partitioned into r independent sets and ℓ cliques. A P versus NP-complete dichotomy for recognizing (r, ℓ)-graphs is well known: the problem is in P if $\max\{r, \ell\} \leq 2$, and is NP-complete otherwise.

The class of (r, ℓ)-graphs and its subclasses have been extensively studied in the literature. For instance, list partitions of (r, ℓ)-graphs were studied by Feder et al [4]. In another paper, Feder et al. [5] proved that recognizing graphs that are both chordal and (r, ℓ) is in P. Kolay et al. [6] and Baste et al. [7] considered the problem of removing a

small number of vertices from a graph so that it becomes (r, ℓ). Faria et al. [8] presented kernelization algorithms for Signed Max Cut on (r, ℓ)-graphs. Alves et al. [9] analyse the complexity of recognizing well-covered (r, ℓ)-graphs.

In this dissertation we developed three studies related to (r, ℓ)-graphs. In Chapter 2, we present a complexity analysis of Graph Coloring on (r, ℓ)-graphs, describing a *Poly* vs. NP-*hard* dichotomy of this problem regarding the parameters r and ℓ. We also analyze the complexity of the problem on (r, ℓ)-graphs under the parameterized complexity perspective, providing a polynomial kernel for the problem in graphs-$(2, 1)$. In Chapter 3, we improved the state of the art on the recognition of $(2, 2)$-graphs, providing a $\mathcal{O}(n^3 \cdot m^2 + m^5)$-time algorithm for recognizing $(2, 2)$-graphs. To the best of our knowledge, the current state of the art on $(2, 2)$-graphs' recognition is the Brandstädt's algorithm [10], which performs in $\mathcal{O}(n^{12})$-time. In Chapter 4, we present a strategy to list all the minimal obstructions of $(2, 1)$-graphs with at most 9 vertices that can be extended to any n vertices. We also produce a web page illustrating all the 1026 minimal obstructions of $(2, 1)$-graphs with at most 9 vertices, which may help future studies on such a class.

Chapter 2

On the coloring of (r,ℓ)-Graphs

In the Graph Coloring problem, we are given as input a graph G, and the objective is to determine the minimum integer k such that G admits a proper vertex k-coloring. In this chapter, we describe a *Poly* vs. NP-*hard* dichotomy of this problem regarding the parameters r and ℓ of (r, ℓ)-graphs, which determine the boundaries of the NP-hardness of Graph Coloring for such classes. We also analyze the complexity of the problem on (r, ℓ)-graphs under the parameterized complexity perspective. We show that given a $(2, 1)$-partition S_1,S_2,K_1 of a graph G, finding an optimal coloring of G is: NP-complete even when K_1 is a maximal clique of size three; XP but W[1]-hard when parameterized by $\min\{|S_1|, |S_2|\}$; FPT and admits a polynomial kernel when parameterized by $\max\{|S_1|, |S_2|\}$. Besides, concerning the case where K_1 is a maximal clique of size three, a P vs. NPc dichotomy regarding the neighborhood of K_1 is provided; furthermore, an FPT algorithm parameterized by the number of vertices having no neighbor in K_1 is presented.

2.1 Overview

Partitioning problems in graphs received attention from several different perspectives. The main goal of a partitioning problem in graphs is to partition the set of vertices of a graph into subsets $V_1, V_2, \ldots, V_k$, where $V_1 \cup V_2 \cup \ldots \cup V_k = V$ and $V_i \cap V_j = \emptyset$, $i \neq j$, $1 \leq i \leq k$ and $1 \leq j \leq k$, however, some properties may be required on these subsets of vertices. Feder et al. (1999) [11] describe some common properties, like *internal*: V_i may be required to be stable (independent set) or complete (clique), or *external*: V_i and V_j may be required to be completely nonadjacent, that is, no vertex of V_i is adjacent to any vertex of V_j, or completely adjacent, i.e., each vertex in V_i is adjacent to any other.

Graph Coloring can be seen as an example of a partitioning problem. In the Graph Coloring problem we receive as input a graph G and we are asked to determine the minimum integer k such that G admits a vertex coloring using at most k colors such that adjacent vertices have different colors. Note that Graph Coloring consists of determining the minimum integer k for which the input graph G can be partitioned into k independent sets. The motivation for studying that problem relies on the strong connections with the research on perfect graphs and the search of polynomial-time algorithms to recognize certain graph classes. From a practical point of view, Lewis (2015) [12] discusses the Graph Coloring problem by relating some applications, e.g., in pattern matching, scheduling, the solution of Sudoku puzzles, and many others. A survey on vertex coloring problems can be found in Malaguti's work [13]. Besides, variants of vertex and edge coloring problems are shown by Dantas, Freitas and Akihiro [14, 15, 16].

In this chapter, we are concerned with Graph Coloring on (r, ℓ)-graphs. We achieve a classical complexity analysis of the problem, determining the boundaries of the NP-hardness for such classes of graphs. This result implies a *Poly* vs. NP-*hard* dichotomy of the problem regarding to the parameters r and ℓ of (r, ℓ)-graphs. Also, we perform a parameterized complexity analysis of the problem. Using reductions between List-Coloring on (r, ℓ)-graphs and Graph Coloring as strategy, we prove that, given a $(2, 1)$-partition S_1,S_2,K_1 of the input graph G, finding an optimal coloring of G is:

1. NP-complete when K_1 is maximal and has size three;
2. para-NP-complete when $|K_1| = 3$ and the number of vertices having *some* neighbors in K_1 is the parameter (we also provide a P vs. NP-hard dichotomy with respect to this parameter);
3. FPT when $|K_1| = 3$ and the number of vertices having *no* neighbors K_1 is the parameter;
4. XP but W[1]-hard when parameterized by $\min\{|S_1|, |S_2|\}$;
5. FPT and admits a polynomial kernel when parameterized by $\max\{|S_1|, |S_2|\}$.

2.2 Preliminaries

Let $G = (V, E)$ be a graph. A *coloring* (or *k-coloring*) of G is a function $c : V \to \{1, \ldots, k\}$. We call $c(u)$ the *color* of $u \in V$. A coloring $c : V \to \{1, \ldots, k\}$ is *proper* if

$c(u) \neq c(v)$ whenever $uv \in E(G)$. If a graph G admits a proper k-coloring, we say that G is *k-colorable*. The smallest number of colors needed to color a graph G properly is called its *chromatic number*, denoted $\chi(G)$. A graph G is *perfect* if $\chi(H) = \omega(H)$, for any induced subgraph H of G.

In what follows, we define the decision problems related to this work.

GRAPH COLORING (DECISION VERSION)
Instance: A graph G; a positive integer k.
Question: Is $\chi(G) \leq k$?

LIST COLORING
Instance: A palette P of colors; a graph G such that each vertex $v \in V(G)$ is equipped with a list $L_v \subseteq P$.
Question: Is there a proper vertex coloring (*list coloring*) c of G such that $c(v) \in L_v$ for each $v \in V(G)$?

Parameterized complexity. We refer the reader to [17, 18, 19, 20] for basic background on parameterized complexity, and we recall here only some definitions.

A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$. For an instance $I = (x, k) \in \Sigma^* \times \mathbb{N}$, k is called the *parameter*. A parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm $\mathcal{A}$, a computable function f, and a constant c such that given an instance $I = (x, k)$, $\mathcal{A}$ (called an FPT-*algorithm*) correctly decides whether $I \in L$ in time bounded by $f(k)|I|^c$.

Within parameterized problems, the class W[1] may be seen as the parameterized equivalent to the class NP of classical optimization problems. Without entering into details (see [17, 18, 19, 20] for the formal definitions), a parameterized problem being W[1]-*hard* may be seen as a strong evidence that this problem is *not* FPT. The canonical example of a W[1]-hard problem is INDEPENDENT SET parameterized by the size of the solution[1].

To transfer W[1]-hardness from one problem to another, one uses an *fpt-reduction*, which given an input $I = (x, k)$ of the source problem, computes in time $f(k)|I|^c$, for some computable function f and a constant c, an equivalent instance $I' = (x', k')$ of the

[1]Given a graph G and a parameter k, the problem is to decide whether there exists an independent set $S \subseteq V(G)$ such that $|S| \geq k$.

target problem, such that k' is bounded by a function depending only on k.

Even if a parameterized problem is W[1]-hard, it may still be solvable in polynomial time for *fixed* values of the parameter; such problems are said to belong to the complexity class XP. Formally, a parameterized problem whose instances consist of a pair (x, k) is in XP if an algorithm can solve it with running time $f(k)|x|^{g(k)}$, where f, g are computable functions depending only on the parameter and $|x|$ represents the input size. For example, Independent Set parameterized by the solution size is easily seen to belong to XP.

2.3 Complexity of Graph Coloring on (r, ℓ)-graphs

In the following, the acronyms P and NPc stand for polynomial-time solvable and NP-hard, respectively.

The chromatic number is easy to determine for null, edgeless, bipartite, complete and split graphs, which are our starting point.

Next, we show our results for other values of r and ℓ. The following result presents two classes of (r, ℓ)-graphs in which Graph Coloring can be solved in polynomial time.

Proposition 1. Graph Coloring *is polynomial-time solvable when restricted to the following classes:*

(a) $(0, 2)$*-graphs;*

(b) $(3, 0)$*-graphs.*

Proof. a) $(0, 2)$-graphs are also known as co-bipartite graphs, which are perfect (see [21]), a well-known class for which the problem is solvable in polynomial time, c.f. [22].

b) Let G be $(3, 0)$-graph. Since it is known that $V(G)$ admits a partition into 3 independent sets, then $\chi(G) \leq 3$. Hence, it suffices to verify if G is an edgeless graph, $\chi(G) = 1$, or if G is bipartite, $\chi(G) = 2$, otherwise $\chi(G) = 3$. Since all of these verifications can be done in polynomial time, we are done. □

We show a complexity result for the class of $(4, 0)$-graphs.

Proposition 2. Graph Coloring *on* $(4, 0)$*-graphs is NP-hard.*

Proof. Since every planar graph is 4-colorable (see [23]), the class of planar graphs is a subclass of $(4,0)$-graphs. Then, the result follows by the NP-completeness of 3-Coloring for planar graphs, c.f. [24]. □

Recall we allow that some parts of the partition of an (r,ℓ)-graph to be empty. This implies that every (r,ℓ)-graph is also an $(r,\ell+1)$-graph and an $(r+1,\ell)$-graph. From this observation and Proposition 2 we obtain the partial classification in Table 2.1.

r \ ℓ	0	1	2	3	4	...	n
0	P	P	P	?	?	...	?
1	P	P	?	?	?	...	?
2	P	?	?	?	?	...	?
3	P	?	?	?	?	...	?
4	NPc	NPc	NPc	NPc	NPc	...	NPc
⋮	⋮	⋮	⋮	⋮	⋮	⋱	NPc
n	NPc	NPc	NPc	NPc	NPc	...	NPc

Table 2.1: Partial P vs. NPc dichotomy for Graph Coloring on (r,ℓ)-graphs.

Note that $(2,1)$-graphs, $(1,2)$-graphs and $(0,3)$-graphs are crucial classes for completely map the complexity of Graph Coloring on (r,ℓ)-graphs, since the complexity of Graph Coloring for their subclasses of (r,ℓ)-graphs are in P.

Next, we discuss bounds for the chromatic number of $(2,1)$-graphs and $(1,2)$-graphs.

Theorem 1. *Let G be either a $(2,1)$-graph or a $(1,2)$-graph. It holds that*

$$\omega(G) \leq \chi(G) \leq \omega(G)+1,$$

and a proper $(\omega(G)+1)$-coloring of G can be obtained in polynomial time.

Proof. For any graph G holds that $\omega(G) \leq \chi(G)$. To determine the upper bound, first observe that a (r,ℓ)-partition of G can be obtained (if any) in polynomial-time when $r,\ell \leq 2$ (see [3]).

Let (S_1,S_2,K_1) be a $(2,1)$-partition of G, without loss of generality, we can assume that K_1 is a maximal clique. Thus, we can obtain a proper $(\omega(G)+1)$-coloring of G as follows: color each vertex of K_1 with a distinct color ($|K_1| \leq \omega(G)$); next for each vertex $v \in S_1$ color v with the same color of some $w \in K_1$ such that $vw \notin E(G)$ (since K_1 is a maximal clique, such a vertex exists); finally color all vertices of S_2 with a novel color $c_{|K_1|+1}$.

Now, let (S_1, K_1, K_2) be a $(1,2)$-partition of G. Since $G[K_1 \cup K_2]$ is a $(0,2)$-graph (subclass of perfect graphs), $G[K_1 \cup K_2]$ can be colored in polynomial time with $\omega(G[K_1 \cup K_2])$ colors (see Proposition 1). Thus, G can be colored with $\omega(G[K_1 \cup K_2]) + 1$ colors. □

By Theorem 1, regarding to $(2,1)$-graphs and $(1,2)$-graphs, it remains to analyse the complexity of checking whether $\chi(G) = \omega(G)$. Theorem 2 provides implications to these questions.

Theorem 2. List Coloring *on* (r, ℓ)*-graphs is polynomial-time reducible to the decision version of* Graph Coloring *on* $(r, \ell + 1)$*-graphs.*

Proof. Let G be an (r, ℓ)-graph, instance of List Coloring. Let L_v be the color list, for every $v \in V(G)$, and $P = \{c_1, c_2, \ldots, c_k\}$ the color palette used in G. From G, create a graph H_G, instance of Graph Coloring, as follows:

- $V(H_G) = V(G) \cup K$ where K is a clique formed by k new vertices $w_1, w_2, \ldots, w_k$ such that each vertex $w_i \in K$ represents the color $c_i \in C$;
- $E(H_G) = E(G) \cup E(K) \cup E'$ where K is a clique, and E' is the set of edges $w_i v$ with $w_i \in K$ and $v \in V(G)$ such that $c_i \notin L_v$.

It remains to show that G has a proper list coloring if and only if H_G is properly k-colorable, for $k = |P|$.

Suppose that G has a proper list k-coloring c. In H_G, for every vertex $v \in V(G)$ assign the same color that has been assigned to v in such list coloring, i.e., $c(v)$. Since K is a clique, K must be colored with k colors. Without loss of generality, we assign color c_i to each vertex w_i, $i \in \{1, \ldots, k\}$. By construction, $w_i v \in E(H_G)$ if and only if $c_i \notin L_v$, hence the assingment of color c_i to vertex w_i, for every $i \in \{1, \ldots, k\}$, completes a proper k-coloring of H_G.

For the converse, suppose that H_G has a proper k-coloring. By construction $|K| = k$ and K is a clique. Then, we assume that color c_i is assigned to vertex w_i, for every $i \in \{1, \ldots, k\}$. Notice that by removing K from H_G, the color assignment to the vertices of $H_G \setminus K$ is also a proper coloring for G. By construction, $w_i v \in E(H_G)$ if and only if $c_i \notin L_v$, hence the assignment for $H_G \setminus K$ is a solution of List Coloring in G. Therefore G has a proper list coloring. □

From Theorem 2 and the NP-completeness of List Coloring for $(1,1)$-graphs ([25]), bipartite graphs ([26]), and co-bipartite graphs ([25]) the following holds.

Corollary 1. Graph Coloring *is NP-hard on* $(1,2)$*-graphs,* $(2,1)$*-graphs and* $(0,3)$*-graphs.*

Since some parts of the partition of an (r,ℓ)-graph may be empty, we accomplish the P vs. NPc dichotomy regarding Graph Coloring on (r,ℓ)-graphs.

2.4 Parameterized complexity analysis for $(2,1)$-graphs

In Theorem 2 we show that List Coloring on (r,ℓ)-graphs is reducible to Graph Coloring on $(r,\ell+1)$-graphs. Next, we remark that such problems are in fact equivalent in many ways.

Theorem 3. *Given a graph* H *together with an* (r,ℓ)*-partition*

$$(S_1, S_2, \ldots, S_r, K_1, K_2, \ldots, K_\ell),$$

such that $\ell \geq 1$*.*

The problem of deciding whether $\chi(H) = |K_\ell|$ *is polynomial-time reducible to* List Coloring *on* $(r,\ell-1)$*-graphs, where the number of colors of the palette is* $|K_\ell|$*.*

Proof. Let $(S_1, S_2, \ldots, S_r, K_1, K_2, \ldots, K_\ell)$ be an (r,ℓ)-partition of a graph H such that $\ell \geq 1$.

An instance of List Coloring can be constructed by setting $P = \{c_w : w \in K_\ell\}$; $L_v = P \setminus \{c_w : w \in N(v) \cap K_\ell\}$ for each vertex v in the input graph $G = H[V \setminus K_\ell]$. At this point, it is easy to see that H is $|K_\ell|$-colorable if and only if G equipped with the color lists is an yes-instance of List Coloring. Note that if K_ℓ is not the largest clique in the given partition, then the resulting instance is a *no*-instance of List Coloring. □

Theorem 3 brings interesting consequences for our study. For example, given Theorem 1, Theorem 2 and Theorem 3, it holds that Graph Coloring on $(2,1)$-graphs is equivalent to List Coloring on bipartite graphs.

Given that List Coloring on bipartite graphs is a very interesting problem. In this section we focus on the parameterized complexity of Graph Coloring on $(2,1)$-graphs.

Let G be a $(2,1)$-graph. Since in polynomial time one can compute a $(2,1)$-partition (S_1, S_2, K_1) of G, we can assume that a $(2,1)$-partition of G is provided together with G as input of Graph Coloring on $(2,1)$-graphs. Then, we may consider three natural parameters for such a problem: $k = |K_1|$, $r_1 = \min\{|S_1|, |S_2|\}$ and $r_2 = \max\{|S_1|, |S_2|\}$. Without loss of generality, we consider $|S_1| \leq |S_2|$.

2.4.1 The size of the clique as a parameter

Recall that, according to Theorem 1, Theorem 2, and Theorem 3, Graph Coloring on $(2,1)$-graphs parameterized by $k = |K_1|$ is equivalent to List Coloring on bipartite graphs parameterized by the number of colors of the palette.

The result of Corollary 2, presented below, suggests that the maximum clique's size is not a useful parameter to produce efficient parameterized algorithms for Graph Coloring on $(2,1)$-graphs. We show that the problem remains NP-complete even if the size of the maximum clique is three. First, we define the PreColoring Extension problem, and we recall a result of [27].

PreColoring Extension
Instance: A positive integer k; a graph G whose some vertices are precolored using at most k colors.
Question: Can the precoloring of G be extended to a proper coloring of G using at most k colors?

Theorem 4 ([27]). PreColoring Extension *is NP-complete for planar bipartite graphs even with color bound* $k = 3$.

Corollary 2. List Coloring *remains NP-complete even when* $|P| = 3$ *and the input graph is planar bipartite.*

Proof. Let G be an instance of PreColoring Extension such that G is planar bipartite and $k = 3$. We know that some vertices $v \in V(G)$ are precolored with color $c_v \in \{1,2,3\}$. By a copy of G we create graph G', which is an instance of List Coloring as follows: If $v \in V(G)$ is precolored with c_v, then $L_v = \{c_v\}$, otherwise $L_v = \{1,2,3\}$. Clearly, if the precoloring of G can be extended to the other vertices of G, then we have a proper 3-coloring for G': it is enough to assign to every $v \in V(G')$ the same color that has been assigned to $v \in V(G)$. The converse is analogous. □

Although the size of the clique/palette is not a useful parameter to produce an FPT algorithm for Graph Coloring/List Coloring, others parameters arise from this. Since List Coloring for bipartite graphs is polynomial-time solvable if either all vertices have lists of size three or all vertices have lists of size two (one can use 2-SAT), two questions regarding the parameterized complexity of List Coloring on bipartite graphs raise:

- (i) the complexity when $|P| = 3$ and the number of vertices with lists of size 1 and 2 is a parameter;
- (ii) the complexity when $|P| = 3$ and the number of vertices with lists of size 3 is a parameter.

We deal with these questions and their relations with Graph Coloring on $(2, 1)$-graphs in the next subsections.

2.4.2 The size of the neighborhood of the clique as a parameter

In this section, we are concerned with $(2, 1)$-graphs with $|K_1| = 3$. If K_1 is maximal and has size 2 then the problem can be solved in polynomial time (see [27, 28]). Thus, $|K_1| = 3$ is the smallest size where Graph Coloring on $(2, 1)$-graphs remains NP-hard.

Let G be a $(2, 1)$-graph with a $(2, 1)$-partition S_1, S_2, K_1 such that K_1 is a maximal clique of size three. By the reduction of Theorem 3, if a vertex $v \in V(G)$ is adjacent to $i \in \{0, 1, 2\}$ vertices of the clique, then $v \in H$ has a list of size $3 - i$. Since the clique K_1 is maximal, then no vertex of H will have a list of size 0.

We show in this section that Graph Coloring on such $(2, 1)$-graphs is para-NP-complete when parameterized by the size of the neighborhood of the clique K_1, i.e., the number of vertices with lists of size one or two considering the equivalent instance of List Coloring. In particular, we show that the problem is NP-hard even when there are only three vertices with a list of size at most two. These hardness results are presented for different cases related to the lists' size, proving a P vs. NPc dichotomy regarding the number of vertices with lists of size one and two for List Coloring on bipartite graphs with $|P| = 3$.

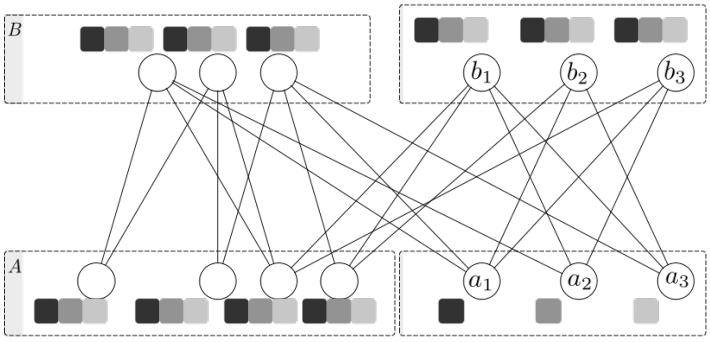


Figure 2.1: Graph G' constructed for Theorem 5.

Instances having only vertices with lists of size one or three

Theorem 5. List Coloring *on bipartite graphs with* $|P| = 3$ *is NP-complete even if every vertex has a list of size three, except for three vertices with unitary lists.*

Proof. Let G be an instance of List Coloring on bipartite graphs with $P = \{c_1, c_2, c_3\}$. In what follows, we construct an instance G' of List Coloring on bipartite graphs with every vertex having lists equal to $\{c_1, c_2, c_3\}$, except for three vertices with unitary lists. See an example in Figure 2.1.

- Let be $G' = G$;
- Add six new vertices to G': a_1, a_2, a_3 and b_1, b_2, b_3, with lists $\{c_1\}$, $\{c_2\}$, $\{c_3\}$, $\{c_1, c_2, c_3\}$, $\{c_1, c_2, c_3\}$, $\{c_1, c_2, c_3\}$, respectively;
- Add to G' the six edges a_2b_1, a_3b_1, a_1b_2, a_3b_2, a_1b_3, a_2, b_3;
- Now, consider A, B a bipartition of G.
  - for every vertex $x \in A$ add xb_i to $E(G')$, for $i \in \{1, 2, 3\}$, if $c_i \notin L_x$;
  - for every vertex $y \in B$ add ya_i to $E(G')$, for $i \in \{1, 2, 3\}$, if $c_i \notin L_y$;
  - for every vertex $v \in A \cup B$ define $L_v = \{c_1, c_2, c_3\}$.

Notice that $|L_{a_i}| = 1$, for every $i \in \{1, 2, 3\}$, then a_1, a_2, and a_3 must be assigned colors 1, 2, and 3, respectively. Consequently, the construction implies that b_1, b_2 and b_3 must be colored with 1, 2, and 3, respectively. Now, it is easy to see that a proper list coloring of G implies a proper list coloring of G' and vice versa. □

We remark that the problem is trivially solvable when the number of vertices with lists of size one and two is 0 since it becomes 3-Coloring on bipartite graphs. However, if that number is 3, the problem is NP-complete. The following result completes this case analysis, where there is no vertex with lists of size two.

Theorem 6. List Coloring *on bipartite graphs with* $|P| = 3$ *can be solved in linear time when the input graph has only two vertices with unitary lists, and the other vertices have lists of size three.*

Proof. Let $G = (A \cup B, E)$ be the input graph. Consider $u, v \in V(G)$ such that $L_u = \{c_u\}$ and $L_v = \{c_v\}$. We consider two cases: (I) $u, v \in A$ (or $u, v \in B$), and (II) $u \in A$, $v \in B$ (or $u \in B$, $v \in A$).

In Case (I), it is enough to assign color c_u to u, color c_v to every vertex in $A \setminus \{u\}$, and a third color to the vertices in B.

In Case (II), if $c_u \neq c_v$, then we assign c_u and c_v to every vertex in A and B, respectively. Otherwise, we may assume that $uv \notin E(G)$, and it is enough to find a proper 2-coloring for the graph $G \setminus \{u, v\}$ avoiding color c_u. □

Instances having only vertices with lists of size two or three

Next, we show that by avoiding unitary lists, six vertices with lists of size two are necessary and sufficient for List Coloring in bipartite graphs to be NP-complete when $|P| = 3$.

Theorem 7. List Coloring *on bipartite graphs with* $|P| = 3$ *can be solved in linear time whenever the input graph has at most five vertices with lists of size two and no vertex with a unitary list.*

Proof. Let $G = (A \cup B, E)$ be the input graph. Since the number of vertices with lists of size two is at most five, then there is a part (that is an independent set) of the bipartition of G, say A, contains at most two vertices with lists of size 2; therefore there is $c \in \bigcap_{v \in A} L_v$. In this case, we assign color c to every vertex $v \in A$, and as every $u \in B$ has a list of size at least two, for each vertex u of the independent set B, there will still be at least one color available in $L_u \setminus \{c\}$. □

The previous proof holds whenever the input is a bipartite graph G having no vertex with a unitary list, $|P| = 3$, and there are not three vertices with distinct lists of size two in some part.

Theorem 8. List Coloring *on bipartite graphs with* $|P| = 3$ *remains* NP*-complete when the input graph has exactly six vertices with lists of size two and no vertex with a unitary list.*

Proof. Let G be an instance of PreColoring Extension on bipartite graphs with color bound $k = 3$. Let $C \subseteq V(G)$ be the set of precolored vertices of G, and let $\{1, 2, 3\}$ be the set of colors of G.

From G, we create a bipartite instance G' of List Coloring as follows.

- Let be $P = \{black, dark\ gray, light\ gray\}$;
- first, set $V(G') = V(G)$ and $E(G') = E(G)$;
- for each current vertex v in $V(G')$ set $L_v = P$;
- add to G' the gadget Γ illustrated in Figure 2.2.

Note that the gadget Γ admits only two possible list colorings. Although different due to the color labels, these colors are isomorphic because they produce the same partition in three independent sets. At this point, we are not interested in the labels assigned to the vertices, but in the three independent sets formed by a partition. Recall that PreColoring Extension can be reinterpreted as a partition problem: given a partition of $G[C]$ into k independent sets, we are asked to determine whether such a partition can be extended to a partition of G into k independent sets.

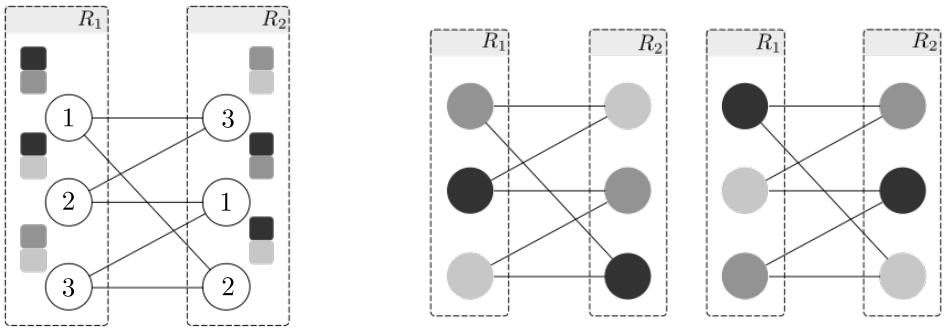


Figure 2.2: Gadget Γ and its possible colorings.

Let I_1, I_2, I_3 be the partition of $V(\Gamma)$ into three independent sets obtained from a list coloring of Γ.

Taking a bipartition of G', we complete the construction of G' by adding edges between the vertices of $V(\Gamma)$ and C as follows.

- for each $v \in V(G') \cap C$ precolored with a color $j \in \{1, 2, 3\}$, add an edge from v to the two vertices of Γ that are neither in I_j nor in the part that contains v.

The edges added in the previous step preserve the bipartition of G' and force any partition of G' into three independent sets to contain the vertices v with $c(v) = j$ in the same part as the vertices in I_j. Thus, if G' is list colorable, then G is 3-colorable respecting the precoloring. The converse is clear. □

Instances having vertices with lists of size one and two

Regarding List Coloring on bipartite graphs G having a palette with three colors and no vertex with a list of size two, Theorem 5 and Theorem 6 provide a dichotomy concerning the number of vertices with a unitary list. (with at most two vertices, it is polynomial; but with three vertices, it is NP-complete) Analogously, Theorem 7 and Theorem 8 provide a dichotomy concerning the number of vertices having a list of size two, when no list of size one is allowed (with at most five vertices, it is polynomial; but with six vertices, it is NP-complete).

At this point, the reader may be wondering about cases where lists of size one and two are allowed. Since one can add isolated vertices with lists of size one or two without changing the instance's answer, we have natural monotonicity. It implies that polynomial cases have at most two vertices with unitary lists and at most five vertices with lists of size two.

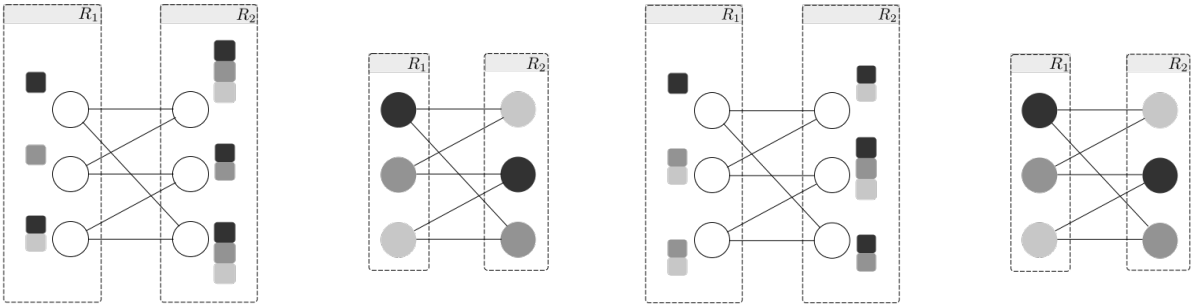
Therefore, to obtain a complete map of the complexity of List Coloring on bipartite graphs having a palette of size three, next, we consider the remaining cases.

Corollary 3. List Coloring *on bipartite graphs G with $|P| = 3$ is NP-complete even restrict to instances*

1. *with exactly two vertices with a unitary list and two vertices with lists of size two;*
2. *with exactly one vertex with a unitary list and four vertices with lists of size two.*

Proof. The proof for cases 1 and 2 is similar to that of Theorem 8 replacing the gadget Γ by the gadgets Γ' and Γ'' shown in Figure 2.3(a) and Figure 2.3(b), respectively. □

Theorem 9. List Coloring *on bipartite graphs with $|P| = 3$ can be solved in linear time whenever the input graph G has at most two vertices with a unitary list and one vertex with a list of size two.*



(a) Γ' and its only possible list coloring

(b) Γ'' and its only possible list coloring

Figure 2.3: Gadget Γ' (two unitary lists and two lists of size two), and gadget Γ'' (one unitary list and four lists of size two).

Proof. Let G' be the subgraph of G induced by the vertices with a list of size one or two. We assume that G' has a list coloring; otherwise, G is a trivial *no*-instance.

Let be $G = (A \cup B, E)$, and let v_1^a, v_1^b be the two vertices of G with a unitary list, and v_2 be the vertex of G with a list of size two. We can assume that these vertices exist since we can add isolated vertices.

If a part of G, say A, contains v_1^a and v_1^b then every vertex in $B \setminus \{v_2\}$ can be colored with a color that is not in $L_{v_1^a} \cup L_{v_1^b}$; since v_2 has a list of size two and G' has a list coloring, regardless of which part contains v_2, a color for v_2 still can be assigned, and after that each vertex of $A \setminus \{v_2\}$ will still have at least one color available.

If $\{v_1^a, v_2\} \subseteq A$ and $v_1^b \in B$ then by assigning the same color $c \in L_{v_2} \setminus L_{v_1^b}$ to all vertices of $A \setminus \{v_1^a\}$, and fixing the color of v_1^a, all vertices of B will still have at least one color available (if $L_{v_1^a} = L_{v_1^b}$, v_1^a and v_1^b are non-adjacent). □

Next, we finish our analysis of cases regarding the number of lists with size one and two (the complete classification is summarized in Table in Concluding remarks).

Theorem 10. List Coloring *on bipartite graphs with* $|P| = 3$ *can be solved in linear time whenever the input* G *has exactly one vertex with a unitary list and at most three vertices with a list of size two.*

Proof. Let be $G = (A \cup B, E)$. Let v_1 be the vertex of G with a unitary list, and let v_2^a, v_2^b, v_2^c be the vertices of G with lists of size two.

If $v_1 \in A$ and $\{v_2^a, v_2^b, v_2^c\} \subseteq B$ then a list coloring can be found assigning the color in L_{v_1} for all vertices of A. If $v_2^a \in A$ and $\{v_1, v_2^b, v_2^c\} \subseteq B$ then, similarly, a list coloring can be found assigning a color in $L_{v_2^a} \setminus L_{v_1}$ for all vertices of A.

Thus, without loss of generality, we can assume that $\{v_1, v_2^a\} \subseteq A$ and $\{v_2^b, v_2^c\} \subseteq B$. If $L_{v_1} \cap Lv_2^a \neq \emptyset$ then a list coloring can be found assigning the color in L_{v_1} for all vertices of A. If $(L_{v_2^b} \cap Lv_2^c) \setminus L_{v_1} \neq \emptyset$ then a list coloring can be found assigning a color in $(L_{v_2^b} \cap Lv_2^c) \setminus L_{v_1}$ for all vertices of B. At this point, we can assume that $L_{v_1} = \{c_1\}$, $L_{v_2^a} = \{c_2, c_3\}$, $L_{v_2^b} = \{c_1, c_2\}$, and $L_{v_2^c} = \{c_1, c_3\}$. If v_1 is non-adjacent to v_2^b then a list coloring can be found assigning the color c_1 to v_1 and v_2^b, color c_2 to all vertices in $A \setminus \{v_1\}$, and color c_3 to all vertices in $B \setminus \{v_2^b\}$ (a similar argument holds to v_2^c). If v_2^a is non-adjacent to v_2^b then a list coloring can be found assigning the color c_2 to v_2^a and v_2^b, color c_1 to all vertices in $A \setminus \{v_2^a\}$, and color c_3 to all vertices in $B \setminus \{v_2^b\}$ (a similar argument holds to v_2^c). Finally, if v_1, v_2^a, v_2^b, v_2^c induce a C_4 then G is a *no*-instance. □

2.4.3 The size of the non neighborhood of the clique as parameter

As seen in the previous section, vertices of Graph Coloring on $(2,1)$-graphs that are nonadjacent to the clique, when transformed to vertices of a List Coloring instance, have lists of size three. Our aim in this section is solving List Coloring on bipartite graphs with $|P| = 3$ when the parameter is the number of vertices with lists of size three. That is equivalent to Graph Coloring on $(2,1)$-graphs parameterized by the number of vertices having no neighbor in K_1, where $|K_1| = 3$.

Theorem 11. List Coloring *on bipartite graphs with* $|P| = 3$ *is* FPT *when parameterized by the number of vertices with lists of size three.*

Proof. Let k be the number of vertices with lists of size three. A bounded search tree can be constructed as follows:

1. Consider the instance of List Coloring as a root vertex.
2. Choose one of the k vertices v with lists of size three.
3. For every available color of v, create a branch representing an instance with $k-1$ vertices with lists of size three, by fixing such a color for v.
4. Perform the previous steps i times until level $k - i = 0$.

Notice that at the end of the algorithm, we have a tree of order 3^k, with leaves corresponding to instances containing no vertex with a list of size three. Then, it is possible to run in polynomial time the algorithm proposed by Hujter [28] for every leaf, completing in $3^k n^{\mathcal{O}(1)}$ time a procedure that solves List Coloring. □

2.4.4 The size of the independent sets as parameters

Contrasting with the NP-hardness when $|K_1| = 3$, Graph Coloring on $(2,1)$-graphs G can be solved in $n^{\mathcal{O}(r_1)}$ ($r_1 = \min\{|S_1|, |S_2|\}$) as follows: construct the equivalent instance of List Coloring; let be $|S_1| \leq |S_2|$; every possible coloring of S_1 can be enumerate in $\mathcal{O}(n^{|S_1|})$; by fixing each coloring of S_1 one can check in linear time if every vertex of S_2 still has an available color in its list; if a list coloring for the bipartite graph is found then the chromatic number of G is $|K_1|$; otherwise, by Theorem 1, the chromatic number of G is $|K_1| + 1$. Therefore, the following holds.

Theorem 12. Graph Coloring *on graphs with a* $(2,1)$*-partition* S_1, S_2, K_1 *is in* XP *when parameterized by* $\min\{|S_1|, |S_2|\}$.

Using a reduction from Multicolored Clique, Fellows [26] showed that List Coloring on bipartite graphs is W[1]-hard when parameterized by the vertex cover number. In such a construction, a minimum vertex cover of the resulting instance G corresponds to a part of the input's bipartition. This fact implies that List Coloring on bipartite graphs is W[1]-hard when parameterized by the size of the smallest part. Therefore, by Theorem 2, the following holds.

Theorem 13. Graph Coloring *on graphs with a* $(2,1)$*-partition* S_1, S_2, K_1 *is* W[1]*-hard when parameterized by* $\min\{|S_1|, |S_2|\}$.

Now, let G be a $(2,1)$-graph with a $(2,1)$-partition S_1, S_2, K_1 where $r_1 = \min\{|S_1|, |S_2|\}$ and $r_2 = \max\{|S_1|, |S_2|\}$. Theorem 13 suggests that parameterizing Graph Coloring by r_1 is not a useful approach to design an FPT algorithm for that problem. However, when taking r_2 as parameter, r_1 is also bounded. This makes possible to produce a simple kernelization algorithm.

Theorem 14. Graph Coloring *on graphs with a* $(2,1)$*-partition* S_1, S_2, K_1 *admits a polynomial kernel when parameterized by* $\max\{|S_1|, |S_2|\}$.

Proof. Let H be a graph with a $(2,1)$-partition S_1, S_2, K_1. Let be $r_2 = \max\{|S_1|, |S_2|\}$. If k is the size of K_1, then we need at least k colors for a proper k-coloring of H. Again, we consider the equivalent instance of List Coloring on bipartite graphs with $|P| = k$ (recall Theorem 3) to obtain results for Graph Coloring on $(2,1)$-graphs.

Let G be a bipartite graph, an instance of List Coloring, obtained from H as described in Theorem 3. If $|L_v| > |N_G(v)|$, for some $v \in V(G)$, there always is an

available color to be assigned to v. This implies that v can be removed from G without affecting a coloring of $G \setminus \{v\}$. Create a graph G' by removing from G every vertex v such that $|L_v| > |N_G(v)|$. We have that $|V(G')| \leq 2 \cdot r_2$ and $|L_u|$, for every $u \in V(G')$, is bounded by r_2. Thus, the palette of G' has at most $2 \cdot r_2^2$ colors. Then, by applying the construction presented in Theorem 2 we obtain an equivalent $(2,1)$-graph G^*such that $|V(G^*)| \leq 2 \cdot r_2^2 + 2 \cdot r_2$. Therefore, G^* is a polynomial kernel for H. □

2.5 Concluding remarks

This chapter present a complexity analysis of Graph Coloring on (r, ℓ)-graphs, and describe a *Poly* vs. NP-*hard* dichotomy of the problem regarding the parameters r and ℓ of (r, ℓ)-graphs, which determine the boundaries of the NP-hardness of Graph Coloring for such classes. Table 2.2 summarizes at high granularity our complete mapping of the complexity Graph Coloring on (r, ℓ)-graphs.

Graph Coloring on (r, ℓ)-graphs

r \ ℓ	0	1	2	3	4	...	n
0	P	P	P	NPc	NPc	...	NPc
1	P	P	NPc	NPc	NPc	...	NPc
2	P	NPc	NPc	NPc	NPc	...	NPc
3	P	NPc	NPc	NPc	NPc	...	NPc
4	NPc	NPc	NPc	NPc	NPc	...	NPc
⋮	⋮	⋮	⋮	⋮	⋮	⋱	NPc
n	NPc	NPc	NPc	NPc	NPc	...	NPc

Table 2.2: P vs. NPc dichotomy regarding the values of r and ℓ for Graph Coloring on (r, ℓ)-graphs.

Since the complement of a (r, ℓ)-graph is a (ℓ, r)-graph, and a coloring (partition into independent sets) correspond to a clique cover in the complement, it holds that our results also provide a complete classification between *Poly* and NP-*hard* for Clique Cover on (r, ℓ)-graphs with respect to r and ℓ.

We also perform a parameterized complexity analysis of the problem. Using reductions between List-Coloring on (r, ℓ)-graphs and Graph Coloring as strategy, we show that given a $(2,1)$-partition S_1,S_2,K_1 of the input graph G the problem of finding an optimal coloring of G is:

- NP-complete when K_1 has size three;

- XP but W[1]-hard when parameterized by $\min\{|S_1|, |S_2|\}$;
- FPT and admits a polynomial kernel when parameterized by $\max\{|S_1|, |S_2|\}$.

The existence of XP-time algorithms implies that for any constant value of the parameter under analysis, the problem will be solved in polynomial time. Such algorithms can be easily obtained for Graph Coloring on $(2,1)$-graphs with respect to $\min\{|S_1|, |S_2|\}$ or $\max\{|S_1|, |S_2|\}$ as parameter. However, by Corollary 2, they cannot be obtained concerning $|K_1|$ unless P= NP (para-NP-hardness). Although XP-time algorithms are of some interest, in Parameterized Complexity we look for FPT algorithms that result in more efficient methods than the typically naive XP-time algorithms. A more refined analysis leads us to conclude that while it is possible to develop an FPT algorithm regarding $\max\{|S_1|, |S_2|\}$ the same is unlikely for $\min\{|S_1|, |S_2|\}$ because such a problem is W[1]-hard. Besides, every FPT problem has a kernel, but such kernels do not necessarily have polynomial-size concerning the parameter, a desirable feature. We also show that Graph Coloring on $(2,1)$-graphs parameterized by $\max\{|S_1|, |S_2|\}$ admits a polynomial kernel. A compendium of these results is presented in Table 2.3.

Graph Coloring on $(2,1)$-graphs

Class \ Parameter	$\lvert K_1\rvert$	$\lvert S_1\rvert$	$\lvert S_2\rvert$
para-NP-hard	✓ (hard even for 3)	–	–
XP (P for const. values)	–	✓	✓
W[1]-hard	–	✓	–
FPT	–	–	✓
Polynomial Kernel	–	–	✓

Table 2.3: Parameterized Complexity of Graph Coloring on $(2,1)$-graphs regarding a $(2,1)$-partition S_1,S_2,K_1.

From Theorem 1, Theorem 2, and Theorem 3 it holds that Graph Coloring on $(2,1)$-graphs parameterized by $|K_1|$ is equivalent to List Coloring on bipartite graphs parameterized by the number of colors of the palette ($|P|$). In particular, the NP-hard case where K_1 is a maximal clique of size three correspond to List Coloring on bipartite graphs with $|P| = 3$.

Although the size of the clique/palette is not a useful parameter to produce an FPT algorithm for Graph Coloring/List Coloring, others parameters arise from this. Since List Coloring for bipartite graphs is polynomial-time solvable if either all vertices have lists of size three or all vertices have lists of size two (one can use 2-SAT), two questions regarding the parameterized complexity of List Coloring on bipartite graphs raise: (i) the complexity when $|P| = 3$ and the number of vertices with lists of size 1 and 2 is a parameter; (ii) the complexity when $|P| = 3$ and the number of vertices with lists of size 3 is a parameter.

Regarding (i) a P vs. NPc dichotomy with respect to the number of vertices with lists of size 1 and 2 is provided. Our complete classification is summarized in Table 2.4. By the equivalence between List Coloring on bipartite graphs and Graph Coloring on $(2,1)$-graphs, this results can be seen as a dichotomy regarding the neighborhood of K_1. Therefore, as a by-product of this mapping, one can conclude that Graph Coloring on $(2,1)$-graphs with a $(2,1)$-partition S_1, S_2, K_1, where K_1 is a maximal clique of size three, can be solved in polynomial time whenever the number of edges crossing from K_1 to $S_1 \cup S_2$ is at most five, but it is NP-hard when there are six edges crossing from K_1 to $S_1 \cup S_2$ (each vertex of $S_1 \cup S_2$ has at most two neighbors in K_1).

List Coloring on bipartite graphs with $|P| = 3$

$\lvert L_v\rvert = 2$ \ $\lvert L_v\rvert = 1$	0	1	2	3	...	n
0	P	P	P	NPc	...	NPc
1	P	P	P	NPc	...	NPc
2	P	P	NPc	NPc	...	NPc
3	P	P	NPc	NPc	...	NPc
4	P	NPc	NPc	NPc	...	NPc
5	P	NPc	NPc	NPc	...	NPc
6	NPc	NPc	NPc	NPc	...	NPc
⋮	⋮	⋮	⋮	⋮	⋱	NPc
n	NPc	NPc	NPc	NPc	...	NPc

Table 2.4: P vs. NPc dichotomy regarding the number of vertices with lists of size one and two for List Coloring on bipartite graphs with $|P| = 3$. In green is highlighted the cases that correspond to instances of Graph Coloring on graphs G having a $(2,1)$-partition S_1, S_2, K_1 with 5 edges crossing from K_1 to $S_1 \cup S_2$, and in red is highlighted the cases that correspond to instances of Graph Coloring on graphs G having a $(2,1)$-partition S_1, S_2, K_1 with 6 edges crossing from K_1 to $S_1 \cup S_2$.

Contrasting with the results presented in Table 2.4 concerning the neighborhood of K_1, we answer (ii) presenting an FPT algorithm parameterized by the number of vertices having no neighbor in K_1.

Chapter 3

On the recognition of (r,ℓ)-Graphs

In the literature it is well known that the recognition of (r, ℓ)-graphs is polynomial-time solvable whenever $\max\{r, \ell\} \leq 2$ and NP-complete otherwise (see [10, 3, 29, 30]). However, there are still some questions to be clarified. In 1996 [3], Brandstädt published in Discrete Applied Mathematics a paper containing a $\mathcal{O}(n^3)$-time algorithm for recognizing $(2, 1)$ and $(1, 2)$-graphs and a $\mathcal{O}(n^4)$-time algorithm for recognizing $(2, 2)$-graphs. However, in 1998 [29], Brandstädt published a corrigendum stating that the previously presented algorithms were not correct. As a reference to the correct versions of such algorithms Brandstädt cites the manuscript [10] of his own, that is unpublished but available in a repository. In such a manuscript the complexity of recognizing $(2, 1)$-graphs and $(1, 2)$-graphs is $\mathcal{O}(n^4)$ while the complexity of recognizing $(2, 2)$-graphs is $\mathcal{O}(n^{12})$. Later, in 1998, Brandstädt published a $\mathcal{O}((n + m)^2)$-time algorithm for $(1, 2)$ and $(2, 1)$-graph recognition. However, in this last paper there is no mention of the recognition of $(2, 2)$-graphs. To the best of our knowledge, nowadays there is still no correct algorithm for $(2, 2)$-graph recognition published in a journal, being the $\mathcal{O}(n^{12})$-time algorithm of 1984, available in repository, the unique reference for this algorithm. Therefore, in this chapter we discuss the recognition of such a class. Our main goal is to develop a more efficient algorithm to recognize $(2, 2)$-graphs.

3.1 State of the art

In this section, we will discuss the current state of the computational complexity to recognize (r, ℓ)-graphs. We will fulfill the following table according we expose the strategies and how those can be used to enlighten the more complex results.

The most trivial result is the recognition of the $(1,0)$-graphs, as in order to recognize it we just need to know whether $|E(G)| > 0$. Thus, $(1,0)$-graphs can be solved in $\mathcal{O}(1)$ time when the set of edges is given. Otherwise, given an adjacency list of the graph, one can decide whether such a graph has empty edge set in $\mathcal{O}(n)$ time.

r \ ℓ	0	1	2	3	4	...
0	-	?	?	?	?	...
1	$\mathcal{O}(n)$	?	?	?	?	...
2	?	?	?	?	?	...
3	?	?	?	?	?	...
4	?	?	?	?	?	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Table 3.1: Incomplete complexity map of the recognition of (r,ℓ)-graphs.

Next, we consider the $\mathcal{O}(n+m)$ running time cases.

In [31], König showed that to recognize bipartite graphs ($(2,0)$-graphs) takes $\mathcal{O}(n+m)$ time. For complete graphs ($(0,1)$-graphs), is enough to check whether $|E(G)| = \frac{n(n-1)}{2}$: if $|E(G)| \neq \frac{n(n-1)}{2}$ then we have a negative answer; and, if $|E(G)| = \frac{n(n-1)}{2}$ by checking vertex by vertex if its neighborhood contains all other vertices (i.e., the graph is simple), we can recognize complete graphs in $\mathcal{O}(n+m)$ time.

For co-bipartite graphs, it is enough to verify whether its complement is a bipartite graph. At this point, we need to take care because if G is sparse then the construction of $\overline{G}$ may takes $\mathcal{O}(n^2)$ time. However, by the pigeonhole principle, we know that co-bipartite graphs have a clique of size $\lceil n/2 \rceil$; therefore, if $m < \frac{n^2}{4} - \frac{n^2}{2}$ then we can safely return a negative answer, otherwise we have $m = \mathcal{O}(n^2)$ and we can compute its complement. Thus, the recognition of co-bipartite graphs can also be performed in $\mathcal{O}(n+m)$ time.

The recognition of split graphs can be done using their vertex degrees sequences [32]. Let the degree sequence of a graph G be $d_1 \geq d_2 \geq \ldots \geq d_n$, , and let k be the largest value of i such that $d_i \geq i-1$. Then G is a split graph if and only if (c.f. [32])

$$\sum_{i=1}^{k} d_i = k(k-1) + \sum_{i=k+1}^{n} d_i$$

If this is the case then the k vertices with the largest degrees form a maximum clique in G, and the remaining vertices constitute an independent set.

Since compute the degree of each vertex take $\mathcal{O}(n+m)$ time, and order the set of vertices with respect to its degree can be done in linear time using the Counting Sort algorithm, it holds that the recognition of split graphs can also be done in $\mathcal{O}(n+m)$ time.

Now, we consider the NP-complete cases.

In [33], it is shown that 3-Coloring is NP-complete; therefore, the recognition of $(3,0)$-graphs as well as the the recognition of $(0,3)$-graphs are NP-complete.

By adding isolated cliques and/or dominating independent sets, it is easy to see that we have monotonicity regarding the complexity of recognizing (r,ℓ)-graphs. Therefore, the recognition of (r,ℓ)-graph is NP-Complete whenever r or ℓ are greater or equal to three.

r \ ℓ	0	1	2	3	4	...
0	-	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	NPc	NPc	...
1	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$	?	NPc	NPc	...
2	$\mathcal{O}(n+m)$	?	?	NPc	NPc	...
3	NPc	NPc	NPc	NPc	NPc	...
4	NPc	NPc	NPc	NPc	NPc	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Table 3.2: Partial complexity mapping of the (r,ℓ)-partite recognition problem.

3.1.1 Brandstädt's strategy

In this section, we consider the frontier cases $(1,2)$, $(2,1)$ and $(2,2)$, that were subject of studies by Brandstädt et al. [10, 3, 29, 30]. First, we will describe the property key of the algorithms designed by Brandstädt to recognize a $(2,1)$-graph.

Let G be a graph. If G has a $(2,1)$-partition then any vertex $v \in V(G)$ satisfies the following condition:

- (N1) $N(v)$ induces a split graph,
- (N2) or $V(G) \setminus N(v)$ induces a bipartite graph.

More specifically, if G has a $(2,1)$-partition I_1, I_2, C then (N2) holds for every vertex $v \in C$ and (N1) holds for the vertices $v \in I_1 \cup I_2$.

Therefore, for a graph G, let A be the set of vertices that satisfies (N1), and let B be the set of vertices that satisfies (N2). If $A \cup B \neq V(G)$ then G is not a $(2,1)$-graph. If $R = A \cap B = \emptyset$ then G has a $(2,1)$-partition if and only if A induces a bipartite graph and B induces a clique. Otherwise, $A \cup B = V(G)$, $R = A \cap B \neq \emptyset$. In the latter case, one must be guessing the vertices of R that must be in the clique as well as in the bipartite part of a $(2,1)$-partition of G, if any. In [10], Brandstädt showed how to perform that in $\mathcal{O}(n^4)$. Thus, the first Brandstädt's recognition algorithm for $(2,1)$-graphs is performed in $\mathcal{O}(n^4)$ time.

Since the complement graph of G can be constructed in $\mathcal{O}(n^2)$ time, from Brandstädt's algorithm also holds that the recognition of a $(1,2)$-graph can be performed in $\mathcal{O}(n^4)$ time. Later, Brandstädt et al. [30] improve such algorithm performing the recognition of these classes in $\mathcal{O}((n+m)^2)$-time.

In 1996 [3], Brandstädt published a paper containing a $\mathcal{O}(n^4)$-time algorithm for $(2,2)$-graph recognition. However, in 1998 [29], Brandstädt published a corrigendum stating that the previously presented algorithm was not correct. As a reference to the correct version of such algorithm Brandstädt cites the manuscript [10] of his own available in a repository. In such a manuscript the complexity of recognizing $(2,2)$-graphs is $\mathcal{O}(n^{12})$. To the best of our knowledge, this is the unique known algorithm to recognize $(2,2)$-graphs.

Therefore, the current map of the complexity of recogning (r,ℓ)-graphs is as follows.

r \ ℓ	0	1	2	3	4	...
0	-	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	NPc	NPc	...
1	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2+m^2)$	NPc	NPc	...
2	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2+m^2)$	$\mathcal{O}(n^{12})$	NPc	NPc	...
3	NPc	NPc	NPc	NPc	NPc	...
4	NPc	NPc	NPc	NPc	NPc	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Table 3.3: Current complexity map of the recognition of (r,ℓ)-graphs.

3.2 Our approach for $(2,2)$-graph recognition

In this section, we will discuss our approach to obtain $(2,2)$-partitions (if any) more efficiently. As Brandstädt we will use the neighborhood and non-neighborhood pattern of the vertices in order to reduce our analysis to instances G for which a $(3,3)$-partition of

G is known.

Our approach differs from Brandstädt in the procedure for given a $(3,3)$-partition of G to decide the vertices of the tripartite part that should be allocated in the co-bipartition of a $(3,2)$-partition (if any).

3.2.1 Finding a (3,3)-partition

As seen on the Brandstädt's algorithm every vertex of a $(2,2)$-graph must compliance at least one of the following rules:

- (N3) $N(v)$ induces a $(1,2)$-graph.
- (N4) or, $V(G) \setminus N(v)$ induces a $(2,1)$-graph.

Similarly to the strategy for $(2,1)$-graphs, if G has a $(2,2)$-partition I_1, I_2, K_1, K_2 then (N3) holds for every vertex $v \in K_1 \cup K_2$ and (N4) holds for the vertices $v \in I_1 \cup I_2$.

If every vertex match exactly one of these rules, we can check the derived partitions to certify that they form a $(2,2)$-partition in $\mathcal{O}(n+m)$ time. On the other hand, if any vertex does not match N3 or N4 then the graph does not have a $(2,2)$-partition. However, when a vertex matches both N3 and N4 the graph could have a $(2,2)$-partition but clearly it has a $(3,3)$-partition as the union of the $(1,2)$-subgraph induced by its closed neighborhood, and the $(2,1)$-subgraph induced by its non-neighborhood.

Therefore, the following holds.

Theorem 15. *There is an algorithm that in $\mathcal{O}(n^3 + n \cdot m^2)$ time either correctly asserts if G is a $(2,2)$-graph or obtains a $(3,3)$-partition of G.*

Proof. For each vertex v in the graph we apply the following:

- Verify if its open neighborhood is $(1,2)$ ($\mathcal{O}(n^2 + m^2)$);
- Verify if the complement of its neighborhood is $(2,1)$ ($\mathcal{O}(n^2 + m^2)$).

Such verification spans three sets (one containing the vertices that matches only the first verification (B), one containing the ones that matches only the second verification (C) and one containing the vertices that matches both conditions (R)). It is easy to see that when some vertex does not match any condition then the graph is not a $(2,2)$-graph.

Therefore, in $\mathcal{O}(n^3 + n \cdot m^2)$ time we can find these sets or conclude that G is not $(2,2)$. If $|R| = 0$ then, by the definition of $(2,2)$-graphs, G is $(2,2)$ if and only if C induces a co-bipartite graph and B induces a bipartite graph. However, if $|R| > 0$ then for any vertex $v \in R$, $N(v)$ is known to induce a $(1,2)$-graph, and $V(G) \setminus N(v)$ induces a $(2,1)$-graph, which gives us a $(3,3)$-partition of G in the desired time. □

At this point, we have an algorithm that either solve the problem or obtains a $(3,3)$-partition of G. Thus, our problem is reducible to determine whether a $(3,3)$-graph G with a $(3,3)$-partition $I_1, I_2, I_3, K_1, K_2, K_3$ also admits a (2,2)-partition. We can see that obtain a $(2,2)$-partition from a $(3,3)$-partition of G is in fact the work of correctly rearranging some vertex from the tripartition (sparse) to the co-tripartition (dense) as well as move a few vertices from the dense part to the sparse one (that should becomes co-bipartite and bipartite, respectively), i.e., from a sparse-dense partition we must find another more particular.

To find these vertices we need to find a set of vertices in the tripartition that when removed from such tripartition makes it a bipartition, this kind of set is known as an *odd cycle transversal*. In addition, we need to assure that these vertices can be placed in the dense part without breaking the desired $(2,2)$-partition. The same holds for the co-tripartition.

3.2.2 Applying an algorithm for Triangle Vertex Deletion

At this point, we are given a graph G with a $(3,3)$-partition $I_1, I_2, I_3, K_1, K_2, K_3$ of $V(G)$.

From the $(3,3)$-partition $I_1, I_2, I_3, K_1, K_2, K_3$ the problem of obtaining a $(2,2)$-partition (if any) can be seen as rearranging the sparse-dense partition moving some vertices from the tripartition to the co-tripartition and vice versa.

Notice that at most 6 vertices can be moved from the tripartition to the dense part, since $G[I_1 \cup I_2 \cup I_3]$ is K_4-free. So, at most 3 vertices from $G[I_1 \cup I_2 \cup I_3]$ should be in the same clique of the resulting co-bipartition (if any).

The Odd Cycle Transversal (OCT) problem can be described as the problem of finding a k-set of vertices of a graph such that when removed from the input graph, makes it a bipartite one. This problem is known to be NP-complete, however, Reed, Smith and Vetta presented a fixed-parameter tractable algorithm for this problem. Such algorithm obtains the desirable set (if any) in time $\mathcal{O}(3^k \cdot kn(n+m))$ [34]. Note that when

k is a constant such a algorithm is performed in $\mathcal{O}(n^2 + n \cdot m)$ time. Thus, in order to identify the vertices that should be moved to the dense part (if any), we first must check, in $\mathcal{O}(n^2 + n \cdot m)$ time, whether $G[I_1 \cup I_2 \cup I_3]$ has an odd cycle transversal of size at most six. If the minimum odd cycle transversal of $G[I_1 \cup I_2 \cup I_3]$ has size greater than three then we are dealing with a *no*-instance.

Now, suppose that G has a $(2,2)$-partition I', I'', K', K''. Let S be the set of vertices to be moved from the tripartition to the dense part, i.e. $S = (I_1 \cup I_2 \cup I_3) \cap (K' \cup K'')$. Let $S' = S \cap K'$ and $S'' = S \cap K''$. Now, we discuss how to find $S = S' \cup S''$:

1. If $|S'| = 3$ or $|S''| = 3$ then $G[I_1 \cup I_2 \cup I_3]$ has triangles. In particular, S contains a minimal *triangle vertex deletion set* (also known as triangle transversal). Therefore, we first "guess" such a minimal triangle vertex deletion set of size at most six. Thus, we can reduce our problem to the case where $|S'| \leq 2$ and $|S''| \leq 2$, by applying the first steps of a simple FPT algorithm for Triangle-free Vertex Deletion parameterized by $k = 6$ as follows:

   - find a triangle T_i (if $G[I_1 \cup I_2 \cup I_3]$ has no triangle, there is no solution with $|S'| = 3$ or $|S''| = 3$);
   - at least one vertex of T_i must be in S, so we can "guess" such a vertex by branching recursively and looking for a solution of size $k - 1$ in the graph resulting from the removal of the guessed vertex.
   - the previously described step is applied until we obtain graphs without triangles or $k = 0$;

Note that the previously described procedure can be performed in $\mathcal{O}(3^k \cdot n \cdot m)$ time. And, since $k = 6$, it holds that the previous step is performed in $\mathcal{O}(n \cdot m)$ time. Also, observe that the previous procedure enumerate every minimal triangle vertex deletion set of size at most k (here $k = 6$) of the input, and there are $\mathcal{O}(3^k)$ minimal triangle vertex deletion set of size at most k.

In addition, since we are looking for a minimal triangle vertex deletion set R contained in S, it holds that R must induce a co-bipartite graph ($R \subseteq S \subseteq (K' \cup K'')$). At this point, we proceed as follows:

1 **For each** triangle vertex deletion set R of $G[I_1 \cup I_2 \cup I_3]$ such that $G[R]$ is co-bipartite and $|R| \leq 6$ **do**

2 **For each** co-bipartition R', R'' of R such that $|R'|, |R''| \leq 3$ **do**

3 **Enumerate each pair** $S' \supseteq R'$, $S'' \supseteq R''$ such that
$|S' \setminus R'|, |S'' \setminus R''| \leq 2$;
$|S'|, |S''| \leq 3$;
S', S'' are cliques.

At this point, since $k = 6$ it holds that steps 1 and 2 are performed in $\mathcal{O}(1)$. After that, given R' and R'', since $|S' \setminus R'|, |S'' \setminus R''|$ must be at most two, and S', S'' should be cliques, it holds that each pair $S' \setminus R', S'' \setminus R''$ can be enumerate in $\mathcal{O}(n^2 + m^2)$ time. Note that $S' \setminus R'$ and $S'' \setminus R''$ are either single vertices or edges. Therefore, each possible $S = S' \cup S''$ can be enumerate in $\mathcal{O}(n^2 + m^2)$ time.

Similarly, let X be the set of vertices to be moved from the co-tripartition to the sparse part, i.e. $X = (K_1 \cup K_2 \cup K_3) \cap (I' \cup I'')$. Let $X' = X \cap I'$ and $X'' = X \cap I''$.

Now, notice that at most 4 vertices can be moved from the co-tripartition to the sparse part, otherwise X introduces a triangle in the bipartition. So, at most 2 vertices from $G[K_1 \cup K_2 \cup K_3]$ should be in the same independent set of the resulting bipartition (if any). Thus, similar to the previous Step 3, we can enumerate in $\mathcal{O}(m^2)$ such vertices.

At this point, it is enough to check for each pair S, X (there are $\mathcal{O}(n^2 \cdot m^2 + m^4)$) whether

- $G[((I_1 \cup I_2 \cup I_3) \setminus S) \cup X]$ is bipartite;
- $G[((K_1 \cup K_2 \cup K_3) \setminus X) \cup S]$ is co-bipartite.

Recall that recognizing bipartite and co-bipartite graphs can be done in $\mathcal{O}(n + m)$, and G has a $(2, 2)$-partition if and only if there are such pair S, X. Since we enumerate all S, X that should be considered (amortizing some cases using triangle vertex deletion sets), the correctness of our procedure is straightforward. Therefore, Theorem 16 holds.

Theorem 16. *The recognition of* $(2, 2)$*-graphs can be performed in* $\mathcal{O}(n^3 \cdot m^2 + m^5)$ *time.*

Table 3.4 shows the improved complexity map of the recognition of $(2,2)$-graphs.

r \ ℓ	0	1	2	3	4	...
0	-	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	NPc	NPc	...
1	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2+m^2)$	NPc	NPc	...
2	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2+m^2)$	$\mathcal{O}(n^3 \cdot m^2 + m^5)$	NPc	NPc	...
3	NPc	NPc	NPc	NPc	NPc	...
4	NPc	NPc	NPc	NPc	NPc	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Table 3.4: Improved complexity map of the recognition of (r,ℓ)-graphs.

Recall that $\mathcal{O}(n^3 \cdot m^2 + m^5)$ is upper bounded by $\mathcal{O}(n^{10})$. Therefore, our algorithm is faster than Brandstädt's algorithm ($\mathcal{O}(n^{12})$) even when $m = \mathcal{O}(n^2)$.

3.3 Concluding Remarks

In this chapter we demonstrated how, by using algorithms well established in the literature from the parameterized complexity framework one can improve known algorithms.

In our approach for a given graph G we first find either the $(2,2)$-partition of G or a $(3,3)$-partition of it given it exists, otherwise, we guarantee that that are no $(2,2)$-partition of G. Once the $(3,3)$-partition is found we developed a strategy using an parameterized algorithm for triangle vertex deletion set to reallocate the partitions in a way tht a $(2,2)$-partition is found if and only if the graph accepts a $(2,2)$-partition.

By using this strategy we were able to enhance the literature results for recognizing graphs $(2,2)$, from $\mathcal{O}(n^{12})$ to $\mathcal{O}(n^{10})$. Our preliminary research on the recognition of graphs $(2,1)$ is implying that the literature established complexity can be improved from $\mathcal{O}(n^2+m^2)$ to $\mathcal{O}(n \cdot m)$ by using a similar approach.

Chapter 4

Building obstructions of $(2, 1)$-graphs

An obstruction for a hereditary graph property Π is any subgraph that imposes a structure where Π cannot be obtained. Although the class of $(2, 1)$-graphs and their characteristics has been extensively studied [10, 3, 35, 36], few is known about the obstructions of this class. In this chapter, for a given integer n, our goal is present a strategy to list all the minimal obstructions of $(2, 1)$-graphs with at most n vertices. Our focus is to list all obstructions of $(2, 1)$-graphs with at most nine vertices. Furthermore, we describe features of $(2, 1)$-graphs that can be used to identify obstructions.

4.1 Overview

In order to be able to recognize minimal obstructions for the property of being a $(2, 1)$-graph, we must identify characteristics that a $(2, 1)$-partition imposes on a graph and how these conflict with obstructions. This section shows that by using some characteristics and applying those to a certain number of vertices, we can build an algorithm to describe all the obstructions in feasible time, in practice. Therefore, we shall start by showing features of an obstruction and how can we use it to obtain all of them.

Theorem 17. *If a graph G is a minimal obstruction of $(2, 1)$-graphs, then it has a $(2, 2)$-partition and also a $(3, 1)$-partition, where one of the parts is a single vertex.*

Proof. By definition, G is a minimal obstruction of $(2, 1)$-graphs if the removal of a vertex $v \in V(G)$ spawns a $(2, 1)$-partition of G (S_1,S_2,K). Therefore, G has a $(2, 2)$-partition $S_1, S_2, K, \{v\}$, and also a $(3, 1)$-partition $S_1, S_2, \{v\}, K$. □

Theorem 17 allow us to build an algorithm based on the following steps:

- generate a graph H that has a $(2,1)$-partition;
- creates the graph G, by inserting a vertex v to K;
- makes G an minimal obstruction by appropriately adding edges from v to $V(H)$.

Notice that, our strategy shall not be a brute force testing the insertion of all subset of edges between v and $V(H)$. We use a smarter approach in order to reduce the number of verifications of possible obstructions. In order to do so, we must pay attention to the following lemmas.

Lemma 1. *There are no obstruction of* $(2,1)$*-graphs with less than six vertices.*

Proof. A graph G has a $(2,1)$-partition if and only if it contains a clique that intersects every odd cycle in G [37]. Therefore, to complete our proof, we shall demonstrate that every graph with at most 5 vertices is $(2,1)$.

Let G be a graph with 5 vertices. If G has a K_3 then we can add such a K_3 in the clique part and the other two vertices induces a bipartite graph, so G has a $(2,1)$-partition. Now assume that G is K_3-free and it is not bipartite (bipartite graphs are $(2,1)$). Therefore, G has only C_5s as odd cycles, thus, by removing any vertex (the clique part), the remaining vertices induce a bipartite graph, so G is $(2,1)$. □

Lemma 2. *If a graph G is a minimal obstruction of $(2,1)$-graphs, then:*

1. *For any vertex $v \in V(G)$ the graph $G' = G[V(G) \setminus \{v\}]$ can be partitioned in two independent sets (S_1, S_2), and a clique (C), such as $|V(C)| \geq 2$ and there is at least one edge between S_1 and S_2;*
2. *There is no vertex $v \in V(G)$ with $d(v) \leq 1$.*

Proof. Let G be a minimal obstruction of $(2,1)$-graphs, and G' be a subgraph of G obtained by removing a single vertex v from G. Let S_1, S_2, C be a $(2,1)$-partition of G'.

1. If that is no edge from S_1 to S_2 then $S_1 \cup S_2$ is also an independent set, thus G' a $(1,1)$-graph, which implies that G has a $(2,1)$-partition where $\{v\}$ is an independent part, which is a contradiction.

   Beside, if $C = \{u\}$ then either:

   - $d_{G'}(u) = 0$: In this case u can be allocated at any independent set. Therefore G' is bipartite and G is a $(2,1)$-graph, which is a contradiction.

- $d_{G'}(v) \geq 1$: In this case we can select a vertex $w \in N(u)(w \neq v)$ and reallocate it to the clique, keeping an edge between S_1 and S_2, otherwise G would be a $(2,1)$-graph, which is a contradiction.

2. If G has a vertex v such that $d(v) \leq 1$. By the minimality property, the graph $G' = G[V \setminus \{v\}]$ has a $(2,1)$-partition S_1, S_2, C. As v is connected to at most to one vertex of $S_1 \cup S_2$, then $G[S_1 \cup S_2\{v\}]$ is bipartite and therefore G would be a $(2,1)$-graph, which is a contradiction.

□

Theorem 17, Lemma 1 and Lemma 2 gives us the basic structure of a $(2,1)$-graph and its obstructions. Every obstruction then shall have a $2K_2$ as subgraph. One K_2 in the clique, and the other K_2 between the independent sets. The following graphs are the non-isomorphic graphs of size four having a such a $2K_2$ as subgraph. We call these structures *bases*.

The structures illustrated in Figure 4.1 will be the starting point of our algorithm, by using them we can incrementally add vertices and edges in order to form our obstructions.

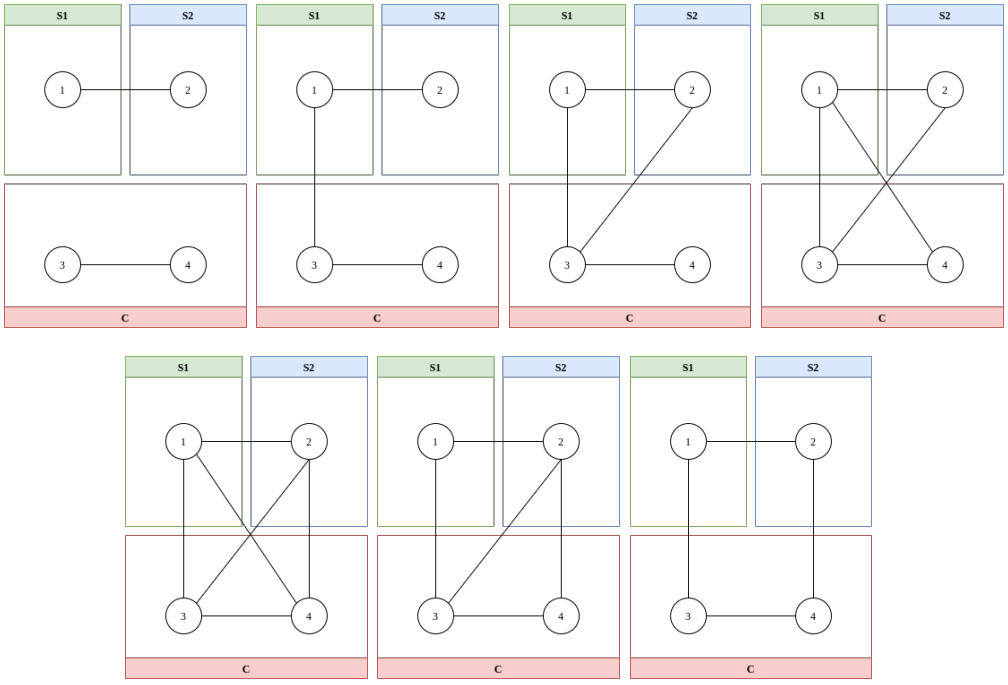


Figure 4.1: Bases #0, . . . , #6, respectively.

4.2 Building the obstructions

At this point, to build obstructions up to n vertices, we already know the configuration of 5 of those vertices: The vertex v vertex that will cause the obstruction and the 4 vertices from the from a base as described in Section 4.1.

Our next step is to allocate the remaining $n-5$ vertices. To achieve this, we list all the non-isomorphic $(2,1)$-graphs with $n-5$ vertices. We called those the *external* graphs.

After the generation of the external graphs is complete, we must now allocate the vertices of a generated external H into the bases.

For example let's assume that we want to generate all the obstructions with $n = 7$ vertices. In this case, we must generate all non-isomorphic graphs H of size exactly 2, Figure 4.2 illustrates these graphs.

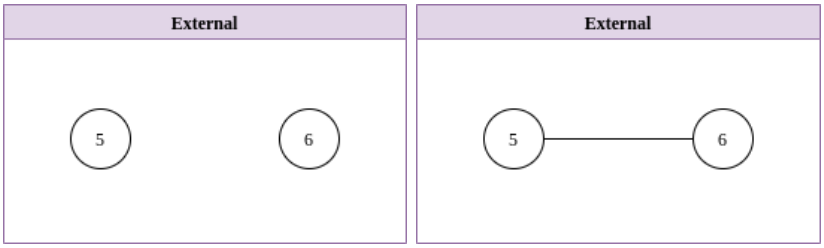


Figure 4.2: Externals #0 and #1 for n=7

After generating the external graphs we must merge the graphs H into the bases I. Such merging shall obey the following constraints:

- If a vertex $v \in V(H)$ is allocated to the clique C of I then for every vertex $u \in C$ an edge (v,u) must be added in $E(I)$;
- If a vertex $v \in V(H)$ is allocated to the clique C of I, then only its neighbors can also be allocated in C;
- If a vertex $v \in V(H)$ is allocated to one of the independent sets of I, then only non-neighbors of v can be allocated to the same independent set.

After the allocation is complete we have one more step before introducing the obstructive vertex. We must enumerate all graphs formed by the addition of allowed edges (those that do not break the properties of the partitions) between the vertices of I and H. We named these graphs the *composite* graphs. Notice that, if any vertex in the composite

graph has a degree equal 0 then we can forwent this graph as it will not be a viable candidate as seem in Lemma 2. Figure 4.3 show us one valid composite graph for $n = 7$.

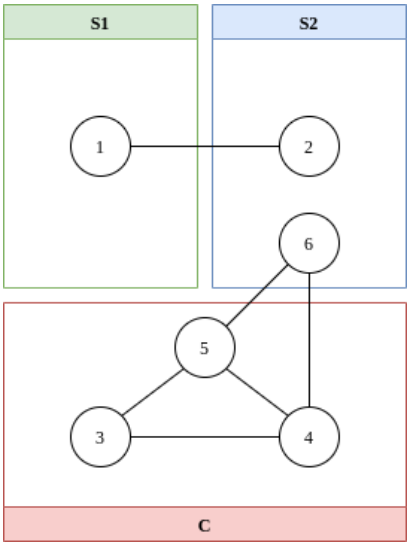


Figure 4.3: A viable composite of External #1 into Base #0

After the development of all the composite graphs we must allocate the obstructive vertex v. To do so, for each composite J we add v to the vertex set of J; After that, we enumerate every possible combination of edges from v to $V(J)$ without regard with the partitioning. The graphs generated by this step was called candidates. The final step in generating all minimal obstructions of size n is applying the recognition of $(2, 1)$-graphs to every candidate and selecting those who fail.

Continuing our example, after the allocation of v, on the subset showed by Figure 4.4 only second candidate would be selected, as it is an obstruction.

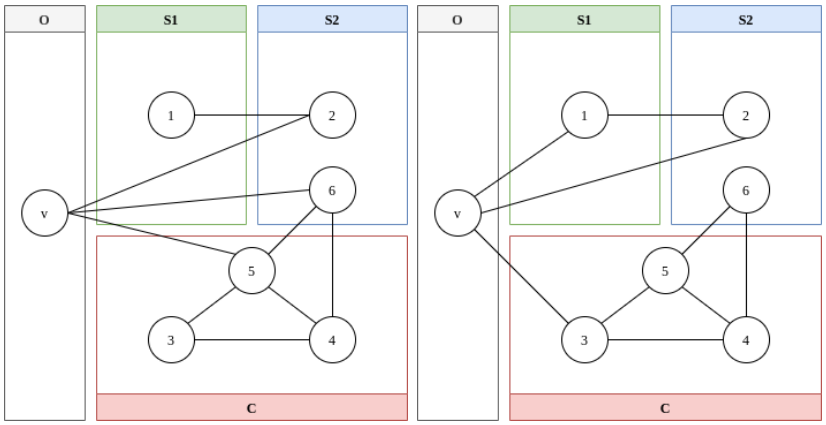


Figure 4.4: Two viable candidates, with one being an obstruction for $n = 7$

Therefore, the only remain step is to demonstrate that this algorithm generates all the minimal obstructions.

Theorem 18. *The algorithm generate all the minimal obstructions for graphs with at most n vertices.*

Proof. Theorem 17 shows that all the minimal obstructions of a $(2,1)$-graph is at the same time in $(3,1)$-graph and $(2,2)$-graph, where one of the parts contains only one vertex v such that $G' = G[V(G) \setminus v]$ is a $(2,1)$-graph.

Our algorithm uses the features shown at Lemma 1 and Lemma 2 to enumerates all possible G', then the obstructive vertex v is added to G' and we generate all graphs formed from additions of edges from v to $V(G')$ and test the result using Brändstadt's algorithm for $(2,1)$-recognition. Therefore, in order to be correct we must demonstrate that all possible G' are generated.

The graphs shown at Figure 4.1 are all the non-isomorphic graphs with 4 vertex having a $2K_2$, the bases. As any obstruction contains one of these bases as a subgraph, the algorithm aim to enumerate all obstructions formed by addition of vertices and edges from those vertices of these bases.

In order to do so, we enumerate all the non-isomorphic $(2,1)$-graphs H with $|V| = n-5$, allocate the vertices of H into the bases I and add all the possible edges from H to I in a manner that the partitions are not violated.

Notice that if any vertex has degree at most one after the introduction of v then it is removed from G', that guarantees that all the candidates with $n-1 \ldots 6$ are generated and so on all G' are generated therefore all obstructions are generated. □

4.3 Results for $n = 9$

Our algorithm is an approach that aim to reduce the set of possible obstructions to be tested using the Brändstadt algorithm, to demonstrate the impact of our strategy we will show our results for $n = 9$.

Notice that are 2^{36} distinct graphs with at most 9 vertices and therefore the enumeration and analysis using Brändstadt's algorithm does not provide an adequate approach. First we performer a naive implementation, without use our bases of size at most four and remove vertices with degree smaller than 2. We obtain about *8GB* of data taking approximately *6h* to complete.

A second implementation used the bases of size 4, remove duplicates, and removed

the vertices of degree at most 1. This run took *94min* to complete, generating about *8MB* of data with a count of 22 thousands of obstructions, a smaller set of graphs.

The third and final implementation is the one described in this chapter, we use the strategies of the second implementation and remove the isomorphic graphs along the way, resulting in an execution time of *80min* generating 1026 graphs in a file of size $500KB$.

The implementation was made using the framework JGrapht[38], and Java. The algorithm to generate only non-isomorphic graphs was the one proposed VF2 algorithm by Cordella[39] as made available by JGrapht. The machine in wich the experiments were executed is a Lenovo e431 with an Intel i5 CPU and 4GB of RAM.

The result of the third run is available at

http://www2.ic.uff.br/~ueverton/obstrucao21/Apendice.pdf

and the visualizations of the results using vis.js[40] can be found at

http://www2.ic.uff.br/~ueverton/obstrucao21/obstrucoes_html/obstrucoes.html

4.4 Concluding Remarks

This chapter present an algorithm that correctly generates all the minimal obstruction of (r, ℓ) graphs up to n vertices. Such algorithm is based both on structural properties that are common to all obstructions of this kind and structural properties of (r, ℓ) graphs.

By identifying such properties and developing tailored approaches to vertex allocation we were able to greatly improve the execution time to generate all the obstructions up to n vertices. As stated on section 4.3 we elaborated test cases with $n = 9$ to demonstrate the impact of our approach, the table 4.1 show these results.

Algorithm generation	1st	2nd	Final
File Size	8GB	8MB	500KB
Execution Time	6h	94Min	80Min
Generated Graphs	-	22000	1026

Table 4.1: Obstructions generation algorithm comparison

Chapter 5

Final remarks

In this dissertation several discoveries was made about the (r, ℓ)-graphs: How the proper coloring of an (r, ℓ)-graph relates to the list-coloring of it's $(r, \ell - 1)$-graph counterpart; the $P \times NP$-*complete* dichotomy of Graph Coloring on such classes; a parameterized ways to find a proper coloring for a given $(2, 1)$-graph; an improvement in the recognition of the $(2, 2)$-graphs; and a program to build minimal obstructions of $(2, 1)$-graphs up to a given number n of vertices. In chapter 2, we unveil the complexity of the Graph Coloring problem (and consequently to clique cover problem) establishing an P vs NPc dichotomy for (r, ℓ)-graphs. We also provided a link between the graph coloring of a (r, ℓ)-graph and the *List-Coloring* of it's $(r, \ell - 1)$-graph counterpart. By analysing the $(2, 1)$ class we determined how the problem behaves using the size of each partition as a parameter under the parameterized complexity approach; Although the size of the dense part is not useful as a parameter to produce a **FPT** algorithm and that the problem remains NPc even with a maximum clique of size 3, we demonstrated how the neighborhood of the clique (under the optics of list coloring, the size of the list for each vertex in the bipartition) plays an important role in finding an **FPT** algorithm.

The third chapter of this work demonstrated how building up on the Brandstädt algorithm that checks the neighborhood of each vertex, one can in polynomial time discard the graph as a $(2, 2)$-graph, finding a $(2, 2)$-partition or a $(3, 3)$-partition that can be used to determine if the found $(3, 3)$-graph allows a $(2, 2)$-partition or not. Once the $(3, 3)$-partition is found we demonstrated how, by how using an FPT algorithm for triangle-free vertex deletion problem as subroutine to find the vertex set in the sparse part that should be reallocated in the dense part of the desired $(2, 2)$-partition, one can build an $\mathcal{O}(n^{10})$ algorithm to recognize graphs in the $(2, 2)$-class. Such algorithm plays an important role, since the former algorithm for recognition of graphs in such class was

$\mathcal{O}(n^{12})$ and lacked proper publication.

Lastly, we studied the structural properties of graphs in the $(2,1)$ class, and how these properties can be used to create an obstruction of such class. We demonstrated by using $n = 9$ how the execution time to generate all the 1026 minimal obstructions can go from 6 hours to 80 minutes by using our approach.

Although several discoveries were made, in this dissertation, we hypothesized some improvements that can be researched to find newer and better results.

5.1 Further research

In addition to providing an upper bound for the chromatic number of $(2,1)$-graphs, Theorem 1 also shows that $\omega(G) \leq \chi(G) \leq \omega(G) + 1$ for a $(1,2)$-graph G.

For any graph G with a $(1,2)$-partition S_1, K_1, K_2, it holds that

$$\max\{|K_1|, |K_2|\} \leq \omega(G) \leq 2 \cdot \max\{|K_1|, |K_2|\} + 1.$$

Thus, for $(1,2)$-graphs we can assume parameterization by $\omega(G)$ instead of $\max\{|K_1|, |K_2|\}$.

Since the maximum clique of a $(1,2)$-graph G can be obtained in polynomial time, we can assume $\omega(G) = |K_2|$ because we can add $\omega(G) - |K_2|$ new vertices to K_2 having no neighbor in $S_1 \cup K_1$, which preserves the chromatic number of G. Therefore, from Theorem 2 and Theorem 3, it holds that given a $(1,2)$-graph G, the problem of determining the chromatic number of G is equivalent to List Coloring on split graphs where $\omega(G)$ is the number of colors of the of the palette ($|P|$).

A similar analysis to the one we developed for Graph Coloring on $(2,1)$-graphs and List Coloring on bipartite graphs also seems interesting for $(1,2)$-graphs and split graphs.

Besides, a multivariate analysis of the parameterized complexity of Graph Coloring on $(0,3)$-graphs and $(4,0)$-graphs is suitable. Finally, to map the P vs. NPc dichotomy regarding the values of r and ℓ of (r,ℓ)-graphs for other classical NP-hard graph problems is an interesting research topic.

A better approach on the obstruction algorithm

In chapter 4 we stated a program to generate all the obstructions up to n vertex. However, an analysis of the execution showed us that there may be a reuse of the discarded

candidates of an obstruction of n vertex in the identification of obstructions of $n+1$ vertex. We conjecture that there is a dynamic programing approach that yield even better results.

On the recognition of $(1, 2)$ and $(2, 1)$-graphs

A major part of our focus through this work was in the recognition part of it. Besides the results declared in chapter 3 we heavily studied a strategy to get better results for the $(1, 2)$ and $(2, 1)$ classes, this study leaded us to some exciting insights.

The Brandstäd algorithm show us that, when identifying a $(2, 1)$-graph the neighborhood of each vertex plays an important role. During Brandstadt execution there are two checks that should be performed for each vertex:

- (N1) If the neighborhood of the vertex is a split graph, then the vertex could belong to the sparse part.
- (N2) If the complement of the neighborhood of the vertex is a bipartite graph, the the vertex could belong to the clique.

If every vertex is compliant to exactly one of this requirements, then the graph is a $(2, 1)$-graph. However, if any vertex does not fit N1 nor N2 then the graph is guaranteed not in the $(2, 1)$ class; When there is a vertex that meets both N1 and N2 then the graph is guaranteed to be an $(3, 1)$-graph spawned by the neighborhood and it's complement spawned by this vertex.

Since we have an $(3, 1)$-graph, if we can reallocate the vertices in the tripartition to coalesce with the found clique in a way that the sparse part becomes a bipartition then we can find the $(2, 1)$ partition of the given graph (and if no reallocation is possible then we can assure that the graph is not in the $(2, 1)$ class).

This leaded us to search for an algorithm that can properly move the vertices between partitions to find a proper $(2, 1)$ partition. In order to do so, the first step was to find a way to make the tripartition a bipartition; This problem can be treated partially as the Odd Cycle Transverse problem, as it can dismantle the tripartition. Even knowing that OCT is in the $\mathcal{NP}$ class, we demonstrated that, by using the parameterized complexity framework we can obtain such answer in polynomial time for our instances.

By identifying this vertices one can determine strategies to reallocate they to the dense part of the pretended $(2, 1)$ partition. Our efforts towards such strategies using a modified version of the Annotated bipartition problem heavily indicates that the complexity of the recognition for the $(2, 1)$ and $(1, 2)$-graphs can be improved to be $\mathcal{O}(n \cdot m)$.

Bibliography

[1] Tomas Feder et al. “Complexity of graph partition problems”. In: *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 1999, pp. 464–472.

[2] Russell Merris. “Split graphs”. In: *European Journal of Combinatorics* 24.4 (2003), pp. 413–430.

[3] Andreas Brandstädt. “Partitions of graphs into one or two independent sets and cliques”. In: *Discrete Mathematics* 152.1-3 (1996), pp. 47–54.

[4] Tomás Feder et al. “List partitions”. In: *SIAM Journal on Discrete Mathematics* 16.3 (2003), pp. 449–478.

[5] Tomás Feder et al. “List matrix partitions of chordal graphs”. In: *Theoretical Computer Science* 349.1 (2005), pp. 52–66.

[6] Sudeshna Kolay et al. “Parameterized algorithms on perfect graphs for deletion to (r, l)-graphs”. In: *41st International Symposium on Mathematical Foundations of Computer Science (MFCS 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2016.

[7] Julien Baste et al. “Parameterized Complexity Dichotomy for (r, ℓ)-Vertex Deletion”. In: *Theory of Computing Systems* 61.3 (2017), pp. 777–794.

[8] Luerbio Faria et al. “Improved kernels for Signed Max Cut parameterized above lower bound on (r, l)-graphs”. In: *arXiv preprint arXiv:1512.05223* (2015).

[9] Sancrey Rodrigues Alves et al. “On the (parameterized) complexity of recognizing well-covered (r, ℓ)-graph”. In: *Theoretical Computer Science* 746 (2018), pp. 36–48.

[10] Andreas Brandstädt. *Partitions of graphs into one or two independent sets and cliques*. Tech. rep. Friederich-Schiller-Universtat Jena, 1984.

[11] Tomas Feder et al. “Complexity of graph partition problems”. In: *STOC*. Citeseer. 1999, pp. 464–472.

[12] Rhyd Lewis. *A guide to graph colouring*. Vol. 7. Springer, 2015.

[13] Enrico Malaguti and Paolo Toth. “A survey on vertex coloring problems”. In: *International transactions in operational research* 17.1 (2010), pp. 1–34.

[14] Simone Dantas et al. “On star and biclique edge-colorings”. In: *International Transactions in Operational Research* 24.1-2 (2017), pp. 339–346.

[15] Rosiane de Freitas et al. “On distance graph coloring problems”. In: *International Transactions in Operational Research* 28.3 (2021), pp. 1213–1241.

[16] Uejima Akihiro et al. “A coloring problem with restrictions of adjacent colors”. In: *International Transactions in Operational Research* 9.2 (2002), pp. 183–194.

[17] Marek Cygan et al. *Parameterized algorithms*. Vol. 4. 8. Springer, 2015.

[18] Rodney G Downey and Michael R Fellows. *Fundamentals of parameterized complexity*. Vol. 4. Springer, 2013.

[19] Jörg Flum and Martin Grohe. *Parameterized complexity theory*. Springer Science & Business Media, 2006.

[20] Rolf Niedermeier. “Invitation to fixed-parameter algorithms”. In: (2006).

[21] Béla Bollobás. *Modern graph theory*. Vol. 184. Springer Science & Business Media, 2013.

[22] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Vol. 2. Springer Science & Business Media, 2012.

[23] Kenneth Appel and Wolfgang Haken. “The solution of the four-color-map problem”. In: *Scientific American* 237.4 (1977), pp. 108–121.

[24] Larry Stockmeyer. “Planar 3-colorability is polynomial complete”. In: *ACM Sigact News* 5.3 (1973), pp. 19–25.

[25] Klaus Jansen and Petra Scheffler. “Generalized coloring for tree-like graphs”. In: *Discrete Applied Mathematics* 75.2 (1997), pp. 135–155.

[26] Michael R Fellows et al. “On the complexity of some colorful problems parameterized by treewidth”. In: *Information and Computation* 209.2 (2011), pp. 143–153.

[27] Jan Kratochvil et al. “Precoloring extension with fixed color bound”. In: *Acta Math. Univ. Comen* 62 (1993), pp. 139–153.

[28] Mihály Hujter and Zsolt Tuza. “Precoloring extension. II. Graph classes related to bipartite graphs”. In: *Acta Mathematica Universitatis Comenianae* 62.1 (1993), pp. 1–11.

[29] Andreas Brandstädt. “Corrigendum”. In: *Discrete Mathematics* 186.1 (1998), p. 295. ISSN: 0012-365X. DOI: https://doi.org/10.1016/S0012-365X(98)00014-4. URL: http://www.sciencedirect.com/science/article/pii/S0012365X98000144.

[30] Andreas Brandstädt, Van Bang Le, and Thomas Szymczak. “The complexity of some problems related to Graph 3-colorability”. In: *Discrete Applied Mathematics* 89.1 (1998), pp. 59–73. ISSN: 0166-218X. DOI: https://doi.org/10.1016/S0166-218X(98)00116-4. URL: http://www.sciencedirect.com/science/article/pii/S0166218X98001164.

[31] Dénes König. *Theorie der endlichen und unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*. Vol. 16. Akademische Verlagsgesellschaft mbh, 1936.

[32] Peter L Hammer and Bruno Simeone. “The splittance of a graph”. In: *Combinatorica* 1.3 (1981), pp. 275–284.

[33] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 29. wh freeman New York, 2002.

[34] Bruce Reed, Kaleigh Smith, and Adrian Vetta. “Finding odd cycle transversals”. In: *Operations Research Letters* 32.4 (2004), pp. 299–301.

[35] Tomas Feder, Pavol Hell, and Jing Huang. “List homomorphisms and circular arc graphs”. In: *Combinatorica* 19.4 (1999), pp. 487–505.

[36] Pavol Hell et al. “Partitioning chordal graphs into independent sets and cliques”. In: *Discrete Applied Mathematics* 141.1-3 (2004), pp. 185–194.

[37] JS Silva. “Obstruções minimais de grafos-(2, 1)”. PhD thesis. Dissertação de mestrado, Universidade Federal Fluminense, 2011.

[38] The JGraphT team. *JGrapht*. 2020. URL: https://jgrapht.org/.

[39] Luigi P Cordella et al. “A (sub) graph isomorphism algorithm for matching large graphs”. In: *IEEE transactions on pattern analysis and machine intelligence* 26.10 (2004), pp. 1367–1372.

[40] Vis.js Community. *Vis.js*. 2020. URL: https://visjs.org/.