Finding Independent Transversals Efficiently

by

Alessandra Graf

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Combinatorics & Optimization

Waterloo, Ontario, Canada, 2019

© Alessandra Graf 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Kathie Cameron Professor, Dept. of Mathematics, Wilfrid Laurier University
Supervisor:	Penny Haxell Professor, Dept. of Combinatorics and Optimization, University of Waterloo
Internal Member:	Joseph Cheriyan Professor, Dept. of Combinatorics and Optimization, University of Waterloo
Internal Member:	Luke Postle Assistant Professor, Dept. of Combinatorics and Optimization, University of Waterloo
Internal-External Member:	Anna Lubiw Professor, School of Computer Science, University of Waterloo

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis is based on collaborations with Penny Haxell and David Harris. These include, but are not limited to, papers [38] and [37].

Abstract

Let G be a graph and $(V_1, \ldots, V_m)$ be a vertex partition of G. An *independent transversal* (IT) of G with respect to $(V_1, \ldots, V_m)$ is an independent set $\{v_1, \ldots, v_m\}$ in G such that $v_i \in V_i$ for each $i \in \{1, \ldots, m\}$.

There exist various theorems that give sufficient conditions for the existence of ITs. These theorems have been used to solve problems in graph theory (e.g. list colouring, strong colouring, delay edge colouring, circular colouring, various graph partitioning and special independent set problems), hypergraphs (e.g. hypergraph matching), group theory (e.g. generators in linear groups), and theoretical computer science (e.g. job scheduling and other resource allocation problems). However, the proofs of the existence theorems that give the best possible bounds do not provide efficient algorithms for finding an IT. In this thesis, we give poly-time algorithms for finding an IT under certain conditions and some applications, while weakening the original theorems only slightly. We also give efficient poly-time algorithms for finding partial ITs and ITs of large weight in vertex-weighted graphs, as well as an application of these weighted results.

Acknowledgements

More than anyone, I would like to thank Penny Haxell. Without her patience, guidance, and support this thesis would not have been possible. I am truly lucky and honoured to have had her as my Ph. D. supervisor and for that I am eternally grateful.

I would also like to thank David Harris for his insightful comments and guidance. It has been a pleasure to work with him.

Next, I would like to thank my committee members, Kathie Cameron, Joseph Cheriyan, Anna Lubiw, and Luke Postle for all of their comments and suggestions. In particular, I would like to thank Anna for all of her insights into writing the algorithms in this thesis in a clear way.

I would also like to thank the C&O faculty, staff, and graduate students for making my time as a graduate student a wonderful experience. I will truly miss all of the lunch time conversations, cryptic crossword puzzles, and evenings at the Grad House.

Last, but certainly not least, I would like to thank my family and friends for all of their love and support. Without all of you, I would not have finished this thesis or have enjoyed living here in Waterloo. Thank you all from the bottom of my heart! I will truly miss our board game days (and nights), trivia nights, movie nights, kayaking adventures, escape rooms, lunches and dinners, and just spending time hanging out with all of you. I especially want to thank Cynthia Rodríguez for being such an understanding office mate and overall great friend.

Table of Contents

List of Figures

Chapter 1

Introduction

Let G be a graph and let $(V_1, \ldots, V_m)$ be a vertex partition of G. A set of vertices (or edges) is *independent if no two of its elements are adjacent.* An *independent transversal* (IT) of G with respect to $(V_1, \ldots, V_m)$ is an independent set $\{v_1, \ldots, v_m\}$ in G such that $v_i \in V_i$ for each i. A *partial independent transversal* (PIT) of G with respect to $(V_1, \ldots, V_m)$ is an independent set M in G such that no two vertices of M are in the same vertex class. This is a very general notion, and many combinatorial problems can be formulated by asking if a given graph with a given vertex partition has an IT (e.g. the SAT problem, see [52]). It follows that determining whether a given graph and vertex partition has an IT is NP-complete.

While we cannot expect an efficient characterisation of precisely which vertex-partitioned graphs have an IT (unless $P = NP$), there are various known results that give sufficient conditions for the existence of an IT. One of the most easily stated and most frequently applied is a result of Haxell from [48, 49] involving the size of the vertex classes in terms of the maximum degree of G (see Theorem 2.3). In particular, it shows that as long as the classes have size at least $2\Delta(G)$, where $\Delta(G)$ denotes the maximum degree of G, G is guaranteed to have an IT with respect to the vertex partition. This answers a question first introduced in 1975 by Bollobás, Erdős and Szemerédi in [21]. A construction of Szabó and Tardos [91] has also shown that the result of Haxell is best possible for every maximum degree.

Another more general existence result, also due to Haxell [48, 49], uses a notion of domination to give a sufficient condition for the existence of an IT (see Theorem 2.4). We say that a subset $D \subseteq V(G)$ *dominates* a subgraph W of G if for all $w \in V(W)$, there exists $uw \in E(G)$ for some $u \in D$. (This definition of domination is quite often referred to as *strong domination* or *total domination*, but since it is the only notion of domination that we will refer to in this thesis, we will use the simpler term.) This general result (often in the form of the maximum degree case) has been applied to obtain many results in various fields, including graph theory (e.g. list colouring [49], strong colouring [50, 3], delay edge colouring [8], circular colouring [57, 58], various graph partitioning and special independent set problems [9, 53, 61, 26]), hypergraphs (e.g. hypergraph matching [48, 69, 11, 12]), group theory (e.g. generators in linear groups [23]), and theoretical computer science (e.g. job scheduling and other resource allocation problems [14, 15]).

Unfortunately, the proofs of these two results are not algorithmic and so do not provide a method of finding ITs efficiently. For certain applications, it is enough to know that class sizes of $c\Delta(G)$ guarantee an IT for some constant c. In these cases, one could obtain algorithmic versions using (for example) the algorithmic Lovász Local Lemma (LLL) (see [17, 78, 79]). However, for many other applications, having the best possible value of the constant c is important. This raises the question of how much the hypotheses of these two existence results need to be strengthened in order to guarantee that an IT can be found efficiently. Such results would give algorithmic proofs of these applications with the constants being as close as possible to their optimal values.

Most of the known results on this question have focused on the maximum degree result and are randomised algorithms obtained as applications of algorithmic versions of the LLL or its lopsided variant. These include algorithms that use the original algorithm of Beck [17] and its improvements (see e.g. [10, 76, 27, 89, 20]) as well as the resampling algorithm of Moser and Tardos [78] and its improvements (see e.g. [65, 81, 66, 45, 1, 44]. The current best result for polynomial expected time is due to Harris [42] with the lower bound on the class size being $4\Delta(G) - 1$. Deterministic algorithms based on derandomising the Moser-Tardos algorithm have also been studied, but they require the class size be $C\Delta$ for some large constant C in order to find an IT efficiently [35, 43].

Recently, a new algorithmic approach that does not rely on the LLL has been introduced by Annamalai [11, 13]. This new approach was developed as a method for finding perfect matchings in r-uniform bipartite hypergraphs (see Section 4.1), which has applications to the restricted max-min fair allocation problem. It is an example of an algorithmic version of an application of the domination existence result of Haxell to hypergraphs (see Theorem 4.4).

In the first main result of this thesis, we address the algorithmic IT question for a large class of graphs, called r-claw-free graphs, without using the LLL or any of its variants. A graph G with vertex partition $(V_1, \ldots, V_m)$ is said to be *r-claw-free with respect to* $(V_1, \ldots, V_m)$ if no vertex of G has r independent neighbours in distinct vertex classes. Our work leads to algorithmic versions of both the maximum degree and domination existence results, with only a slight strengthening of the hypotheses. We then use these algorithms on some applications previously solved by the non-algorithmic existence results to get algorithmic versions of these applications. Our main algorithm, called FindITorBD (which finds an IT or two sets $\mathcal{B}$ and D with certain properties), incorporates some key ideas from Annamalai's work in [11] as well as from the original proofs of [48, 49].

Aharoni, Berger, and Ziv [3] gave a generalisation of the IT existence theorems of [48, 49] to (partial) ITs in vertex-weighted graphs. Their work could be viewed as an analogue of the theory of weighted matchings in graphs. Their proofs are based on [48, 49] and again are not algorithmic. Our second main result of this thesis is to give an algorithmic version of their result, called FindWeightPIT. It uses ideas from [3] and relies on FindITorBD as a subroutine.

The degree of the runtime of FindITorBD is dependent on r, or on Δ in the maximum degree case. Hence our algorithms are only efficient when r (or Δ) is constant. To address this limitation, our last main result in this thesis develops a randomised algorithm, called FindWeightIT, that is efficient even when the maximum degree is unbounded. FindWeightIT incorporates FindWeightPIT as a subroutine. We demonstrate the use of FindWeightIT by giving an application to finding strong colourings and fractional strong colourings of graphs efficiently.

This thesis is organised as follows. In Chapter 2, we give a more thorough presentation of the known results related to ITs. In particular, we discuss the history of existence results for ITs (Section 2.2) as well as the algorithmic results (Section 2.3) in more detail. This includes formally stating the results in [48, 49] (Theorems 2.3 and 2.4). We also present an exponential time algorithm based on the original proofs of Theorem 2.4 and some modifications of it based on several key notions introduced by Annamalai in [11, 13]. We also include a section providing the notation used throughout this thesis (Section 2.1).

In Chapter 3, we prove the first main result of this thesis (Theorem 3.1). In particular, we provide the algorithm FindITorBD and prove that FindITorBD efficiently finds an IT in an r-claw-free graph, providing an algorithmic version of Theorem 2.4. The algorithm FindITorBD also proves an algorithmic version of Theorem 2.3 (Corollary 3.2).

In Chapter 4, we apply the algorithm FindITorBD to some applications of Theorems 2.3 and 2.4. In particular, we look at some known results in graph colourings, graph partitionings, and hypergraph matchings. This includes results due to Annamalai [11, 13] (Section 4.1); Kaiser, Král, and Škrekovski [57] (Section 4.2); King [61] (Section 4.3); Aharoni, Berger, and Ziv [3] (Section 4.4); and Alon, Ding, Oporowski, and Vertigan [9] (Section 4.5). We present modifications of these results that make them algorithmic.

In Chapter 5, we prove the second main result of this thesis (Theorem 5.5). In particular, we provide the algorithm FindWeightPIT and prove that FindWeightPIT efficiently finds a PIT of weight at least $\tau_w^{r,\epsilon}$ in an r-claw-free vertex-weighted graph, where $\tau_w^{r,\epsilon}$ is a parameter of vertex-weighted graphs (see Section 5.2). This proves an algorithmic version of Theorem 5.2, the generalisation of Theorem 2.4 due to Aharoni, Berger, and Ziv from [3].

In Chapter 6, we prove the final main result of this thesis (Theorem 6.10). In particular, we provide the randomised algorithm FindWeightIT that finds an IT in vertex-weighted graphs of maximum degree Δ in expected polynomial time, where this runtime does not depend on Δ. We then apply FindWeightIT to find strong colourings and fractional strong colourings of graphs efficiently. These applications use ideas from the applications of Theorem 5.2 presented in [3].

The results in Chapters 3 and 5 are joint work with my supervisor Penny Haxell. The content of Chapter 6 is joint work with David Harris and Penny Haxell. The rest of this thesis is my own unaided work.

Chapter 2

Background

In this chapter, we discuss the main definitions and known results that will be referred to throughout this thesis. This includes an overview of results concerning the existence of independent transversals (ITs), both algorithmic and non-algorithmic. In particular, we will introduce two non-algorithmic results due to Haxell [48, 49] (Theorems 2.3 and 2.4). We will see algorithmic versions of these results in Chapter 3 as well as algorithmic versions of results that used Theorems 2.3 and 2.4 in Chapter 4. In Section 2.3, we discuss the previously known algorithmic results for determining the existence of ITs in a graph. This includes results derived from algorithmic versions of the Lovász Local Lemma (LLL) as well as the new result of Annamalai [11, 12, 13] that helped us develop our algorithm.

This chapter is organised as follows. In Section 2.1, we review the graph theory terms and notation used throughout this thesis. In Section 2.2, we provide some key definitions and a history of (non-algorithmic) results relating to ITs. We conclude in Section 2.3 with a discussion of the previous algorithmic results relating to ITs.

2.1 Notation

In this section, we provide a brief overview of some terms and notation used throughout this thesis. Some non-standard definitions and notation will appear later in the thesis when needed. For all other terms and notation, we refer the reader to Diestel's graph theory text [28].

Let G be a graph. We denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. The maximum degree of G is denoted $\Delta(G)$.

For a vertex $v \in V(G)$, we denote the set of neighbours of v by $N(v)$ and the degree of v by $\deg(v)$. For a subgraph H of G and vertex $v \in V(G)$, we write $N_H(v)$ to denote the set of neighbours of v in H and $\deg_H(v)$ to denote the degree of v in H.

Suppose $U \subseteq V(G)$. The graph induced by U is denoted $G[U]$. Recall from Chapter 1 that U *dominates* a subgraph H of G if for all $v \in V(H)$, there exists some $u \in U$ such that $uv \in E(G)$. This definition of domination is quite often referred to as *strong domination* or *total domination*. However, since it is the only notion of domination that we will discuss, we use the simpler term.

Let $(V_1, \dots, V_m)$ be a vertex partition of G. For any subset $\mathcal{B} \subseteq \{V_1, \dots, V_m\}$, let $G_{\mathcal{B}}$ denote the subgraph of G whose vertex set is the union of the vertex classes in $\mathcal{B}$ and whose edge set consists of all edges $v_iv_j \in E(G)$ where $v_i \in V_i$, $v_j \in V_j$, and $V_i, V_j \in \mathcal{B}$ are distinct classes. Specifically, $G_{\mathcal{B}} = G\left[\bigcup_{V_i \in \mathcal{B}} V_i\right] - \{uv \in E(G) : u, v \in V_i, V_i \in \mathcal{B}\}$. Let $\mathrm{Vclass}(v)$ denote the vertex class that contains v. The set of vertex classes containing vertices of $U \subseteq V(G)$ is given by $\mathrm{Vclass}(U) = \{\mathrm{Vclass}(v) : v \in U\}$.

An *n-star* is the complete bipartite graph $K_{1,n}$. We call the vertex of degree n the *centre* of the star and the other vertices of the graph the *leaves* of the star. When $n = 1$, we designate one vertex as the centre and the other as a leaf. Note that the 3-star is sometimes referred to as the *claw graph*. We will use this definition of an n-star to define a constellation in Definition 2.5 of Section 2.2.

We use the notation $\mathrm{poly}(x)$ to denote a polynomial in x and $\mathrm{poly}(x, y)$ to denote a polynomial in x and y. We will use this notation throughout this thesis to describe the runtimes of various algorithms.

2.2 Independent Transversals

In this section, we discuss independent transversals (ITs). Recall the following definition from Chapter 1.

Definition 2.1. Let G be a graph and $(V_1, \dots, V_m)$ be a vertex partition of G. An *independent transversal* (IT) of G with respect to $(V_1, \dots, V_m)$ is an independent set $\{v_1, \dots, v_m\}$ in G such that $v_i \in V_i$ for each $i \in \{1, \dots, m\}$.

We have the following generalisation of Definition 2.1.

Definition 2.2. Let G be a graph and $(V_1, \ldots, V_m)$ be a vertex partition of G. A *partial independent transversal* (PIT) of G with respect to $(V_1, \ldots, V_m)$ is an independent set M in G such that no two vertices in M are in the same vertex class.

Thus an IT is a PIT that contains a vertex from every vertex class.

Many combinatorial problems can be formulated by asking if a given graph and vertex partition has an IT. For example, ITs have appeared in the context of hypergraph matchings [48, 11, 12], graph colourings [57, 3], list colourings [49], graph partitioning and special independent set problems [9, 53, 61], and job scheduling and other resource allocation problems [14, 15]. However, the problem of determining whether a given graph and vertex partition has an IT is NP-complete. This can be shown by reductions to it from SAT (see e.g. [52]) or from the decision problem for perfect matchings in 3-uniform 3-partite hypergraph with parts of equal size. The latter is one of Karp's 21 NP-complete problems [59].

Although we cannot expect an efficient characterisation of precisely which vertex-partitioned graphs have an IT (unless $P = NP$), there are various known results that give sufficient conditions for the existence of an IT (e.g. [5, 34, 48, 49, 10, 20]). Furthermore, some of these results have algorithmic versions, mostly relying on algorithmic versions of the Lovász Local Lemma (LLL), that can in expected polynomial time return an IT given a graph G and vertex partition $(V_1, \ldots, V_m)$ (see [25, 13, 42, 47, 35, 43]). We will discuss these algorithmic versions and the LLL in more detail in Section 2.3. For the remainder of this section, we will focus on one of the most commonly applied sufficient conditions for the existence of an IT. The proof of this result does not give an efficient algorithm for finding an IT.

Given a graph and vertex partition, it is natural to ask how large the vertex classes need to be, in terms of the maximum degree, to guarantee the existence of an IT. This question was first introduced and studied in 1975 by Bollobás, Erdős and Szemerédi [21]. Further progress on this bound was made by many authors, including Alon [5], Fellows [34], and Haxell [49]. For a graph G with maximum degree Δ, Alon [5] showed, using the LLL, that an IT exists if each vertex class has size at least 25Δ. Fellows [34] independently proved a bound of 16Δ. This bound was later improved to $2e\Delta$ using the LLL (see e.g. Alon and Spencer [10]) and to 2Δ by Haxell [48, 49] using a combinatorial argument.

Several authors also found constructions for graphs and vertex partitions where the vertex classes had a particular size in terms of the maximum degree and no IT existed. Work on this lower bound includes results of Jin [56], Yuster [95], and Alon [7]. In 2006, Szabó and Tardos [91] gave constructions for every Δ in which there is a graph G of maximum degree Δ and a vertex partition where every class has size $2\Delta - 1$, but G has no IT with respect to the partition. Because of this, the following result of Haxell from [48, 49] is known to be best possible for every Δ.

Theorem 2.3 ([48, 49])**.** *Let G be a graph with maximum degree Δ. Then for any vertex partition $(V_1, \ldots, V_m)$ of G where $|V_i| \geq 2\Delta$ for each i, there exists an IT of G.*

Theorem 2.3 is an immediate consequence of a more general statement described in terms of domination. Recall that a subset $D \subseteq V(G)$ dominates a subgraph W of G if for all $w \in V(W)$, there exists some $u \in D$ such that $uw \in E(G)$. Also, recall that for $\mathcal{B}$ a subset of the set of vertex classes in the partition, the graph $G_{\mathcal{B}}$ is given by
$G_{\mathcal{B}} = G\left[\bigcup_{V_i \in \mathcal{B}} V_i\right] - \{uv \in E(G) : u, v \in V_i, V_i \in \mathcal{B}\}$. The general theorem of Haxell from [49] (which easily follows from the argument in [48]) is stated as Theorem 2.4.

Theorem 2.4 ([48, 49])**.** *Let G be a graph with a vertex partition $(V_1, \ldots, V_m)$. Suppose that, for each $\mathcal{B} \subseteq \{V_1, \ldots, V_m\}$, the subgraph $G_{\mathcal{B}}$ is not dominated in $G_{\mathcal{B}}$ by any set of size at most $2(|\mathcal{B}| - 1)$. Then G has an IT.*

It is easy to see that Theorem 2.4 implies Theorem 2.3. This is because the union of $|\mathcal{B}|$ vertex classes in G contains a total of at least $2\Delta|\mathcal{B}|$ vertices. Thus $G_{\mathcal{B}}$ cannot be dominated by $2|\mathcal{B}| - 2$ vertices whose degree is at most Δ.

The proof of Theorem 2.4 also provides some insight into the structure of one possible dominating set for $G_{\mathcal{B}}$, for some $\mathcal{B} \subseteq \{V_1, \ldots, V_m\}$, when G does not have an IT. Specifically, if G does not have an IT, then there exists $\mathcal{B} \subseteq \{V_1, \ldots, V_m\}$ such that $G_{\mathcal{B}}$ is dominated by the vertex set of a *constellation* for $\mathcal{B}$.

Definition 2.5. Let G be a vertex-partitioned graph and let $\mathcal{B}$ be a set of vertex classes in that partition. A *constellation* for $\mathcal{B}$ is an induced subgraph K of $G_{\mathcal{B}}$, whose components are stars each containing a centre and a non-empty set of leaves distinct from the centre, and such that the set of all leaves of K, denoted $\mathrm{Leaf}(K)$, forms an IT of $|\mathcal{B}| - 1$ vertex classes of $\mathcal{B}$.

Figure 2.1 shows an example of a constellation. Note that if K is a constellation for $\mathcal{B}$, then $|V(K)| \leq 2(|\mathcal{B}| - 1)$.

The proof of Theorem 2.4 implies the following result.

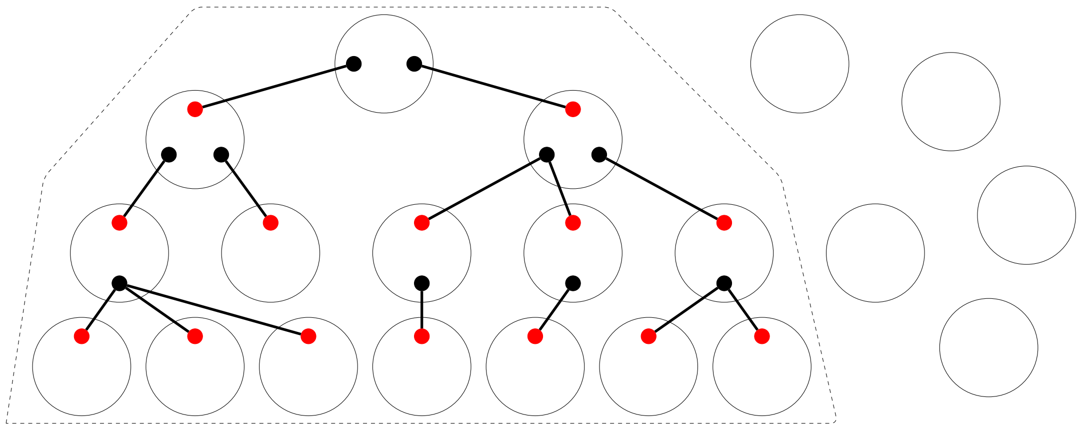
Figure 2.1: A constellation K for the set $\mathcal{B}$ of classes enclosed by the dotted border. Each circle represents a vertex class. The centres of the stars appear in black and the leaves appear in red.

Theorem 2.6. *Let G be a graph with a vertex partition $(V_1, \dots, V_m)$. Then at least one of the following holds:*

1. *G has an IT with respect to $(V_1, \dots, V_m)$,*
2. *there is a set $\mathcal{B}$ of vertex classes and constellation K for $\mathcal{B}$ such that $V(K)$ dominates $G_{\mathcal{B}}$ and $|V(K)| \leq 2(|\mathcal{B}| - 1)$,*

Recall from Chapter 1 that the proofs of Theorems 2.3 and 2.4 are not algorithmic. In Chapter 3, we will present an algorithm that, for a large family of graphs, is able to efficiently find either an IT or a set $\mathcal{B}$ of vertex classes with a small dominating set that contains a constellation. This algorithm, called FindITorBD, applies to graphs that are *r-claw-free.*

Definition 2.7. A graph G with vertex partition $(V_1, \dots, V_m)$ is said to be *r-claw-free with respect to* $(V_1, \dots, V_m)$ if no vertex of G has r independent neighbours in distinct vertex classes.

Note that graphs with maximum degree Δ are $(\Delta + 1)$-claw-free.

We will discuss some applications of FindITorBD in Chapter 4, to problems in which r (or Δ) is constant. However, the runtime of FindITorBD is exponential in r (see Section 3.4). In Section 6.4, we will see a randomised algorithm, called FindWeightIT, that overcomes this limitation for graphs with maximum degree Δ, and how FindWeightIT provides algorithmic results for a more general class of problems.

2.3 Algorithmic IT Results

In Section 2.2, we discussed some known results for the existence of an IT for a given graph and vertex partition. In this section, we discuss the known algorithmic results that, given a graph and vertex partition, find an IT in the graph.

Most of the known algorithmic results have focused on finding an IT in graphs with maximum degree Δ under stronger vertex class size constraints than Theorem 2.3. Of these results, most have been obtained as applications of algorithmic versions of the Lovász Local Lemma (LLL) or its lopsided variant. In particular, these algorithmic constructions give an IT under the condition that the vertex classes have size at least $c\Delta$, where c is a constant strictly larger than 2.

The LLL, first proved by Erdős and Lovász [32], is a powerful probabilistic tool that can be used to show that a certain event holds with positive probability. In particular, it is well known that if a large number of events are independent (i.e. the occurrence of one does not affect the occurrence of another) and each occurs with probability less than 1, then there is a positive probability that none of the events occur. The LLL allows the independence condition to be relaxed slightly so that if the events are "mostly" independent and are not "too likely" to occur, then there is a positive probability that none of the events occur. The symmetric case of the LLL is the following.

Theorem 2.8 (Lovász Local Lemma (LLL); Symmetric [88])**.** *Let $A_1, \ldots, A_n$ be events in an arbitrary probability space. Suppose that each event A_i is mutually independent of all but at most d of the other events and that $\Pr[A_i] \leq p$ for all $1 \leq i \leq n$. If $ep(d+1) \leq 1$, then* $\Pr\left[\bigwedge_{i=1}^{n} \overline{A_i}\right] > 0$.

The first algorithmic proof of the LLL was due to Beck [17]. With somewhat more restrictive assumptions, Beck was able to efficiently find a particular type of 2-colouring of hypergraphs. Some improvements to this technique were made by Alon [10], Molloy and Reed [76], Czumaj and Scheideler [27], Srinivasan [89], and Bissacot, Fernández, Procacci and Scoppola [20].

In 2009, Moser and Tardos [79] introduced a new algorithmic proof for the LLL that used resampling. This new proof technique was much simpler than Beck's and is applicable to almost all applications of the LLL. In particular, Moser and Tardos proved the following.

Theorem 2.9 (Moser-Tardos Algorithm [79])**.** *There is a randomised algorithm which takes as input a probability space Ω in k independent variables $X_1, \ldots, X_k$ along with a* *collection $\mathcal{B}$ of "bad" events B in that space such that each event $B \in \mathcal{B}$ is a Boolean function of a subset of the variables* $\mathrm{Var}(B)$. *If the algorithm terminates, it returns a configuration $X = (X_1, \dots, X_k)$ for which all $B \in \mathcal{B}$ are false.*

If for inputs Ω and $\mathcal{B}$ there are parameters $p, d \geq 0$ such that $epd \leq 1$ and such that each $B \in \mathcal{B}$ has probability at most p and there are at most d bad events $B' \in \mathcal{B}$ such that $\mathrm{Var}(B) \cap \mathrm{Var}(B') \neq \emptyset$, *then the algorithm terminates with probability one and the expected runtime is polynomial in $|\mathcal{B}|$ and k.*

As Theorem 2.9 states, the algorithm of Moser and Tardos (or *Moser-Tardos algorithm*) converts the LLL to an efficient randomised algorithm. It has since been improved by several authors (see e.g. [65, 81, 66, 45, 1, 44]). Before discussing how the Moser-Tardos algorithm has been applied to the problem of finding ITs, we state some more properties of the Moser-Tardos algorithm that will be useful in the proof of our randomised algorithm in Chapter 6.

The Moser-Tardos algorithm is a randomised process which terminates with probability one in some configuration X. This configuration X can be regarded as a random variable, and we call the distribution on X the *MT-distribution.* The MT-distribution was first analysed by Haeupler, Saha, and Srinivasan in [39]. Additional bounds on the MT-distribution were shown by Harris and Srinivasan in [46]. One such bound from [46] is the following.

Theorem 2.10 ([46]). *Suppose Ω and $\mathcal{B}$ satisfy the conditions of Theorem 2.9. Let E be an event in the probability space Ω which is a Boolean function of a subset of variables* $\mathrm{Var}(E)$. *Suppose that there are r bad events $B \in \mathcal{B}$ with the property that* $\mathrm{Var}(E) \cap \mathrm{Var}(B) \neq \emptyset$. *Then, the probability that event E holds in the MT-distribution is at most* $e^{epr} \Pr_{\Omega}(E)$.

We will use these properties to devise a "degree-splitting" algorithm in Section 6.3.

In terms of finding ITs efficiently, using the Moser-Tardos approach (and the work of Bissacot et al. [20] and Pegden [81]), Harris and Srinivasan [47] gave a randomised algorithm that finds an IT in expected time $O(m\Delta)$ in vertex-partitioned graphs with m classes of size 4Δ. The current best result for polynomial expected time is due to Harris [42] who improved the bound on the class size to $4\Delta - 1$.

Deterministic algorithms to find ITs based on derandomizing the Moser-Tardos algorithm have also been studied, but they require the class sizes to be $C\Delta$ for some large constant C in order to find an IT efficiently [35, 43]. Some of these deterministic algorithms are also known to be parallelisable [25, 43].

More recently, an algorithmic result due to Annamalai [11, 13] has provided a new method for finding ITs efficiently without using the LLL. While studying problems related to the restricted max-min fair allocation problem (or Santa Claus problem), Annamalai [11, 13] proved an algorithmic version of Theorem 3.1 for the specific case of matchings in bipartite hypergraphs. We discuss this result and how it follows as a consequence of our main theorem (Theorem 3.1) in Section 4.1. For now, we will discuss the main ideas of Annamalai's algorithm that we will use in our own algorithm. To do so, consider the following sketch of the proof of Theorem 2.4.

Sketch of the proof of Theorem 2.4: Let M be a PIT and A be a vertex class such that $A \cap M = \emptyset$. We aim to alter M until it can be augmented by a vertex in A.

We build a "tree-like structure" T (which we describe as a vertex set inducing a forest of stars) as follows. Choose $x_1 \in A$ and set $T = \{x_1\}$. If $\deg_M(x_1) = 0$, then *improve* M by adding x_1 to M, and stop. Otherwise add $N_M(x_1)$ to T. Let $\mathcal{T} = \text{Vclass}(T)$.

In the general i^{th} step: it can easily be shown that $|T| \leq 2(|\mathcal{T}| - 1)$. This is because $\mathcal{T}$ can be treated as the vertex set of a tree-like structure where $V_i, V_j \in \mathcal{T}$ are adjacent if some $x_i \in V_i$ is adjacent in M to some $y_j \in V_j$. By construction, this $\mathcal{T}$ has $|\mathcal{T}| - 1$ edges, each of which corresponds to at most 2 vertices of T. Thus by assumption the subgraph $G_{\mathcal{T}}$ of G induced by $\bigcup\limits_{V_i \in \mathcal{T}} V_i$ is not dominated by T. Therefore there exists a vertex of $G_{\mathcal{T}}$ that is not adjacent to any vertex in T. Choose such a vertex x_i arbitrarily.

If $\deg_M(x_i) = 0$, then *improve* M by adding x_i to M and removing (if it exists) the M-vertex y in $\text{Vclass}(x_i)$. This forms a new PIT M, and is an improvement in the following sense: it reduces $\deg_M(x_j)$ where y was added to T because it was in $N_M(x_j)$ in an earlier step. Truncate T to $\{x_1\} \cup N_M(x_1) \cup \cdots \cup \{x_j\} \cup N_M(x_j)$.

Otherwise, add x_i and $N_M(x_i)$ to T. Thus $|\mathcal{T}|$ increases by $\deg_M(x_i) > 0$ and $|T|$ increases by $\deg_M(x_i) + 1$, which maintains $|T| \leq 2(|\mathcal{T}| - 1)$. See Figure 2.2 for T (the set of vertices shown) and $\text{Vclass}(T)$ (the set of classes enclosed by the dotted border). Note that T is a constellation for $\mathcal{T}$ whose centres are given by the x_i and whose leaves are given by the $N_M(x_i)$.

At each step we either grow T or reduce $\deg_M(x_j)$ for some j, until the current M can be extended to include a vertex of A. Therefore progress can be measured by a *signature vector*

$$(\deg_M(x_1), \ldots, \deg_M(x_t), \infty).$$

Note that this signature vector has at most $|M| + 1 \leq m$ entries since each $N_M(x_i)$ is a nonempty subset of M and all such sets are mutually disjoint. Each step reduces the lexicographic order of the signature vector. Thus the process terminates, and so we succeed in extending M to a larger PIT and eventually to an IT. □

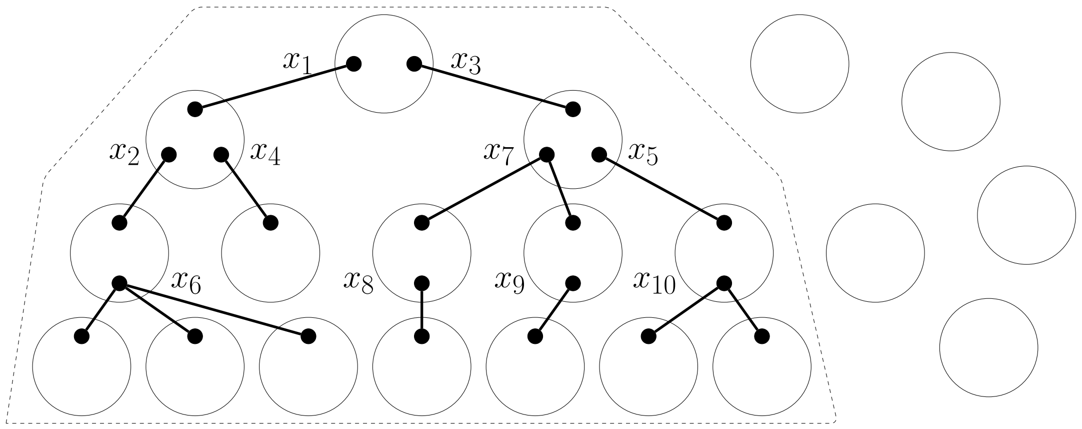

Figure 2.2: T after the 10th step. The classes enclosed by the dotted border form $\mathcal{T}$.

The drawback of the above procedure is that the number of signature vectors (and hence the number of steps) could potentially be as large as $(r-1)^m$ when G is r-claw-free. To make this approach into an efficient algorithm, we use Annamalai's idea from [11, 13] of a "lazy update," which essentially amounts to performing updates in "clusters" (large subsets of vertices) rather than for individual vertices (which change the $\deg_M(x_i)$ only one at a time). In particular, we make the following three modifications to the algorithm of the proof sketch.

1. Maintain layers: at each growth step, instead of choosing x_i arbitrarily, choose it to be a vertex in a class at smallest possible "distance" from the root class A, similar to a breadth-first search. Vertices x_i that are added into classes at the same distance from A are in the same *layer*.
2. Update in "clusters": instead of updating M when a single x_i satisfies $\deg_M(x_i) = 0$, update only when at least a *positive proportion* μ of an entire layer satisfies $\deg_M(x) = 0$. Discard later layers.
3. Rebuild layers in "clusters": after an update, add new vertices x_i to a layer of T only if doing so would add a μ proportion of that layer. Then discard later layers.

We will see these steps in more detail in Chapter 3. However, we give a basic overview of the benefits of these modifications here.

A consequence of *maintaining layers* is that the vertices in $G_{\mathcal{T}}$ that do not have a neighbour in T tend to "pile up" towards the bottom layer of T. This is because we try to add undominated vertices to the structure as soon as they appear. This results in the set of vertices of T associated with the bottom layer having size a positive proportion ρ of $|T|$. This implies that the total number of layers is always logarithmic in m since with each new layer, the total size of T increases by a fixed factor larger than one.

Updating in clusters and *rebuilding layers in clusters* allows for a different signature vector to be used. In particular, the new signature vector will measure the sizes of layers rather than degrees of individual vertices $\deg_M(x_i)$. It will have two entries per layer: the first is essentially $-\lfloor \log x \rfloor$ where x is the number of vertices x_i associated with that layer, and the second is essentially $\lfloor \log y \rfloor$ where y is the total size of their neighbourhoods in M (which is also the size of the layer). Updating M in a cluster (Modification 2) decreases the value of y for a layer by a positive proportion. Rebuilding a layer in a cluster (Modification 3) increases the value of x for a layer by a positive proportion. Hence (with suitably chosen bases for the logarithms) these updates always *decrease* the relevant entry by an integer amount. Therefore, as in the proof of Theorem 2.4, each update decreases the signature vector lexicographically.

Since the length of the signature vector is proportional to the number of layers, as noted above this is logarithmic in m. The entries are also of the order $\log m$. While this gives a very significant improvement over the signature vector from the proof of Theorem 2.4, it still does not quite give a polynomial number of signature vectors. However, as in [11, 13] it can be shown with a suitable alteration to the suggested signature vector, each signature vector can be associated with a subset of integers from 1 to x, where x is of order $\log m$. It then follows that the number of signature vectors, and hence the number of steps in the algorithm, is $\text{poly}(m)$.

As mentioned before, similar modifications were used by Annamalai [11, 13] only in the context of efficiently finding perfect matchings in bipartite hypergraphs. In Chapter 3, we will use the above modifications to prove an algorithmic version of Theorem 2.4 for r-claw-free graphs. This will allow us to improve the best known algorithmic versions of Theorem 2.3 to requiring classes of size at least $2\Delta + 1$, which is 1 away from being best possible.

Chapter 3

The Algorithm FindITorBD

In this chapter, we prove the first main result (Theorem 3.1) of this thesis. It is an algorithmic version of Theorem 2.4 (in a form similar to Theorem 2.6) for r-claw-free graphs. In particular, we provide an algorithm, called FindITorBD, for r-claw-free graphs that efficiently finds either an IT or a set of vertex classes dominated by a small set of vertices. We refer the reader to Section 2.2 for the definition of r-claw-free graphs (Definition 2.7) and constellations (Definition 2.5).

Theorem 3.1. *There exists an algorithm* FindITorBD *that takes as input any graph G with vertex partition $(V_1, \ldots, V_m)$ such that G is r-claw-free with respect to $(V_1, \ldots, V_m)$ and finds either:*

1. *an IT in G, or*
2. *a non-empty set $\mathcal{B}$ of vertex classes and a set D of vertices of G such that D dominates $G_{\mathcal{B}}$ in G and $|D| < (2+\epsilon)(|\mathcal{B}|-1)$. Moreover D contains $V(K)$ for a constellation K for some $\mathcal{B}_0 \supseteq \mathcal{B}$, where $|D \setminus V(K)| < \epsilon(|\mathcal{B}|-1)$.*

The runtime is $O(|V(G)|^{f(r,\epsilon)})$, which for fixed r and ϵ, is $\text{poly}(|V(G)|)$.

We remark that for fixed r, it is possible to adjust the algorithm FindITorBD to return either outcome (1), outcome (2), or a set of $r+1$ vertices S such that there exists $v \in S$ for which $S \setminus \{v\} \subseteq N(v)$ is an independent set of vertices in distinct vertex classes. This last outcome shows that G is not r-claw-free with respect to the provided vertex partition. However, we will not consider this generalisation.

Recall from Theorem 2.4, that for any graph G and vertex partition $(V_1, \dots, V_m)$, it is possible for both an IT with respect to $(V_1, \dots, V_m)$, as well as a set $\mathcal{B}$ of vertex classes where $V(K)$ for a constellation K for $\mathcal{B}$ dominates $G_{\mathcal{B}}$, to exist simultaneously. In such instances, the algorithm FindITorBD will terminate after providing only one of outcomes (1) and (2) of Theorem 3.1. However, for certain pairs of graphs and vertex partitions, it can be shown that outcome (2) of Theorem 3.1 never occurs, thus ensuring that FindITorBD always returns an IT on such inputs.

For example, consider a graph with maximum degree Δ. From Section 2.2, we know that any graph with maximum degree Δ is $(\Delta+1)$-claw-free with respect to any partition. Let G be a graph with maximum degree Δ and $(V_1, \dots, V_m)$ a vertex partition such that $|V_i| \geq 2\Delta + 1$ for each i. Then for $r = \Delta + 1$ and $\epsilon = \frac{1}{\Delta}$, outcome (2) of Theorem 3.1 would imply that there exists a non-empty set $\mathcal{B}$ of vertex classes and a set D of vertices of G such that D dominates $G_{\mathcal{B}}$ in G and $|D| < \left(2 + \frac{1}{\Delta}\right)(|\mathcal{B}| - 1)$. Since the maximum degree in G is Δ, D can dominate a subgraph on at most $\Delta|D|$ vertices. Hence

$$|V(G_{\mathcal{B}})| \leq \Delta|D| < \Delta\left(2 + \frac{1}{\Delta}\right)(|\mathcal{B}| - 1) = (2\Delta + 1)(|\mathcal{B}| - 1).$$

This is a contradiction since every vertex class has size at least $2\Delta + 1$ and so $|V(G_{\mathcal{B}})| \geq (2\Delta + 1)|\mathcal{B}|$. This gives the following algorithmic version of Theorem 2.3.

Corollary 3.2. *The algorithm* FindITorBD *on inputs G with maximum degree Δ and vertex partition $(V_1, \dots, V_m)$ such that $|V_i| \geq 2\Delta + 1$ for each i will return an IT in G. The runtime is $O(|V(G)|^{f(\Delta)})$, which for fixed Δ, is* poly$(|V(G)|)$.

We will see more examples of graphs and vertex partition pairs for which FindITorBD will never return outcome (2) in Chapter 4.

The proof of Theorem 3.1 will use the proof of Theorem 2.4 (as sketched in Section 2.3) as well as the modifications discussed in Section 2.3. The main idea is to "grow" a PIT using a "tree-like" structure until either you find an IT or the vertex classes in the tree are "too dominated" to grow, in which case you can find a set of vertex classes in the tree that have a small dominating set. To show that this process completes in polynomial time for fixed r and ϵ, we use a similar *signature* to that of Annamalai [11, 13] (see Section 3.4).

As mentioned in Chapter 1, the results in this chapter are joint work with Penny Haxell. The content of this chapter constitutes the main result of our joint paper [38].

This chapter is organised as follows. In Section 3.1, we introduce the terms and new notation used throughout the chapter. In Section 3.2, we present and discuss the three subroutines that are used by FindITorBD. We begin the analysis of FindITorBD in Section 3.3 by analysing the main subroutine of FindITorBD, called GrowTransversal. In Section 3.4, we define the signature vector used to prove that GrowTransversal terminates in polynomial time when r and ϵ are fixed. We conclude in Section 3.5 by providing the algorithm FindITorBD and using the results of earlier sections to prove Theorem 3.1.

3.1 Preliminaries

In this section we introduce the terminology and notation used throughout this chapter. Much of the terminology follows that of Annamalai from [11, 13]. For terms and notation not defined in this section, we refer the reader to Section 2.1 and [28].

For the remainder of this chapter, let G be a graph and let $(V_1, \ldots, V_m)$ be a vertex partition of G such that G is r-claw-free with respect to $(V_1, \ldots, V_m)$. Since the case $m = 1$ is trivial, we assume from now on that $m \geq 2$.

We claim that we may also assume that each vertex class V_i is an independent set of vertices. To see this, let F be the set of edges $uv \in E(G)$ such that $\mathrm{Vclass}(u) = \mathrm{Vclass}(v)$ and let $G' = G - F$. As every edge $uv \in E(G)$ such that $\mathrm{Vclass}(u) \neq \mathrm{Vclass}(v)$ is in $E(G')$, it is clear that a set M is an IT of G' if and only if it is an IT of G. Also, recall from Section 2.1 that $G_{\mathcal{B}}$ does not include any edges between vertices in the same vertex class in $\mathcal{B}$. Thus $G_{\mathcal{B}} = G'_{\mathcal{B}}$ for all $\mathcal{B} \subseteq \{V_1, \ldots, V_,\}$. Hence we assume without loss of generality that each vertex class is an independent set of vertices.

For a PIT M of G, note that any isolated vertex v in a vertex class where $\mathrm{Vclass}(v) \cap M = \emptyset$ can be added to M to create a larger PIT. Thus, we may remove the vertex classes from $(V_1, \ldots, V_m)$ that contain at least one isolated vertex and consider the induced subgraph of the remaining vertex classes as G under the partition formed by the remaining classes. Moreover, any subset $\mathcal{B}$ of the remaining vertex classes and set D of vertices satisfying outcome (2) of Theorem 3.1 under the new graph and vertex partition will still satisfy the conditions of outcome (2) for G and $(V_1, \ldots, V_m)$. Hence we assume without loss of generality that G does not contain an isolated vertex. Furthermore, we may assume that $r \geq 2$ (a graph is 1-claw-free if every vertex has no neighbour).

For the definitions that follow, let M be a PIT of G.

Definition 3.3. A vertex u *blocks* a vertex v if $u \in M$ and $uv \in E(G)$.

Definition 3.4. A vertex v is *immediately addable* with respect to M if $v \notin M$ and it has no vertices in $V(G)$ blocking it. For $W \subseteq V(G)$, $I_M(W)$ denotes the set of vertices in W

that are immediately addable with respect to M. Thus

$$I_M(W) = \{v \in W \setminus M : M \cup \{v\} \text{ is an independent set}\}.$$

Definition 3.5. Let A be a vertex class in the vertex partition of G that does not contain a vertex in M. An *alternating tree* T with respect to M and A is a structure that consists of some vertices $X \cup Y$ with $X \subseteq V(G) \setminus M$ and $Y \subseteq M$ and some edges xy where $x \in X$ and $y \in Y$ that satisfy the following properties:

(T1) The edges of T are all of the edges in $G[X \cup M]$.

(T2) Every vertex of Y has degree 1 in T.

(T3) Identifying the vertices of T in the same vertex class produces a tree $\mathcal{T}(T)$ whose vertices are $\mathrm{Vclass}(X \cup Y)$.

(T4) The *root* of T is the vertex class A.

Figure 3.1 provides an example of an alternating tree.

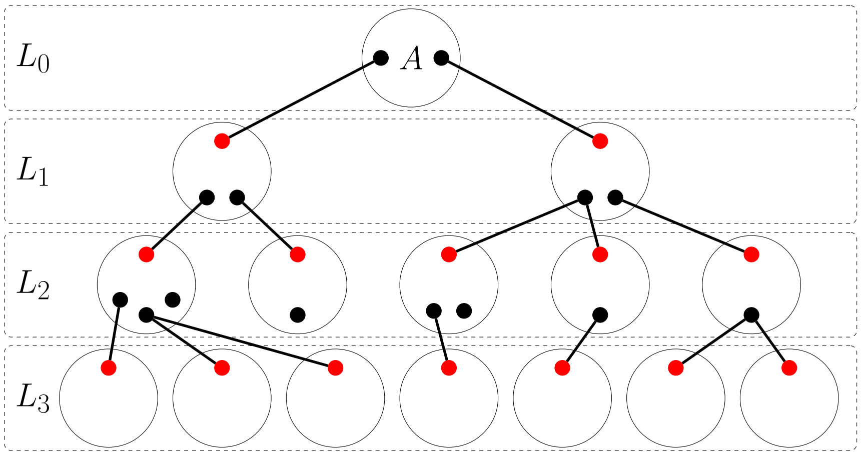

Figure 3.1: An alternating tree T for a PIT M and vertex class A. The circles are vertex classes of T and vertex classes in the same layer of T are enclosed with a dotted border. The vertices in X are shown in black and the vertices in Y are shown in red.

Note that property (T1) implies that X is an independent set since every edge of T is of the form xy for $x \in X$ and $y \in Y$ (note $X \cap Y = \emptyset$). Also, property (T2) implies that the connected components of T are stars whose leaves are in Y and whose centres are in

X. Moreover, it follows that for vertex classes V_i and V_j in T with V_i the "parent" of V_j in $\mathcal{T}(T)$, then V_j must contain a vertex, say $y_j \in M$, and T contains a unique edge $x_i y_j$ with $x_i \in V_i \setminus M$. It therefore follows that the subgraph of G induced by the vertices in $(X \cup Y) \setminus I_M(X)$ is a constellation for $\text{Vclass}(Y) \cup \{A\}$ (see Definition 2.5).

The vertex classes of T (which are $\text{Vclass}(X \cup Y)$) can be partitioned into *layers* by distance from the root as follows. Let L_i be the set of vertex classes of T in layer i, with $L_0 = \{A\}$. For each $i \geq 1$, let X_i be the vertices in X in vertex classes of L_{i-1} and let Y_i be the vertices in Y in vertex classes of L_i. Hence the edges xy with $x \in X_i$ and $y \in Y_i$ join a vertex from a vertex class in L_{i-1} to a vertex from a vertex class in L_i. We also define $X_0 = \emptyset$ and $Y_0 = \emptyset$.

Figure 3.1 also shows the layers of the alternating tree, as well as shows the vertices in X_i and Y_i in different colours.

We now describe how to construct the next layer of an alternating tree. Suppose we are given an alternating tree T with layers $L_0, \ldots, L_\ell$ for some $\ell \geq 0$. Then $L_{\ell+1}$, $X_{\ell+1}$, and $Y_{\ell+1}$ are built as follows. For each vertex x_i in the union of the classes in L_ℓ, we add x_i to $X_{\ell+1}$ and $N_M(x_i)$ to $Y_{\ell+1}$ if doing so preserves (T1)-(T4). To make this process deterministic, we use a vertex order $v_1, \ldots, v_n$ (where $n = |V(G)|$) so that for each $i = 1, \ldots, n$, if $\text{Vclass}(v_i) \in L_\ell$, $v_i \notin V(T)$, and $V(T) \cup \{v_i\}$ is an independent set, then we add v_i to $X_{\ell+1}$ and to T. Then for every edge $v_i y$ with $y \in M$, we add y to $Y_{\ell+1}$ and to T. Note that any such y is in a new vertex class, which we add to $L_{\ell+1}$ and to T. The final sets $L_{\ell+1}$, $X_{\ell+1}$, and $Y_{\ell+1}$ after v_n has been tested satisfy the definitions provided above for $i = \ell + 1$.

Let T be an alternating tree of G with respect to M and a vertex class A and let ℓ be the index of the final layer of T. For all $0 \leq j \leq \ell$, we define $X_{\leq j}$ to be the union of the X_i for $0 \leq i \leq j$, i.e. $X_{\leq j} = \bigcup_{i=0}^{j} X_i$. Similarly, we define $Y_{\leq j} = \bigcup_{i=0}^{j} Y_i$. Note that $\text{Vclass}(Y_{\leq \ell}) \cup \{A\}$ is the set of vertex classes of T, i.e. $\text{Vclass}(Y_{\leq \ell}) \cup \{A\} = \bigcup_{i=0}^{\ell} L_i$. It follows from Definitions 3.5 and 2.5 that the subgraph of G induced by $(X_{\leq \ell} \cup Y_{\leq \ell}) \setminus I_M(X_{\leq \ell})$ is a constellation for $\text{Vclass}(Y_{\leq \ell}) \cup \{A\}$ (see Figures 2.1 and 3.1).

The algorithm FindITorBD uses alternating trees as the "tree-like structure" described in the sketch of the proof of Theorem 2.4 (found in Section 2.3). Recall from Section 2.3 that the three changes FindITorBD makes to the algorithm of the sketch were to maintain layers, perform updates to M in clusters, and rebuild layers in clusters after an update occurs. From the description of how the next layer of an alternating tree is constructed, it is clear that alternating trees maintain layers. For the other two modifications, we will need to define certain properties an alternating tree may have. These are defined in terms of fixed parameters U and μ (which are defined in Definition 3.8). Before giving these properties, we introduce two more definitions.

Definition 3.6. Let T be an alternating tree of G with respect to M and a vertex class A, ℓ be the index of the final layer of T, and $X, Y \subseteq V(G)$ be from a partially built layer $\ell + 1$. A vertex v in a class in L_ℓ is *addable* for X, Y, and T if $v \notin V(T) \cup X \cup Y$, $|\operatorname{Vclass}(v) \cap X| < U$, and there does not exist a vertex $u \in V(T) \cup X \cup Y$ such that $uv \in E(G)$.

Definition 3.7. Layer i is *collapsible* if $I_M(X_i) > \mu|X_i|$.

We now introduce the following properties an alternating tree may have with respect to fixed parameters U and μ.

(P1) For each i, we have that $|X_i \cap V_j| \leq U$ for each $V_j \in L_{i-1}$.

(P2) For each i, layer i is not collapsible (i.e. $|I_M(X_i)| \leq \mu|X_i|$).

(P3) For each i there are fewer than $\mu|X_i|$ addable vertices with respect to X_i, Y_i, and T.

Property (P1) means that instead of adding every vertex of X in a vertex class in L_{i-1} that preserves all of the properties of an alternating tree to X_i, we only permit ourselves to add a limited amount. Specifically, we add at most U vertices of X per vertex class in L_{i-1} to X_i. By limiting the number of vertices in X_i a vertex class may contain, we are able to give bounds on the size of certain sets of vertices. This allows us to show some of the properties of outcome (2) of Theorem 3.1 on $\mathcal{B}$ and D hold as well as prove that the runtime of FindITorBD is polynomial in $|V(G)|$ when r and ϵ are fixed.

Property (P2) is used to determine when an update of M should be performed and when a layer should be rebuilt. Specifically, when layer ℓ is constructed, if $|I_M(X_\ell)| > \mu|X_\ell|$, then we have a significant number of immediately addable vertices in X_ℓ. We will see that we can make a significant enough update to M that Y_ℓ decreases by a positive proportion. This decreases the signature vector lexicographically. Hence we will update M and adjust the alternating tree accordingly.

Property (P3) ensures that after a modification is made to M, we check each layer i of the tree to see if X_i can be increased by $\mu|X_i|$ vertices while maintaining properties (T1)-(T4) and (P1). If so, then we can rebuild layer i so that X_i is significantly larger. This will decrease the signature vector lexicographically. Hence we will rebuild layer i and remove all later layers from our tree.

We will show that at the start and end of each iteration of GrowTransversal (which is a subroutine of FindITorBD), the structure T satisfies properties (T1)-(T4) as well as (P1)-(P3) in Section 3.3.

Along with the fixed parameters U and μ, there is a third fixed parameter, called ρ, that GrowTransversal will use. The values of U, μ, and ρ are chosen in advance and depend only on the inputs r and ϵ. The following notion formalises a suitable choice for these values.

Definition 3.8. The tuple (U, μ, ρ) of positive real numbers is *feasible* for (r, ϵ), where $r \geq 2$ and $\epsilon > 0$, if the following hold:

1. $(2+\epsilon)\left[1 - \frac{1}{U}\left(\frac{1+\mu U}{1-\mu} + \rho\right)\right] > \left(\frac{2+\mu(r+2)+\rho(r+1)}{1-\mu}\right)$,
2. $\epsilon\left[1 - \frac{1}{U}\left(\frac{1+\mu U}{1-\mu} + \rho\right)\right] > \frac{\mu(r+4)+\rho(r+2)}{1-\mu}$, and
3. $U - \mu\rho > \rho$.

Note that when r and ϵ are fixed, U, μ, and ρ are also fixed constants. As an example, $(U, \mu, \rho) = \left(\frac{10r}{\epsilon}, \frac{\epsilon}{10r}, \frac{\epsilon}{10r}\right)$ is feasible for (r, ϵ) when $r \geq 2$ and $0 < \epsilon < 1$. We do not make an attempt here to choose the constants in a way that optimises the running time.

3.2 Subroutines of FindITorBD

Let G and $(V_1, \ldots, V_m)$ be a graph and vertex partition such that:

(A1) $m \geq 2$ and $r \geq 2$,

(A2) G is r-claw free with respect to $(V_1, \ldots, V_m)$,

(A3) Each V_i is an independent set of vertices, and

(A4) G does not contain an isolated vertex.

(See Section 3.1 for explanations as to why we can assume (A1)-(A4) hold.) We will also assume that the vertices of G have been assigned an ordering $v_1, \ldots, v_n$, where $|V(G)| = n$.

Let M be a PIT in G with respect to $(V_1, \ldots, V_m)$ and let A be a vertex class that does not contain a vertex in M. The main idea of FindITorBD will be to perform a series of modifications to M that will allow us to augment M with a vertex in A. If we are not successful, then we will find a subset of classes (based on an alternating tree of G) that has a small dominating set.

We hold off on presenting the algorithm FindITorBD until Section 3.5. Instead, we present the subroutines that FindITorBD will use to perform these modifications. In the following subsections, we describe three algorithms that are used by FindITorBD. The first two algorithms, called BuildLayer (Section 3.2.1) and SuperposedBuild (Section 3.2.2), are used as subroutines in the third algorithm, called GrowTransversal (Section 3.2.3). GrowTransversal appears as the main subroutine of FindITorBD.

For the remainder of Section 3.2, let T be an alternating tree of G with respect to M and A. Let U, μ, and ρ be fixed and chosen in advance so that (U, μ, ρ) is feasible for (r, ϵ) (see Definition 3.8). Also, we will refer to properties (T1)-(T4) of Definition 3.5 and (P1)-(P3) from Section 3.1.

3.2.1 The subroutine BuildLayer

BuildLayer is a subroutine in the main algorithm for augmenting M that helps construct new layers for T. The function takes as inputs an alternating tree T that satisfies property (P1), the index ℓ of the final layer in T, and two sets X and Y of vertices which are $X_{\ell+1}$ and $Y_{\ell+1}$ for a "partially built" layer $\ell+1$ that also satisfies property (P1). It then creates a new layer $\ell+1$ by augmenting X and Y and returning the resulting pair X' and Y' together with Vclass(Y), which will be $X_{\ell+1}$, $Y_{\ell+1}$, and $L_{\ell+1}$ respectively in an alternating tree that satisfies (P1) whose final layer is indexed by $\ell+1$.

We give the algorithm BuildLayer in 3.2.1. Note that the conditions in line 6 can be rewritten as v_i is addable for X', Y', and T.

Claim 3.9. *Let T, ℓ, X, and Y be the inputs and X', Y', and L' be the outputs of* BuildLayer*. The structure T' with vertex set $V(T) \cup X' \cup Y'$ and edge set $E(G[V(T) \cup X' \cup M])$ is an alternating tree with layers $L_0, \ldots, L_\ell, L'$ that satisfies (P1).*

Proof: The proof is by induction on $|X' \setminus X|$. As T is an alternating tree satisfying (P1) by the conditions on the input, the claim holds when $|X' \setminus X| = 0$ (which necessarily implies $|Y' \setminus Y| = 0$).

Suppose T' with $V(T') = V(T) \cup X' \cup Y'$ and $E(T') = E(G[V(T) \cup X' \cup M])$ is an alternating tree satisfying (P1). It suffices to show that adding an addable vertex v for X',

3.2.1 BuildLayer

Input: An alternating tree T that satisfies (P1), the index ℓ of the final layer in T, and two sets X and Y of vertices which are $X_{\ell+1}$ and $Y_{\ell+1}$ for a partially built layer $\ell+1$ that satisfies (P1).

Output: The sets $X_{\ell+1}$, $Y_{\ell+1}$, and $L_{\ell+1}$ for an alternating tree that satisfies (P1).

```
function BuildLayer($T, \ell, X, Y$)
    $X' := X$
    $Y' := Y$
    $L' := \text{Vclass}(Y)$
    for $i = 1, \ldots, n$ do
        if $\text{Vclass}(v_i) \in L_\ell$, $v_i \notin V(T) \cup X' \cup Y'$, $v_iu \notin E(G)$ for all $u \in V(T) \cup X' \cup Y'$, and $|\text{Vclass}(v_i) \cap X'| < U$ then
            $X' := X' \cup \{v_i\}$
            $Y' := Y' \cup \{y \in M : v_iy \in E(G)\}$
            $L' := \text{Vclass}(Y')$
    return $X', Y', L'$
```

Y', and T (as well as $N_M(v)$) to T' preserves this. We begin by verifying the structure is an alternating tree. (See Definition 3.5.)

Let v be an addable vertex for X', Y', and T. If $\ell = 0$, then T consists only of L_0 and so v being addable for X', Y', and T means that $\text{Vclass}(v) = A$. As $A \cap M = \emptyset$, we therefore have $v \in V(G) \setminus M$ and $vu \notin E(G)$ for all $u \in X' \cup Y'$ (recall $X_0 = \emptyset$ and $Y_0 = \emptyset$). Thus $N_M(v) \cap Y' = \emptyset$ and so the edge set of the graph induced by $X' \cup Y' \cup \{v\} \cup N_M(v)$ consists only of the edges in $G[X' \cup \{v\} \cup M]$. Hence (T1) is satisfied. Furthermore, $N_M(v) \cap Y' = \emptyset$ and T' being an alternating tree means that every vertex of Y' still has degree 1 in the new structure. Every vertex of $N_M(v)$ also has degree 1 since M is an independent set. Thus every vertex of $Y' \cup N_M(v)$ has degree 1 in the structure and so (T2) is satisfied. Identifying the vertices in the same vertex class produces a star whose centre is A and whose leaves are the elements of $\text{Vclass}(Y' \cup N_M(v))$, hence (T3) is also satisfied. The root is still clearly A as it is the only vertex class that does not contain a vertex in M, so (T4) also holds. Therefore the structure formed by adding v and $N_M(v)$ to T' is an alternating tree. Moreover, since v is addable for X', Y', and T, there are fewer than U vertices in $X' \cap \text{Vclass}(v) = X' \cap A$. Hence $|(X' \cup \{v\}) \cap A| \leq U$ and $|X' \cap V_i| = 1$ for all other vertex classes V_i in the structure. Thus (P1) is also satisfied.

Now suppose $\ell \geq 1$. Then $\text{Vclass}(v) \in L_\ell$ and $v \notin V(T')$. Hence $v \in V(G) \setminus M$ since $\text{Vclass}(v) \cap M \subseteq V(T')$. Also, $vu \notin E(G)$ for all $u \in V(T')$ since v is addable for X',

Y', and T. Hence $N_M(v) \cap Y = \emptyset$ and so $E(G[V(T') \cup \{v\} \cup N_M(v)])$ consists only of the edges in $G[V(T') \cup \{v\} \cup M]$. Hence (T1) is satisfied. Furthermore, $N_M(v) \cap V(T') = \emptyset$ since v is not adjacent to any $u \in V(T')$. Since T' an alternating tree, every vertex of Y' still has degree 1 in the new structure and every vertex of $N_M(v)$ also has degree 1 since M is an independent set. Thus every vertex of $Y' \cup N_M(v)$ has degree 1 in the structure and so (T2) is satisfied. Identifying the vertices in the same vertex class produces a tree since the only added vertex classes are those of Vclass($N_M(v)$), which are all adjacent to Vclass(v) (which is a vertex of $\mathcal{T}(T')$ since Vclass$(v) \in L_\ell$). Hence (T3) is also satisfied. The root is still clearly A as it is the only vertex class that does not contain a vertex in M, so (T4) also holds. Therefore the structure formed by adding v and $N_M(v)$ to T' is an alternating tree. Moreover, since v is addable for X', Y', and T, there are fewer than U vertices in $(V(T') \setminus M) \cap \text{Vclass}(v)$. Hence $|((V(T') \setminus M) \cup \{v\}) \cap \text{Vclass}(v)| \leq U$ and $|(V(T') \setminus M) \cap V_i| \leq U$ for all other vertex classes V_i in the structure. Thus (P1) is also satisfied.

This concludes the proof. □

3.2.2 The subroutine SuperposedBuild

SuperposedBuild is a subroutine in the main algorithm for augmenting M that, after a modification of M occurs in the algorithm, modifies T so that it remains an alternating tree with respect to the new PIT M and A as well as satisfies (P1) and (P3). SuperposedBuild possibly augments T by adding some vertices that are no longer blocked due to the modification of M. The function takes as input an alternating tree T that satisfies (P1) and the index ℓ of the final layer of T. It then performs some tests on the layers of T, to see if any X_i could be substantially enlarged, and returns a possibly modified alternating tree T' that satisfies (P1) and (P3) as well as the index ℓ' of the final layer of T'. We give the algorithm SuperposedBuild in 3.2.2.

Note that SuperposedBuild performs BuildLayer on every layer of T. This may seem strange as SuperposedBuild is only used when the main algorithm for augmenting M (GrowTransversal) modifies the vertices in M in some vertex classes in L_ℓ. However, the reason SuperposedBuild must perform BuildLayer on every layer and not just layer ℓ is to ensure the output T' satisfies property (P3). We can see this in the following example.

We give an example for how a modification to layer ℓ can affect a layer $i < \ell$ when layer i had been modified previously. Recall X' and Y' replace X_i and Y_i only when $|X'| \geq (1 + \mu)|X_i|$. Let x_1 and x_2 be vertices in vertex classes of L_{i-1} that are not in $V(T)$ and that share neighbours $y \in Y_i$ and $w \in Y_\ell$. Suppose that during an iteration in

3.2.2 SuperposedBuild

Input: An alternating tree T that satisfies (P1) and the index ℓ of the final layer of T.
Output: An alternating tree T' that satisfies (P1) and (P3) and the index ℓ' of the final layer of T'.

```
1:  function SuperposedBuild($T, \ell$)
2:      for $i = 1, \ldots, \ell$ do
3:          $T_i :=$ the alternating tree with layers $L_0, \ldots, L_{i-1}$
4:          $(X', Y', L') :=$ BuildLayer$(T_i, i-1, X_i, Y_i)$
5:          if $|X'| \geq (1+\mu)|X_i|$ then
6:              $X_i := X'$
7:              $Y_i := Y'$
8:              $L_i := L'$
9:              $T' :=$ the alternating tree with layers $L_0, \ldots, L_i$
10:             return $T', i$
11:     return $T, \ell$
```

GrowTransversal, a modification to M in layer i removes y from $M \cap Y_i$ and that doing so makes x_1 and x_2 addable for X_i, $Y_i \setminus \{y\}$, and T_i, where T_i is the alternating tree with layers $L_0, \ldots, L_{i-1}$. Since $w \in N_M(x_1) \cap N_M(x_2)$, BuildLayer$(T_i, i-1, X_i, Y_i \setminus \{y\})$ can only add one of x_1 and x_2 to X'. If the resulting output X' satisfies $|X'| < (1+\mu)|X_i|$, then SuperposedBuild leaves X_i unchanged and GrowTransversal continues its sequence of building layers and modifying M. Hence neither x_1 nor x_2 is in the alternating tree (note that this is why w can appear in Y_ℓ). Now suppose a modification to layer ℓ removes w from $M \cap Y_\ell$. Again, x_1 and x_2 are addable for X_i, $Y_i \setminus \{y\}$, and T_i. However, now adding x_1 to X' does not prevent x_2 from being added to X'. This increases the size of the output X' returned by BuildLayer$(T_i, i-1, X_i, Y_i \setminus \{y\})$, which makes it possible for $|X'| \geq (1+\mu)|X_i|$ after this modification. Therefore, to maintain property (P3), every layer must be checked after any modification to M.

Claim 3.10. *The structure T' returned by* SuperposedBuild(T, ℓ) *is an alternating tree that satisfies properties (P1) and (P3) whose final layer is indexed by ℓ'.*

Proof: Let T be an alternating tree that satisfies (P1) and let ℓ be the index of the final layer of T. We show that the output T' of SuperposedBuild(T, ℓ) is an alternating tree that satisfies properties (P1) and (P3) and ℓ' is the index of its final layer.

Suppose that for each i, BuildLayer$(T_i, i-1, X_i, Y_i)$ returns an X' where
$|X'| < (1+\mu)|X_i|$. Then SuperposedBuild returns T and ℓ, the index of its final layer.

Hence the output satisfies (P1) be assumption. It satisfies (P3) because each X' is a union of X_i and some addable vertices for X_i, Y_i, and T_i. Therefore $|X'| < (1+\mu)|X_i|$ means that there are fewer than $\mu|X_i|$ vertices $v \in V(G) \setminus (M \cup X_i)$ such that v is not blocked, $\text{Vclass}(v) \in L_{i-1}$, and $|\text{Vclass}(v) \cap X_i| < U$.

Now suppose BuildLayer$(T_i, i-1, X_i, Y_i)$ returns an X' where $|X'| \geq (1+\mu)|X_i|$ for some i. Then the outputs X', Y', and L' returned by BuildLayer replace X_i, Y_i and L_i respectively. By Claim 3.9, the structure T' formed by adding X', Y', and L' to T_i is an alternating tree T' with layers $L_0, \ldots, L_{i-1}, L'$ that satisfies (P1). As SuperposedBuild outputs T' and i (which is the index of the final layer of T'), we therefore know that it outputs an alternating tree satisfying (P1). The alternating tree satisfies (P3) because for SuperposedBuild to perform BuildLayer$(T_i, i-1, X_i, Y_i)$, it performed BuildLayer$(T_j, j-1, X_j, Y_j)$ for all $1 \leq j < i$ and did not stop. Hence the condition $|X'| \geq (1+\mu)|X_j|$ was not satisfied for any $1 \leq j < i$. Moreover, $(X', Y', L') =$ BuildLayer$(T_i, i-1, X_i, Y_i)$, so BuildLayer$(T_i, i-1, X', Y')$ would not return a larger tree (see BuildLayer). Hence $|X'| < (1+\mu)|X'|$ and so (P3) holds.

This concludes the proof. □

3.2.3 The subroutine GrowTransversal

GrowTransversal is the main algorithm for augmenting M. It takes as inputs M and a vertex class A and performs a series of modifications to M until either a vertex in A is added to M or an iteration constructs a layer i with too small an X_i relative to the size of T. When GrowTransversal terminates, it returns the final PIT M', a graph T, and an integer ℓ. If GrowTransversal terminates due to an iteration constructing a layer i with at most $\rho|Y_{\leq i-1}|$ vertices in X_i, then T is an alternating tree with respect to M' and A that satisfies properties (P1)-(P3) whose final layer is indexed by i. Otherwise, GrowTransversal returns a PIT M' containing a vertex in A (with $|M'| = |M| + 1$), an empty graph T, and 0. If the output of GrowTransversal has $i > 0$, we will show in the next section that T contains a subset $\mathcal{B}$ of vertex classes whose vertices are dominated by a set of fewer than $(2+\epsilon)|\mathcal{B}|$ vertices with the properties stated in outcome (2) of Theorem 3.1. We give the algorithm GrowTransversal in 3.2.3.

GrowTransversal begins by initialising the alternating tree T with respect to M and A as well as its number of layers ℓ. While A does not contain a vertex in the PIT M', the algorithm repeats a building layer operation (line 6) followed by a loop of *collapsing* operations (lines 12-24) that modify M' when enough immediately addable vertices with

3.2.3 GrowTransversal

Input: A PIT M and a vertex class A that does not contain a vertex in M.

Output: Either a PIT M', an alternating tree T with respect to M' and A satisfying properties (P1)-(P3), and $\ell+1$ the index of the final layer of T, or a PIT M' containing a vertex in each of Vclass$(M) \cup \{A\}$, an empty graph, and 0.

```
1:  function GrowTransversal($M, A$)
2:      Initialise $X_0 = \emptyset$, $Y_0 = \emptyset$, $L_0 = \{A\}$, $M' := M$.
3:      $T :=$ the alternating tree with layer $L_0$
4:      $\ell := 0$
5:      while $A \cap M' = \emptyset$ do
6:          $(X_{\ell+1}, Y_{\ell+1}, L_{\ell+1}) :=$ BuildLayer$(T, \ell, \emptyset, \emptyset)$
7:          $T :=$ the alternating tree with layers $L_0, \ldots, L_{\ell+1}$
8:          if $|X_{\ell+1}| \leq \rho |Y_{\leq \ell}|$ then
9:              return $(M', T, \ell + 1)$
10:         else
11:             $\ell := \ell + 1$
12:             while $|I_{M'}(X_\ell)| > \mu |X_\ell|$ do
13:                 if $\ell = 1$ then
14:                     $M' := M' \cup \{u\}$ for $u \in I_{M'}(X_1)$ of smallest index.
15:                     $T := \emptyset$
16:                     return $(M', T, 0)$
17:                 else
18:                     for $i = 1, \ldots, n$ do
19:                         if $v_i \in Y_{\ell-1}$ and $I_{M'}(X_\ell) \cap \text{Vclass}(v_i) \neq \emptyset$ then
20:                             $M' := (M' \setminus \{v_i\}) \cup \{u\}$ for $u \in I_{M'}(X_\ell) \cap \text{Vclass}(v_i)$ of smallest
    index.
21:                             $Y_{\ell-1} := Y_{\ell-1} \setminus \{v_i\}$
22:                             $L_{\ell-1} := L_{\ell-1} \setminus \text{Vclass}(v_i)$
23:                     $T' :=$ the alternating tree with layers $L_0, \ldots, L_{\ell-1}$
24:                     $(T, \ell) :=$ SuperposedBuild$(T', \ell - 1)$
```

respect to M' are present in the newly constructed layer. Figure 3.2 shows an example of one collapse operation (lines 19-22).

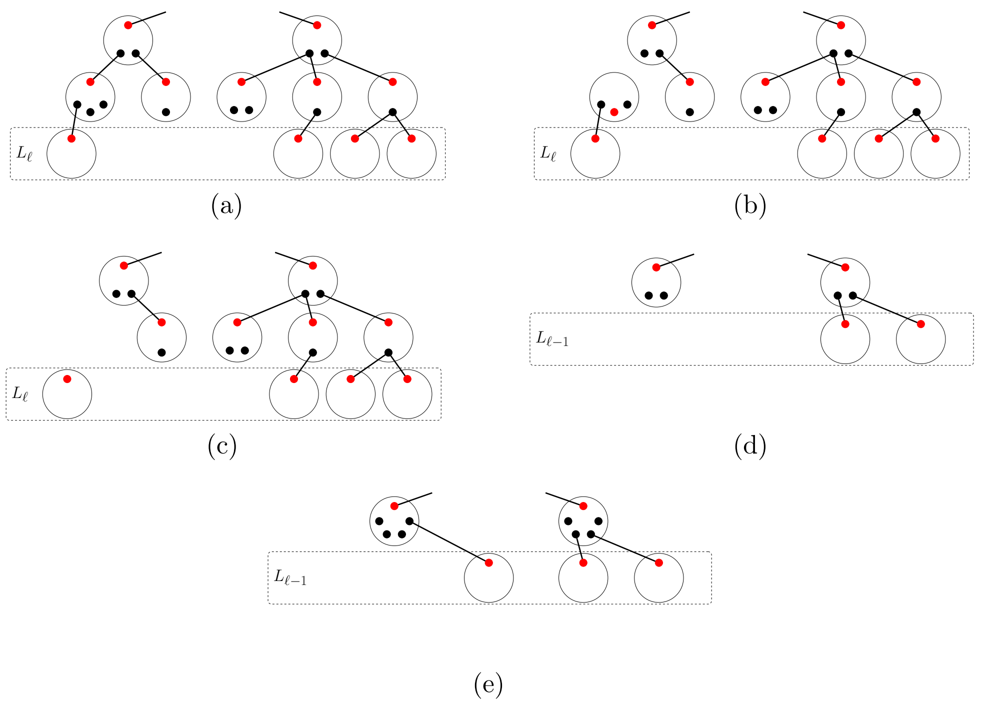

Figure 3.2: An example of an iteration of the loop of collapsing operations, where layer $\ell > 1$ is collapsed. The relevant part of alternating tree T at the start of the iteration is shown in (a). Images (b) and (c) together show one iteration of the for loop that changes M', $Y_{\ell-1}$, and $L_{\ell-1}$. Specifically, (b) shows T after M' and $Y_{\ell-1}$ are modified and (c) shows the same section at the end of the iteration of the for loop. The result of the for loop (i.e. T') is shown in (d) and this section in the final alternating tree after SuperposedBuild is applied is shown in (e).

Note that performing one collapse operation may result in other vertices in the vertex classes of earlier layers becoming addable with respect to the new M'. Hence one collapse operation can lead to a sequence of collapse operations. It remains to show that after one iteration of the while loop of collapsing operations completes, the structure T is an alternating tree that satisfies properties (P1)-(P3) or the empty graph (which only occurs when lines 13-16 are implemented).

Claim 3.11. *Suppose that at the start of the while loop of collapsing operations, T is an alternating tree with respect to M' and A whose final layer is indexed by ℓ and satisfies* *property (P1) for all layers and property (P2) for all but possibly layer ℓ. Then after the while loop completes, the structure T is either an alternating tree with respect to (a possibly different) M' and A that satisfies properties (P1)-(P3) or the empty graph.*

Proof: Let T be an alternating tree with respect to M' and A that satisfies property (P1) for all layers and property (P2) for all but possibly layer ℓ, where ℓ is the index of the final layer of T. If lines 13-16 are implemented, then T is set to the empty graph and the algorithm terminates. Otherwise, $\ell > 1$ and X_ℓ contains more than $\mu|X_\ell|$ vertices in vertex classes of $L_{\ell-1}$ that are immediately addable vertices with respect to M'.

Each vertex class in $L_{\ell-1}$ contains exactly one vertex in $Y_{\ell-1}$ (since M' is a PIT and $Y_{\ell-1} \subseteq M$). Thus lines 19-22 exchange exactly two vertices per $V_j \in L_{\ell-1}$ where $|V_j \cap I_{M'}(X_\ell)| \geq 1$. As each vertex $u \in I_{M'}(X_\ell)$ is immediately addable with respect to M' and the vertex $v_i \in M'$ is removed from M when the vertex u of smallest index in $I_{M'}(X_\ell) \cap \text{Vclass}(v_i)$ is added to M', M' remains a PIT. Moreover, Vclass(v_i) is removed from $L_{\ell-1}$, and so the structure T' is a subgraph of T. Hence T' satisfies properties (T2)-(T4) since T satisfies them. Also, since each $u \in X_\ell$ is not adjacent to any $V(T) \setminus Y_\ell$ by construction, we have (T1) holds (any new edge to M' would be to a vertex in X_ℓ). Thus T' is an alternating tree with respect to the new M' and A.

As we have not changed the number of vertices in any vertex class in $\bigcup_{i=0}^{\ell-1} L_i$ by line 22, we know T' is an alternating tree satisfying (P1) since T satisfied (P1). Hence by Claim 3.10, the output T of SuperposedBuild$(T', \ell - 1)$ is an alternating tree satisfying (P1) and (P3). Therefore each iteration of the while loop ends with T being an alternating tree with respect to M' and A that satisfies (P1) and (P3) (unless lines 13-16 are implemented).

Let T and M' be the alternating tree and PIT at the completion of the while loop of collapse operations. Note that each layer $0 \leq i < \ell$ is left unchanged by the while loop. This is because the BuildLayer operation for layer i in the algorithm SuperposedBuild is the only operation applied to layer i. As SuperposedBuild removes all layers after the one it adjusts and the while loop of collapse operations does not add layers to the tree, we can therefore conclude that $|I_{M'}(X_i)|$ remains unchanged for all $0 \leq i < \ell$. Since the loop continues until $|I_{M'}(X_\ell)| \leq \mu|X_\ell|$, we can conclude (P2) holds for T.

This concludes the proof. □

Claim 3.12. *At the start of any iteration of the main while loop of* GrowTransversal*, T is an alternating tree with respect to M' and A that satisfies properties (P1)-(P3).*

Proof: For the first iteration of the while loop, the statement holds trivially as T is an alternating tree with respect to $M' = M$ and A that contains no vertices. We show that, given an iteration of the while loop starts with an alternating tree T with respect to M' and A that satisfies properties (P1)-(P3), it ends with an alternating tree T' with respect to a (possibly different) M' and A or terminates.

Let T be an alternating tree with respect to M' and A that satisfies properties (P1)-(P3) and let ℓ be the index of the final layer of T. Then by Claim 3.9, the structure T' formed by adding $X_{\ell+1}$, $Y_{\ell+1}$, and $L_{\ell+1}$ to T is an alternating tree that satisfies (P1). Moreover BuildLayer does not affect the vertices in layers $L_0, \ldots, L_\ell$, and so $|I_{M'}(X_i)| \leq \mu|X_i|$ for all $0 \leq i \leq \ell$. If lines 8-9 are implemented, the while loop terminates. Thus suppose they are not implemented.

If $|I_{M'}(X_{\ell+1})| \leq \mu|X_{\ell+1}|$, then T' satisfies (P2) and the loop of collapsing operations is not performed. Hence (P3) follows from T' satisfying (P1) and (P2) together with the fact that BuildLayer constructs the entire layer. Therefore T' satisfies (P1)-(P3). Now suppose $|I_{M'}(X_{\ell+1})| > \mu|X_{\ell+1}|$. Then by Claim 3.10, the output T' (and M') at the end of the loop of collapsing operations is an alternating tree with respect to M' and A that satisfies properties (P1)-(P3). This concludes the proof. □

Note that in GrowTransversal, the PIT M' is only changed during iterations of the loop of collapsing operations. Moreover, a vertex in A can only be added to M' if $\ell = 1$ at the start of an iteration of this loop. This is because $A \in L_0$ and for $\ell \neq 1$, the vertices v_i have $\text{Vclass}(v_i) \in L_j$ for some $j \geq 1$. Hence $A \cap M' = \emptyset$ throughout the main while loop unless lines 13-16 are performed, which terminates the loop. Thus GrowTransversal returns either a PIT M' and an alternating tree T with respect to M' and A whose final layer is indexed by $i > 0$, or GrowTransversal returns a PIT M' with $A \cap M' \neq \emptyset$ and an empty graph T (with index 0).

We will show in the next section that given an alternating tree T with respect to M and A such that $|X_{\ell+1}| \leq \rho|Y_{\leq \ell}|$, where $\ell+1$ is the index of the final layer of T, there exists some set $\mathcal{B}$ of the vertex classes in T such that $G_{\mathcal{B}}$ is dominated by a set D of vertices with the properties stated in outcome (2) of Theorem 3.1. Our analysis will provide a specific $\mathcal{B}$ and its corresponding D given that GrowTransversal returns $(M, T, \ell+1)$ for some $\ell + 1 > 0$.

3.3 Analysis of GrowTransversal

Recall from Section 3.2 that we assume G is an r-claw-free graph with respect to vertex partition $(V_1, \ldots, V_m)$, each vertex class V_i is an independent set of vertices, $r \geq 2$, and

$\epsilon > 0$ (i.e. assume (A1)-(A4)). We also assume that U, μ, and ρ are fixed such that (U, μ, ρ) is feasible for (r, ϵ). Also, note that the algorithms in Section 3.2 assume that the vertices of the graph G have been assigned some arbitrary but fixed ordering, and the vertices are processed by our algorithms subject to this ordering.

In this section, we prove that when BuildLayer constructs a layer $\ell + 1$ where $|X_{\ell+1}| \leq \rho|Y_{\leq \ell}|$, we can give a set $\mathcal{B}$ of vertex classes and set D of vertices that satisfy the conditions of outcome (2) of Theorem 3.1. To do so, we first give some consequences of the alternating trees in GrowTransversal satisfying properties (P1)-(P3).

Lemma 3.13. *Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop in* GrowTransversal*. Then the layers of T are not collapsible and $|Y_i| \geq (1 - \mu)|X_i|$ for each $i \neq 0$.*

Proof: The proof is by induction on the number of iterations of the main while loop in GrowTransversal. In the first iteration, T is the alternating tree with respect to $M' = M$ and A with $X_0 = \emptyset$, $Y_0 = \emptyset$, and $L_0 = \{A\}$. Hence $|I_{M'}(X_0)| = |X_0| = 0$ and so layer 0 is not collapsible (see Definition 3.7). Also, there is no layer i in T for which $i \neq 0$ and so the conditions hold.

Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop and let ℓ be the index of the final layer of T. Suppose the layers of T are not collapsible and $|Y_i| \geq (1 - \mu)|X_i|$ for each layer $i \neq 0$ in T. We show that the alternating tree T' at the end of iteration satisfies these two properties as well. (Note that T' is also the alternating tree at the start of the next iteration of the main while loop.)

During this iteration, note that layer $\ell + 1$ is constructed. If GrowTransversal does not terminate and layer $\ell + 1$ is not collapsible, then the first property holds for T' since none of the earlier layers are modified.

If GrowTransversal does not terminate and layer $\ell + 1$ is collapsible, let T' be the alternating tree with respect to M' at the end of the loop of collapsing operations and let ℓ' be the index of the final layer of T'. Unless the algorithm terminates, $\ell' \geq 1$.

Each iteration of this loop reduces the number of layers in the tree, but does not modify any layer j with $0 \leq j < \ell'$. Thus layer j of T' is not collapsible for all $0 \leq j < \ell'$ since it is also layer j of T. Layer ℓ' of T' may be different than layer ℓ' of T, however the condition of the loop of collapsing operations implies that layer ℓ' is not collapsible (since ℓ' is the final layer of T' when the loop completes). Hence none of the layers in T' are collapsible (unless the algorithm terminates during the iteration).

To show the second property holds for T', recall that Y_i contains all of the blocking vertices of all of the vertices in X_i by (T1) and every vertex in Y_i is adjacent to exactly one vertex in X_i by (T2). Thus, there are at most $|Y_i|$ vertices in $X_i \setminus I_{M'}(X_i)$ and at most $\mu|X_i|$ vertices in $I_{M'}(X_i)$ (since (P2) holds) for each $i = 1, \ldots, \ell'$. Hence $|X_i| \leq |Y_i| + \mu|X_i|$ for each $i \in \{1, \ldots, \ell'\}$. □

Lemma 3.14. *Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop in* GrowTransversal *and let ℓ be the index of the final layer of T. Then for each $i \neq 0$,*

$$(X_i', Y_i', L_i') := \text{BuildLayer}(T_i, i-1, X_i, Y_i)$$

satisfies $|X_i'| < (1+\mu)|X_i|$, where T_i is the alternating tree with respect to M' and A with layers $L_0, \ldots, L_{i-1}$.

Proof: Consider layer i of T for some $i \neq 0$. Note that the vertex sets X_i and Y_i were either constructed during an implementation of BuildLayer or an implementation of SuperposedBuild in some previous iteration of the main while loop of GrowTransversal. In either case, an (additional) application of BuildLayer$(T_i, i-1, X_i, Y_i)$ could not increase the size of X_i since both BuildLayer and SuperposedBuild created X_i to be as large as possible with respect to M' during its construction.

Suppose no layer built between the iteration that constructed layer i of T and the current iteration is collapsible. Then the loop of collapsing operations is never performed, and so BuildLayer$(T_i, i-1, X_i, Y_i)$ would return $X_i' = X_i$, $Y_i' = Y_i$, and $L_i' = L_i$. Hence $|X_i'| < (1+\mu)|X_i|$.

Now suppose some layer built between the iteration that constructed layer i of T and the current iteration was collapsible. Note that the index of the collapsible layer must be greater than i as otherwise L_i would be discarded. Thus, for each $j \geq i+1$ such that layer j is collapsible, SuperposedBuild performs BuildLayer on layer i to try to augment X_i by at least a μ proportion of its size. However, since L_i is a layer in T at the start of the current iteration, SuperposedBuild does not succeed in changing X_i. Hence BuildLayer$(T_i, i-1, X_i, Y_i)$ does not increase the number of vertices in X_i by $\mu|X_i|$, and so $|X_i'| < (1+\mu)|X_i|$. □

Lemma 3.15. *Suppose M', T, and $\ell+1$ are the outputs of* GrowTransversal *with $\ell+1 > 0$. Then*

$$|Y_{\leq \ell}' \setminus Y_{\leq \ell}| < \mu(r-1)|X_{\leq \ell}|,$$

where $Y_0' = \emptyset$ and all other Y_i' are given by $(X_i', Y_i', L_i') :=$ BuildLayer$(T_i, i-1, X_i, Y_i)$ for T_i the alternating tree with respect to M' and A with layers $L_0, \ldots, L_{i-1}$.

Proof: Let $v \in Y'_{\leq \ell} \setminus Y_{\leq \ell}$. Then by definition $v \in M$, and hence v blocks its neighbours. If $uv \in E(G)$ for any $u \in X_i$ such that $0 \leq i \leq \ell$, v blocks u and so v would be included in Y_i (see BuildLayer). This implies $v \in Y_{\leq \ell}$, which is a contradiction. Therefore $N(v) \cap X_{\leq \ell} = \emptyset$. However, $v \in Y'_i \setminus Y_i$ for some $1 \leq i \leq \ell$ and, by the construction of X'_i and Y'_i, v is adjacent to exactly one $u \in X'_i$ (see BuildLayer). Thus v has a neighbour in $X'_i \setminus X_i$ and so v has a neighbour in $X'_{\leq \ell} \setminus X_{\leq \ell}$.

Note that for each $1 \leq i \leq \ell$, Y'_i is a set of independent vertices in distinct vertex classes. As G is r-claw-free, each vertex of X'_i has at most $r-1$ independent neighbours in different vertex classes. By Lemma 3.14, $|X'_i \setminus X_i| < (1+\mu)|X_i| - |X_i| = \mu|X_i|$ since layer i is a layer of the alternating tree at the start of the iteration in which GrowTransversal terminates (recall $i < \ell + 1$). Hence $|X'_{\leq \ell} \setminus X_{\leq \ell}| < \mu |X_{\leq \ell}|$, and so

$$|Y'_{\leq \ell} \setminus Y_{\leq \ell}| \leq (r-1)|X'_{\leq \ell} \setminus X_{\leq \ell}| < \mu(r-1)|X_{\leq \ell}|. \qquad \square$$

We are now ready to prove the main result of this section.

Lemma 3.16. *Suppose M', T, and $\ell+1$ are the outputs of* GrowTransversal *with $\ell+1 > 0$. Then*

$$\mathcal{B} = (\text{Vclass}(Y_{\leq \ell}) \setminus \mathcal{U}) \setminus \left(\bigcup_{i=1}^{\ell} \text{Vclass}(X'_i \setminus X_i) \right)$$

and

$$D = X'_{\leq \ell} \cup Y'_{\leq \ell} \cup X_{\ell+1} \cup Y_{\ell+1} \cup S$$

satisfy the conditions of outcome (2) of Theorem 3.1, where

- *$\mathcal{U} \subseteq \text{Vclass}(Y_{\leq \ell})$ is the set of vertex classes V_j such that $|V_j \cap X_{\leq \ell+1}| = U$,*
- *$(X'_i, Y'_i, L'_i) = \text{BuildLayer}(T_i, i-1, X_i, Y_i)$ for each $1 \leq i \leq \ell$ with T_i being the alternating tree with respect to M' and A with layers $L_0, \ldots, L_{i-1}$,*
- *$W = X'_{\leq \ell} \cup Y'_{\leq \ell} \cup X_{\ell+1} \cup Y_{\ell+1}$, and*
- *S is the set of all $u \in V(G)$ for which $u \in N(v)$ for some $v \in I_{M'}(W)$ and u has the smallest index amongst the vertices in $N(v)$.*

Proof: Let T, M', A, $\mathcal{B}$, $\mathcal{U}$, X'_i, Y'_i, T_i, W, and S be as defined in the lemma statement. Define $B_0 = \text{Vclass}(Y_{\leq \ell}) \cup \{A\}$ and note $\mathcal{B} \subseteq \mathcal{B}_0$. We will show D contains the vertex set of a constellation for $\mathcal{B}_0$ (Claim 3.17), D dominates $G_{\mathcal{B}}$ (Claim 3.18), $|D| < (2+\epsilon)(|\mathcal{B}|-1)$ (Claim 3.22), and $|D \setminus V(K)| < \epsilon(|\mathcal{B}|-1)$ (Claim 3.23).

Claim 3.17. *$K = G[(X_{\le \ell} \cup Y_{\le \ell}) \setminus I_M(X_{\le \ell})]$ is a constellation for $\mathcal{B}_0$.*

Proof: As T is an alternating tree with respect to M' and A, it is clear T_ℓ is an alternating tree with respect to M' and A with vertex set $X_{\le \ell} \cup Y_{\le \ell}$. Hence K is a constellation for $\mathcal{B}_0 = \mathrm{Vclass}(Y_{\le \ell}) \cup \{A\}$ (see discussion after Definition 3.5). □

Clearly $V(K) \subseteq D$ since $X_i \subseteq X_i'$ and $Y_i \subseteq Y_i'$ by construction (see BuildLayer).

Claim 3.18. *The set $W \cup S$ dominates $G_{\mathcal{B}}$.*

Proof: Let $B = \bigcup\limits_{V_i \in \mathcal{B}} V_i$. We show $W \cup S$ dominates $G_{\mathcal{B}}$ by showing W dominates $G[B \setminus I_{M'}(W)]$ and S dominates $G[I_{M'}(W)]$.

Let $u \in B \setminus I_{M'}(W)$. Note that by the choice of $\mathcal{B}$, we know $\mathcal{B}$ does not contain a vertex class in $\mathrm{Vclass}(X_i' \setminus X_i)$ for any $1 \le i \le \ell$. As $X_0 = \emptyset$, we therefore have that $u \notin X_{\le \ell}' \setminus X_{\le \ell}$.

Suppose $u \in W \setminus I_{M'}(W)$. Then $u \in (X_{\le \ell+1} \cup Y_{\le \ell}' \cup Y_{\ell+1}) \setminus I_{M'}(W)$. If $u \in Y_i'$ for some $i \in \{1, \dots, \ell\}$, then the construction of (X_i', Y_i', L_i') and (X_i, Y_i, L_i) by BuildLayer implies that u has a neighbour v in X_i' (see BuildLayer), which means $v \in W$. Similarly, if $u \in Y_{\ell+1}$, then u has a neighbour v in $X_{\ell+1}$. If $u \in X_i$ for some $i \in \{1, \dots, \ell+1\}$, then since $u \notin I_{M'}(W)$, u has a neighbour v that blocks u. By the construction of X_i and Y_i in GrowTransversal, $v \in Y_i$ and so $v \in W$. Therefore, every $u \in W \setminus I_{M'}(W)$ has a neighbour in W. Thus we may assume $u \in B \setminus (W \cup I_{M'}(W))$.

Note that each vertex class in $\mathcal{B}$ has at most one vertex in M' and that these vertices are in $Y_{\le \ell}'$. Thus $u \notin M'$. Since $\mathrm{Vclass}(u) \in \mathcal{B}$, let i be the layer of T such that $\mathrm{Vclass}(u) \in L_{i-1}$.

Suppose $i < \ell+1$ and u has no neighbours in $X_{\le i}' \cup Y_{\le i}'$. As $\mathcal{B}$ contains no vertex classes in $\mathcal{U}$, $\mathrm{Vclass}(u)$ contains fewer than U vertices in X_i. Furthermore, $\mathrm{Vclass}(u)$ contains no vertices in $X_i' \setminus X_i$. Thus by Definition 3.6, u is an addable vertex for X_i, Y_i, and T_i. Hence $\mathrm{BuildLayer}(T_i, i-1, X_i, Y_i)$ would not stop until either u is added to X_i' or u has a neighbour in $X_i' \cup Y_i'$. As $u \in B \setminus (W \cup I_{M'}(W))$ and $X_i' \subseteq W$, we know that $u \notin X_i'$. Therefore u has a neighbour in $X_i' \cup Y_i'$.

Now suppose $i = \ell+1$ and u has no neighbours in W. Then again $\mathrm{Vclass}(u)$ contains fewer than U vertices in X_i and $\mathrm{Vclass}(u)$ contains no vertices in $X_i' \setminus X_i$. Thus by Definition 3.6, u is an addable vertex for $X_{\ell+1}$, $Y_{\ell+1}$, and $T_{\ell+1}$. Hence $\mathrm{BuildLayer}(T_{\ell+1}, \ell, \emptyset, \emptyset)$ would not stop until either u is added to $X_{\ell+1}$ or u has a neighbour in $X_{\ell+1} \cup Y_{\ell+1}$. As

$u \in B\backslash(W \cup I_{M'}(W))$ and $X_{\ell+1} \subseteq W$, we know that $u \notin X_{\ell+1}$. Therefore u has a neighbour in $X_{\ell+1} \cup Y_{\ell+1}$.

Thus W dominates $G[B \setminus I_{M'}(W)]$. For $u \in I_{M'}(W)$, note that $|N(u)| \geq 1$ by (A4). As the neighbour $v \in N(u)$ with the smallest index in the ordering is in S, clearly S dominates $G[I_{M'}(W)]$. Thus $W \cup S$ dominates $G_{\mathcal{B}}$. □

To bound $|D|$ and $|D \setminus V(K)|$, we break the calculations into a few steps. Let

$$Q = (X'_{\leq\ell} \setminus X_{\leq\ell}) \cup (Y'_{\leq\ell} \setminus Y_{\leq\ell}) \cup X_{\ell+1} \cup Y_{\ell+1} \cup S.$$

Note that $D = X_{\leq\ell} \cup Y_{\leq\ell} \cup Q$ and $D \setminus V(K) = Q \cup I_M(X_{\leq\ell})$. We therefore begin our calculations by bounding $|\mathcal{B}| - 1$ (Claim 3.19), $|S|$ (Claim 3.20) and $|Q|$ (Claim 3.21). With these claims and Lemma 3.15, we can then bound $|D|$ (Claim 3.22) and $|D \setminus V(K)|$ (Claim 3.23).

Claim 3.19. *We have*

$$|\mathcal{B}| - 1 \geq \left[1 - \frac{1}{U}\left(\frac{1+\mu U}{1-\mu} + \rho\right)\right] |Y_{\leq\ell}|.$$

Proof: Note that the vertex classes in $\mathcal{B}$ are the vertex classes of T that do not contain U vertices in $X_{\leq\ell+1}$ and do not contain any addable vertices for X_i, Y_i, and T_i for all $1 \leq i \leq \ell$. We use these facts to bound $|\mathcal{B}|$ from below as follows.

Recall that $|\operatorname{Vclass}(Y_{\leq\ell}) \cup \{A\}| = |Y_{\leq\ell}| + 1$. It is clear that $|\mathcal{U}| \leq \frac{|X_{\leq\ell+1}|}{U}$. By Lemma 3.14, $|X'_i| < (1+\mu)|X_i|$ for each $i \in \{1, \ldots, \ell\}$. As $X_i \subseteq X'_i$, this implies that there are at most $\mu|X_i|$ vertices in $X'_i \backslash X_i$. Thus $|\operatorname{Vclass}(X'_i \backslash X_i)| \leq \mu|X_i|$ for all $1 \leq i \leq \ell$ and so $\sum_{i=1}^{\ell} |\operatorname{Vclass}(X'_i \backslash X_i)| \leq \mu|X_{\leq\ell}|$. Also by Lemma 3.13, $|X_{\leq\ell}| \leq \frac{1}{1-\mu}|Y_{\leq\ell}|$. As GrowTransversal terminates with $\ell + 1 > 0$, we know from the algorithm that $|X_{\ell+1}| \leq \rho|Y_{\leq\ell}|$. Therefore,

$$\begin{aligned}
|\mathcal{B}| &\geq |\operatorname{Vclass}(Y_{\leq \ell}) \cup \{A\}| - |\mathcal{U}| - \left| \bigcup_{i=1}^{\ell} \operatorname{Vclass}(X'_i \setminus X_i) \right| \\
&\geq |\operatorname{Vclass}(Y_{\leq \ell})| - \frac{1}{U}|X_{\leq \ell+1}| - \mu |X_{\leq \ell}| \\
&= (|Y_{\leq \ell}| + 1) - \left(\frac{1}{U}|X_{\leq \ell}| + \frac{1}{U}|X_{\ell+1}| + \mu|X_{\leq \ell}| \right) \\
&\geq |Y_{\leq \ell}| + 1 - \left[\left(\frac{1}{U} + \mu \right) |X_{\leq \ell}| + \frac{\rho}{U}|Y_{\leq \ell}| \right] \\
&\geq |Y_{\leq \ell}| + 1 - \left[\left(\frac{1}{U} + \mu \right) \left(\frac{1}{1-\mu} \right) |Y_{\leq \ell}| + \frac{\rho}{U}|Y_{\leq \ell}| \right] \\
&= 1 + \left[1 - \frac{1}{U} \left(\frac{1+\mu U}{1-\mu} + \rho \right) \right] |Y_{\leq \ell}|.
\end{aligned}$$

□

Claim 3.20. *We have* $|S| \leq |I_{M'}(W)| < \frac{2\mu+\rho}{1-\mu}|Y_{\leq \ell}|$.

Proof: As S contains the neighbour with the smallest index in the ordering for each $v \in I_{M'}(W)$, we have $|S| \leq |I_{M'}(W)|$. By Lemma 3.14, $|X'_{\leq \ell}| < (1+\mu)|X_{\leq \ell}|$ and so $|X'_{\leq \ell} \setminus X_{\leq \ell}| < \mu |X_{\leq \ell}|$. Also, by definition, $I_{M'}(W) \subseteq X'_{\leq \ell} \cup X_{\ell+1}$ (see Definition 3.4). Thus,

$$\begin{aligned}
|I_{M'}(W)| &= |I_{M'}(X'_{\leq \ell} \cup X_{\ell+1})| \\
&= |I_{M'}(X'_{\leq \ell})| + |I_{M'}(X_{\ell+1})| \\
&\leq |I_{M'}(X_{\leq \ell})| + |I_{M'}(X'_{\leq \ell} \setminus X_{\leq \ell})| + |X_{\ell+1}| \\
&< \mu |X_{\leq \ell}| + \mu |X_{\leq \ell}| + |X_{\ell+1}| \\
&= 2\mu |X_{\leq \ell}| + |X_{\ell+1}|.
\end{aligned}$$

Recall that $|X_{\leq \ell}| \leq \frac{1}{1-\mu}|Y_{\leq \ell}|$ by Lemma 3.13. Since $\ell + 1 > 0$, we know that $|X_{\ell+1}| \leq \rho |Y_{\leq \ell}|$ (see GrowTransversal). Thus we obtain

$$|I_{M'}(W)| < \frac{2\mu}{1-\mu}|Y_{\leq \ell}| + \rho |Y_{\leq \ell}|,$$

from which the claim follows. □

Claim 3.21. *We have*

$$|Q| < \frac{\mu(r+2) + \rho(r+1)}{1-\mu}|Y_{\leq \ell}|.$$

Proof: Since G is r-claw-free we know that $|Y_{\ell+1}| \le (r-1)|X_{\ell+1}|$. Also, since $\ell+1>0$, we know that $|X_{\ell+1}| \le \rho|Y_{\le\ell}|$ and therefore $|Y_{\ell+1}| \le \rho(r-1)|Y_{\le\ell}|$. We bound each of the remaining three summands below using (respectively) Lemma 3.14, Claim 3.15, and Claim 3.20, to obtain

$$\begin{aligned}|Q| &\le |X'_{\le\ell}\setminus X_{\le\ell}| + |Y'_{\le\ell}\setminus Y_{\le\ell}| + |X_{\ell+1}| + |Y_{\ell+1}| + |S| \\ &\le \mu|X_{\le\ell}| + \mu(r-1)|X_{\le\ell}| + \rho|Y_{\le\ell}| + \rho(r-1)|Y_{\le\ell}| + \frac{2\mu+\rho}{1-\mu}|Y_{\le\ell}| \\ &= \mu r|X_{\le\ell}| + \rho r|Y_{\le\ell}| + \frac{2\mu+\rho}{1-\mu}|Y_{\le\ell}| \\ &< \mu r|X_{\le\ell}| + \frac{2\mu+\rho(r+1)}{1-\mu}|Y_{\le\ell}|.\end{aligned}$$

Since $|X_{\le\ell}| \le \frac{1}{1-\mu}|Y_{\le\ell}|$ by Lemma 3.13, we conclude $|Q| < \frac{\mu(r+2)+\rho(r+1)}{1-\mu}|Y_{\le\ell}|$. □

We can now bound $|D|$ and $|D\setminus V(K)|$.

Claim 3.22. *We have $|D| < (2+\epsilon)(|\mathcal{B}|-1)$.*

Proof: Claim 3.21 and Lemma 3.13 combine to give

$$\begin{aligned}|D| &= |X_{\le\ell}\cup Y_{\le\ell}\cup Q| \\ &< \left(\frac{1}{1-\mu} + 1 + \frac{\mu(r+2)+\rho(r+1)}{1-\mu}\right)|Y_{\le\ell}| \\ &< \left(\frac{2+\mu(r+2)+\rho(r+1)}{1-\mu}\right)|Y_{\le\ell}|.\end{aligned}$$

Thus by Claim 3.19 and Condition (1) of Definition 3.8 (feasibility of (U,μ,ρ)),

$$\begin{aligned}(2+\epsilon)(|\mathcal{B}|-1) &\ge (2+\epsilon)\left[1-\frac{1}{U}\left(\frac{1+\mu U}{1-\mu}+\rho\right)\right]|Y_{\le\ell}| \\ &> \left(\frac{2+\mu(r+2)+\rho(r+1)}{1-\mu}\right)|Y_{\le\ell}| \\ &> |D|.\end{aligned}$$

□

Claim 3.23. *We have $|D\setminus V(K)| < \epsilon(|\mathcal{B}|-1)$.*

Proof: To bound $|D \setminus V(K)|$ using Claim 3.21, note that $I_{M'}(X_{\leq \ell}) \subseteq I_{M'}(W)$. Thus Claim 3.20 implies that

$$\begin{aligned}|D \setminus V(K)| &= |Q \cup I_{M'}(X_{\leq \ell})| \\ &< \frac{\mu(r+2)+\rho(r+1)}{1-\mu}|Y_{\leq \ell}| + \frac{2\mu+\rho}{1-\mu}|Y_{\leq \ell}| \\ &= \frac{\mu(r+4)+\rho(r+2)}{1-\mu}|Y_{\leq \ell}|.\end{aligned}$$

Using Claim 3.19 again, and Condition (2) of Definition 3.8, we find that

$$\begin{aligned}\epsilon(|\mathcal{B}|-1) &\geq \epsilon\left[1 - \frac{1}{U}\left(\frac{1+\mu U}{1-\mu} + \rho\right)\right]|Y_{\leq \ell}| \\ &> \frac{\mu(r+4)+\rho(r+2)}{1-\mu}|Y_{\leq \ell}| \\ &> |D \setminus V(K)|.\end{aligned}$$

□

This completes the proof of Lemma 3.16. □

Note that by Lemma 3.16, if the size of the layer constructed at the beginning of an iteration of the main while loop of GrowTransversal is too small (less than a ρ proportion of the number of classes already in the tree), then GrowTransversal has found a set of vertex classes with a small dominating set satisfying the desired properties of outcome (2) of Theorem 3.1. Thus, it will be important for the proof of the runtime of FindITorBD to bound the number of layers an alternating tree in an iteration of GrowTransversal can have. We show that this is $c\log(m)$ (and give this c) in Lemma 3.24.

Lemma 3.24. *Suppose (U, μ, ρ) is feasible for (r, ϵ). The number of layers in the alternating tree T with respect to PIT M' and vertex class A maintained during the execution of* GrowTransversal *is always bounded by $c\log(m)$, where $c = \frac{1}{\log[1+\rho(1-\mu)]}$.*

Proof: Suppose T is an alternating tree with respect to M' and A at the beginning of some iteration of the main while loop in GrowTransversal and let ℓ be the index of the final layer of T. Consider any layer i where $1 \leq i \leq \ell$. By Lemma 3.13, $|Y_i| > (1-\mu)|X_i|$. Since GrowTransversal did not terminate in the iteration in which layer i was constructed, we know that

$$|Y_i| > (1-\mu)|X_i| > \rho(1-\mu)|Y_{\leq i-1}|.$$

Therefore, since $|Y_1| \geq 1$, we find $m \geq |Y_{\leq \ell}| = \sum_{i=0}^{\ell} |Y_i| > [1 + \rho(1-\mu)]^\ell$. Thus the number of layers at any moment of the algorithm is bounded above by $\frac{\log m}{\log[1+\rho(1-\mu)]}$. As μ and ρ are fixed parameters, this is $c \log(m)$ for $c = \frac{1}{\log[1+\rho(1-\mu)]}$. □

3.4 Signature Vectors

We begin this section by defining the signature vector of an alternating tree. Recall that U, μ, and ρ are fixed parameters such that (U, μ, ρ) is feasible for (r, ϵ). We will use these signature vectors to prove that GrowTransversal terminates after a polynomial in m number of iterations where the degree is a function of r and ϵ.

Definition 3.25. Let T be an alternating tree with respect to PIT M and vertex class A and let ℓ be the index of the final layer of T. The *signature* of layer i is defined to be

$$(s_{2i-1}, s_{2i}) = \left(- \left\lfloor \log_b \frac{\rho^{-i}}{(1-\mu)^{i-1}} |X_i| \right\rfloor , \left\lfloor \log_b \frac{\rho^{-i}}{(1-\mu)^{i}} |Y_i| \right\rfloor \right),$$

where $b = \frac{U}{U - \mu\rho}$. The *signature vector* of T is $s = (s_1, s_2, \ldots, s_{2\ell-1}, s_{2\ell}, \infty)$.

The above definition for the signature of a layer i is chosen so that the lexicographic value of the signature vector decreases whenever $|X_i|$ increases significantly (see Lemma 3.26 subcase 2.2) as well as decreases whenever $|Y_i|$ decreases significantly (Lemma 3.26 subcase 2.1). The factors of $\frac{\rho^{-i}}{(1-\mu)^{i-1}}$ and $\frac{\rho^{-i}}{(1-\mu)^{i}}$ exist to ensure that the coordinates of the signature vector are non-decreasing in absolute value, which we will show in Lemma 3.27.

We begin by showing that the lexicographic value of the signature vector decreases during each iteration of GrowTransversal.

Lemma 3.26. *The lexicographic value of the signature vector reduces across each iteration of the main while loop of* GrowTransversal *unless the algorithm terminates during that iteration.*

Proof: Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop of GrowTransversal and let ℓ be the index of the final layer of T. Let $s = (s_1, \ldots, s_{2\ell}, \infty)$ be the signature vector of T. There are two cases.

Case 1. No collapse operation occurs in this iteration, i.e. $|I_{M'}(X_{\ell+1})| \leq \mu |X_{\ell+1}|$.

Then the only modification to T in this iteration is a new layer $\ell+1$ is added to T. The new signature vector for T is therefore $s' = (s'_1, \ldots, s'_{2\ell+2}, \infty)$ where $s'_i = s_i$ for all $1 \le i \le 2\ell$ and $(s'_{2\ell+1}, s'_{2\ell+2})$ is the signature of layer $\ell+1$. Hence the lexicographic value is reduced.

Case 2. At least one collapse operation occurs in this iteration, i.e.
$|I_{M'}(X_{\ell+1})| > \mu|X_{\ell+1}|$.

Then the iteration contains a loop of collapsing operations. Let T' be the alternating tree with respect to M' and A at the start of the last iteration of this loop of collapsing operations and let ℓ' be the index of the final layer of T'. If $\ell' = 1$, then $|I_{M'}(X_1)| > \mu|X_1|$ and the algorithm terminates. Hence we may assume $\ell' > 1$.

Let T^* be the alternating tree with respect to M' and A at the end of this last iteration, i.e. (T^*, ℓ^*) is the output of SuperposedBuild$(T'_{\ell'}, \ell'-1)$, where $T'_{\ell'}$ is the alternating tree with layers $L_0, \ldots, L_{\ell'-1}$ after the modifications to M' are performed. There are two cases.

Subcase 2.1. We have $T^* = T'_{\ell'}$.

Then SuperposedBuild makes no modifications to $T'_{\ell'}$. Thus the only layer of T' modified during this last iteration is layer $\ell'-1$ (see GrowTransversal). Moreover, layers $L_0, \ldots, L_{\ell'-2}$ of T' are the same layers as T (see SuperposedBuild). Therefore the signature vector $s' = (s'_1, \ldots, s'_{2\ell'-2}, \infty)$ of T^* satisfies $s'_i = s_i$ for all $1 \le i \le 2\ell'-4$ and

$$(s'_{2\ell'-3}, s'_{2\ell'-2}) = \left(-\left\lfloor \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-2}} |X'_{\ell'-1}| \right\rfloor, \left\lfloor \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}} |Y'_{\ell'-1}| \right\rfloor \right).$$

As the modifications performed in the final iteration before SuperposedBuild is applied do not modify vertices of layer $\ell'-1$ not in M, $X_{\ell'-1}$ is not modified and so $X'_{\ell'-1} = X_{\ell'-1}$.

Note that each vertex class containing a vertex in $I_{M'}(X_\ell)$ must contain a vertex in $Y_{\ell-1}$. Since a vertex class contains at most U vertices in X_ℓ, there are at least $\frac{\mu}{U}|X_\ell|$ blocking vertices in $Y_{\ell-1}$ that are not in $T'_{\ell'}$ because of the modifications to M'. Since GrowTransversal did not terminate when layer ℓ' was constructed initially, we have $|X'_\ell| > \rho|Y_{\le \ell'-1}| \ge \rho|Y_{\ell'-1}|$ and so

$$|Y'_{\ell'-1}| \le |Y_{\ell'-1}| - \frac{\mu}{U}|X'_\ell| < \left(1 - \frac{\mu\rho}{U}\right)|Y_{\ell'-1}|.$$

Therefore,

$$\begin{aligned}\log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}}|Y'_{\ell'-1}| &< \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}}\left(1-\frac{\mu\rho}{U}\right)|Y_{\ell'-1}| \\ &\le \log_b \left(1-\frac{\mu\rho}{U}\right) + \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}}|Y_{\ell'-1}| \\ &\le \log_b \left(\frac{U-\mu\rho}{U}\right) + \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}}|Y_{\ell'-1}| \\ &= -1 + \log_b \frac{\rho^{-(\ell'-1)}}{(1-\mu)^{\ell'-1}}|Y_{\ell'-1}|.\end{aligned}$$

Hence $s'_{2\ell'-3} = s_{2\ell'-3}$ and $s'_{2\ell'-2} < s_{2\ell'-2}$, so the lexicographic value of the signature vector is reduced.

Subcase 2.2. We have $T^* \neq T'_{\ell'}$.

Then SuperposedBuild makes a modification to some layer ℓ^* of $T'_{\ell'}$. Hence $|X'_{\ell^*}| \ge (1+\mu)|X_{\ell^*}|$.

As (U,μ,ρ) is feasible, $\rho \le U - \mu\rho$. Hence $\frac{U}{U-\mu\rho} = 1 + \frac{\mu\rho}{U-\mu\rho} \le 1+\mu$. Thus,

$$\begin{aligned}\log_b \frac{\rho^{-\ell^*}}{(1-\mu)^{\ell^*-1}}|X'_{\ell^*}| &\ge \log_b \frac{\rho^{-\ell^*}}{(1-\mu)^{\ell^*-1}}(1+\mu)|X_{\ell^*}| \\ &= \log_b (1+\mu) + \log_b \frac{\rho^{-\ell^*}}{(1-\mu)^{\ell^*-1}}|X_{\ell^*}| \\ &\ge 1 + \log_b \frac{\rho^{-\ell^*}}{(1-\mu)^{\ell^*-1}}|X_{\ell^*}|,\end{aligned}$$

and so $s'_{2\ell^*-1} < s_{2\ell^*-1}$. Since SuperposedBuild and the loop of collapsing operations do not modify layers $L_0, \dots, L_{\ell^*-1}$, we see that $s'_i = s_i$ for all $1 \le i \le 2\ell^* - 2$. Hence the signature vector of T^* is $(s'_1, \dots, s'_{2\ell^*-1}, s'_{2\ell^*}, \infty)$. Thus the lexicographic value of the signature vector is reduced. □

Lemma 3.27. *The coordinates of the signature vector are non-decreasing in absolute value at the beginning of each iteration of the main while loop of* GrowTransversal*.*

Proof: Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop of GrowTransversal and let ℓ be the index of the final layer of T. Let $s = (s_1, \dots, s_{2\ell}, \infty)$ be the signature vector of T.

Consider layer i for some $1 \leq i \leq \ell$. Since $|Y_i| \geq (1-\mu)|X_i|$ by Lemma 3.13,

$$|s_{2i-1}| = \left\lfloor \log_b \frac{\rho^{-i}}{(1-\mu)^{i-1}} |X_i| \right\rfloor \leq \left\lfloor \log_b \frac{\rho^{-i}}{(1-\mu)^{i}} |Y_i| \right\rfloor = |s_{2i}|.$$

Hence the coordinates of the signature vector for a layer of T are non-decreasing in absolute value. Now consider layers i and $i+1$ for some $0 \leq i \leq \ell - 1$. As GrowTransversal did not terminate when layer $i+1$ was constructed initially, we have $|X_{i+1}| \geq \rho|Y_i|$ and so

$$|s_{2i}| = \left\lfloor \log_b \frac{\rho^{-i}}{(1-\mu)^{i}} |Y_i| \right\rfloor \leq \left\lfloor \log_b \frac{\rho^{-(i+1)}}{(1-\mu)^{i}} |X_{i+1}| \right\rfloor = |s_{2i+1}|.$$

Thus consecutive coordinates of the signature vector for different layers of T are also non-decreasing in absolute value. Hence the coordinates of the signature vector of T are non-decreasing in absolute value. □

We may now use Lemmas 3.26 and 3.27 to bound the total number of possible signature vectors.

Lemma 3.28. *Let T be the alternating tree with respect to PIT M' and vertex class A at the start of some iteration of the main while loop of* GrowTransversal *and let ℓ be the index of the final layer of T. The number of possible signature vectors for T is bounded by $m^{f(r,\epsilon)}$.*

Proof: For each layer of T, the signature vector of T contains two coordinates. Thus by Lemma 3.24, the signature vector of T has at most $2c\log m$ coordinates, where $c = \frac{1}{\log[1+\rho(1-\mu)]}$. Also, by Lemma 3.27, the coordinates are non-decreasing in absolute value and so the absolute value of the final (finite) coordinate is an upper bound on the absolute value of each coordinate in the signature vector. By Definition 3.25, the final coordinate is $\left\lfloor \log_b \left(\left\lceil \frac{\rho^{-\ell}}{(1-\mu)^{\ell}} \right\rceil |Y_\ell| \right) \right\rfloor$. As $\ell \leq c\log(m)$ (by Lemma 3.24) and $|Y_\ell| \leq m$, the absolute value of each coordinate of the signature vector is bounded above by

$$\begin{aligned}
\log_b \left[\frac{\rho^{-c\log(m)}}{(1-\mu)^{c\log(m)}} |Y_{c\log(m)}| \right] &\leq \log_b \left[\frac{\rho^{-c\log(m)}}{(1-\mu)^{c\log(m)}} (m) \right] \\
&= \log_b m + \log_b \rho^{-c\log(m)} - \log_b (1-\mu)^{c\log(m)} \\
&= \log_b m - c[\log_b(\rho)]\log(m) - c[\log_b(1-\mu)]\log(m) \\
&= \left[\frac{1}{\log(b)} - c[\log_b(\rho)] - c[\log_b(1-\mu)] \right] \log(m).
\end{aligned}$$

Let $R = \left[\frac{1}{\log(b)} - c[\log_b(\rho)] - c[\log_b(1-\mu)]\right]$ and note that R is fixed and depends only on r and ϵ (since b and c depend only on U, μ, and ρ, which in turn depend only on r and ϵ).

To each signature vector $s = (s_1, s_2, \ldots, s_{2\ell-1}, s_{2\ell}, \infty)$, we associate the vector

$$s^+ = (s_1 - 1, s_2 + 2, \ldots, s_{2\ell-1} - (2\ell-1), s_{2\ell} + 2\ell, \infty).$$

The final coordinate of s^+ is at most $R \log m + 2\ell \leq (R+2c) \log m$. Since the coordinates of s are non-decreasing in absolute value (and considering the sign pattern), the coordinates of s^+ are strictly increasing in absolute value. Thus each vector s^+ corresponds to a distinct subset of the set $\{1, \ldots, \lfloor (R+2c) \log m \rfloor\}$. Hence the total number of vectors s^+ (and therefore the total number of signature vectors) is at most $2^{(R+2c)\log m}$.

It is easy to verify that $2^{(R+2c)\log m}$ is $m^{f(r,\epsilon)}$ for some function f of r and ϵ, which completes the proof. $\square$

The idea of the penultimate paragraph in the proof of Lemma 3.28 was suggested by a referee of [13] (see [12, 13].) Although one can find the exact function f needed for any choice of feasible (U, μ, ρ), the function is very large and does not have a "simple" interpretation. We therefore do not specify f. However, note that for fixed r and ϵ, $m^{f(r,\epsilon)}$ is polynomial in m (with a very large degree).

3.5 The Algorithm FindITorBD and the Proof of Theorem 3.1

With the results of Sections 3.3 and 3.4, we are ready to state FindITorBD and prove Theorem 3.1.

Let FindITorBD be the algorithm defined in 3.5.1. Note that the sets $\mathcal{U}$ and S are as defined in Lemma 3.16.

Although not necessary for proving Theorem 3.1, the set L returned by FindITorBD will be used by our algorithm FindWeightPIT in Chapter 5. We therefore ignore the set L for the time being, other than to say that $L = \text{Leaf}(K)$ for the constellation K for $\mathcal{B}_0 \supseteq \mathcal{B}$ with $V(K) \subseteq D$.

We now prove Theorem 3.1.

3.5.1 FindITorBD

Input: A graph G and vertex partition $(V_1, \ldots, V_m)$ satisfying (A1)-(A4).

Output: Either an IT M of G with respect to $(V_1, \ldots, V_m)$, or a set $\mathcal{B}$ of vertex classes and set D of vertices satisfying outcome (2) of Theorem 3.1, together with the set L of leaves in the constellation contained in D.

```
1:  function FindITorBD($G; V_1, \ldots, V_m$)
2:      Initialise $M := \emptyset$.
3:      for $i = 1, \ldots, m$ do
4:          if $V_i \cap M = \emptyset$ then
5:              $(M, T, \ell + 1) :=$ GrowTransversal$(M, V_i)$
6:              if $\ell + 1 > 0$ then
7:                  for $j = 1, \ldots, \ell$ do
8:                      $(X'_j, Y'_j, L'_j) :=$ BuildLayer$(T_j, j-1, X_j, Y_j)$
9:                  $\mathcal{B} := (\mathrm{Vclass}(Y_{\leq \ell}) \setminus \mathcal{U}) \setminus \left( \bigcup_{i=1}^{\ell} \mathrm{Vclass}(X'_i \setminus X_i) \right)$
10:                 $D := X'_{\leq \ell} \cup Y'_{\leq \ell} \cup X_{\ell+1} \cup Y_{\ell+1} \cup S$.
11:                 $L := Y'_{\leq \ell} \cup Y_{\ell+1}$
12:                 return $\mathcal{B}$, $D$, and $L$ and terminate.
13:     return $M$
```

Proof of Theorem 3.1: By Lemma 3.26, every iteration of GrowTransversal reduces the lexicographic value of the signature vector of an alternating tree T with respect to a PIT M' and vertex class A. Furthermore, Lemma 3.28 implies that the number of such signature vectors is bounded by $m^{f(r,\epsilon)}$. Thus GrowTransversal terminates after at most $m^{f(r,\epsilon)}$ iterations. Moreover, it is clear that the algorithms BuildLayer and SuperposedBuild can be implemented in time $O(|V(G)|^3)$ for fixed r and ϵ. Hence, when r and ϵ are fixed, each iteration of GrowTransversal can be implemented in time $O(|V(G)|^4)$. As GrowTransversal is implemented at most m times in FindITorBD and the other steps of FindITorBD take an additional $O(|V(G)|^4)$ operations, the runtime of FindITorBD is

$$O(|V(G)|^4) + m\left[m^{f(r,\epsilon)}O(|V(G)|^4)\right],$$

which is poly$(|V(G)|)$ for fixed r and ϵ. It remains to show that FindITorBD returns one of the two stated outcomes.

Note that FindITorBD starts with $M = \emptyset$ and runs GrowTransversal on at most m vertex classes. At the end of each of these iterations of GrowTransversal, the PIT M covers one more vertex class than the PIT at the start of the iteration. Also, the PIT at the end of one iteration of GrowTransversal is the initial PIT of the next iteration.

Suppose FindITorBD terminates during iteration i of the main loop of FindITorBD. Then $V_i \cap M = \emptyset$ and GrowTransversal(M, V_i) returns $(M', T, \ell + 1)$ for some alternating tree T with respect to PIT M' and vertex class V_i. Hence by Lemma 3.16, the sets $\mathcal{B}$ and D have the properties stated in outcome (2) of Theorem 3.1. (The sets $\mathcal{B}$ and D are defined by FindITorBD to be the same as in the statement of Lemma 3.16.)

Suppose FindITorBD does not terminate during any iteration of the main loop of FindITorBD. Then $\ell + 1 = 0$ and so GrowTransversal(M, V_i) returns a PIT M' containing a vertex in each of Vclass$(M) \cup \{V_i\}$. Hence at the end of iteration m, every vertex class contains a vertex in the final PIT M. Hence M is an IT of G and so FindITorBD returns an IT in G. $\square$

Recall from Lemma 3.28 that the number of signature vectors for T is bounded by $m^{f(r,\epsilon)}$. Thus the degree of the runtime of FindITorBD depends on r and ϵ as well. In particular, this means that the degree of the runtime of FindITorBD depends on Δ in the case of Corollary 3.2. Hence FindITorBD is only efficient when the parameters r and ϵ (or Δ) are fixed.

In Chapter 4, we will explore some problems where the parameters r and ϵ are constant. This allows us to use FindITorBD to introduce new efficient algorithmic results for these problems. However, there are other applications where the parameters r and ϵ are not constant, such as problems where the maximum degree of the graphs is unbounded. We will give a new randomised algorithm that overcomes the dependence on Δ in the case of Corollary 3.2 as well as an application of this algorithm to fractional strong colourings in Chapter 6.

Chapter 4

Applications

As mentioned in Chapter 1, Theorem 2.4 has been applied to a variety of problems. In this chapter, we will look at some known results in graph colourings, graph partitionings, and hypergraph matchings which have applied Theorem 2.4. In particular, we will look at results due to Annamalai [11, 13]; Kaiser, Král, and Škrekovski [57]; King [61]; Aharoni, Berger, and Ziv [3]; and Alon, Ding, Oporowski, and Vertigan [9]. We aim to present modifications to these results that make them algorithmic. Outlines of some of these modifications appear in [38].

We are not the first to present algorithmic versions of applications of Theorem 2.4. Annamalai [11, 13] provided an algorithm that found perfect matchings in bipartite r-uniform hypergraphs. Our algorithm FindITorBD is a generalisation of this algorithm, which we will show in Section 4.1.

All of the results presented in this chapter are modifications of the results of other authors. As such, we will focus on presenting the changes necessary to make the original results algorithmic. The interested reader may find more detailed information on the runtimes for these algorithmic results in the appendices.

This chapter is organised as follows. In Section 4.1, we discuss the hypergraph matching result of Annamalai [11, 13] and how it follows from Theorem 3.1. In Section 4.2, we modify the proof of a circular colouring result due to Kaiser, Král, and Škrekovski [57] to make their result fully algorithmic. In Section 4.3, we present a fully algorithmic proof of a result due to King [61] which finds an independent set meeting every maximum clique in a graph. In Section 4.4, we modify the proof of a strong colouring result due to Aharoni, Berger, and Ziv [3] to prove a slightly weaker algorithmic version of their result. In Section 4.5, we present algorithmic versions of two results of Alon, Ding, Oporowski, and Vertigan [9] that find vertex and edge colourings that induce small monochromatic components.

4.1 Hypergraph Matchings

In this section, we will discuss the result of Annamalai in more detail. Annamalai [11, 13] was the first to present an algorithmic version of an application of Theorem 2.4. In particular, he provided an algorithmic result for a hypergraph version of Hall's Theorem for bipartite graphs. The equivalent non-algorithmic result is due to Haxell [48].

We aim to show that Annamalai's result can be derived using FindITorBD. Before doing so, we introduce the following notions for hypergraphs.

A hypergraph is *r-uniform* if every edge in the hypergraph contains exactly r vertices. For the remainder of this section, we will only consider hypergraphs that are r-uniform for some $r \geq 2$. We have the following notion for a "bipartite" hypergraph.

Definition 4.1. An *r-uniform bipartite hypergraph* $H = (A, B, E)$ is a hypergraph on a vertex set that is partitioned into two sets A and B such that $|e \cap A| = 1$ and $|e \cap B| = r - 1$ for each edge $e \in E$.

A hypergraph is a *bipartite hypergraph* if it is an r-uniform bipartite hypergraph for some r.

Let $H = (A, B, E)$ be a bipartite hypergraph. We define a matching and cover of H as follows.

Definition 4.2. A *matching* in H is a subset $M \subseteq E$ of pairwise disjoint edges of H. A matching M is *perfect* if it saturates A, i.e. $|M| = |A|$.

Definition 4.3. Let $F \subseteq E$. A subset $T \subseteq B$ is a *B-cover of F* if $|e \cap T| \neq \emptyset$ for every $e \in F$. The smallest cardinality of a B-cover of F is denoted by $\tau_B(F)$.

For a subset $S \subseteq A$, let E_S be the set of edges in H incident to S, i.e.

$$E_S = \{e \in E : |e \cap S| = 1\}.$$

The following generalisation of Hall's Theorem due to Haxell [48] provides a condition under which H admits a perfect matching.

Theorem 4.4 ([48])**.** *Let $H = (A, B, E)$ be an r-uniform bipartite hypergraph. If for all $S \subseteq A$,*

$$\tau_B(E_S) > (2r-3)(|S|-1),$$

then H admits a perfect matching.

Note that when $r = 2$, Theorem 4.4 is (the nontrivial direction of) Hall's Theorem. It was shown in [48] that Theorem 4.4 is best possible for every r. Furthermore, Theorem 4.4 is a special case of Theorem 2.4. This can be seen as follows.

Given an r-uniform bipartite hypergraph $H = (A, B, E)$, construct an auxiliary graph G^H with vertex set E, in which vertices e and f are adjacent if and only if $e \cap f \cap B \neq \emptyset$. Consider the vertex partition of G^H given by assigning e and f to the same vertex class if and only if $e \cap f \cap A \neq \emptyset$. We index the vertex classes by A. Thus, a set $M \subseteq E$ is a perfect matching of H if and only if $M \subseteq V(G^H)$ is an IT of G^H.

By Theorem 2.4 applied to G^H, if H does not have a perfect matching, then there exists a subset $\mathcal{B}$ of vertex classes (indexed by a set $S(\mathcal{B}) \subseteq A$) such that $(G^H)_{\mathcal{B}}$ is dominated by $V(K)$ for a constellation K for $\mathcal{B}$. Thus $T = \bigcup\limits_{e \in V(K)} (e \cap B)$ is a set of vertices of H that form a B-cover of $E_{S(\mathcal{B})}$. Claim 4.5 then gives an immediate contradiction to the assumption of Theorem 4.4, thus completing the proof.

Claim 4.5. *Let $\mathcal{B}$ be a subset of the vertex classes of G^H and K be a constellation for $\mathcal{B}$. Then,*

$$\left| \bigcup_{e \in V(K)} (e \cap B) \right| \leq (2r-3)(|S(\mathcal{B})|-1).$$

Proof: Each component C of K corresponds to a set of edges of H, consisting of the centre e_C of the star C and a nonempty set L_C of leaves, all of which intersect e_C in B. Hence the total number of vertices of B contained in $\{e_C\} \cup L_C$ is at most $(r-1) + (r-2)|L_C|$. By definition, $\bigcup\limits_C L_C$ is an IT of $|\mathcal{B}| - 1$ classes of $\mathcal{B}$, which implies that K has at most $|S(\mathcal{B})| - 1$ components, and that $\sum\limits_C |L_C| = |S(\mathcal{B})| - 1$. Therefore,

$$\begin{aligned}\left| \bigcup_{e \in V(K)} (e \cap B) \right| &\leq \sum_C (r - 1 + (r-2)|L_C|) \\ &\leq (r-1)(|S(\mathcal{B})|-1) + (r-2)(|S(\mathcal{B})|-1) \\ &= (2r-3)(|S(\mathcal{B})|-1),\end{aligned}$$

where the sum is over all components C of K. $\square$

Annamalai [11, 13] proved the following algorithmic version of Theorem 4.4.

Theorem 4.6 ([11, 13])**.** *There exists an algorithm that finds for fixed $r \geq 2$ and $\epsilon > 0$, in time polynomial in the size of the input, a perfect matching in r-uniform bipartite hypergraphs $H = (A, B, E)$ satisfying*

$$\tau_B(E_S) > (2r - 3 + \epsilon)(|S| - 1)$$

for all $S \subseteq A$.

We now show that Theorem 3.1 is a generalisation of Theorem 4.6. Note that for every r-uniform bipartite hypergraph $H = (A, B, E)$, the graph G^H is r-claw-free with respect to any partition. This follows from the facts that the neighbours of e which form an independent set in G^H must contain distinct vertices of $e \cap B$ and $|e \cap B| = r - 1$. Thus for fixed r and ϵ, an algorithm $\mathcal{A}$ that takes as input $H = (A, B, E)$, finds G^H, and applies FindITorBD for $r' = r$ and $\epsilon' = \frac{\epsilon}{r-1}$ to G^H and any vertex partition of G^H is a polynomial-time algorithm that finds for each input $H = (A, B, E)$ either:

(1) an IT in G^H (which is a perfect matching in H), or

(2) a set $\mathcal{B}$ of vertex classes and a set D of vertices of G^H such that D dominates $(G^H)_{\mathcal{B}}$ in G^H and $|D| < (2 + \epsilon')(|\mathcal{B}| - 1)$. Moreover D contains $V(K)$ for a constellation K for some $\mathcal{B}_0 \supseteq \mathcal{B}$, where $|D \setminus V(K)| < \epsilon'(|\mathcal{B}| - 1)$.

If (1) is the outcome for every input H, then $\mathcal{A}$ is the algorithm in Theorem 4.6. Therefore, suppose (2) holds for some input H. Then D is a set of edges of H such that every edge of $E_{S(\mathcal{B})}$ intersects $T = \bigcup_{e \in D} (e \cap B)$. For $u = |D \setminus V(K)|$ it is clear that

$$\left| \bigcup_{e \in D \setminus V(K)} (e \cap B) \right| \leq u(r - 1).$$

We now estimate $\left| \bigcup_{e \in V(K)} (e \cap B) \right|$. As in the proof of Claim 4.5, for each component C of the constellation K, the number of vertices of B contained in $\{e_C\} \cup L_C$ is at most $(r - 1) + (r - 2)|L_C| = 1 + (r - 2)|V(C)|$. Each component of K has at least two vertices, so the number of components is at most $\frac{|V(K)|}{2} = \frac{|D|-u}{2}$. Since $|D| < (2+\epsilon')(|\mathcal{B}|-1)$, we have

$$\begin{aligned}\left|\bigcup_{e\in V(K)} (e\cap B)\right| &\le \sum_{C} (1+(r-2)|V(C)|)\\ &\le |V(K)|/2 + (r-2)|V(K)|\\ &= (2r-3)\tfrac{|V(K)|}{2}\\ &= (2r-3)\tfrac{|D|-u}{2}\\ &< (2r-3)\left(1+\tfrac{\epsilon'}{2}\right)(|\mathcal{B}|-1) - u\left(r-\tfrac{3}{2}\right),\end{aligned}$$

where again the sum is over all components C of K. Therefore,

$$\begin{aligned}|T| &< u(r-1) + (2r-3)\left(1+\tfrac{\epsilon'}{2}\right)(|\mathcal{B}|-1) - u\left(r-\tfrac{3}{2}\right)\\ &= \tfrac{u}{2} + \left[2r-3+\epsilon'\left(r-\tfrac{3}{2}\right)\right](|\mathcal{B}|-1)\\ &< [2r-3+\epsilon'(r-1)](|\mathcal{B}|-1)\\ &= (2r-3+\epsilon)(|\mathcal{B}|-1),\end{aligned}$$

where the last line holds because $u < \epsilon'(|\mathcal{B}|-1)$ and $\epsilon' = \frac{\epsilon}{r-1}$. This contradicts the assumption that $\tau_B(E_S) > (2r-3+\epsilon)(|S|-1)$ for $S = S(\mathcal{B})$. Hence only (1) occurs, thus proving Theorem 4.6.

4.2 Circular Edge Colourings

In this section, we discuss some modifications that can be made to a result by Kaiser, Král, and Škrekovski [57] (stated as Theorem 4.8 in Section 4.2.1) to make their result fully algorithmic. In particular, we will provide an algorithm that, given a cubic bridgeless graph G with girth at least $f(p)$, will return a proper circular $(3p+1)/p$-edge-colouring of G in polynomial time, where f is the same function of p as the original Kaiser, Král, and Škrekovski result. We therefore present a fully algorithmic proof of their result.

This section is organised as follows. We begin with a brief discussion of circular colourings and their relation to Jaeger and Swart's Girth conjecture [55]. In Section 4.2.1, we discuss the result of Kaiser, Král, and Škrekovski [57]. This includes necessary definitions in Section 4.2.1.1, an outline of the colouring in Section 4.2.1.2, and the algorithm in Section 4.2.1.3. We conclude in Section 4.2.2 with another algorithmic proof of a result from [57] for the case $p = 2$. However, this result does not provide the best known bound on the girth when $p = 2$. The best bound, due to Král, Máčajová, Mazák, and Sereni [68], also has an algorithmic proof, so our discussion of this second result from [57] will be brief.

Circular colourings of graphs were introduced in 1988 by Vince [92] under the name "star colourings". We define circular p/q-edge-colourings as follows.

Definition 4.7. A *proper circular p/q-edge-colouring* of a graph G is a colouring of the edges of G with colours in $\{0, \dots, p-1\}$ such that the difference modulo p of the colours assigned to two adjacent edges is not in $\{-(q-1), -(q-2), \dots, q-1\}$. The *circular chromatic index* of G is given by

$$\chi'_c(G) = \inf\left\{\tfrac{p}{q} : \text{there is a proper circular } p/q\text{-edge-colouring of } G\right\}.$$

Figure 4.1 gives two examples of a proper circular 10/3-edge-colouring.

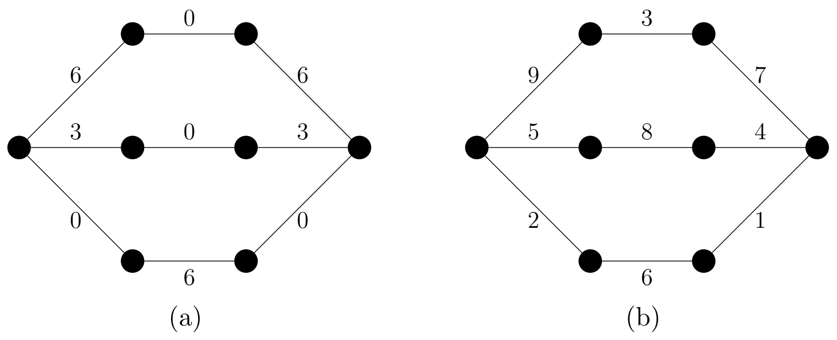

Figure 4.1: Two proper circular 10/3-edge-colourings of the same graph.

It is well known that for all graphs G, the infimum in the definition of $\chi'_c(G)$ is always obtained (see [22, 92]). Hence we may think of $\chi'_c(G)$ as the smallest ratio $\frac{p}{q}$ for which there is a proper circular p/q-edge-colouring of G. Furthermore, it can be shown that $\chi'(G) - 1 < \chi'_c(G) \leq \chi'(G)$, where $\chi'(G)$ is the edge chromatic number of G (see [92, 96]). Thus for all graphs G, $\chi'(G) = \lceil \chi'_c(G) \rceil$. It is also easy to see that for each p and q such that $\chi'_c(G) \leq \frac{p}{q}$, there is a proper circular p/q-edge-colouring of G. The interested reader may find more results on circular colourings in the surveys by Zhu [96, 97].

For the remainder of Section 4.2, we will be interested in graphs which are both cubic and bridgeless. Note that for a cubic bridgeless graph G, we have that $2 < \chi'_c(G) \leq 4$ since $3 \leq \chi'(G) \leq 4$ by Vizing's theorem. However, it can be shown that $\chi'_c(G) = 3$ if and only if $\chi(G) = 3$, where $\chi(G)$ is the chromatic number of G (see [90]). Hence $3 \leq \chi'_c(G) \leq 4$.

Cubic bridgeless graphs for which $\chi'(G) = 4$ are referred to as *snarks*. It was conjectured by Jaeger and Swart [55] that there exists a number g such that each snark has girth at most g. This conjecture is known as the *Girth conjecture*. It was disproved by Kochol [64] through the construction of cyclically 5-edge-connected snarks with arbitrarily large girth. However, Kaiser, Král, and Škrekovski [57] showed that the Girth conjecture "almost holds" for circular edge-colourings, i.e. for every $\epsilon > 0$, there exists an integer g such that each cubic bridgeless graph G with girth at least g has $\chi'_c(G) \leq 3 + \epsilon$. This result is best possible for every g; there is a snark S with girth at least g and $\chi'_c(S) \neq 3$ [57].

4.2.1 Circular $(3p + 1)/p$-edge-colourings

Our work in this section is based on the following result of Kaiser, Král, and Škrekovski [57].

Theorem 4.8 ([57]). *Let $p \in \mathbb{N}$ with $p \geq 2$ and G be a cubic bridgeless graph with girth*

$$g \geq f(p) = \begin{cases} 2(2p)^{2p-2} & \textit{if } p \geq 2 \textit{ is even} \\ 2(2p)^{2p} & \textit{if } p \geq 3 \textit{ is odd.} \end{cases}$$

Then G admits a proper circular $(3p + 1)/p$-edge-colouring.

Theorem 4.8 almost implies the Girth conjecture for circular edge-colourings as follows. For any $\epsilon > 0$, choose an integer $p > 0$ such that $\frac{1}{p} \leq \epsilon$. Then for any cubic bridgeless graph G whose girth is at least $f(p)$, G has a proper circular $(3p + 1)/p$-edge-colouring. As $\frac{3p+1}{p} = 3 + \frac{1}{p} \leq 3 + \epsilon$, this implies $\chi'_c(G) \leq 3 + \epsilon$ for all cubic bridgeless graphs G with girth at least $f(p)$.

The proof of Theorem 4.8 relies on a particular auxiliary graph H that is constructed using G, p, and a fixed 1-factor F of G. Kaiser, Král, and Škrekovski proved that H has an IT using Theorem 2.4. Then, given such an IT, they explicitly provided the required proper circular $(3p+1)/p$-edge-colouring of G. Thus, to make their result fully algorithmic, all that is necessary is to find an IT in the graph H. We instead will use the same technique and colouring for a subgraph G' of H to prove Corollary 4.9. Both G' and H are defined in Section 4.2.1.2.

Corollary 4.9. *There exists an algorithm* CircularP *that for integers $p \geq 2$, takes as input any cubic bridgeless graph G with girth*

$$g \geq f(p) = \begin{cases} 2(2p)^{2p-2} & \textit{if } p \geq 2 \textit{ is even} \\ 2(2p)^{2p} & \textit{if } p \geq 3 \textit{ is odd,} \end{cases}$$

and finds a proper circular $(3p+1)/p$-edge-colouring of G. For fixed p, the runtime is $\operatorname{poly}(|V(G)|)$.

Note that the function $f(p)$ in Corollary 4.9 is the same as in Theorem 4.8. Thus no weakening of the constraints in Theorem 4.8 is necessary to make the result algorithmic.

We prove Corollary 4.9 in Section 4.2.1.3. To do so, we require several definitions provided in Section 4.2.1.1 and the colouring provided in Section 4.2.1.2.

4.2.1.1 Definitions

We now define some necessary structures for the proof of Corollary 4.9. These definitions may also be found in [57]. For all of the definitions, we will assume G is a cubic bridgeless graph with girth g, F is a fixed 1-factor of G, and k is always an integer in $\{1,\dots,g\}$. Note that such an F exists for G by Petersen's theorem (see, for ex., [28]). Moreover $G-F$ is a collection of cycles.

Definition 4.10. A *k-segment* of G is a k-tuple $(v_1,\dots,v_k)$ of vertices $v_i \in V(G)$ for which there is a cycle C in $G-F$ such that $v_1v_2\dots v_k$ is a path in C.

As the cycles of G are not oriented, we consider $(v_1,v_2,\dots,v_k)$ and $(v_k,v_{k-1},\dots,v_1)$ to be the same k-segment of G.

Definition 4.11. Define $G_{F,k}$ to be the graph whose vertices are the k-segments of G and two k-segments $(v_1,\dots,v_k)$ and $(w_1,\dots,w_k)$ are adjacent in $G_{F,k}$ if there exist $i,j \in \{1,\dots,k\}$ such that $v_iw_j \in F$.

Note that each vertex of G is contained in at most k of the k-segments. Furthermore, each vertex of G is incident with a unique edge in F and so has a unique neighbour that is incident with the same edge in F. Thus the maximum degree of $G_{F,k}$ is at most k^2.

Let n be the number of cycles in $G-F$ and let $C_1,\dots,C_n$ be these cycles. For each $i \in \{1,\dots,n\}$, let V_i be the set of all k-segments whose vertices are in cycle C_i. Thus $(V_1,\dots,V_n)$ is a vertex partition of $G_{F,k}$ and $|V_i| = |C_i| \geq g$ for each $i \in \{1,\dots,n\}$.

Definition 4.12. Let $r \in \mathbb{N}$. A *system of r-independent k-segments* is a set $\{s_1,\dots,s_n\}$ of k-segments such that $s_i \in V_i$ for each i and the distance in $G_{F,k}$ between each pair s_i, s_j with $i \neq j$ is at least $r+1$.

From Definition 4.12, it is clear that a system of 1-independent k-segments is an IT of $G_{F,k}$ with respect to $(V_1, \ldots, V_n)$. In general, a system of r-independent k-segments is an IT of $(G_{F,k})^r$ with respect to $(V_1, \ldots, V_n)$, where $(G_{F,k})^r$ is the r^{th} *power* of $G_{F,k}$. The r^{th} power of a graph G is a graph defined on $V(G)$ so that two vertices are adjacent if and only if the distance between them in G is at most r.

For the next definition, let $p \geq 2$ be an integer and assume $g \geq f(p)$, where $f(p)$ is as defined in Theorem 4.8 (and Corollary 4.9).

Definition 4.13. An *octopus* is a subgraph of G with the following structure, which we define algorithmically. First, let C be a cycle of $G - F$ and let $Q = v_1 v_2 \ldots v_{2p}$ be a path on $2p$ vertices in C. Define $\mathcal{P}(1) = \{Q\}$.

We construct $\mathcal{P}(2)$ from $\mathcal{P}(1)$ as follows. First, for each $v_i \in V(Q)$, let v_i^F be the neighbour of v_i such that $v_i v_i^F \in F$. Then, if p is even, let $P(v_i) = u_1 \ldots u_{2p-3}$ be the path in $G - F$ on $2p - 3$ vertices such that $u_{p-1} = v_i^F$. Otherwise, p is odd and so let $P(v_i) = u_1 \ldots u_{2p-1}$ be the path in $G - F$ on $2p - 1$ vertices such that $u_p = v_i^F$. Define $\mathcal{P}(2)$ to be the set of all paths $P(v_i)$ for which $v_i \in V(Q)$, i.e. $\mathcal{P}(2) = \{P(v_i) : v_i \in V(Q)\}$. Note that each path in $\mathcal{P}(2)$ is a path on $2p + 5 - 4(2)$ vertices when p is even and is a path on $2p + 7 - 4(2)$ vertices if p is odd.

Now, assume $\mathcal{P}(\ell)$ has been constructed for some $2 \leq \ell \leq \lceil \frac{p}{2} \rceil$ and that all the paths in $\mathcal{P}(\ell)$ are on k vertices, where

$$k = \begin{cases} 2p + 5 - 4\ell & \text{if } p \text{ is even} \\ 2p + 7 - 4\ell & \text{if } p \text{ is odd.} \end{cases}$$

We construct $\mathcal{P}(\ell + 1)$ from $\mathcal{P}(\ell)$ as follows. First, for each path $Q \in \mathcal{P}(\ell)$, let $Q = v_{Q,1} \ldots v_{Q,k}$. Then for each $v_{Q,i} \in V(Q)$, we have $v_{Q,i}^F$ as the neighbour of $v_{Q,i}$ such that $v_{Q,i} v_{Q,i}^F \in F$. For each $i \neq \lceil \frac{k}{2} \rceil$, define $P(v_{Q,i}) = u_1 \ldots u_{k-4}$ to be the path in $G - F$ on $k - 4$ vertices such that $u_j = v_{Q,i}^F$ for $j = \lceil \frac{k-4}{2} \rceil$. Define $\mathcal{P}(\ell + 1)$ to be the set of all paths $P(v_{Q,i})$ such that $Q \in \mathcal{P}(\ell)$ and $i \neq \lceil \frac{k}{2} \rceil$, i.e.

$$\mathcal{P}(\ell + 1) = \left\{ P(v_{Q,i}) : Q \in \mathcal{P}(\ell), i \neq \lceil \tfrac{k}{2} \rceil \right\}.$$

Note that these paths have $k - 4$ vertices, which is $2p + 5 - 4(\ell + 1)$ or $2p + 7 - 4(\ell + 1)$ vertices if p is even or odd respectively. Moreover, for $\ell = \lceil \frac{p}{2} \rceil$, the paths in $\mathcal{P}(\ell + 1)$ contain $2p + 5 - 4\left(\frac{p}{2} + 1\right) = 1$ vertex if p is even and $2p + 7 - 4\left(\frac{p+1}{2} + 1\right) = 1$ vertex if p is odd. Hence we do not construct $\mathcal{P}(\ell)$ for any $\ell > \lceil \frac{p}{2} \rceil + 1$.

We define the octopus to be the structure formed by the union of the vertices in the paths in $\bigcup_{\ell=1}^{\lceil p/2\rceil+1} \mathcal{P}(\ell)$ together with every edge of G that is incident to at least one of these vertices. (Note that for an edge incident with exactly one vertex in this union, the endpoint not in this union is not considered to be in the octopus.)

Figure 4.2 gives an example of an octopus.

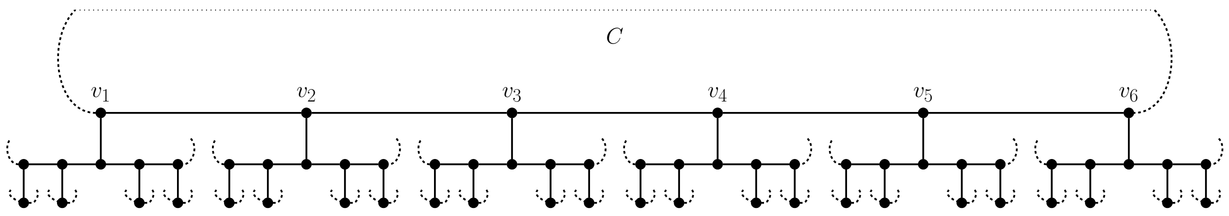

Figure 4.2: The octopus with head $(v_1, v_2, v_3, v_4, v_5, v_6)$. The inner edges of the octopus have solid lines and the contact edges have dashed lines. The dotted edge represents the remainder of head cycle C, which is not part of the octopus.

We refer to the cycle C in Definition 4.13 as the *head cycle* and the $2p$-segment $(v_1, \ldots, v_{2p})$ as the *head* of the octopus.

For each $\ell \in \left\{1, \ldots, \left\lceil \frac{p}{2} \right\rceil + 1\right\}$, we say that $\mathcal{P}(\ell)$ contains the set of paths in *level* ℓ of the octopus and refer to the paths in $\mathcal{P}(\ell)$ as *blocks*. Edges that are in a block are called *inner edges* and edges of $G - F$ that are not inner edges but are adjacent to an inner edge are called *contact edges*. We say that a contact edge is *adjacent* to block B if it is adjacent to an inner edge of B.

Note that for $\ell \geq 2$, the centre vertex of each block in $\mathcal{P}(\ell)$ is incident with the same edge of F as some vertex in a block in $\mathcal{P}(\ell - 1)$. Given a block $A \in \mathcal{P}(\ell)$, the edge of F between a vertex in A and a vertex in some $B \in \mathcal{P}(\ell - 1)$ is called an *input edge* and the edges of F between a vertex in A and a vertex in some $B \in \mathcal{P}(\ell + 1)$ are called *output edges*. Hence an output edge of level ℓ is an input edge of level $\ell + 1$.

Before discussing the colouring, it is important to note that no octopus is self-intersecting, i.e. no vertex in the octopus appears in two distinct blocks of the octopus. If it did, then the octopus contains a vertex v in distinct blocks A and B. By the construction in Definition 4.13, it is clear that the octopus contains a path P_A from its head h to A and a path P_B from h to B. As $v \in A \cap B$, this implies that the subgraph $h \cup P_A \cup P_B \cup A \cup B$

contains a cycle. However, the maximum length of such a cycle is less than

$$|V(h)| \cdot |E(P_A \cup P_B)| = 2p\left[2\left(\left\lceil \tfrac{p}{2}\right\rceil + 1\right)\right] = \begin{cases} 2p^2 + 4p & \text{if } p \text{ is even} \\ 2p^2 + 6p & \text{if } p \text{ is odd.} \end{cases}$$

Since $2p^2 + 4p < 2(2p)^{2p-2}$ and $2p^2 + 6p < 2(2p)^{2p}$ for $p \geq 2$, this contradicts g being the girth of G. Hence the subgraph induced by the vertices of the octopus must be acyclic and so no vertex appears in multiple blocks.

4.2.1.2 The Colouring Technique

In this section, we discuss the technique our algorithm will use to colour the graph G. It relies on the following result of Kaiser, Král, and Škrekovski from [57].

Lemma 4.14 ([57])**.** *Suppose c is a partial proper circular $(3p+1)/p$-edge-colouring of G and o is an octopus in G such that c colours the contact edges of o by p and $2p$ so that each pair of contact edges adjacent to the same block receive opposite colours and c does not colour any of the inner edges of o. Then c can be extended to a proper circular $(3p+1)/p$-edge-colouring of o.*

The exact colouring used to prove Lemma 4.14 can be found in Appendix A.1. For our purposes, it is enough to know that, given a partial proper circular $(3p+1)/p$-edge-colouring satisfying the hypotheses of Lemma 4.14, the edges of an octopus o can be coloured in such a way that maintains the proper circular $(3p+1)/p$-edge-colouring.

Let G be a cubic bridgeless graph with girth $g \geq f(p)$ for $f(p)$ as in Corollary 4.9 and F be a fixed 1-factor of G. Suppose $\mathcal{O}$ is a set of disjoint octopi in G. If the edges of $G - F$ that are not inner edges of the octopi of $\mathcal{O}$ can be 2-coloured with the colours p and $2p$ such that contact edges adjacent to the same block (for each block of each octopus $o \in \mathcal{O}$) receive distinct colours, we can extend the partial colouring to a proper circular $(3p+1)/p$-edge-colouring of all of G by applying Lemma 4.14 to each $o \in \mathcal{O}$. It therefore remains to find such a set of octopi $\mathcal{O}$ so that $G - F$ can be coloured in such a manner. To do so, we will use an IT of an auxiliary graph G'.

Let $C_1, \ldots, C_n$ be the cycles of $G - F$ and for each $i \in \{1, \ldots, n\}$, let V_i be the set of all $2p$-segments whose vertices are in cycle C_i. We define $G_{F,2p}$ as in Definition 4.11 for $k = 2p$ and define $(G_{F,2p})^{p-1}$ and $(G_{F,2p})^p$ to be the $(p-1)$th and pth power, respectively, of $G_{F,2p}$.

Let $S \subseteq \{1, \dots, n\}$ be the set of indices such that $i \in S$ if and only if $|V(C_i)|$ is odd. Define

$$H = \begin{cases} (G_{F,2p})^{p-1} & \text{if } p \text{ is even} \\ (G_{F,2p})^{p} & \text{if } p \text{ is odd.} \end{cases}$$

Let G' be the subgraph of H induced by the $2p$-segments in the odd cycles of $G - F$, i.e.

$$G' = H\left[\bigcup_{i \in S} V_i\right].$$

Let $(V_1', \dots, V_m')$ be the vertex partition of G' whose classes are also vertex classes in $(V_1, \dots, V_n)$, so for each $j \in \{1, \dots, m\}$, we have $V_j' = V_i$ for some $i \in \{1, \dots, n\}$. Recall that $|V_i| = |C_i| \geq g \geq f(p)$.

Claim 4.15. *For fixed p, an IT M of G' with respect to $(V_1', \dots, V_m')$ can be found in time* $\text{poly}(|V(G')|)$.

Proof: We aim to use Corollary 3.2. To do so, we require the maximum degree of G' as well as $|V_i'|$ for all $i \in \{1, \dots, m\}$. We will determine these values by cases involving the parity of p using the following application of the Binomial theorem:

$$\begin{aligned}\sum_{i=0}^{k} [(2p)^2 - 1]^i &= \frac{1 - [(2p)^2 - 1]^{k+1}}{1 - [(2p)^2 - 1]} \\ &= \frac{[(2p)^2 - 1]^{k+1} - 1}{[(2p)^2 - 1] - 1} \\ &< [(2p)^2 - 1]^k \\ &< (2p)^{2k}.\end{aligned}$$

Note that the inequalities hold because $p \geq 2$. Hence $[(2p)^2 - 1] - 1 \geq [16 - 1] - 1 > 1$ and $[(2p)^2 - 1]^{k+1} - 1 > [(2p)^2 - 1]^{k+1}$, which implies the first inequality, as well as $(2p)^2 - 1 \geq 16 - 1 > 0$, which implies the second inequality ($a^k < b^k$ for $0 < a < b$).

Case 1. p is even.

By assumption, the girth of G is at least $2(2p)^{2(p-1)}$ and $|V_i'|$ is odd, so $|V_i'| \geq 2(2p)^{2(p-1)} + 1$ for all $i \in \{1, \dots, m\}$. It therefore remains to show that the maximum degree Δ of G' is at most $(2p)^{2(p-1)}$.

Recall from the discussion after Definition 4.11 that the maximum degree of $G_{F,2p}$ is at most $(2p)^2$. Hence the power graph $(G_{F,2p})^{p-1}$ has maximum degree at most

$$\left[(2p)^2\right] \sum_{i=0}^{p-2} \left[(2p)^2 - 1\right]^i < [(2p)^2][(2p)^{2(p-2)}] = (2p)^{2(p-1)}.$$

As G' is a subgraph of $H = (G_{F,2p})^{p-1}$, we have $\Delta \leq (2p)^{2(p-1)}$ and $|V_i'| \geq 2\Delta + 1$.

Case 2. p is odd.

The girth of G is at least $2(2p)^{2p}$ and, as in Case 1, $|V_i'| \geq 2(2p)^{2p} + 1$ for all $i \in \{1, \dots, m\}$. It remains to show that the maximum degree Δ of G' is at most $(2p)^{2p}$.

As the maximum degree of $G_{F,2p}$ is at most $(2p)^2$, the power graph $(G_{F,2p})^p$ has maximum degree at most

$$\left[(2p)^2\right] \sum_{i=0}^{p-1} \left[(2p)^2 - 1\right]^i < [(2p)^2][(2p)^{2(p-1)}] = (2p)^{2p}.$$

Since G' is a subgraph of $H = (G_{F,2p})^p$, we have $\Delta \leq (2p)^{2p}$ and $|V_i'| \geq 2\Delta + 1$.

In both Case 1 and Case 2, G' is a graph with maximum degree Δ and the vertex partition $(V_1', \dots, V_m')$ of G' satisfies $|V_i'| \geq 2\Delta + 1$ for each i. Hence for fixed p, by Corollary 3.2, an IT M of G' with respect to $(V_1', \dots, V_m')$ can be found in time poly$(|V(G')|)$. □

We choose $\mathcal{O}$ to be the set of octopi o such that the head of o is in M. By the discussion after Definition 4.12, we know that M is a system of $(p-1)$-independent or p-independent $2p$-segments of $G_{F,2p}\left[\bigcup_{i \in S} V_i\right]$ with respect to $(V_1', \dots, V_m')$ when p is even or odd, respectively. Hence $\mathcal{O}$ is a set of octopi such that:

(O1) The head cycle of each octopus $o \in \mathcal{O}$ is odd.

(O2) Each odd cycle of $G - F$ is the head cycle of exactly one $o \in \mathcal{O}$.

(O3) The heads of any two octopi in $\mathcal{O}$ are at distance at least p in $G_{F,2p}$ if p is even and at distance at least $p + 1$ in $G_{F,2p}$ if p is odd.

Property (O3) implies the following.

Claim 4.16. *The octopi of $\mathcal{O}$ are vertex-disjoint.*

Proof: Suppose not. Let o_1 and o_2 be two octopi in $\mathcal{O}$ that share a common vertex v.

By Definition 4.13, the edge $e \in F$ incident with v (as well as its other endpoint) is always in any octopus that contains v. Thus e is an edge of both o_1 and o_2 and so there is a path from the head of o_1 to the head of o_2 using only edges in the octopi. Therefore an upper bound on the distance between the heads of o_1 and o_2 occurs when e is assumed to be an edge between level $\lceil \frac{p}{2} \rceil$ and level $\lceil \frac{p}{2} \rceil + 1$ in both octopi. But then the distance between the head of o_1 and head of o_2 is at most $\frac{p}{2} + \left(\frac{p}{2} - 1\right) = p - 1$ if p is even and at most $\frac{p+1}{2} + \left(\frac{p+1}{2} - 1\right) = p$ if p is odd. Both contradict (O3), and so the octopi of $\mathcal{O}$ are vertex-disjoint. $\square$

In addition to being vertex-disjoint, consider the set I_H of inner edges of the heads of the octopi in $\mathcal{O}$. The graph $G^* = G - (F \cup I_H)$ consists only of the even cycles of $G - F$ and a set of even paths, one for each odd cycle of $G - F$. Thus G^* is bipartite and so G^* can be 2-coloured with colours p and $2p$. By applying such a 2-colouring to the edges of G that are not inner edges of the octopi in $\mathcal{O}$, we obtain the desired partial colouring of G that can be extended to the inner edges of the octopi in $\mathcal{O}$. The only edges that are uncoloured will be edges in F that are not in the octopi in $\mathcal{O}$. Such edges are necessarily adjacent to edges that are coloured p and $2p$ (from the 2-colouring) and so may be coloured 0 to obtain a proper circular $(3p+1)/p$-edge-colouring of all of G.

In the next section, we present an algorithm that uses a partial 2-colouring of $G - F$ and an IT of G' to extend the partial colouring to the edges of the octopi in $\mathcal{O}$ using the colouring of Claim 4.14. It then assigns the remaining edges of F the colour 0 to complete the proper circular $(3p+1)/p$-edge-colouring. We will show that all of these steps can be accomplished in time poly$(|V(G)|)$.

4.2.1.3 Proof of Corollary 4.9

We now present an algorithm that uses the technique outlined in Section 4.2.1.2 to colour the graph G. The graphs G' and G^* as well as the sets S and $\mathcal{O}$ are as defined in Section 4.2.1.2. Let CircularP be an algorithm that takes as input a cubic bridgeless graph G with girth $g \geq f(p)$ and performs the following steps:

1. Find a 1-factor F of G.

2. Form ordered lists of vertices V_i such that each V_i is a unique cycle in $G - F$ and the vertices are stored in V_i in cyclic order.

3. Find an IT M of G' with respect to the vertex partition associated with the vertex classes in S.

4. Find a 2-colouring c of G^* with colours p and $2p$.

5. Determine the set of edges in the octopi of $\mathcal{O}$ and remove the colours c assigned to these edges.

6. Colour the edges of the octopi in $\mathcal{O}$ as in Claim 4.14.

7. Colour the remaining uncoloured edges 0.

8. Return the colouring.

We will show that such an algorithm returns a proper circular $(3p+1)/p$-edge-colouring of G (Lemma 4.17). The proof that CircularP runs in time poly($|V(G)|$) for fixed p can be found in Appendix A.2.

Lemma 4.17. *Let $p \geq 2$ be given. The algorithm* CircularP *takes as input any cubic bridgeless graph G with girth*

$$g \geq f(p) = \begin{cases} 2(2p)^{2p-2} & \text{if } p \geq 2 \text{ is even} \\ 2(2p)^{2p} & \text{if } p \geq 3 \text{ is odd,} \end{cases}$$

and returns a proper circular $(3p+1)/p$-edge-colouring of G.

Proof: As G is cubic and bridgeless, G has a 1-factor F by Petersen's theorem. Thus it is possible to find cycles $C_1, \ldots, C_n$ of $G - F$ and define the sets of $2p$-segments $V_1, \ldots, V_n$ for these cycles. From this, the set S of indices for which $|V(C_i)|$ is odd can be determined (recall $|V(C_i)| = |V_i|$) and so the subgraph G' is well-defined. Let $\{V_1', \ldots, V_m'\}$ be the subset of vertex classes in $\{V_1, \ldots, V_n\}$ whose indices are in S.

By applying FindITorBD (for $\Delta = \Delta(G')$) on inputs G' and $(V_1', \ldots, V_m')$, we find an IT M of G' in time poly($|V(G')|$) when p is fixed (recall $\Delta(G')$ is determined by p). Given this M, the set of octopi $\mathcal{O}$ whose heads are in M is uniquely determined by Definition 4.13. By removing the edges of F and the set I_H of inner edges of the heads of the octopi in $\mathcal{O}$, the resulting graph G^* is bipartite and so can be 2-coloured with colours p and $2p$. CircularP then checks the inner edges of the octopi in $\mathcal{O}$ and removes the colours assigned to these edges. The result is a partial colouring c' of $G - F$ where the contact edges of all octopi in $\mathcal{O}$ are coloured with p and $2p$.

Recall that the head of each octopus in $\mathcal{O}$ contains an odd number of edges. Hence for each octopus in $\mathcal{O}$, the head of the octopus is adjacent to one contact edge coloured p and another coloured $2p$ by c' (the head cycle is odd and the head has $2p-1$ edges). Furthermore, each block of an octopus has an even number of edges. Hence for each octopus in $\mathcal{O}$, each block of the octopus is also adjacent to one contact edge coloured p and another coloured $2p$ by c'. Thus c' satisfies the conditions of Claim 4.14 for each $o \in \mathcal{O}$. Moreover, since the octopi of $\mathcal{O}$ are vertex-disjoint (Claim 4.16), extending c' to colour the octopi in any set $\mathcal{O}' \subseteq \mathcal{O}$ results in a colouring that still satisfies the conditions of Claim 4.14 for each $o \in \mathcal{O} \setminus \mathcal{O}'$. Hence by repeatedly applying the colouring of Claim 4.14 to each octopus in $\mathcal{O}$, c' can be extended to a proper circular $(3p+1)/p$-edge-colouring of $G - (F \setminus I)$, where I is the set of inner edges of the octopi in $\mathcal{O}$.

Note that for an edge $uv \in F \setminus I$, we have that u and v are not in any octopus in $\mathcal{O}$. Also, c' colours the edges of $G - (F \cup I)$ either p or $2p$ in such a way that G^* $(= G - (F \cup I_H))$ is properly 2-coloured. Thus u is incident with an edge coloured p and an edge coloured $2p$ in $G - F$ and the same holds for v. Colouring the edges of $F \setminus I$ with 0 maintains the proper circular $(3p+1)/p$-edge-colouring, so CircularP returns a proper circular $(3p+1)/p$-edge-colouring of G. □

Corollary 4.9 follows from Lemma 4.17 and Lemma A.6 in Appendix A.2.

4.2.2 Circular $7/2$-edge-colourings when $g \geq 16$

In this section, we discuss a different approach for $p = 2$, which is the smallest interesting case of $(3p+1)/p$-edge-colourings (since $p = 1$ is equivalent to finding a 4-edge-colouring of cubic graph G).

When $p = 2$, Corollary 4.9 states that CircularP takes as an input any bridgeless cubic graph with girth $g \geq f(2) = 2(4)^{2(1)} = 32$ and returns a proper circular $7/2$-edge-colouring of G. Kaiser, Král, and Škrekovski [57] showed a different technique that found such colourings for any cubic bridgeless graph with girth $g \geq 14$. This result is not best possible as it has been shown by Král, Máčajová, Mazák, and Sereni [68] that every cubic bridgeless graph with girth at least 6 has a proper circular $7/2$-edge-colouring. Moreover, the proof of the girth 6 result is algorithmic.

Even though it is not the best known algorithmic result when $p = 2$, we briefly outline a modification of the technique from [57] that gives the following algorithmic result.

Corollary 4.18. *There is an algorithm that takes as input any cubic bridgeless graph G with girth $g \geq 16$ and returns, in time* $\text{poly}(|V(G)|)$*, a proper circular $7/2$-edge-colouring of G.*

As in [57], we use the auxiliary graph $G_{F,\pm}$. Let G be a cubic bridgeless graph and F be a fixed 1-factor of G. For each 4-segment $(\alpha, \beta, \gamma, \delta)$ of a cycle C in $G - F$, let $G_{F,\pm}$ contain two vertices $v^{\pm}(\alpha, \beta, \gamma, \delta)$ and $v^{\mp}(\alpha, \beta, \gamma, \delta)$. The vertex $v^{\pm}(\alpha, \beta, \gamma, \delta)$ represents the requirement that the edges of F incident with α and β are assigned "+" and the edges of F incident with γ and δ are assigned "$-$." Similarly, the vertex $v^{\mp}(\alpha, \beta, \gamma, \delta)$ represents the requirement that the edges of F incident with α and β are assigned "$-$" and the edges of F incident with γ and δ are assigned "+." Two vertices of $G_{F,\pm}$ are adjacent in $G_{F,\pm}$ if their corresponding requirements contradict each other.

From the above construction, it is clear that $G_{F,\pm}$ has $2|V(G)|$ vertices and is 16-regular. Let G' be the subgraph of $G_{F,\pm}$ induced by the vertices derived from the 4-segments of odd cycles of $G - F$. Then $|V'_i| \geq 2(17) = 34$ for each set V'_i consisting of the vertices of G' that are derived from the vertices of cycle C_i in $G - F$ (recall girth $g \geq 16$ and C_i is odd). As the maximum degree of G' is at most 16, we have $2\Delta(G') + 1 = 33 < 34$ and so an IT M of G' with respect to $(V'_1, \ldots, V'_m)$, where m is the number of odd cycles in $G - F$, can be found in time $\text{poly}(|V(G')|)$.

For each vertex in M, assign the appropriate sign to the corresponding edges of F in G. Then, for each unsigned edge in F, assign it the sign "+." We now colour the edges of G as follows. For each $e \in F$, colour e with 6 if e is "+" and 0 otherwise. For each even cycle C_i of $G - F$, 2-colour the edges of C_i with colours 2 and 4. For each odd cycle C_i of $G - F$, let (v_1, v_2, v_3, v_4) be the 4-segment of C_i in M. Colour v_1v_2 with 1, v_2v_3 with 3, and v_3v_4 with 5. Then, colour the remaining edges of C_i cyclically by alternating between colours 2 and 4, starting at the uncoloured edge incident with v_3v_4. The result is a colouring of G that is clearly a proper circular $7/2$-edge-colouring.

Since the above colouring can be applied in time $O(|V(G)|)$, it suffices to show that G' and the assignment of signs to edges in F can be found in time $\text{poly}(|V(G)|)$. As a better algorithmic result is known, we merely state the runtimes of these tasks and omit their proofs. However, we note that FindITorBD is a subroutine of Sign.

Claim 4.19. *There is an algorithm* MakeGFPM *that takes as inputs a cubic bridgeless graph G with girth g and a 1-factor F of G and returns, in time $O(|V(G)|)$, the graph $G_{F,\pm}$.*

Lemma 4.20. *There exists an algorithm* Sign *that takes as input a cubic bridgeless graph G with girth $g \geq 16$ and returns, in time* $\text{poly}(|V(G)|)$*, a 1-factor F of G and an assignment* *of "+" and "−" to the edges of F such that for every odd cycle C of $G - F$, C contains a 4-segment (v_1, v_2, v_3, v_4) where v_1, v_2 are incident with edges assigned "+" and v_3, v_4 are incident with edges assigned "−."*

4.3 Hitting All Maximum Cliques with an Independent Set

In this section, we discuss some modifications that can be made to a result by King [61] (stated as Theorem 4.22) to find independent sets that meet every maximum clique in a graph. In particular, we will provide an algorithm that, given a graph G with clique number $\omega(G) > \frac{2}{3}(\Delta(G) + 1)$, will return, in polynomial time for fixed $\Delta(G)$, an independent set meeting every maximum clique. Note that a set U *meets* (or *hits*) a graph G if $U \cap V(G) \neq \emptyset$.

This section is organised as follows. We begin with a brief history of the problem of finding an independent set that meets every maximum clique in the graph and how it relates to Reed's conjecture. Next, we prove an algorithmic version of a lemma from [61] (Lemma 4.24) as well as a new corollary of Theorem 3.1 (Corollary 4.25). Finally, we use this new corollary and the original proof of Theorem 4.22 from [61] to make Theorem 4.22 fully algorithmic (Corollary 4.23).

The problem of finding an independent set that meets every maximum clique in a graph is often related to vertex colouring problems. This is because such a set can be assigned its own colour class, which reduces the size of the maximum clique in the graph induced by the set of uncoloured vertices. Thus, finding such independent sets efficiently can lead to finding colourings of certain types of graphs in polynomial time, such as *perfect graphs* [85]. A graph G is *perfect* if for each induced subgraph H of G, the clique number is equal to the chromatic number (i.e. $\omega(H) = \chi(H)$). Some examples of perfect graphs include bipartite graphs, chordal graphs, line graphs of bipartite graphs, and complements of all of these.

In addition to vertex colouring problems, finding an independent set that meets every maximum clique has also been used to approach the following conjecture due to Reed [86].

Conjecture 4.21 (Reed's Conjecture)**.** *For any graph G, $\chi(G) \leq \left\lceil \frac{1}{2}(\Delta(G) + 1 + \omega(G)) \right\rceil$.*

Reed's conjecture has been verified for several families of graphs, including line graphs [63], quasi-line graphs [62, 60], claw-free graphs [60], and graphs with disconnected complements [83].

It is well known that any minimum counterexample G to Reed's conjecture does not contain an independent set M that meets every maximum clique. This is because such a G would contain a maximal independent set M' that meets every maximum clique. Hence

$$\left\lceil \tfrac{1}{2}(\Delta(G-M')+1+\omega(G-M')) \right\rceil + 1 \leq \left\lceil \tfrac{1}{2}(\Delta(G)+1+\omega(G)) \right\rceil.$$

Since M' is an independent set, $\chi(G) \leq \chi(G-M')+1$ and so the minimality of G is contradicted.

Rabern [84] showed that for graphs G with $\omega(G) \geq \frac{3}{4}(\Delta(G)+1)$, the graph G contains an independent set that meets every maximum clique. The following result, due to King [61], is best possible for this form, i.e. the quantity $\frac{2}{3}(\Delta(G)+1)$ can't be improved (see e.g. [67]).

Theorem 4.22 ([61]). *If a graph G satisfies $\omega(G) > \frac{2}{3}(\Delta(G)+1)$, then G contains an independent set that meets every maximum clique.*

To prove Theorem 4.22, King used a corollary of Theorem 2.4 that finds an IT in a graph and vertex partition where there is a k such that each vertex v has at most $\min\{k, |\mathrm{Vclass}(v)|-k\}$ neighbours outside $\mathrm{Vclass}(v)$. He used this result to find an IT in a particular auxiliary graph, called a *clique graph* (Definition 4.26), with respect to a certain partition. For a particular choice of k, this IT of the clique graph gives the desired hitting set in the original graph. The choice of k used by King (which we use as well) is based on results of Hajnal [40] and Kostochka [67] which ensure that each component of the clique graph is sufficiently large and the intersection of the cliques in a component of the clique graph is also sufficiently large.

In this section, we use the same ideas to prove the following algorithmic result.

Corollary 4.23. *There exists an algorithm* HitCliques *that takes as input any graph G with maximum degree Δ and $\omega(G) > \frac{2}{3}(\Delta+1)$ and returns, in time* $\mathrm{poly}(|V(G)|)$ *for fixed Δ, an independent set that meets every maximum clique in G.*

Note that no weakening of the constraints in Theorem 4.22 is necessary to make the result algorithmic.

To prove Corollary 4.23, we require a modification of the corollary of Theorem 2.4 used by King in [61], which we state in Lemma 4.24. We use this modification because the proof of the original corollary is not algorithmic, but our proof of Lemma 4.24 is algorithmic and uses Theorem 3.1.

Lemma 4.24. *Let k be a positive integer and let G be a graph with vertex partition $(V_1, \ldots, V_m)$. If for every i and every $v \in V_i$, the vertex v has at most $\min\{k-1, |V_i| - k\}$ neighbours outside V_i, then G has an IT.*

Recall that the corollary from [61] has a slightly stronger statement, with $\min\{k-1, |V_i| - k\}$ replaced by $\min\{k, |V_i| - k\}$. The algorithmic proof of Lemma 4.24 is as follows.

Proof: We may assume each V_i is independent for the same reasons given at the start of Section 3.1. Thus each vertex of G has at most $k-1$ neighbours, so we have that G is k-claw-free.

Let $\epsilon = \frac{1}{k-1}$. We apply FindITorBD of Theorem 3.1 with $r = k$ and ϵ to inputs G and $(V_1, \ldots, V_m)$. As k is fixed, the running time of FindITorBD is poly$(|V(G)|)$, where the degree of the polynomial depends only on k (since r and ϵ depend only on k). We obtain either:

1. an IT of G, or
2. a set $\mathcal{B}$ of vertex classes and a set D of vertices of G such that D dominates $G_{\mathcal{B}}$ in G and $|D| < (2+\epsilon)(|\mathcal{B}| - 1)$. Moreover D contains $V(K)$ for a constellation K for some $\mathcal{B}_0 \supseteq \mathcal{B}$, where $|D \setminus V(K)| < \epsilon(|\mathcal{B}| - 1)$.

To prove the statement, we must show that for every G and $(V_1, \ldots, V_m)$, outcome (2) never occurs. Suppose for the sake of contradiction that (2) holds for some G and $(V_1, \ldots, V_m)$. Recall from Definition 2.5 that Leaf(K) forms an IT of $|\mathcal{B}_0| - 1$ vertex classes of $\mathcal{B}_0$ and from the algorithm of FindITorBD (see Section 3.5) that FindITorBD returns $L = \text{Leaf}(K) \subseteq D$. Hence D contains a set $Y \subseteq L$ that forms an IT of a set $\mathcal{B}' \subseteq \mathcal{B}$ of vertex classes where $|\mathcal{B}'| \geq |\mathcal{B}| - 1$.

Since D dominates $G_{\mathcal{B}}$, we know

$$\sum_{v \in D} \deg(v) \geq \sum_{V_i \in \mathcal{B}} |V_i| \geq \sum_{V_i \in \mathcal{B}'} |V_i|.$$

Since Y is an IT of $\mathcal{B}'$, we have that $|Y| \geq |\mathcal{B}| - 1$ and so

$$|D \setminus Y| < (1+\epsilon)(|\mathcal{B}| - 1).$$

For any vertex $v \in D \setminus Y$, we know $\deg(v) \leq k-1$. Also, for any vertex $v \in Y$, we have $\deg(v) \leq |\operatorname{Vclass}(v)| - k$. Hence,

$$\begin{aligned}\sum_{v \in D} \deg(v) &\leq (k-1)|D \setminus Y| + \sum_{v \in Y} \deg(v) \\ &< (k-1)[(1+\epsilon)(|\mathcal{B}|-1)] + \sum_{V_i \in \mathcal{B}'} (|V_i| - k) \\ &\leq \epsilon(k-1)(|\mathcal{B}|-1) - (|\mathcal{B}|-1) + \sum_{V_i \in \mathcal{B}'} [(|V_i| - k) + k] \\ &= \sum_{V_i \in \mathcal{B}'} |V_i|.\end{aligned}$$

This is a contradiction and so outcome (2) never occurs. This completes the proof. □

From the proof of Lemma 4.24, we obtain Corollary 4.25. Note that the algorithm of Corollary 4.25 is the algorithm FindITorBD of Theorem 3.1 for $r = k$ and $\epsilon = \frac{1}{k-1}$ on the inputs G and $(V_1, \ldots, V_m)$.

Corollary 4.25. FindITorBD *for* $r = k$ *and* $\epsilon = \frac{1}{k-1}$ *finds, in time* $\operatorname{poly}(|V(G)|)$ *for fixed* k*, an IT in any graph* G *with vertex partition* $(V_1, \ldots, V_m)$ *such that for each* i *and each* $v \in V_i$*, the vertex* v *has at most* $\min\{k-1, |V_i| - k\}$ *neighbours outside* V_i*.*

We are now ready to discuss the *clique graph* $G(\mathcal{C})$ of graph G.

Definition 4.26. Let G be a graph and $\mathcal{C}$ be the set of maximum cliques in G. The *clique graph* $G(\mathcal{C})$ of G is defined as follows. For each clique $C \in \mathcal{C}$, there is a vertex v_C in $G(\mathcal{C})$. Two vertices v_C and v_D of $G(\mathcal{C})$ are adjacent if C and D intersect in G.

For each component i of $G(\mathcal{C})$, let $\mathcal{C}_i$ denote the set of all cliques $C \in \mathcal{C}$ such that v_C is in component i of $G(\mathcal{C})$. Let V_i be the set of vertices in $V(G)$ in the mutual intersection of the cliques in $\mathcal{C}_i$. Hence each maximum clique C of G necessarily contains every vertex in V_i for some unique i (i.e. C does not contain vertices in V_i and V_j for $i \neq j$). Suppose there are m components of $G(\mathcal{C})$. Then we have that an IT M of $G' = G\left[\bigcup_{i=1}^{m} V_i\right]$ with respect to $(V_1, \ldots, V_m)$ is an independent set of vertices that meets every maximum clique in G. (Note that M being an IT implies $V_i \neq \emptyset$ for all $1 \leq i \leq m$.) Thus to prove Corollary 4.23, it suffices to show that G' and $(V_1, \ldots, V_m)$ satisfy Corollary 4.25 for an appropriate choice of k.

To establish this choice of k, we require the following two results due to Hajnal [40] and Kostochka [67].

Lemma 4.27 ([40])**.** *Let G be a graph and $C_1, \dots, C_r$ be a collection of maximum cliques in G. Then*

$$\left|\bigcap_{i=1}^{r} V(C_i)\right| + \left|\bigcup_{i=1}^{r} V(C_i)\right| \geq 2\omega(G).$$

Lemma 4.28 ([67])**.** *Let G be a graph with maximum degree Δ and $\omega(G) > \frac{2}{3}(\Delta+1)$. Let $\mathcal{C}$ be the set of maximum cliques in G. Then for each component i of $G(\mathcal{C})$,*

$$\left|\bigcap_{C \in \mathcal{C}_i} V(C)\right| \geq 2\omega(G) - (\Delta+1).$$

Claim 4.29. *$(V_1, \dots, V_m)$ is a vertex partition of G' such that for each i and $v \in V_i$, the vertex v has at most*

$$\min\left\{\tfrac{1}{3}(\Delta+1) - 1, |V_i| - \tfrac{1}{3}(\Delta+1)\right\}$$

neighbours outside V_i.

Proof: For each $i \in \{1, \dots, m\}$, let U_i be the union of the vertex sets of the cliques in $\mathcal{C}_i$, i.e. $U_i = \bigcup_{C \in \mathcal{C}_i} V(C)$. Since $\omega(G) > \frac{2}{3}(\Delta+1)$, it is clear that $|U_i| > \frac{2}{3}(\Delta+1)$ as each U_i contains the vertex set of at least one maximum clique. Because each $v \in V_i$ is necessarily adjacent to every $u \in U_i$ (excluding itself), this implies that v has at most $\frac{1}{3}(\Delta+1) - 1$ neighbours in G' outside of V_i.

By Lemma 4.28, we have

$$|V_i| > 2\left[\tfrac{2}{3}(\Delta+1)\right] - (\Delta+1) = \tfrac{1}{3}(\Delta+1).$$

Also, by Lemma 4.27, we have

$$|V_i| + |U_i| > 2\left[\tfrac{2}{3}(\Delta+1)\right],$$

and so

$$|V_i| - \tfrac{1}{3}(\Delta+1) > \Delta + 1 - |U_i|.$$

Thus each $v \in V_i$ has fewer than $|V_i| - \frac{1}{3}(\Delta+1)$ neighbours in G' outside of V_i. This completes the proof. □

From Claim 4.29, we see that G' and $(V_1, \dots, V_m)$ satisfy the degree conditions of Corollary 4.25 for $k = \frac{\Delta+1}{3}$. Unfortunately, this choice of k is not guaranteed to be an integer. However, it is easy to show that $k = \left\lceil \frac{\Delta+1}{3} \right\rceil$ will also satisfy the degree conditions.

Claim 4.30. *$(V_1, \ldots, V_m)$ is a vertex partition of G' such that for each i and $v \in V_i$, the vertex v has at most*

$$\min\left\{\left\lceil \tfrac{\Delta+1}{3} \right\rceil - 1, |V_i| - \left\lceil \tfrac{\Delta+1}{3} \right\rceil\right\}$$

neighbours outside V_i.

Proof: By Claim 4.29, we have that for each i and $v \in V_i$, the vertex v has at most $\min\left\{\frac{1}{3}(\Delta+1) - 1, |V_i| - \frac{1}{3}(\Delta+1)\right\}$ neighbours outside V_i. Thus it suffices to show that $\frac{1}{3}(\Delta+1)$ can be replaced by the integer $\left\lceil \frac{\Delta+1}{3} \right\rceil$ without decreasing the number of neighbours v is permitted to have outside V_i. We do so by analysing the cases of $\Delta \mod 3$.

Case 1. $\Delta \equiv 0 \mod 3$.

Then

$$\frac{1}{3}(\Delta+1) - 1 = \frac{\Delta}{3} - \frac{2}{3},$$

and

$$|V_i| - \frac{1}{3}(\Delta+1) = |V_i| - \frac{\Delta}{3} - \frac{1}{3}.$$

As v can only have an integer number of neighbours, v has at most $\frac{\Delta}{3} - 1$ or $|V_i| - \frac{\Delta}{3} - 1$ neighbours outside V_i. Since $\left\lceil \frac{\Delta+1}{3} \right\rceil = \frac{\Delta}{3} + 1$, we have $\left\lceil \frac{\Delta+1}{3} \right\rceil - 1 = \frac{\Delta}{3}$ and $|V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil = |V_i| - \frac{\Delta}{3} - 1$. Hence v has at most $\min\left\{\left\lceil \frac{\Delta+1}{3} \right\rceil - 1, |V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil\right\}$ neighbours outside V_i.

Case 2. $\Delta \equiv 1 \mod 3$.

Then

$$\frac{1}{3}(\Delta+1) - 1 = \frac{\Delta-1}{3} - \frac{1}{3},$$

and

$$|V_i| - \frac{1}{3}(\Delta+1) = |V_i| - \frac{\Delta-1}{3} - \frac{2}{3}.$$

As v can only have an integer number of neighbours, v has at most $\frac{\Delta-1}{3} - 1$ or $|V_i| - \frac{\Delta-1}{3} - 1$ neighbours outside V_i. Since $\left\lceil \frac{\Delta+1}{3} \right\rceil = \frac{\Delta-1}{3} + 1$, we have $\left\lceil \frac{\Delta+1}{3} \right\rceil - 1 = \frac{\Delta-1}{3}$ and $|V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil = |V_i| - \frac{\Delta-1}{3} - 1$. Hence v has at most $\min\left\{\left\lceil \frac{\Delta+1}{3} \right\rceil - 1, |V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil\right\}$ neighbours outside V_i.

Case 3. $\Delta \equiv 2 \mod 3$.

Then

$$\frac{1}{3}(\Delta+1) - 1 = \frac{\Delta+1}{3} - 1,$$

and

$$|V_i| - \frac{1}{3}(\Delta+1) = |V_i| - \frac{\Delta+1}{3}.$$

Since $\left\lceil \frac{\Delta+1}{3} \right\rceil = \frac{\Delta+1}{3}$, we have $\left\lceil \frac{\Delta+1}{3} \right\rceil - 1 = \frac{\Delta+1}{3} - 1$ and $|V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil = |V_i| - \frac{\Delta+1}{3}$. Hence v has at most $\min\left\{ \left\lceil \frac{\Delta+1}{3} \right\rceil - 1, |V_i| - \left\lceil \frac{\Delta+1}{3} \right\rceil \right\}$ neighbours outside V_i. □

We now have all of the required ingredients to prove Corollary 4.23. Let HitCliques be the algorithm defined in 4.3.1.

4.3.1 HitCliques

Input: A graph G with $\omega(G) > \frac{2}{3}(\Delta(G)+1)$.
Output: An independent set M that meets every maximum clique in G.

```
1: function HitCliques($G$)
2:     $\mathcal{C}$ := MaximumCliques($G$)
3:     $G(\mathcal{C})$ := MakeCliqueGraph($G, \mathcal{C}$)
4:     $\mathcal{V}$ := CliqueIntersect($G, G(\mathcal{C})$)
5:     $G'$ := MakeG'($G, \mathcal{V}$)
6:     $k := \left\lceil \frac{\Delta(G)+1}{3} \right\rceil$
7:     $M$ := FindITorBD($G'; \mathcal{V}$) for $r = k$ and $\epsilon = \frac{1}{k-1}$.
8:     return $M$
```

The subroutines used by HitCliques perform the necessary tasks outlined in the discussion after Definition 4.26. In particular, MaximumCliques will find the set $\mathcal{C}$ of all maximum cliques in G. This can be accomplished by first finding all maximal cliques in the graph and then restricting the set to those of maximum size. For fixed $\Delta(G)$, it is known that the set of maximal cliques can be found in time $O(|V(G)|)$ and that there are at most $O(|V(G)|)$ maximal cliques (see Appendix B). Given G and $\mathcal{C}$, the subroutine MakeCliqueGraph will construct the clique graph $G(\mathcal{C})$.

CliqueIntersect is an algorithm that, given a graph G and its clique graph $G(\mathcal{C})$, will return a set $\mathcal{V}$ of lists V_i of vertices of G such that each component i of $G(\mathcal{C})$ has a unique V_i and each V_i contains all of the vertices that are in every clique $C \in \mathcal{C}_i$. Given this vertex partition $\mathcal{V}$ and G, MakeG' creates the graph G' induced by the vertices in the union of lists in $\mathcal{V}$. This choice of G' and $\mathcal{V}$ satisfy the conditions of Corollary 4.25 with $k = \left\lceil \frac{\Delta(G)+1}{3} \right\rceil$ (Claim 4.30). Thus FindITorBD for this k returns an independent set M that hits all maximum cliques in G.

For fixed $\Delta(G)$, the majority of the runtime of HitCliques is spent implementing FindITorBD, which takes $\text{poly}(|V(G')|) = \text{poly}(|V(G)|)$. We refer the reader to Appendix B to verify the runtime of the other subroutines, which are also $\text{poly}(|V(G)|)$ for fixed $\Delta(G)$.

4.4 Strong Colourings

In this section, we discuss a result of Aharoni, Berger, and Ziv [3], which we state in Theorem 4.33. In particular, we will show that a slight strengthening of the constraints of Theorem 4.33 leads to an algorithmic version of the result (see Corollary 4.34). Since the original result of Aharoni, Berger, and Ziv uses Theorem 2.3, this slightly weaker algorithmic result is due to the stronger constraints in Corollary 3.2. We will see that a difference of 1 in the constraints of Theorem 4.33 is all that is required to make the result algorithmic (just as with Corollary 3.2).

This section is organised as follows. We begin with a brief discussion of strong colourings (defined in Definition 6.12) and the result of Aharoni, Berger, and Ziv [3]. We then present a theorem from [3] (Theorem 4.35) that is a corollary of Theorem 2.4 and prove an algorithmic version of this theorem (Corollary 4.36). Using this algorithmic result and the original proof of Theorem 4.33, we prove Corollary 4.34.

We begin with some definitions. For these definitions, let k and n be positive integers, G be a graph on n vertices, and $\mathcal{V} = (V_1, \ldots, V_m)$ be a vertex partition of G such that $|V_i| \leq k$ for all i.

Definition 4.31. A graph G is *strongly k-colourable with respect to $\mathcal{V}$* if there is a k-vertex colouring of G such that for each V_i, each colour is assigned to at most one vertex of V_i.

We call a k-colouring that shows G is strongly k-colourable with respect to a given partition a *strong k-colouring* of G. Similarly, a partial k-colouring of G is a *partial strong k-colouring* if for each colour, each vertex class contains at most one vertex assigned that colour. Note that the vertex classes can contain any number of vertices that are not assigned a colour in a partial strong k-colouring.

Definition 4.32. A graph G is *strongly k-colourable* if for every vertex partition $\mathcal{V}$ of G into classes of size at most k, G is strongly k-colourable with respect to $\mathcal{V}$. The *strong chromatic number* of a graph G, denoted $s\chi(G)$, is the minimum k such that G is strongly k-colourable.

The notion of a strong chromatic number was introduced independently by Alon [5] and Fellows [34]. It has since been widely studied [36, 77, 50, 16, 3, 72, 51]. The best known general bound for $s\chi(G)$ in terms of its maximum degree is $s\chi(G) \leq 3\Delta(G)-1$, which was proved by Haxell [50]. (See also [51] for an asymptotically better bound due to Haxell.) However, it is conjectured (see e.g. [3]) that the correct general bound in $s\chi(G) \leq 2\Delta(G)$. If true, this would be the best possible general bound [91].

Using a nice simplification of the proof in [50], Aharoni, Berger, and Ziv [3] proved the following.

Theorem 4.33 ([3])**.** *Every graph G with maximum degree Δ satisfies $s\chi(G) \leq 3\Delta$.*

In this section, we will prove the following algorithmic result.

Corollary 4.34. *There exists an algorithm* StrongColour *that takes as input any graph G with maximum degree Δ and vertex partition $(V_1, \dots, V_m)$ where $|V_i| \leq 3\Delta + 1$ for each i, and finds, in time* $\text{poly}(|V(G)|)$ *for fixed Δ, a strong $(3\Delta+1)$-colouring of G with respect to $(V_1, \dots, V_m)$.*

Note that Corollary 4.34 implies that every graph G with maximum degree Δ satisfies $s\chi(G) \leq 3\Delta + 1$. This is because for any such graph G, StrongColour will return a strong $(3\Delta + 1)$-colouring of G so long as the provided vertex partition $(V_1, \dots, V_m)$ satisfies $|V_i| \leq 3\Delta + 1$ for each i. Hence G is strongly $(3\Delta + 1)$-colourable with respect to every vertex partition of G into classes of size at most $3\Delta + 1$. Thus Corollary 4.34 is a slightly weaker algorithmic version of Theorem 4.33.

To prove Corollary 4.34, we will use the same techniques Aharoni, Berger, and Ziv used to prove Theorem 4.33 in [3]. The main idea of this technique is to increase the size of a colour class α by taking an uncoloured vertex v such that α is missing on $\text{Vclass}(v)$. We then find an IT containing v and assign every vertex in this IT the colour α. Some adjustments are needed for the colouring to remain a strong colouring, but this is the main idea of the technique. Thus, we require some notion for when we can find an IT containing a specified vertex. This leads to the following slight strengthening of Theorem 2.3 from [3].

Theorem 4.35 ([3])**.** *Let G be a graph with maximum degree Δ. Then in any vertex partition $(V_1, \dots, V_m)$ of G where $|V_i| \geq 2\Delta$, for each vertex $v \in V(G)$ there exists an IT of G containing v.*

Theorem 4.35 follows immediately from Theorem 2.4 applied to G with vertex partition $(\{v\}, V_2, \dots, V_m)$, assuming without loss of generality that $v \in V_1$. Thus, by applying Theorem 3.1 instead of Theorem 2.4, we obtain the following slightly stronger version of Corollary 3.2.

Corollary 4.36. *There exists an algorithm that takes as input any graph G with maximum degree Δ, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| \geq 2\Delta + 1$ for each i, and any $v \in V(G)$, and finds, in time* $\mathrm{poly}(|V(G)|)$ *for fixed Δ, an IT in G that contains v.*

Proof: Let G be a graph with maximum degree Δ, $(V_1, \dots, V_m)$ a vertex partition of G such that $|V_i| \geq 2\Delta + 1$ for each i, and $v \in V(G)$. Without loss of generality, assume $v \in V_1$.

Define $G' = G\left[\{v\} \cup \left(\bigcup_{i=2}^{m} V_i\right)\right]$ and consider the vertex partition $(\{v\}, V_2, \dots, V_m)$ of G'. As G' is a subgraph of G, the maximum degree Δ' of G' is at most Δ and so G' is $(\Delta{+}1)$-claw-free with respect to $(\{v\}, V_2, \dots, V_m)$. Taking $\epsilon = \frac{1}{\Delta}$, we have that FindITorBD returns either an IT of G' (which necessarily includes v) or a set $\mathcal{B}$ of vertex classes and a set D of vertices of G' such that D dominates $G'_{\mathcal{B}}$ in G' and $|D| < (2 + \epsilon)(|\mathcal{B}| - 1)$.

Claim 4.37. FindITorBD$(G'; \{v\}, V_2, \dots, V_m)$ *always returns an IT of G'.*

Proof: Suppose not. Let $\mathcal{B}$ be the set of vertex classes and D the set of vertices returned by FindITorBD. As $\Delta' \leq \Delta$, we have that D dominates at most

$$\Delta'|D| < \Delta\left(2 + \tfrac{1}{\Delta}\right)(|\mathcal{B}| - 1) = (2\Delta + 1)(|\mathcal{B}| - 1)$$

vertices. However, $G_{\mathcal{B}}$ contains at least $(2\Delta + 1)|\mathcal{B}|$ vertices, which contradicts D dominating $G_{\mathcal{B}}$. Hence no such $\mathcal{B}$ and D exist, so FindITorBD always returns an IT of G'. □

Let M be the IT of G' returned by FindITorBD. Since each vertex class of the partition of G' is a subset of a vertex class in the partition of G, M is also an IT in G. Moreover, M clearly contains v. For fixed Δ, determining G' and the appropriate vertex partition takes time $O(|V(G)|)$, so finding M takes time $\mathrm{poly}(|V(G)|)$ (since $|V(G')| \leq |V(G)|$ and Δ is fixed). □

We now prove Corollary 4.34 using the proof of Theorem 4.33 and Corollary 4.36. Let StrongColour be the algorithm defined in 4.4.1. We will discuss what each of the subroutines do later.

The vector c only has entries in $\{0, \dots, 3\Delta + 1\}$. We think of c as a colouring of $V(G)$ such that each $v \in V(G)$ is assigned colour $c(v)$, with $c(v) = 0$ indicating that the vertex v is uncoloured. We prove that the returned vector c is a strong $(3\Delta + 1)$-colouring in Lemma 4.38.

Before discussing each subroutine individually, we give an outline of the re-colouring process in the while loop. This process is identical to the technique used by Aharoni,

4.4.1 StrongColour

Input: A graph G with maximum degree Δ and a vertex partition $(V_1, \ldots, V_m)$ where $|V_i| \leq 3\Delta + 1$ for each $1 \leq i \leq m$.

Output: A strong $(3\Delta + 1)$-colouring of G with respect to $(V_1, \ldots, V_m)$.

```
1:  function StrongColour($G; V_1, \ldots, V_m$)
2:      for all $v \in V(G)$ do $c(v) := 0$
3:      while there is a vertex $v$ such that $c(v) = 0$ do
4:          Choose a vertex $v$ such that $c(v) = 0$.
5:          Choose a colour $\alpha$ missing on Vclass($v$).
6:          $(W, V'_1, \ldots, V'_m) :=$ RemoveColoured($G; c; \alpha; V_1, \ldots, V_m$)
7:          $\mathcal{V} := \{V'_i : V_i \neq \text{Vclass}(v)\}$
8:          $G' := G\left[\{v\} \cup \left(\bigcup_{V'_j \in \mathcal{V}} V'_j\right)\right]$
9:          $M :=$ FindITorBD($G'; \{v\}, \mathcal{V}$) for $r = \Delta + 1$ and $\epsilon = \frac{1}{\Delta}$.
10:         $c :=$ ReColour($c; \alpha; M; W; V_1, \ldots, V_m$)
11:     return $c$
```

Berger, and Ziv in [3] to prove Theorem 4.33. Let G be a graph with maximum degree Δ and $(V_1, \ldots, V_m)$ a vertex partition such that $|V_i| \leq 3\Delta + 1$ for all i. By adding some isolated vertices to G and the vertex classes, we may assume $|V_i| = 3\Delta + 1$ for each i.

Let c be a partial strong $(3\Delta + 1)$-colouring of G and let $v \in V(G)$ be a vertex not coloured by c. Without loss of generality, we may assume $v \in V_1$. As $|V_1| = 3\Delta + 1$, there must be at least one colour missing on V_1. Let α be such a colour. Define W to be the set of all vertices coloured α by c. For each V_i containing a vertex $w_i \in W$, let $V'_i \subseteq V_i$ be the set of vertices that are not assigned a colour that appears on $N(w_i)$. (Note that 0 is not a "colour.") For all other V_i, let $V'_i = V_i$. As the maximum degree of G is Δ, at most Δ vertices are removed from V_i to obtain V'_i. Thus $|V'_i| \geq 2\Delta + 1$ for all i. Hence by Corollary 4.36, we can find an IT M containing v in the graph $G' = G\left[\bigcup_{i=1}^{m} V'_i\right]$ with respect to vertex partition $(V'_1, \ldots, V'_m)$.

Let $u_i = M \cap V_i$ for each i, noting that $u_1 = v$. By re-colouring each $w_i \in W$ with $c(u_i)$ and (re-)colouring each u_i with α, we have a new colouring c'. As $M \subseteq \bigcup_{i=1}^{m} V'_i$, no vertex adjacent to w_i is coloured $c(u_i)$ by c' and so c' is a proper colouring. Also, the number of vertices coloured by c' is greater than the number coloured by c. This is because no step of this process decreases the total number of coloured vertices (colours are only re-assigned or traded) and $c'(v) = \alpha$ but $c(v) = 0$.

The subroutines RemoveColoured and ReColour perform the operations their names suggest. RemoveColoured returns the set W of vertices coloured α and the subsets $V_i' \subseteq V_i$ of vertices not assigned a colour that appears on the neighbourhood $N(w_i)$ of $w_i \in W \cap V_i$. ReColour performs the colouring modification outlined in the proceeding paragraph.

We are now ready to prove Lemma 4.38.

Lemma 4.38. StrongColour *returns a strong* $(3\Delta + 1)$*-colouring.*

Proof: Consider an iteration of the while loop. Let c be the colouring at the start of the iteration and c' be the colouring at the end of the iteration. We show that if c is a partial strong $(3\Delta + 1)$-colouring, then so is c'.

Suppose V_i is a vertex class and $w \in V_i$ is a vertex such that $c(w) = \alpha$. By the choice of V_i', it is known that for $u \in M \cap V_i$, the colour $c(u)$ is not assigned to any neighbour of w. Thus setting $c'(w) = c(u)$ maintains a proper colouring of G. We claim that re-assigning u the colour α maintains the colouring. This is because any $w' \in N(u)$ with $c(w') = \alpha$ is a vertex that will have $c'(w') = c(u')$, where $u' \in M \cap \text{Vclass}(w')$. Thus the only vertices that are coloured α by c' are the vertices in M, which is an independent set. This re-assignment also implies that if c is a colouring such that no two vertices in the same class are assigned the same colour, then no two vertices in the same vertex class are assigned the same colour under c'. Hence c' is a partial strong $(3\Delta + 1)$-colouring if c is a partial strong $(3\Delta + 1)$-colouring.

Note that for every vertex class V_i that contains a vertex w such that $c(w) = \alpha$, switching $c(w)$ and $c(u)$, where $u \in M \cap V_i$, does not change the number of vertices in V_i that are not coloured. In vertex classes V_i where c did not assign any vertex the colour α, there are two possibilities. First, if the vertex $u \in M \cap V_i$ has $c(u) \neq 0$, the number of vertices in V_i that are not coloured remains unchanged. Otherwise, $u \in M \cap V_i$ has $c(u) = 0$ and so $c'(u) = \alpha$ implies the number of uncoloured vertices in V_i decreases by 1. Since the vertex chosen as v is in the latter category, the number of uncoloured vertices decreases by at least 1 during the iteration. Thus StrongColour will eventually terminate and every vertex is assigned some non-zero colour by c'. As StrongColour starts with an empty colouring c, which is trivially a partial strong $(3\Delta + 1)$-colouring, StrongColour will return a strong $(3\Delta + 1)$-colouring of G. □

It remains to show that StrongColour runs in time $\text{poly}(|V(G)|)$ for fixed Δ. As with CircularP and HitCliques, the majority of the runtime of StrongColour is due to implementing FindITorBD, which completes in time $\text{poly}(|V(G')|) = \text{poly}(|V(G)|)$ for fixed Δ. The interested reader can find a discussion of the runtimes of the other steps of StrongColour in Appendix C.

We will revisit the problem of finding strong colourings algorithmically in Chapter 6 when we discuss randomised algorithms.

4.5 Bounded Size Monochromatic Components

In this section, we discuss algorithmic versions of two results on colourings with small monochromatic components due to Alon, Ding, Oporowski, and Vertigan [9], which we state in Theorems 4.40 and 4.41. As with the result of Aharoni, Berger, and Ziv discussed in Section 4.4, we will show that a slight strengthening of the constraints in Theorems 4.40 and 4.41 will lead to algorithmic versions of these results (see Corollaries 4.51 and 4.54).

This section is organised as follows. We begin by defining monochromatic components (see Definition 4.39) and a brief discussion of known results related to colourings with small monochromatic components. We then present a lemma from [9] (Lemma 4.42) that follows from Theorem 2.3 and prove an algorithmic version of the lemma (Corollary 4.43). Using Corollary 4.43 and the original proofs of Theorems 4.40 and 4.41, we prove Corollaries 4.51 and 4.54.

Before providing a formal definition of a monochromatic component, consider the following interpretation of vertex and edge colourings. Let G be a graph. We can interpret a vertex partition $(V_1, \ldots, V_m)$ of G as a collection $(G_1, \ldots, G_m)$ of vertex-disjoint induced subgraphs G_i of G such that $V(G_i) = V_i$ for each i. Note that although $\bigcup_{i=1}^{m} E(G_i) \subseteq E(G)$, it is not usually the case that $\bigcup_{i=1}^{m} E(G_i) = E(G)$ since there can be edges in G whose endpoints are in different V_i. Thus a proper k-vertex colouring of G is a vertex partition $(G_1, \ldots, G_k)$ of G where each G_i is an edgeless graph. Similarly, we can interpret an edge partition $(E_1, \ldots, E_m)$ of G as a collection $(G_1, \ldots, G_m)$ of edge-disjoint subgraphs G_i such that $\bigcup_{i=1}^{m} E(G_i) = E(G)$. Hence a proper k-edge colouring of G is an edge partition $(G_1, \ldots, G_k)$ where each G_i is a matching. Therefore, we can interpret any vertex or edge partition as a vertex or edge colouring respectively. This leads to the following definition.

Definition 4.39. Let $(G_1, \ldots, G_m)$ be a vertex (or edge) partition of a graph G. A subgraph H of G is a *monochromatic component* of G if H is a component of some G_i.

By Definition 4.39, the monochromatic components of G in a proper k-vertex colouring of G contain only one vertex. Similarly, the monochromatic components of G in a proper k-edge colouring of G contain at most one edge. (Isolated vertices are still monochromatic components of G_i.) By allowing for larger monochromatic components, we obtain a natural relaxation of proper vertex and edge colourings.

Colourings that allow for monochromatic components of a bounded size are sometimes referred to as *clustered colourings*. Much work has been done for clustered colourings [9, 53, 30, 18, 19, 70, 33, 74, 71]. In particular, for 2-vertex colourings of graphs G with maximum degree Δ, it is known that the maximum size c of the monochromatic components of G satisfy $c = 2$ when $\Delta = 3$ [9], $c = 6$ when $\Delta = 4$ [53], and $c < 20000$ when $\Delta = 5$ [53]. See [94] for a survey by Wood on clustered colourings.

In this section, we will focus on the following two results of Alon, Ding, Oporowski, and Vertigan from [9]. Both results provide a colouring of a graph G with maximum degree Δ so that the monochromatic components have size bounded by a function in Δ.

Theorem 4.40 ([9])**.** *Every graph with maximum degree Δ can be $\lceil \frac{\Delta+2}{3} \rceil$-vertex coloured so that every monochromatic component has at most*

$$f(\Delta) = \begin{cases} 1 & \text{if } \Delta = 0 \\ 2 & \text{if } \Delta = 1, 2 \\ 12\Delta^2 - 36\Delta + 9 & \text{if } \Delta \geq 3 \end{cases}$$

vertices.

Theorem 4.41 ([9])**.** *Every loopless graph G with maximum degree Δ can be $\lceil \frac{\Delta+1}{2} \rceil$-edge coloured such that every monochromatic component has at most*

$$g(\Delta) = \begin{cases} 0 & \text{if } \Delta = 0 \\ 1 & \text{if } \Delta = 1 \\ 60\Delta - 63 & \text{if } \Delta \geq 2 \end{cases}$$

edges.

The proofs of Theorems 4.40 and 4.41 use the following lemma from [9].

Lemma 4.42 ([9])**.** *Let $d \geq 3$ and $r \geq 1$ be integers and let G be a graph with maximum degree $\Delta(G) \leq d$. Let (A, B) be a vertex partition of G and let $(B_1, \ldots, B_m)$ be a partition of B. Suppose:*

(i) $\Delta(G[A]) \leq 1$,

(ii) $\Delta(G[B]) \leq d - 2$,

(iii) *each* $G[B_i]$ *is either a cycle or a path, and*

(iv) *for each* $i \in \{1, \ldots, m\}$ *and* $v \in B_i$ *with* $\deg_{G[B_i]}(v) = 2$, *there are at most* r *components of* $G[A]$ *that contain neighbours of* v.

Then there is a set $M \subseteq B$ *such that every component of* $G[B_i \setminus M]$ *for each* $i \in \{1, \ldots, m\}$ *and every component of* $G[A \cup M]$ *has at most*

$$K = (12r + 6)d - (18r + 27)$$

vertices.

Figure 4.3 gives an example of such a vertex partition (A, B) with partition (B_1, B_2, B_3) of B.

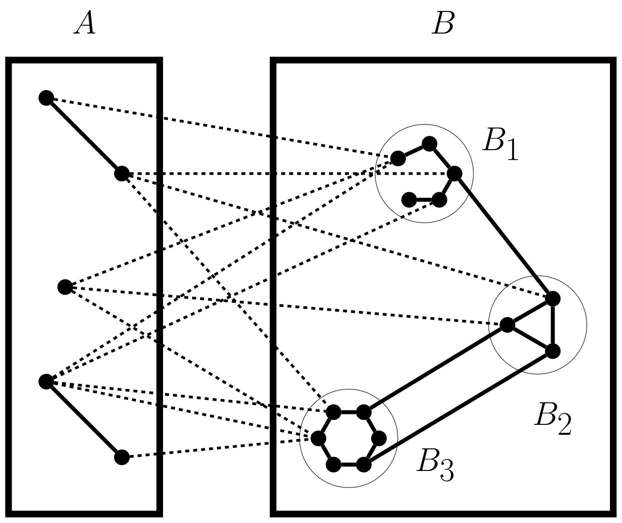

Figure 4.3: A vertex partition (A, B) of a graph G satisfying the conditions of Lemma 4.42 for $d = 5$ and $r = 2$.

The proof of Lemma 4.42 uses the choice of K and Theorem 2.3 to find an IT of a particular auxiliary graph. This IT is the desired set M that makes the lemma hold. For an algorithmic version of Lemma 4.42, we only require a slight weakening of the value of K. Specifically, we increase to $K = (12r + 6)d - (18r + 24)$. This leads to the following result.

Corollary 4.43. *There exists an algorithm* SmallComponents *that for $d \geq 3$ and $r \geq 1$, takes as inputs any graph G with maximum degree $\Delta(G) \leq d$, a vertex partition (A, B) of G, and a partition $(B_1, \ldots, B_m)$ of B such that:*

(i) $\Delta(G[A]) \leq 1$,

(ii) $\Delta(G[B]) \leq d - 2$,

(iii) each $G[B_i]$ is either a cycle or a path, and

(iv) for each $i \in \{1, \ldots, m\}$ and each $v \in B_i$ with $\deg_{G[B_i]}(v) = 2$, there are at most r components of $G[A]$ that contain neighbours of v.

For fixed d and r, it returns in time $\text{poly}(|V(G)|)$, *a set $M \subseteq B$ such that every component of $G[B_i \setminus M]$ for each $i \in \{1, \ldots, m\}$ and every component of $G[A \cup M]$ has at most*

$$K = (12r + 6)d - (18r + 24)$$

vertices.

We will prove Corollary 4.43 as follows. Let SmallComponents be the algorithm defined in 4.5.1.

The first subroutine of SmallComponents, called Order, arranges the vertices of B_i of degree 2 in $G[B_i]$ in the order of their appearance in the path or cycle of $G[B_i]$, returning this set as B'_i. The next subroutine, called SmallerParts, uses the ordering of B'_i to make ordered lists $B'_{i,j}$ of size exactly $2k + 1$ (which are returned in the set $\mathcal{B}'_i$) and (a possibly empty) list B^*_i of size at most $2k$ (which is not included in $\mathcal{B}'_i$). The union of the elements in the $B'_{i,j}$ is returned as B''_i. The subroutine MakeG' creates the graph G' whose vertex set is V and whose edge set consists only of edges uv such that either u and v are neighbours of vertices in the same component of $G[A]$ or $uv \in E(G)$ with $\text{Vclass}(u) \neq \text{Vclass}(v)$. Thus $\mathcal{B}'$ is a vertex partition of G' whose vertex classes are the $B'_{i,j}$ for the B'_i with $|B'_i| > K$.

Proposition 4.44. FindITorBD$(G', \mathcal{B}')$ *always returns an IT M of G' with respect to $\mathcal{B}'$.*

Proof: Let G be the graph with maximum degree $\Delta \leq d$, (A, B) the vertex partition, and $(B_1, \ldots, B_m)$ the partition of B given as inputs to SmallComponents. For each component C of $G[A]$, let $N^*(C)$ denote the set of vertices in B that are adjacent to vertices in C, i.e. $N^*(C) = \bigcup_{v \in V(C)} (N(v) \cap B)$.

4.5.1 SmallComponents

Input: A graph G with maximum degree $\Delta(G) \leq d$, a vertex partition (A, B), and a partition $(B_1, \ldots, B_m)$ of B satisfying the conditions of Corollary 4.43 for some $d \geq 3$ and $r \geq 1$.

Output: A set $M \subseteq B$ such that every component of $G[B_i \setminus M]$ for each $i \in \{1, \ldots, m\}$ and every component of $G[A \cup M]$ has at most $(12r+6)d - (18r+24)$ vertices.

```
1: function SmallComponents($G; A; B; B_1, \ldots, B_m$)
2:     $K := (12r+6)d - (18r+24)$
3:     $k := (2r+1)d - (3r+4)$
4:     for $i$ such that $|B_i| > K$ do
5:         $B_i' := \text{Order}(G, B_i)$
6:         $(\mathcal{B}_i', B_i'') := \text{SmallerParts}(2k+1, B_i')$
7:     $V := \bigcup_{|B_i'|>K} B_i''$
8:     $\mathcal{B}' := \bigcup_{|B_i'|>K} \mathcal{B}_i'$
9:     $G' := \text{MakeG'}(V; G; A; B_1, \ldots, B_m)$
10:    $M := \text{FindITorBD}(G'; \mathcal{B}')$ for $r = k+1$ and $\epsilon = \frac{1}{k}$.
11:    return $M$
```

Claim 4.45. *For each component C of $G[A]$, we have $|N^*(C)| \leq 2(d-1)$.*

Proof: Let C be a component of $G[A]$. By (i), $\Delta(G[A]) \leq 1$ and so $|V(C)| \leq 2$. Thus $|N^*(C)| \leq \Delta$ if $|V(C)| = 1$ and $|N^*(C)| \leq 2\Delta - 2$ if $|V(C)| = 2$. Since $\Delta \leq d$ and $d \geq 3$, we have that $|N^*(C)| \leq 2(d-1)$ in either case. $\square$

Claim 4.46. *$|B_i'| \geq 2k+1$ for all i such that $|B_i| > K$.*

Proof: By (iii), $G[B_i]$ is either a path or a cycle. As Order(G, B_i) returns B_i', the ordered set of vertices of degree 2 in $G[B_i]$, we have $|B_i'| \geq |B_i| - 2 > K - 2$.

Recall that $k = (2r+1)d - (3r+4)$. Since $d \geq 3$ and $r \geq 1$, we have that $k > 0$ and

$$K - (2k+1) = (8r+4)d - (12r+16) - 1 \geq 12r - 5 > 0.$$

Thus $K > 2k+1 > 0$. Hence $|B_i'| \geq K - 1 \geq 2k+1$ for each i such that $|B_i| > K$. $\square$

By Claim 4.46, $|B_i'| \geq 2k+1$ for all i such that $|B_i| > K$. Thus $\mathcal{B}_i'$ contains at least one $B_{i,j}'$, and so $\mathcal{B}_i'$ is non-empty for all i such that $|B_i| > K$.

Let $V = \bigcup_{|B_i|>K} B_i''$ and $\mathcal{B}' = \bigcup_{|B_i|>K} \mathcal{B}_i'$. The graph G' returned by MakeG' has vertex set V and edge set consisting of all edges uv such that either $u, v \in N^*(C)$ for some component C of $G[A]$ (type 1) or $uv \in E(G)$ with Vclass$(u) \neq$ Vclass(v) (type 2).

Claim 4.47. *$\Delta(G') \leq k$.*

Proof: Consider a vertex $v \in V$. By (iv), $v \in N^*(C)$ for at most r components C of $G[A]$. Thus by Claim 4.45, v has at most $r[|N^*(C)| - 1] = r(2d-3)$ neighbours via edges of type 1. Also, since $v \in B_i'$ for some i such that $|B_i| > K$, we know that $\deg_{G[B_i]}(v) = 2$. By (ii), $\deg_{G[B]}(v) \leq d-2$ and so v has at most $(d-2) - 2 = d-4$ neighbours via edges of type 2. Thus

$$\Delta(G') \leq r(2d-3) + (d-4) = (2r+1)d - (3r+4) = k. \qquad \square$$

By the definition of the $\mathcal{B}_i'$, we have that each vertex class $B_{i,j}'$ of $\mathcal{B}'$ has size exactly $2k+1$. Hence by Corollary 3.2, FindITorBD returns an IT M of G' with respect to $\mathcal{B}'$. $\square$

By Proposition 4.44, SmallComponents always returns an IT of G', which is a set $M \subseteq B$. It remains to show that the M returned by SmallComponents satisfies the desired conditions (Lemma 4.48) and completes in time poly$(|V(G)|)$ for fixed d and r. We prove Lemma 4.48 now and discuss the runtime of SmallComponents in Appendix D.1.

Lemma 4.48. *The set M returned by* SmallComponents *is a subset of B such that every component of $G[A \cup M]$ and every component of $G[B_i \setminus M]$ for each $i \in \{1, \ldots, m\}$ has at most K vertices.*

Proof: Let G be the graph with maximum degree $\Delta \leq d$, (A, B) the vertex partition, and $(B_1, \ldots, B_m)$ the partition of B given as inputs to SmallComponents and suppose they satisfy the conditions of Corollary 4.43. Let M be the set returned by SmallComponents. For each component C of $G[A]$, we again use $N^*(C)$ to denote the set of vertices in B that are adjacent to vertices in C, so $N^*(C) = \bigcup_{v \in V(C)} (N(v) \cap B)$.

Claim 4.49. *Every component of $G[A \cup M]$ has fewer than K vertices.*

Proof: Note that $E(G')$ contains all edges of types 1 and 2, so any edge of $G[M]$ is an edge of $G[B_i]$ for some i where $|B_i| > K$. Hence (iii) implies that the components of $G[M]$ contain at most two vertices. (Note that at least one neighbour in G of $v \in M$ is in the same $B'_{i,j}$ as v.) Furthermore, since M is an IT of G' (Proposition 4.44), the edges of type 1 imply that for each component C of $G[A]$, we have $|N^*(C) \cap M| \leq 1$. Hence the components of $G[A \cup M]$ have at most two vertices in M, which are vertices in the same B'_i but different $B'_{i,j}$ and whose neighbours in A are in different components of $G[A]$. Each vertex in M is adjacent to at most r components of $G[A]$ by (iv). By (i), the components of $G[A]$ each have at most two vertices. Thus the components of $G[A \cup M]$ have size at most $2[1 + r(2)] = 4r + 2$. As $d \geq 3$ and $r \geq 1$, we have

$$K - (4r + 2) \geq 14r - 8 > 0.$$

Hence every component of $G[A \cup M]$ has fewer than K vertices. □

Claim 4.50. *Every component of $G[B_i \setminus M]$ has size at most K.*

Proof: For B_i such that $|B_i| \leq K$, it is clear that any component of $G[B_i \setminus M]$ has size at most $|B_i \setminus M| = |B_i| \leq K$. Thus we must show the statement holds for B_i where $|B_i| > K$. Note that for these B_i, the components of $G[B_i \setminus M]$ are paths. Thus for a component P of $G[B_i \setminus M]$, either both ends of P are adjacent in G to a vertex in M or only one end of P is adjacent in G to a vertex in M. The latter only occurs when $G[B_i]$ is a path.

Let P be a component of $G[B_i \setminus M]$. Suppose both ends of P are adjacent in G to vertices in M. Let u and v be these vertices in M and let $B'_{i,a} = \text{Vclass}(u)$ and $B'_{i,b} = \text{Vclass}(v)$ be their classes in $\mathcal{B}'$. As M is an IT with respect to $\mathcal{B}'$, there cannot exist a vertex class $B'_{i,j} \in \mathcal{B}'$ such that the vertices in $B'_{i,j}$ occur between the vertices of $B'_{i,a}$ and $B'_{i,b}$ in the ordering of B'_i. However, it is possible for the vertices of B^*_i to occur between the vertices of $B'_{i,a}$ and $B'_{i,b}$ in the path/cycle order of $G[B_i]$. Thus there are at most $2k$ vertices of B_i between the vertices in $B'_{i,a}$ and vertices in $B'_{i,b}$ in $G[B_i]$. Since $u, v \in B'_{i,a} \cup B'_{i,b}$,

$$\begin{aligned}|V(P)| &\leq (|B'_{i,a}| - 1) + |B^*_i| + (|B'_{i,b}| - 1)\\ &\leq [(2k+1) - 1] + 2k + [(2k+1) - 1]\\ &= 6k\\ &= (12r+6)d - (18r+24)\\ &= K.\end{aligned}$$

Suppose only one end of P is adjacent to a vertex in M and let v be that vertex in M. Then $G[B_i]$ is a path and P contains an end x of $G[B_i]$. Let $B'_{i,a} = \text{Vclass}(v)$ in $\mathcal{B}'$. As M is an IT with respect to $\mathcal{B}'$, there cannot exist a vertex class $B'_{i,j} \in \mathcal{B}'$ such that the vertices in $B'_{i,j}$ occur between the vertices of $B'_{i,a}$ and x in the path order of $G[B_i]$. However, it is possible for the vertices of B^*_i to occur between the vertices of $B'_{i,a}$ and x in the path order of $G[B_i]$. Thus there are at most $2k$ vertices of B_i between the vertices in $B'_{i,a}$ and x in $G[B_i]$. Hence,

$$\begin{aligned}|V(P)| &\leq (|B'_{i,a}| - 1) + |B^*_i| + 1\\ &\leq [(2k+1) - 1] + 2k + 1\\ &= 4k + 1\\ &< K.\end{aligned}$$

Hence every component of $G[B_i \setminus M]$ has size at most K. □

The result follows from Proposition 4.44 and Claims 4.49 and 4.50. □

Hence Corollary 4.43 follows from Lemma 4.48 and Lemma D.4 in Appendix D.1.

We now use Corollary 4.43 to prove some algorithmic results. These results are slightly weaker than the non-algorithmic Theorems 4.40 and 4.41. Again, we leave the discussion of the runtimes of these algorithms to Appendices D.2 and D.3.

Corollary 4.51. *There exists an algorithm* BoundedVert *that takes as inputs any graph G with maximum degree Δ and, for fixed $\Delta \geq 3$, returns in time* poly$(|V(G)|)$ *a $\lceil \frac{\Delta+2}{3} \rceil$-vertex colouring of G such that the monochromatic components have size at most* $K = 12\Delta^2 - 36\Delta + 12$.

4.5.2 BoundedVert

Input: A graph G with maximum degree $\Delta \geq 3$.

Output: A $\lceil \frac{\Delta+2}{3} \rceil$-vertex colouring of G such that the monochromatic components have size at most $K = 12\Delta^2 - 36\Delta + 12$.

```
 1: function BoundedVert($G$)
 2:     $h := \lceil \frac{\Delta+2}{3} \rceil$
 3:     $(A, B)$ := DegreePartition($G; 1, \Delta - 2$) for $\ell = 2$.
 4:     $(B_1, \ldots, B_{h-1})$ := DegreePartition($G[B]; 2, 2, 2, ..., 2$) for $\ell = h - 1$.
 5:     for $i = 1, \ldots, h - 1$ do
 6:         $\mathcal{C}_i$ := Components($G; B_i$)
 7:     $\mathcal{C} := \bigcup_{i=1}^{h-1} \mathcal{C}_i$
 8:     $M$ := SmallComponents($G; A; B; \mathcal{C}$) for $d = \Delta$ and $r = \Delta - 2$.
 9:     for $i = 1, \ldots, h - 1$ do
10:         for all $v \in B_i \setminus M$ do
11:             $c(v) := i$
12:     for all $v \in A \cup M$ do
13:         $c(v) := h$
14:     return $c$
```

Let BoundedVert be the algorithm defined in D.2.1.

The subroutine DegreePartition is based on the following theorem due to Lovász [73].

Theorem 4.52 ([73])**.** *Let G be a graph and let $n_1, \dots, n_\ell$ be nonnegative integers such that $n_1 + n_2 + \cdots + n_\ell \geq \Delta(G) - \ell + 1$. Then $V(G)$ can be partitioned into sets $V_1, \dots, V_\ell$ such that $\Delta(G[V_i]) \leq n_i$ for all $1 \leq i \leq \ell$.*

In Corollary 4.51, we have $\Delta \leq 3h - 2$. Hence for G, $n_1 = 1$, $n_2 = \Delta - 2$, and $\ell = 2$, Theorem 4.52 implies that there is a partition (A, B) where $\Delta(G[A]) \leq 1$ and $\Delta(G[B]) \leq \Delta - 2$. Moreover, for $G[B]$, $n_i = 2$ for all $1 \leq i \leq h - 1$, and $\ell = h - 1$, Theorem 4.52 implies that there is a vertex partition $(B_1, \dots, B_{h-1})$ such that $\Delta(G[B_i]) \leq 2$ for each i. The original proof of Theorem 4.52 is algorithmic, which is shown in Appendix D.2. We therefore take DegreePartition to be this algorithm that finds the partition of Theorem 4.52 in polynomial time.

The subroutine Components of BoundedVert returns the set of all vertex sets of the components of $G[B_i]$. We now show that the c returned by BoundedVert is an h-vertex colouring with monochromatic components of size at most K (Claim 4.53).

Claim 4.53. BoundedVert *returns an h-vertex colouring of G with monochromatic components of size at most K.*

Proof: Note that for $d = \Delta$ and $r = \Delta - 2$, we have that the K of Corollary 4.43 is

$$\begin{aligned} K &= (12r + 6)d - (18r + 24) \\ &= [12(\Delta - 2) + 6]\Delta - [18(\Delta - 2) + 24] \\ &= 12\Delta^2 - 36\Delta + 12. \end{aligned}$$

Thus, assuming the hypotheses are satisfied, Corollary 4.43 implies that the set M returned by SmallComponents makes the components of $G[B_i \setminus M]$ and $G[A \cup M]$ have size at most $K = 12\Delta^2 - 36\Delta + 12$. Since c assigns the vertices in $A \cup M$ the same unique colour and a different unique colour for vertices in each B_i, the colouring c returned by BoundedVert is an h-vertex colouring with the desired properties. It therefore remains to show that for $d = \Delta$ and $r = \Delta - 2$, this choice of G, (A, B), and partition $\mathcal{C}$ of B satisfies the hypotheses of Corollary 4.43.

First, note the following properties of the vertex partitions (A, B) of G and $(B_1, \dots, B_{h-1})$ of $G[B]$ due to DegreePartition:

(DP1) $\Delta(G[A]) \leq 1$,

(DP2) $\Delta(G[B]) \leq \Delta - 2$, and

(DP3) $\Delta(G[B_i]) \leq 2$ for each $i \in \{1, \ldots, h-1\}$.

Properties (DP1)-(DP3) clearly follow from the choice of bounds in DegreePartition used to define (A, B) and $(B_1, \ldots, B_{h-1})$. Furthermore, as $G[B_i] \leq 2$ for all $1 \leq i \leq h-1$, each component of $G[B_i]$ must be a path or a cycle. Thus, using Components to partition each B_i into classes $B_{i,j}$ such that each $B_{i,j}$ is the vertex set of a component of $G[B_i]$ will produce a partition $\mathcal{C}_i$ of B_i. The union of all the partitions $\mathcal{C}_i$ is then a partition $\mathcal{C}$ of B satisfying condition (iii) of Corollary 4.43. Hence conditions (i)-(iii) are satisfied by vertex partition (A, B) of G and the partition $\mathcal{C}$ of B. It remains to show that this choice of A, B, and $\mathcal{C}$ also satisfies condition (iv) of Corollary 4.43.

Let $B_{i,j} \in \mathcal{C}$. As $\Delta(G) = \Delta$, clearly each $v \in B_{i,j}$ with $\deg_{G[B_{i,j}]}(v) = 2$ has at most $\Delta - 2$ neighbours in A. Thus for each $B_{i,j} \in \mathcal{C}$ and each $v \in B_{i,j}$ with $\deg_{G[B_{i,j}]}(v) = 2$, there are at most $\Delta - 2$ components of $G[A]$ that contain neighbours of v. Hence condition (iv) is satisfied for $r = \Delta - 2$ and so $(G; A; B; \mathcal{C})$ satisfies the hypotheses of Corollary 4.43 when $d = \Delta$ and $r = \Delta - 2$. $\square$

The proof of Corollary 4.51 follows from Claim 4.53 and Claim D.10 in Appendix D.2.

We now continue with an algorithmic version of Theorem 4.41. Note that for a graph G and set $S \subseteq E(G)$, the graph induced by S, denoted $G[S]$, is the subgraph of G with edge set S and whose vertex set consists of all the endpoints of the edges in S.

Corollary 4.54. *There exists an algorithm* BoundedEdge *that takes as inputs any loopless graph G with maximum degree Δ and, for fixed $\Delta \geq 3$, returns in time* poly$(|E(G)|)$ *a $\left\lceil \frac{\Delta+1}{2} \right\rceil$-edge colouring of G such that the monochromatic components have size at most $K = 60\Delta - 60$.*

As with Corollary 4.51, we will use SmallComponents to prove Corollary 4.54. In particular, we will use SmallComponents on the line graph $L(G)$ of G (after finding the necessary partitions) to get our desired colouring.

Let BoundedEdge be the algorithm defined in D.3.1.

The subroutines TwoPartition and TwoFactor are based on the following lemmas from [9].

Lemma 4.55 ([9]). *Let ℓ be an integer and G be a loopless graph with $\Delta(G) \leq 2\ell$. Then there exists an edge partition $(E_1, \ldots, E_\ell)$ such that $\Delta(G[E_i]) \leq 2$ for all i.*

4.5.3 BoundedEdge

Input: A loopless graph G with maximum degree $\Delta \geq 3$.

Output: A $\lceil \frac{\Delta+1}{2} \rceil$-edge colouring of G such that the monochromatic components have size at most $K = 60\Delta - 60$.

```
function BoundedEdge($G$)
    $h := \lceil \frac{\Delta+1}{2} \rceil$
    $(A, B) :=$ TwoPartition($G$)
    $(B_1, \ldots, B_{h-1}) :=$ TwoFactor($G[B], h-1$)
    $H :=$ LineGraph($G$)
    for $i = 1, \ldots, h-1$ do
        $\mathcal{C}_i :=$ Components($H; B_i$)
    $\mathcal{C} := \bigcup_{i=1}^{h-1} \mathcal{C}_i$
    $M :=$ SmallComponents($H; A; B; \mathcal{C}$) for $d = 4h - 4$ and $r = 2$.
    for $i = 1, \ldots, h-1$ do
        for all $e \in B_i \setminus M$ do
            $c(e) := i$
    for all $e \in A \cup M$ do
        $c(e) := h$
    return $c$
```

Lemma 4.56 ([9])**.** *Every loopless graph G has a set $A \subseteq E(G)$ such that $\Delta(G - A) < \Delta(G)$ and each component of $G(A)$ is a path containing at most two edges.*

Lemmas 4.55 and 4.56 are both reformulations of a theorem due to Petersen [82], which is proved via Euler tours. As Euler tours can be found efficiently, the original proofs of Lemmas 4.55 and 4.56 are algorithmic. The details of these algorithms (TwoFactor for Lemma 4.55 and TwoPartition for Lemma 4.56) can be found in Appendix D.3.

Finally, the subroutine LineGraph returns the line graph $L(G)$ of G. We are now ready to show that the c returned by BoundedEdge is an h-edge colouring with monochromatic components of size at most K (Claim 4.57).

Claim 4.57. BoundedEdge *returns an h-edge colouring of G with monochromatic components of size at most K.*

Proof: Note that for $d = 4h - 4$ and $r = 2$, we have that the K of Corollary 4.43 is

$$\begin{aligned} K &= (12r + 6)d - (18r + 24) \\ &= [12(2) + 6](4h - 4) - [18(2) + 24] \\ &= 120h - 120 - 60 \\ &= 120 \left\lceil \frac{\Delta + 1}{2} \right\rceil - 180. \end{aligned}$$

For even Δ, this K is $60\Delta - 60$ and for odd Δ it is $60\Delta - 120$. Hence $K \leq 60\Delta - 60$. Thus, assuming the hypotheses are satisfied, Corollary 4.43 implies that the set M returned by SmallComponents makes the components of $H[B_i \setminus M]$ and $H[A \cup M]$ have size at most $K = 60\Delta - 60$.

Note that for each $S \subseteq E(G)$, $G[S]$ is connected if and only if $H[S]$ is connected. Hence for each $i \in \{1, \ldots, h-1\}$, every component of $G[B_i \setminus M]$ is a component of $H[B_{i,j} \setminus M]$ for some $B_{i,j} \in \mathcal{C}$. Also, every component of $H[A \cup M]$ is a component of $G[A \cup M]$. Therefore the components of $G[B_i \setminus M]$ and $G[A \cup M]$ contain at most $60\Delta - 60$ edges for all $i \in \{1, \ldots, h-1\}$.

Since c assigns the vertices in $A \cup M$ the same unique colour and a different unique colour for vertices in each B_i, the colouring c returned by BoundedEdge is an h-edge colouring with the desired properties. It therefore remains to show that for $d = 4h - 4$ and $r = 2$, this choice of H, (A, B), and partition $\mathcal{C}$ of B satisfies the hypotheses of Corollary 4.43.

First, note the following properties of edge partition (A, B) and $(B_1, \ldots, B_{h-1})$ due to TwoPartition, TwoFactor, and h:

(EP1) $\Delta(G-A) < \Delta$,

(EP2) $\Delta(G[B]) \leq 2(h-1)$,

(EP3) each component of $G[A]$ is a path with at most two edges, and

(EP4) $\Delta(G[B_i]) \leq 2$ for each $i \in \{1, \ldots, h-1\}$.

Properties (EP1) and (EP3) hold by the conditions on A returned by TwoPartition. Property (EP4) holds by the condition on the B_i returned by TwoFactor. We show (EP2) holds for $G[B]$ as follows. Note that every vertex in G has degree at most $\Delta - 1$ in $G - A$ (by (EP1)). As (A, B) is an edge partition of G, this implies that each endpoint of an edge $e \in B$ has degree at most $\Delta - 1$ in $G[B] = G - A$. Thus $\Delta(G[B]) \leq \Delta - 1$ and $2(h-1) = 2\left(\lceil \frac{\Delta+1}{2} \rceil - 1\right) \geq \Delta - 1$. Hence (EP2) holds. We now show that these properties imply the hypotheses of Corollary 4.43.

By (EP3), each edge in A is in a component with at most one other edge in $G[A]$. Thus $\Delta(H[A]) \leq 1$ and so condition (i) of Corollary 4.43 holds. By (EP2), each vertex $v \in V(G)$ is incident with at most $2(h-1)$ edges in B. Thus each edge in B containing v is adjacent in H to at most $2(h-1) - 1 = 2h - 3$ other edges of G that contain v as an endpoint. Each edge in B has exactly two endpoints (recall G is loopless), so $\Delta(H[B]) \leq 2(2h-3) = 4h - 6 = d - 2$. Hence condition (ii) of Corollary 4.43 also holds.

By (EP4), each component of $G[B_i]$ is a cycle or path in G and so its corresponding component in $H = L(G)$ is also a cycle or path. Thus, using Components to partition each B_i into classes $B_{i,j}$ such that each $B_{i,j}$ is the vertex set of a component of $H[B_i]$ will produce a partition $\mathcal{C}_i$ of B_i. The union of all the partitions $\mathcal{C}_i$ is then a partition $\mathcal{C}$ of B satisfying condition (iii) of Corollary 4.43.

For condition (iv) of Corollary 4.43, note that H is a line graph and so H is $K_{1,3}$-free. Hence each $e \in B$ is adjacent in H to vertices of at most two components of $H[A]$. Thus for each $B_{i,j} \in \mathcal{C}$ and each $e \in B_{i,j}$ with $\deg_{H[B_{i,j}]}(e) = 2$, there are at most 2 components of $H[A]$ that contain neighbours of e. Therefore H, A, B, and $\mathcal{C}$ satisfy all of the conditions of Corollary 4.43 for $d = 4h - 4$ and $r = 2$. □

The proof of Corollary 4.54 follows from Claim 4.57 and Claim D.13 in Appendix D.3.

Chapter 5

Independent Transversals in Weighted Graphs

In this chapter, we consider the problem of finding an IT in vertex-weighted graphs. In particular, we will discuss finding a PIT with large (but not necessarily maximum) weight τ, where the value of τ depends on the weight function on the graph. The problem of finding ITs in vertex-weighted graphs was first studied by Aharoni, Berger, and Ziv in [3]. Their work generalised Theorem 2.4 to vertex-weighted graphs, in a way similar to that in which the theory of matchings in weighted graphs generalises Hall's theorem. Their main theorem is stated as Theorem 5.2, which we discuss in Section 5.1. As with Theorem 2.4, the proof of Theorem 5.2 is not algorithmic. We aim to generalise FindITorBD from Theorem 3.1 to vertex-weighted graphs in order to prove an algorithmic version of Theorem 5.2. As mentioned in Chapter 1, the new results in this chapter are joint work with Penny Haxell.

In Section 5.2, we introduce our new generalised algorithm, called FindWeightPIT, for efficiently finding PITs of weight at least $\tau_w^{r,\epsilon}$, where $\tau_w^{r,\epsilon}$ is a parameter of vertex-weighted graphs that is defined in Section 5.2. FindWeightPIT uses the LP formulation of the problem studied by Aharoni, Berger, and Ziv in [3]. This new formulation, which is different but equivalent to that in [3], is presented in Section 5.2 and was suggested by David Harris [41]. We will see some applications of FindWeightPIT (via the randomised algorithm FindWeightIT) in Chapter 6.

This chapter is organised as follows. In Section 5.1, we discuss the work of Aharoni, Berger, and Ziv from [3]. We state Theorem 5.2 as it was presented in [3]. In Section 5.2, we restate Theorem 5.2 using the LP formulation and introduce our algorithm FindWeightPIT. We prove the correctness of FindWeightPIT in Section 5.3.

5.1 A Generalisation of Theorem 2.4

In this section, we discuss the work of Aharoni, Berger, and Ziv in [3]. We begin with some notation. For any set S, subset $U \subseteq S$, and function $F : S \to \mathbb{R}$, we write $F(U) = \sum_{x \in U} F(x)$ and $|F| = F(S)$.

Let G be a graph with vertex partition $(V_1, \ldots, V_m)$ and w a weight function $w : V(G) \to \mathbb{R}^+$. We have the following important definition.

Definition 5.1. A pair of non-negative functions $g : \{V_1, \ldots, V_m\} \to \mathbb{R}^+$ and $f : V(G) \to \mathbb{R}^+$ is *w-dominating* if for every $v \in V(G)$,

$$g(\text{Vclass}(v)) + f(N(v)) \geq w(v).$$

We write $|(g, f)| = |g| + \frac{|f|}{2}$.

Let $\tau_w = \min\{|(g, f)| : (g, f) \text{ is } w\text{-dominating}\}$ and let ν_w be the maximum total weight of a PIT in G with respect to $(V_1, \ldots, V_m)$. For every $i \in \{1, \ldots, m\}$, let $t_i = \max\{w(v) : \text{Vclass}(v) = V_i\}$. Aharoni, Berger, and Ziv proved the following.

Theorem 5.2 ([3]). *For G a graph with vertex partition $(V_1, \ldots, V_m)$ and w a weight function $w : V(G) \to \mathbb{R}^+$, we have $\tau_w \leq \nu_w$. Furthermore, it is possible to find a dominating pair of functions (g, f) of weight at most ν_w such that for all $v \in V(G)$, $f(v) + g(V_i) \leq t_i$, where $V_i = \text{Vclass}(v)$. If all weights $w(v)$ are integral, then g and f can also be assumed to be integral.*

We remark that the existence of a dominating pair of functions satisfying the condition $f(v) + g(V_i) \leq t_i$ for all $v \in V(G)$ is unnecessary for proving $\tau_w \leq \nu_w$. Thus an equivalent condition will not be present in the LP formulation of τ_w presented in Section 5.2. However, the condition is necessary for Aharoni, Berger, and Ziv's proof from [3] that Theorem 2.4 follows from Theorem 5.2. We present this proof in the following claim. Note that the support of a function $f : X \to \{0, 1\}$ is the subset $\{x \in X : f(x) = 1\}$.

Claim 5.3 ([3]). *Theorem 2.4 follows from Theorem 5.2.*

Proof: Let G be a graph with vertex partition $(V_1, \ldots, V_m)$. Define weight function $w : V(G) \to \mathbb{R}^+$ by $w(v) = 1$ for all $v \in V(G)$.

Suppose there does not exist an IT in G with respect to $(V_1, \ldots, V_m)$. Then $\nu_w < m$. By Theorem 5.2, there exists an integer-valued dominating pair of functions (g, f) such that $|(g,f)| \leq \nu_w < m$. Furthermore, we claim we can find such a dominating pair with g and f both having codomain $\{0,1\}$. This is because $t_i = \max\{w(v) : \text{Vclass}(v) = V_i\} = 1$ for all $1 \leq i \leq m$ in Theorem 5.2 implies that $f(v) + g(\text{Vclass}(v)) \leq 1$ for all $v \in V(G)$. As g and f are both integer-valued functions into the non-negative real numbers, we have that the codomain of both f and g is $\{0,1\}$.

Let $\mathcal{B} = \{V_i : g(V_i) = 0\}$. As (g,f) is w-dominating, $g(\text{Vclass}(v)) + f(N(v)) \geq w(v) = 1$ for all $v \in \bigcup\limits_{V_i \in \mathcal{B}} V_i$. Hence $f(N(v)) \geq 1$ for all $v \in \bigcup\limits_{V_i \in \mathcal{B}} V_i$. This implies that every vertex v where $\text{Vclass}(v) \in \mathcal{B}$ has at least one neighbour u such that $f(u) = 1$. Thus the support of f dominates $G_{\mathcal{B}}$. Furthermore, since for all $v \in V(G)$ we know that $f(v) + g(\text{Vclass}(v)) \leq 1$, we have that $g(\text{Vclass}(v)) = 0$ for all v in the support of f. Hence the support of f is contained in $V(G_{\mathcal{B}})$.

As $|(g,f)| < m$, we have that $|g| + \frac{|f|}{2} \leq m - 1$. Hence

$$|f| \leq 2(m - 1 - |g|) = 2[m - 1 - (m - |\mathcal{B}|)] = 2(|\mathcal{B}| - 1).$$

Thus there exists a set $\mathcal{B}$ of vertex classes that is dominated by a set of size at most $2(|\mathcal{B}| - 1)$ (namely the vertices in the support of f). This proves Theorem 2.4. $\square$

5.2 The Algorithm FindWeightPIT

In this section, we present an algorithm, called FindWeightPIT, that proves an algorithmic version of Theorem 5.2, which we state in Theorem 5.5. Before doing so, we consider a different definition of τ_w, which is formulated in terms of the dual LP of the formulation presented in Section 5.1.

Let G be a graph, $(V_1, \ldots, V_m)$ a vertex partition of G, and w a weight function $w : V(G) \to \mathbb{R}$. Let $\mathcal{P}_w$ be the following LP and let τ_w be the largest objective function value to $\mathcal{P}_w$.

$$\begin{array}{rrcll} \max & \sum_{v \in V(G)} w(v)\gamma_v & & & \\ \text{subject to} & \sum_{u \in N(v)} \gamma_u & \leq & \frac{1}{2} & \forall v \in V(G) \\ & \sum_{v \in V_i} \gamma_v & \leq & 1 & \forall i \in \{1, \ldots, m\} \\ & 0 \leq \gamma_v & \leq & 1 & \forall v \in v(G). \end{array}$$

We claim that for $w : V(G) \to \mathbb{R}^+$ the optimal value of $\mathcal{P}_w$ is the same as the value of τ_w in Section 5.1. Recall that in Section 5.1, τ_w is defined to be the minimum value of $\sum_{i=1}^{m} g(V_i) + \sum_{v \in V(G)} \frac{f(v)}{2}$ amongst the w-dominating pairs (g, f). Also, recall that (g, f) is w-dominating if for all $v \in V(G)$, $g(\text{Vclass}(v)) + f(N(v)) \geq w(v)$. Thus by duality, this value of τ_w is also the maximum value of $\sum_{v \in V(G)} w(v)\gamma_v$ for some vector γ that is subject to the constraints $\sum_{v \in V_i} \gamma_v \leq 1$ for each $i \in \{1, \ldots, m\}$ and $\sum_{u \in N(v)} \gamma_u \leq \frac{1}{2}$ for each $v \in V(G)$. Hence the new definition of τ_w is simply the optimal solution to the dual LP formulation of what was presented in Section 5.1.

Using this new formulation of τ_w, Theorem 5.2 yields the following.

Theorem 5.4 ([3]). *For any graph G, vertex partition $(V_1, \ldots, V_m)$, and weight function $w : V(G) \to \mathbb{R}^+$, there exists a PIT of G of weight at least τ_w.*

As mentioned before, the proof of Theorem 5.2 (and therefore Theorem 5.4) is not fully algorithmic. To make the result fully algorithmic for r-claw-free graphs, we require a slight modification to the choice of τ_w, which we call $\tau_w^{r,\epsilon}$. Let $r \in \mathbb{N}$ and $0 < \epsilon < \frac{1}{r}$. Let G be a graph, $(V_1, \ldots, V_m)$ a vertex partition such that G is r-claw-free with respect to $(V_1, \ldots, V_m)$, and w a weight function $w : V(G) \to \mathbb{R}$. Let $\mathcal{P}_w^{r,\epsilon}$ be the following LP and let $\tau_w^{r,\epsilon}$ be the largest objective function value to $\mathcal{P}_w^{r,\epsilon}$.

$$\begin{array}{rrrcll}
\max & \sum_{v \in V(G)} w(v)\gamma_v & & & & \\
\text{subject to} & \sum_{u \in N(v)} & \gamma_u & \leq & \frac{1-r\epsilon}{2+\epsilon} & \forall v \in V(G) \\
& \sum_{v \in V_i} & \gamma_v & \leq & 1 & \forall i \in \{1, \ldots, m\} \\
& 0 \leq & \gamma_v & \leq & 1 & \forall v \in v(G).
\end{array}$$

For a subset $U \subseteq V(G)$, let $w(U) = \sum_{v \in U} w(v)$. We define $|w| = \sum_{v \in V(G)} |w(v)|$, and so $-|w| \leq w(V(G)) \leq |w|$. Using these definitions and $\tau_w^{r,\epsilon}$, we obtain the following algorithmic result.

Theorem 5.5. *There exists an algorithm* FindWeightPIT *that takes as inputs any graph G, vertex partition $(V_1, \ldots, V_m)$ such that G is r-claw-free with respect to $(V_1, \ldots, V_m)$, and weight function $w : V(G) \to \mathbb{Z}$ and, for fixed $r \geq 1$ and $0 < \epsilon < \frac{1}{r}$, finds, in time* $\text{poly}(|V(G)|, |w|)$*, a PIT M in G of weight at least $\tau_w^{r,\epsilon}$.*

Note that FindWeightPIT is a pseudo-polynomial time algorithm as it takes roughly $\log_2(|w|)$ bits to write $|w|$. It is possible to remove this dependence on $|w|$, however doing so weakens the conclusion slightly. In particular, instead of finding a PIT of weight at least $\tau_w^{r,\epsilon}$, we find a PIT of weight at least $(1-\eta)\tau_w^{r,\epsilon}$ in time $\text{poly}\left(|V(G)|, \frac{1}{\eta}\right)$ for some input parameter $\eta > 0$ (and fixed $r \geq 1$ and $0 < \epsilon < \frac{1}{r}$). We will see this in Section 6.1 where we discuss some algorithmic consequences of FindWeightPIT.

From the definition of $\tau_w^{r,\epsilon}$, it is clear that Theorem 5.5 is an algorithmic version of Theorem 5.4 for r-claw-free graphs and integer weight functions w. In Chapter 6, we will introduce a randomised algorithm that uses FindWeightPIT to give another algorithmic version of Theorem 5.4. This randomised algorithm will be useful for proving some fractional strong colouring results, which are also discussed in Chapter 6.

Before presenting the algorithm FindITorBD, we give some intuition for the algorithm. FindITorBD first restricts itself to the induced subgraph G' of the input G whose vertex set consists only of the vertices v with $w(v) > 0$ and $w(v) \geq w(u)$ for all $u \in \text{Vclass}(v)$ (i.e. the vertices of positive weight that have maximum weight within their vertex class). We then apply FindITorBD to G' and the vertex partition $\mathcal{W}$ formed by restricting the input partition to the vertices in G'. If this returns an IT, then we are done as this IT has weight at least $\tau_w^{r,\epsilon}$ (Lemma 5.11). Otherwise, we will use the vertex set D returned by FindITorBD to decrease the weights of the vertices in G to get a new weight function w'. We then recursively apply FindWeightPIT to the original input graph and partition, but with the new weight function w'. After some modifications to the resulting set M, it can be shown that we obtain a PIT (Lemma 5.7) of weight at least $\tau_w^{r,\epsilon}$ (Lemma 5.11).

We now present the algorithm FindWeightPIT. We hold off on proving that FindWeightPIT satisfies Theorem 5.5 until Section 5.3. Let FindWeightPIT be the algorithm defined in 5.2.1.

5.2.1 FindWeightPIT

Input: A graph G, vertex partition $(V_1, \dots, V_m)$, and weight function $w : V(G) \to \mathbb{Z}$ with parameters r and ϵ.

Output: A PIT M of weight at least $\tau_w^{r,\epsilon}$.

```
1:  function FindWeightPIT($G; w; V_1, \dots, V_m$)
2:      $\mathcal{V} := \{V_i : \max\{w(v) : \mathrm{Vclass}(v) = V_i\} > 0\}$
3:      for all $V_i \in \mathcal{V}$ do
4:          $W_i := \{v \in V_i : w(v) = \max\{w(u) : u \in V_i\}\}$
5:      $\mathcal{W} := \{W_i : V_i \in \mathcal{V}\}$
6:      $G' := G\left[\bigcup_{V_i \in \mathcal{V}} W_i\right]$
7:      Apply FindITorBD($G'; \mathcal{W}$) with parameters $r$ and $\epsilon$.
8:      if FindITorBD($G'; \mathcal{W}$) returns an IT $M'$ then
9:          $M := M'$
10:     else FindITorBD($G'; \mathcal{W}$) returns $\mathcal{B} \subseteq \mathcal{W}$, $D \subseteq V(G')$, and $L = \mathrm{Leaf}(K) \subseteq D$.
11:         for all $v \in V(G)$ do
12:             $w'(v) := \max\{0, w(v) - |N(v) \cap D|\}$
13:         $M :=$ FindWeightPIT($G; w'; V_1, \dots, V_m$) with parameters $r$ and $\epsilon$.
14:         $Y := \{v \in L : \mathrm{Vclass}(v) \in \mathcal{B}, |N(v) \cap D| = 1\}$
15:         while there is some $v \in Y \setminus M$ with $N(v) \cap M = \emptyset$ do
16:             Choose any such $v \in Y \setminus M$.
17:             $M := (M \cup \{v\}) \setminus \{b \in M : \mathrm{Vclass}(b) = \mathrm{Vclass}(v)\}$
18:     return $M$
```

In the next section, we will analyse FindWeightPIT to show that it satisfies Theorem 5.5.

5.3 Analysing FindWeightPIT

In this section, we analyse FindWeightPIT and show that it returns a PIT of weight at least $\tau_w^{r,\epsilon}$ in time poly$(|V(G)|, |w|)$ for fixed $r \geq 1$ and $0 < \epsilon < \frac{1}{r}$. We begin our analysis with a few definitions. For each $1 \leq i \leq m$, let $t_i^w = \max\{w(v) : v \in V_i\}$. Define the *index* $\text{Ind}(G; w; V_1, \dots, V_m)$ to be $\text{Ind}(G; w; V_1, \dots, V_m) = \sum\limits_{i=1}^{m} \max\{0, t_i^w\}$.

Proposition 5.6. *We have*

$$\text{Ind}(G; w'; V_1, \dots, V_m) < \text{Ind}(G; w; V_1, \dots, V_m).$$

Proof: Recall that w' is only defined when $|w| > 0$ and FindITorBD$(G', \mathcal{W})$ returns a set $\mathcal{B} \subseteq \mathcal{W}$ of vertex classes, set $D \subseteq V(G')$ of vertices where D dominates $G'_{\mathcal{B}}$, and the PIT $L = \text{Leaf}(K)$ for the constellation K for $\mathcal{B}_0 \supseteq \mathcal{B}$ with $V(K) \subseteq D$ (see FindITorBD in Section 3.5). Furthermore, from the choice of vertices in the W_i, we know that for all $v \in V(G')$, $w(v) = t_i^w > 0$, where $V_i = \text{Vclass}(v)$.

As D dominates $G'_{\mathcal{B}}$, for each vertex $v \in W_i$ such that $W_i \in \mathcal{B}$, we have $N(v) \cap D \neq \emptyset$. Hence $w'(v) \leq w(v) - 1$ for all such v. Since $w(v) = t_i^w$, this implies that

$$t_i^{w'} = \max\{w'(u) : u \in V_i\} \leq \max\{w(u) : u \in V_i\} - 1 < t_i^w.$$

Thus $t_i^{w'} < t_i^w$ for all V_i such that $W_i \in \mathcal{B}$. As $\mathcal{B} \neq \emptyset$, this implies

$$\text{Ind}(G; w'; V_1, \dots, V_m) = \sum_{i=1}^{m} t_i^{w'} < \sum_{i=1}^{m} t_i^w = \text{Ind}(G; w; V_1, \dots, V_m). \qquad \square$$

Lemma 5.7. *The set M returned by* FindWeightPIT *is a PIT of G with respect to $(V_1, \dots, V_m)$.*

Proof: If FindITorBD$(G'; \mathcal{W})$ returns an IT, then M is defined to be the IT returned by FindITorBD. Since G' is an induced subgraph of G and the W_i are subsets of the vertex classes V_i, this set M is a PIT of G with respect to $(V_1, \dots, V_m)$.

Suppose FindITorBD$(G'; \mathcal{W})$ returns a set $\mathcal{B}$ of vertex classes, a set D of vertices, and a set L of vertices where $L = \text{Leaf}(K)$ for the constellation K for $\mathcal{B}_0 \supseteq \mathcal{B}$ with $V(K) \in D$. Then FindWeightPIT is recursively applied to obtain a PIT M of G with respect to $(V_1, \dots, V_m)$. By Proposition 5.6 and induction on the index, we know that M is a PIT.

The set M remains a partial transversal throughout the while loop of FindWeightPIT because the loop adds a vertex $v \in Y \setminus M$ to M and removes the vertex b of M with $\text{Vclass}(b) = \text{Vclass}(v)$ (if it exists). To check that M is independent, note that each time M is modified in the loop, the new vertex v satisfies $N(v) \cap M = \emptyset$. Thus M is a PIT of G with respect to $(V_1, \ldots, V_m)$. □

By Lemma 5.7, FindWeightPIT always returns a PIT. Before analysing the weight of this PIT, we need a few more propositions.

Proposition 5.8. *The value of $w'(M)$ does not decrease during any iteration of the while loop in* FindWeightPIT*.*

Proof: Let v be the vertex chosen in an iteration of the while loop. By the definition of Y, we know that v has exactly one neighbour in D and $v \in W_i$ for $V_i = \text{Vclass}(v)$. Hence $w'(v) = w(v) - 1 = t_i^w - 1$. As $W_i \in \mathcal{W}$, we have $t_i^w > 0$ and so $t_i^w - 1 \geq 0$.

Let $b \in M$ be the vertex of M such that $\text{Vclass}(b) = \text{Vclass}(v)$ (if it exists). If b does not exist, then clearly $w'(M)$ does not decrease in this iteration because $w'(v) \geq 0$. If b exists, then $\text{Vclass}(b) = \text{Vclass}(v) = V_i$.

Suppose $b \in W_i$. By the definition of Y, we have $W_i \in \mathcal{B}$, and so $b \in V(G'_{\mathcal{B}})$. Since D dominates $G'_{\mathcal{B}}$, this means that b has at least one neighbour in D, which implies that $w'(b) \leq w(b) - 1 \leq t_i^w - 1$.

Now suppose $b \notin W_i$. Then $b \in V_i \setminus W_i$ and so $w(b) < t_i^w$. Hence $w'(b) \leq w(b) \leq t_i^w - 1$.

In either case, adding vertex v to M increases $w'(M)$ by $w'(v) = t_i^w - 1$, while removing b decreases $w'(M)$ by $w'(b) \leq t_i^w - 1$. Hence the statement holds. □

Proposition 5.9. *Suppose* FindITorBD$(G'; \mathcal{W})$ *returns $\mathcal{B}$, D, and L instead of an IT M. Then the output M of* FindWeightPIT *satisfies*

$$\sum_{v \in M} |N(v) \cap D| > (1 - r\epsilon)(|\mathcal{B}| - 1).$$

Proof: Let $Y = \{v \in L : \text{Vclass}(v) \in \mathcal{B}, |N(v) \cap D| = 1\}$ (as in FindWeightPIT). Because of the termination condition of the while loop in FindWeightPIT, we know that that each vertex $v \in Y \setminus M$ has an edge to some vertex $u \in M$. Since $Y \subseteq \text{Leaf}(K)$, we know that Y is independent and so $u \in M \setminus Y$. This implies that there are at least $|Y \setminus M|$ edges from $M \setminus Y$ to $Y \setminus M$. Since $Y \subseteq D$, this in turn shows that $\sum_{v \in M \setminus Y} |N(v) \cap D| \geq |Y \setminus M|$.

Next, consider some vertex $v \in M \cap Y$. Since D dominates $G'_{\mathcal{B}}$ and $Y \subseteq V(G'_{\mathcal{B}})$, there is at least one vertex in $N(v) \cap D$, and hence $|N(v) \cap D| \geq 1$. This implies that $\sum_{v \in M \cap Y} |N(v) \cap D| \geq \sum_{v \in M \cap Y} 1 = |M \cap Y|$.

We therefore have that

$$\sum_{v \in M} |N(v) \cap D| = \sum_{v \in M \setminus Y} |N(v) \cap D| + \sum_{v \in M \cap Y} |N(v) \cap D| \geq |Y \setminus M| + |Y \cap M| = |Y|.$$

To complete the proof, it suffices to show that $|Y| > (1 - r\epsilon)(|\mathcal{B}| - 1)$. We have

$$|Y| \geq |\operatorname{Leaf}(K)| - |\{v \in \operatorname{Leaf}(K) : W_{i(v)} \notin \mathcal{B} \text{ or } |N(v) \cap D| \neq 1\}|,$$

where $W_{i(v)}$ denotes the vertex class $W_i \in \mathcal{W}$ such that $\operatorname{Vclass}(v) = V_i$.

By Theorem 3.1, K is a constellation for some $\mathcal{B}_0 \supseteq \mathcal{B}$ and thus $|\operatorname{Leaf}(K)| = |\mathcal{B}_0| - 1$. Since $\operatorname{Leaf}(K)$ is a PIT, the set $\{v \in \operatorname{Leaf}(K) : W_{i(v)} \notin \mathcal{B}\}$ has size at most $|\mathcal{B}_0| - |\mathcal{B}|$.

Each vertex in $\operatorname{Leaf}(K)$ has exactly one neighbour in K. Thus, if $y \in \operatorname{Leaf}(K)$ has more than one neighbour in D, then it has a neighbour in $J = D \setminus V(K)$. (Since D dominates $G'_{\mathcal{B}}$, it must have at least one neighbour in D.) Theorem 3.1 ensures that $|J| < \epsilon(|\mathcal{B}| - 1)$. Since G is r-claw-free, there are fewer than $r\epsilon(|\mathcal{B}| - 1)$ edges from J to $\operatorname{Leaf}(K)$. Thus, there are fewer than $r\epsilon(|\mathcal{B}| - 1)$ vertices $v \in \operatorname{Leaf}(K)$ with $|N(v) \cap D| \neq 1$.

Hence,

$$|Y| > (|\mathcal{B}_0| - 1) - (|\mathcal{B}_0| - |\mathcal{B}|) - r\epsilon(|\mathcal{B}| - 1) = (1 - r\epsilon)(|\mathcal{B}| - 1). \qquad \square$$

Proposition 5.10. *Suppose* FindITorBD$(G'; \mathcal{W})$ *returns* $\mathcal{B}$, D, *and* L *instead of an IT* M. *Let* γ *be an optimal solution in* $\mathcal{P}_w^{r,\epsilon}$ *and let* w' *be as defined in* FindWeightPIT. *Then*

$$\tau_{w'}^{r,\epsilon} \geq \left[\sum_{v \in V(G)} w(v)\gamma_v\right] - (1 - r\epsilon)(|\mathcal{B}| - 1).$$

Proof: Recall that $\sum_{u \in N(v)} \gamma_u \leq \frac{1-r\epsilon}{2+\epsilon}$ for all $v \in V(G)$ and $|D| < (2+\epsilon)(|\mathcal{B}|-1)$. Hence,

$$\begin{aligned}
\tau_{w'}^{r,\epsilon} &\geq \sum_{v \in V(G)} w'(v)\gamma_v \\
&\geq \sum_{v \in V(G)} w(v)\gamma_v - \sum_{v \in V(G)} |N(v) \cap D|\gamma_v \\
&= \sum_{v \in V(G)} w(v)\gamma_v - \sum_{v \in D} \sum_{u \in N(v)} \gamma_u \\
&\geq \sum_{v \in V(G)} w(v)\gamma_v - \sum_{v \in D} \frac{1-r\epsilon}{2+\epsilon} \\
&= \left[\sum_{v \in V(G)} w(v)\gamma_v\right] - \frac{(1-r\epsilon)|D|}{2+\epsilon} \\
&\geq \left[\sum_{v \in V(G)} w(v)\gamma_v\right] - (1-r\epsilon)(|\mathcal{B}|-1). \qquad \square
\end{aligned}$$

Lemma 5.11. *Let M be the PIT returned by* FindWeightPIT$(G; w; V_1, \ldots, V_m)$. *Then* $w(M) \geq \tau_w^{r,\epsilon}$.

Proof: We prove the statement by induction on $\mathrm{Ind}(G; w; V_1, \ldots, V_m)$.

For $\mathrm{Ind}(G; w; V_1, \ldots, V_m) = 0$, FindWeightPIT returns $M = \emptyset$ as its PIT. Hence $w(M) = 0$. However, $\mathrm{Ind}(G; w; V_1, \ldots, V_m) = 0$ implies that $w(v) \leq 0$ for all $v \in V(G)$ and so $\tau_w^{r,\epsilon} = 0$ for every solution γ. Thus $w(M) = \tau_w^{r,\epsilon} = 0$.

For the inductive step, suppose $\mathrm{Ind}(G; w; V_1, \ldots, V_m) > 0$. We have $W_i = \{v \in V_i : w(v) = t_i^w\}$ for all $V_i \in \mathcal{V}$, $\mathcal{W} := \{W_i : V_i \in \mathcal{V}\}$, and $G' = G\left[\bigcup_{W_i \in \mathcal{W}} W_i\right]$. FindWeightPIT runs FindITorBD on inputs G' and $\mathcal{W}$, which results in two cases.

Case 1. FindITorBD returns an IT M'.

For any optimal solution γ in $\mathcal{P}_w^{r,\epsilon}$, the constraint $\sum_{v \in V_i} \gamma_v \leq 1$ implies that

$$\sum_{v \in V_i} w(v)\gamma_v \leq \sum_{v \in V_i} t_i^w \gamma_v \leq t_i^w.$$

Hence

$$\tau_w^{r,\epsilon} = \sum_{v \in V(G)} w(v)\gamma_v = \sum_{i=1}^{m} \sum_{v \in V_i} w(v)\gamma_v \leq \sum_{i=1}^{m} t_i^w = w(M').$$

Thus M' is a PIT of G of weight at least $\tau_w^{r,\epsilon}$. As FindWeightPIT$(G; w; V_1, \ldots, V_m)$ returns $M = M'$ in this case, FindWeightPIT returns a PIT M with $w(M) \geq \tau_w^{r,\epsilon}$.

Case 2. FindITorBD returns $\mathcal{B}$, D, and L.

Then D dominates $G'_{\mathcal{B}}$ and D contains the vertices of some constellation K for $\mathcal{B}_0$, where $\mathcal{B} \subseteq \mathcal{B}_0 \subseteq \mathcal{W}$.

Let w' be the weight function as defined in FindWeightPIT. Then by Lemma 5.7, the recursive call FindWeightPIT$(G; w'; V_1, \ldots, V_m)$ returns a PIT M and the final set M is also a PIT. By Proposition 5.6, Ind$(G; w', V_1, \ldots, V_m) <$ Ind$(G; w; V_1, \ldots, V_m)$. Hence by the induction hypothesis, $w'(M) \geq \tau_{w'}^{r,\epsilon}$. By Proposition 5.8, the value of $w'(M)$ does not decrease during the while loop in FindWeightPIT. Thus the final output M also satisfies $w'(M) \geq \tau_{w'}^{r,\epsilon}$.

By the definition of w' and Proposition 5.9, we have

$$w(M) - w'(M) = \sum_{v \in M} [w(v) - w'(v)] = \sum_{v \in M} |N(v) \cap D| > (1 - r\epsilon)(|\mathcal{B}| - 1).$$

Also, by Proposition 5.10, $\tau_{w'}^{r,\epsilon} \geq \left[\sum_{v \in V(G)} w(v)\gamma_v \right] - (1 - r\epsilon)(|\mathcal{B}| - 1)$ for any optimal solution γ in $\mathcal{P}_w^{r,\epsilon}$. Hence,

$$\begin{aligned} w(M) &> w'(M) + (1 - r\epsilon)(|\mathcal{B}| - 1) \\ &\geq \tau_{w'}^{r,\epsilon} + (1 - r\epsilon)(|\mathcal{B}| - 1) \\ &\geq \left[\left[\sum_{v \in V(G)} w(v)\gamma_v \right] - (1 - r\epsilon)(|\mathcal{B}| - 1) \right] + (1 - r\epsilon)(|\mathcal{B}| - 1) \\ &= \sum_{v \in V(G)} w(v)\gamma_v \\ &= \tau_w^{r,\epsilon}. \end{aligned}$$

As FindWeightPIT returns this M, it returns a PIT M with $w(M) \geq \tau_w^{r,\epsilon}$. □

By Lemmas 5.7 and 5.11, we have shown FindWeightPIT returns a PIT M of weight at least $\tau_w^{r,\epsilon}$. It remains to show that for fixed r and ϵ, FindWeightPIT runs in time poly$(|V(G)|, |w|)$. As the runtime of FindITorBD typically dominates the runtime of all other steps in a recursive call of FindWeightPIT, we make no effort to minimise the runtime of these other steps.

Lemma 5.12. *For fixed r and ϵ,* FindWeightPIT *terminates in time* poly$(|V(G)|, |w|)$.

Proof: Let $q(n, x)$ denote the maximum runtime of FindWeightPIT on input graphs G with n vertices and weight function w where $|w| = x$. Let G, $(V_1, \ldots, V_m)$, and w be the given inputs. We prove the result by induction on $|w|$ (recall w is an integer weight function and $|w| \geq 0$).

For $|w| = 0$, $\mathcal{V} = \emptyset$ and so $\mathcal{W} = \emptyset$ and G' is the empty graph. Hence FindITorBD$(G'; \mathcal{W})$ returns $M = \emptyset$ in time poly$(|V(G')|) = O(1)$. Determining $\mathcal{V} = \emptyset$ takes time $O(|V(G)|)$ and the other lines take an additional $O(1)$ operations (since the sets are empty). Hence $q(|V(G)|, 0) = O(|V(G)|)$.

Now suppose $|w| > 0$. Determining the classes in $\mathcal{V}$ can be accomplished by for each V_i, checking the weight of each $v \in V_i$ until a $v \in V_i$ is found such that $w(v) > 0$. Then V_i is added to $\mathcal{V}$ and the next vertex class is tested. This takes time $O(|V(G)|)$. The sets W_i can be found by finding the largest weight assigned to a vertex in V_i and then adding every $v \in V_i$ with that weight to W_i. Hence finding all of the W_i takes time $|\mathcal{V}| \cdot O(|V(G)|) = O(|V(G)|^2)$. Determining the graph G' can also be found using $\mathcal{W}$ in time $O(|V(G)|^2)$. Thus the initialising steps of FindWeightPIT can be implemented in time $O(|V(G)|^2)$.

Recall from Theorem 3.1 that FindITorBD on an r-claw-free graph H with vertex partition $(U_1, \ldots, U_n)$ runs in time poly$(|V(H)|)$ for fixed r and ϵ. Let p denote this polynomial. As $V(G') \subseteq V(G)$ and $W_i \subseteq V_i$ for each $W_i \in \mathcal{W}$, G' is r-claw-free with respect to $\mathcal{W}$ and so FindITorBD$(G', \mathcal{W})$ completes in time $p(|V(G')|) \leq p(|V(G)|)$.

Determining if FindITorBD returns an IT M or a set $\mathcal{B}$ of vertex classes and sets D and L of vertices takes time $O(1)$ (determine the number of vectors in the output). If FindITorBD returns an IT M, returning M takes time $|V(G)|$ and so FindWeightPIT ends after an additional $O(|V(G)|)$ operations. Thus FindWeightPIT completes in time $O(|V(G)|^2) + p(|V(G)|)$, which is poly$(|V(G)|)$ and therefore poly$(|V(G)|, |w|)$.

Suppose FindITorBD returns $\mathcal{B}$, D, and L. Determining the number of neighbours in D of a vertex v and then $w'(v)$ for each $v \in V(G)$ takes time $O(|V(G)| \cdot |D|) = O(|V(G)|^2)$. Defining the set Y takes another $|L| \cdot O(|\mathcal{B}| + O(|V(G)|^2)) = O(|V(G)|^3)$ operations. The while loop of FindWeightPIT has at most $|Y| \leq |V(G)|$ iterations and each iteration can be implemented in time $O(|V(G)|)$. Thus the final PIT M is returned after an additional $O(|V(G)|^3)$ operations. This leads to the following claim.

Claim 5.13. $q(|V(G)|, |w|) \leq q(|V(G)|, |w| - 1) + p(|V(G)|) + O(|V(G)|^3)$.

Proof: By construction, every vertex in D has a neighbour in D and $w(v) > 0$ for all $v \in D \subseteq V(G')$. As $D \neq \emptyset$, we have

$$|w'| \leq |w| - |D| \leq |w| - 1.$$

Hence the recursive application of FindWeightPIT takes time

$$q(|V(G)|, |w'|) \leq q(|V(G)|, |w| - 1).$$

From the discussion above, implementing FindITorBD$(G', \mathcal{W})$ takes time $p(|V(G)|)$ and implementing the remaining lines takes time $O(|V(G)|^3)$. Thus FindWeightPIT on inputs G, w, and $(V_1, \ldots, V_m)$ completes in time

$$q(|V(G)|, |w| - 1) + p(|V(G)|) + O(|V(G)|^3). \qquad \square$$

Therefore $q(|V(G)|, |w|) = O(|w| \cdot [p(|V(G)|) + |V(G)|^3])$, and so the lemma holds. $\square$

Theorem 5.5 follows from Lemmas 5.7, 5.11, and 5.12.

Chapter 6

A Randomised Algorithm and Strong Colourings

In this chapter, we introduce a new randomised algorithm, called FindWeightIT, for graphs with maximum degree Δ, where Δ can vary freely (Theorem 6.10). This is different from the previous algorithms whose runtimes are exponential in Δ, making them only efficient when Δ is fixed. We prove Theorem 6.10 using FindWeightPIT and an algorithmic version of the the Lovász Local Lemma (LLL) due to Moser and Tardos [79]. Unlike FindWeightPIT, the runtime of FindWeightIT is not limited by a dependence on Δ. Moreover, FindWeightIT returns an IT of certain weight rather than a PIT.

We also apply FindWeightIT to the problem of finding strong colourings and fractional strong colourings (formally defined in Section 6.4) in graphs. These applications were studied by Aharoni, Berger, and Ziv in [3] as an application of Theorem 5.2. Recall from Section 4.4 that the general bound on the strong chromatic number of a graph G is conjectured to be $2\Delta(G)$ (see e.g. [3]). Aharoni, Berger, and Ziv [3] used ITs in weighted graphs to show that the fractional version of this conjectured bound holds (Theorem 6.13). We prove an algorithmic version of this result (Theorem 6.14), with asymptotically the same upper bound. We also give an algorithm using FindWeightIT that finds a strong colouring of a graph G using $(3+\epsilon)\Delta(G)$ colours, where $\epsilon > 0$ is fixed (Theorem 6.16). This result generalises a similar result from Section 4.4 since $\Delta(G)$ can be unbounded, however this new algorithm uses FindWeightIT and so is not deterministic.

As mentioned in Chapter 1, the new algorithms and algorithmic results discussed in this chapter are joint work with David Harris and Penny Haxell. Modified versions of these algorithms can be found in [37].

This chapter is organised as follows. In Section 6.1, we derive some algorithmic consequences of FindWeightPIT that will lead to FindWeightIT and the proof of Theorem 6.10. In Section 6.2, we use the LLL to develop some degree reduction results that allow us to remove the condition that $\Delta(G)$ be constant. This will require some of the properties of the Moser-Tardos algorithm discussed in Section 2.3. We prove Theorem 6.10 in Section 6.3. In Section 6.4, we discuss Theorem 6.13 of Aharoni, Berger, and Ziv [3] and prove an algorithmic version (Theorem 6.14) of it using Theorem 6.10. We prove some immediate consequences (Corollaries 6.15 and 6.16) of Theorem 6.10 for strong colourings in Section 6.5.

6.1 Algorithmic Consequences of FindWeightPIT

This section contains some algorithmic consequences of FindWeightPIT. The algorithm FindWeightPIT can be found in 5.2.1 in Section 5.2.

Recall that FindWeightPIT takes as inputs a graph G, a vertex partition $\mathcal{V}$ such that G is r-claw-free with respect to $\mathcal{V}$, and a weight function $w : V(G) \to \mathbb{Z}$. It returns a PIT M in G of weight at least $\tau_w^{r,\epsilon}$ in time $\text{poly}(|V(G)|, |w|)$ for fixed r and $0 < \epsilon < \frac{1}{r}$, where $|w| = \sum\limits_{v \in V(G)} |w(v)|$. (The LP $\mathcal{P}_w^{r,\epsilon}$ defining $\tau_w^{r,\epsilon}$ can also be found in Section 5.2.)

We begin with the following analogue of Corollary 3.2 for vertex-weighted graphs with integral weights.

Proposition 6.1. *There exists an algorithm $\mathcal{A}_1$ that takes as inputs any graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b > 2\Delta(G)$ for each i, and weight function $w : V(G) \to \mathbb{Z}^{\geq 0}$, and finds an IT in G of weight at least $\frac{w(V(G))}{b}$. For fixed b, the runtime is* $\text{poly}(|V(G)|, |w|)$.

Proof: Let the algorithm $\mathcal{A}_1$ be as defined in 6.1.1.

We show that M is an IT with $w(M) \geq \frac{w(V(G))}{b}$ and that for fixed b, the runtime is $\text{poly}(|V(G)|, |w|)$. To do so, we must verify G, $(V_1, \dots, V_m)$, and w' are valid inputs for FindWeightPIT with parameters $r = \lceil \frac{b}{2} \rceil + 1$ and $\epsilon = \frac{1}{b^2}$.

As $b > 2\Delta(G)$, clearly G is r-claw-free with respect to any vertex partition. Also, the vector γ defined by $\gamma_v = \frac{1}{b}$ for each $v \in V(G)$ is feasible for $\mathcal{P}_w^{r,\epsilon}$. This is because $|V_i| = b$ for each i and

$$\sum_{u \in N(v)} \gamma_u = \frac{\deg(v)}{b} \leq \frac{\lceil \frac{b}{2} \rceil - 1}{b} \leq \frac{1 - \left(\lceil \frac{b}{2} \rceil + 1\right)\left(\frac{1}{b^2}\right)}{2 + \left(\frac{1}{b^2}\right)} = \frac{1 - r\epsilon}{2 + \epsilon}.$$

6.1.1 $\mathcal{A}_1$

Input: A graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b > 2\Delta(G)$ for each i, and weight function $w : V(G) \to \mathbb{Z}^{\geq 0}$ with parameter b.

Output: An IT M of weight at least $\frac{w(V(G))}{b}$.

```
1: function $\mathcal{A}_1(G; w; V_1, \dots, V_m)$
2:     $r := \left\lceil \frac{b}{2} \right\rceil + 1$
3:     $\epsilon := \frac{1}{b^2}$
4:     for all $v \in V(G)$ do
5:         $w'(v) := w(v) + m|w|$
6:     $M :=$ FindWeightPIT$(G; w'; V_1, \dots, V_m)$ with parameters $r$ and $\epsilon$.
7:     return $M$
```

For the weight function $w' : V(G) \to \mathbb{Z}^{\geq 0}$ defined by $w'(v) = w(v) + m|w|$, FindWeightPIT for r and ϵ returns a PIT M of weight at least $\tau_{w'}^{r,\epsilon} \geq \frac{w'(V(G))}{b}$ in time poly$(|V(G)|, |w'|)$ by Theorem 5.5. Note that $|w'| = \text{poly}(|V(G)|, |w|)$.

We claim that M is an IT. Suppose M were not an IT. Then $|M| < m$ and so

$$w'(M) \leq (m-1)\left(\max\{w'(u) : u \in V(G)\}\right) \leq (m-1)(|w| + m|w|) = (m^2 - 1)|w|.$$

On the other hand, every vertex class has size b and so $|V(G)| = mb$. Thus,

$$w'(M) \geq \frac{w'(V(G))}{b} = \frac{w(V(G))}{b} + mb\left(\frac{m|w|}{b}\right) \geq \frac{|w|}{b} + m^2|w| \geq m^2|w|.$$

This is a contradiction and so M is an IT.

Since M is an IT, we have $w(M) = w'(M) - m(m|w|)$. Also, $w'(M) \geq \frac{w'(V(G))}{b} = \frac{w(V(G))}{b} + m^2|w|$. Thus $w(M) \geq \frac{w(V(G))}{b}$ as desired, and so $\mathcal{A}_1$ is the desired algorithm. □

The next lemma removes the runtime dependence on $|w|$ in Lemma 6.1, but weakens the conclusion slightly. We note that the algorithm $\mathcal{A}_2$ of Lemma 6.2 takes as input a real-valued weight function. To be rigorous, this weight function should be discretised to some finite precision, which would make $\mathcal{A}_2$ have some dependence on the number of bits of precision used. However, we ignore such numerical issues for simplicity of exposition.

Lemma 6.2. *There exists an algorithm $\mathcal{A}_2$ that takes as inputs any $\eta > 0$, graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b > 2\Delta(G)$ for each i, and weight function*

$w : V(G) \to \mathbb{R}^{\geq 0}$, and finds an IT in G of weight at least $(1-\eta)\frac{w(V(G))}{b}$. For fixed b, the runtime is $\operatorname{poly}\left(|V(G)|, \frac{1}{\eta}\right)$.

Proof: Let the algorithm $\mathcal{A}_2$ be as defined in 6.1.2.

6.1.2 $\mathcal{A}_2$

Input: A parameter $\eta > 0$, graph G, vertex partition $(V_1, \ldots, V_m)$ such that $|V_i| = b > 2\Delta(G)$ for each i, and weight function $w : V(G) \to \mathbb{R}^{\geq 0}$ with parameter b.

Output: An IT M of weight at least $(1-\eta)\frac{w(V(G))}{b}$.

```
1: function $\mathcal{A}_2(G; \eta; w; V_1, \ldots, V_m)$
2:     if $w(V(G)) = 0$ then
3:         $M := \mathcal{A}_1(G; w; V_1, \ldots, V_m)$ with parameter $b$.
4:     else
5:         $\alpha := \frac{|V(G)|}{\eta w(V(G))}$
6:         for all $v \in V(G)$ do
7:             $w'(v) := \lfloor \alpha w(v) \rfloor$
8:         $M := \mathcal{A}_1(G; w'; V_1, \ldots, V_m)$ with parameter $b$.
9:     return $M$
```

We show that M is an IT with $w(M) \geq (1-\eta)\frac{w(V(G))}{b}$ and that for fixed b, the runtime is $\operatorname{poly}\left(|V(G)|, \frac{1}{\eta}\right)$. To do so, we must verify G, $(V_1, \ldots, V_m)$, and w' are valid inputs for $\mathcal{A}_1$ with parameter b.

If $w(V(G)) = 0$, then $w(v) = 0$ for all vertices v. Hence $w : V(G) \to \mathbb{Z}^{\geq 0}$ is integral and so we can apply $\mathcal{A}_1$ directly to get an IT of weight at least $\frac{w(V(G))}{b} > (1-\eta)\frac{w(V(G))}{b}$. For fixed b, the runtime is $\operatorname{poly}(|V(G)|)$ since $|w| = 0$. Thus we may assume $w(V(G)) > 0$.

Let $\alpha = \frac{|V(G)|}{\eta w(V(G))}$ and $w' : V(G) \to \mathbb{Z}^{\geq 0}$ be defined by $w'(v) = \lfloor \alpha w(v) \rfloor$. Then applying $\mathcal{A}_1$ for b to $(G; w'; V_1, \ldots, V_m)$ returns an IT M of weight at least $\frac{w'(V(G))}{b}$. As $|w'| \leq \alpha w(V(G)) = \frac{|V(G)|}{\eta}$, the runtime of $\mathcal{A}_1$ is $\operatorname{poly}(|V(G)|, \frac{1}{\eta})$.

It remains to show M has the desired weight. We have

$$w'(V(G)) = \sum_{v \in V(G)} \lfloor \alpha w(v) \rfloor > \sum_{v \in V(G)} (\alpha w(v) - 1) = \alpha w(V(G)) - |V(G)|.$$

Thus,

$$w(M) = \sum_{v \in M} w(v) \geq \sum_{v \in M} \frac{w'(v)}{\alpha} = \frac{w'(M)}{\alpha} \geq \frac{w'(V(G))}{b\alpha} \geq (1-\eta)\frac{w(V(G))}{b}. \qquad \square$$

The final step to reach Theorem 6.10 will be to remove the requirement that b is a fixed constant in Lemma 6.2. To do so, we will need an algorithm that, given a vertex-partitioned weighted graph G, is able to efficiently find a subgraph H of G such that the degree, vertex class size, and total weight of the vertices is reduced (see Lemma 6.3). We will see this algorithm in Section 6.3, as well as the remainder of the proof of Theorem 6.10. Note that this algorithm uses the properties of the Moser-Tardos algorithm [79] discussed in Section 2.3.

6.2 Degree Reduction Using LLL

In this section, we discuss some results that use the LLL to select a constant-sized piece of each vertex class V_i, such that the subgraph of G induced on all the pieces is an appropriately "scaled-down model of G." We then apply Lemma 6.2 in this model, in which the maximum degree (and thus b) is constant, to find an IT. This is another important step for defining the randomised algorithm FindWeightIT.

In the following lemma, we introduce a "degree-splitting" algorithm. This algorithm uses the Moser-Tardos algorithm (Theorem 2.9) to take a weighted vertex-partitioned graph and reduce the maximum degree, vertex class size, and sum of the vertex weights by a factor of approximately $\frac{1}{2}$. Applying this algorithm $\log \Delta(G)$ times results in a scaled-down model of G that has constant degree.

Lemma 6.3. *There is a randomised algorithm $\mathcal{A}_3$ that takes as input a graph G, a vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b$ for each i, a real-valued parameter $\hat{\Delta} \geq 5000$ such that $\Delta(G) \leq \hat{\Delta}$ and $2\hat{\Delta} \leq b \leq 3\hat{\Delta}$, and a weight function $w : V(G) \to \mathbb{R}^{\geq 0}$. It generates a subset $V' \subseteq V(G)$ such that the induced subgraph $G[V']$ has maximum degree at most $t = \frac{\hat{\Delta}}{2} + 10\left(\sqrt{\hat{\Delta}\log\hat{\Delta}}\right)$, $|V_i \cap V'| = b' = \left\lceil \frac{b}{2} \right\rceil$ for each vertex class V_i, and $\frac{w(V')}{b'} \geq \frac{w(V(G))}{b}(1 - \hat{\Delta}^{-10})$. The expected runtime is* $\mathrm{poly}(|V(G)|)$.

Proof: Let the algorithm $\mathcal{A}_3$ be as defined in 6.2.1.

6.2.1 $\mathcal{A}_3$

Input: A graph G, vertex partition $(V_1, \ldots, V_m)$ such that $|V_i| = b$ for each i, a real-valued parameter $\hat{\Delta} \geq 5000$ such that $\Delta(G) \leq \hat{\Delta}$ and $2\hat{\Delta} \leq b \leq 3\hat{\Delta}$, and weight function $w : V(G) \to \mathbb{R}^{\geq 0}$.

Output: An subset $V' \subseteq V(G)$ satisfying the conditions of Lemma 6.3.

```
1: function $\mathcal{A}_3(G; \hat{\Delta}; w; V_1, \ldots, V_m)$
2:     $b' := \left\lceil \frac{b}{2} \right\rceil$
3:     $t := \frac{\hat{\Delta}}{2} + 10\left(\sqrt{\hat{\Delta} \log \hat{\Delta}}\right)$
4:     for $i = 1, \ldots, m$ do
5:         $X_i$ := a random variable whose distribution is to select at random a subset $V_i' \subseteq V_i$ of size exactly $b'$.
6:     for all $v \in V(G)$ do
7:         $B_v$ := the bad event that $v$ has more than $t$ neighbours in $\{v \in X_i : 1 \leq i \leq m\}$.
8:     $\Omega$ := the probability space in variables $X_1, \ldots, X_m$
9:     $\mathcal{B} := \{B_v : v \in V(G)\}$
10:    $V' := \emptyset$
11:    while $w(V') < \frac{b' w(V(G))}{b}(1 - \hat{\Delta}^{-10})$ do
12:        $V'$ := Moser-Tardos$(\Omega, \mathcal{B})$
13:    return $V'$
```

Note that $\mathcal{A}_3$ repeatedly uses the Moser-Tardos algorithm, and so we must show this choice of Ω and $\mathcal{B}$ satisfies Theorem 2.9 for some parameters $p, d \geq 0$. We begin by determining these parameters p and d.

Consider some vertex v and bad event B_v. This bad event B_v is affected by the choice of X_i for each V_i such that $V_i = \text{Vclass}(u)$ for some $u \in N(v)$. Each choice of X_i, in turn, affects B_w for any vertex w with a neighbour in V_i. In total, B_v can affect at most $b\Delta(G)^2$ other vertices. Thus the conditions that $b \leq 3\hat{\Delta}$ and $\Delta(G) \leq \hat{\Delta}$ ensure that this is at most $3\hat{\Delta}^3$. Hence $d = 3\hat{\Delta}^3$ will suffice.

We now examine the probability of event B_v. For $N(v) = \{y_1, \ldots, y_k\}$, the degree of v in V' is the sum $Y = \sum_{i=1}^{k} Y_i$, where Y_i is the indicator that $y_i \in V'$. It is not hard to see that the random variables Y_i are negatively correlated, and that Y has mean $\mu = \frac{kb'}{b} \leq \frac{\hat{\Delta}}{2} + \frac{\hat{\Delta}}{b}$. Hence the condition that $b \geq 2\hat{\Delta}$ ensures that this is at most $\hat{\mu} = \frac{\hat{\Delta}+1}{2}$.

We use Chernoff's bound, which applies to sums of negatively correlated random variables (see [80]), to compute the probability $Y \geq t$. Note that $\hat{\mu}$ is an upper bound on the true mean μ of Y and that t represents a multiplicative deviation of $\delta = \frac{t}{\hat{\mu}} - 1 = \frac{20\sqrt{\hat{\Delta}\log\hat{\Delta}} - 1}{\hat{\Delta}+1}$ over the value $\hat{\mu}$. Simple analysis shows that $0 < \delta < 1$ for $\hat{\Delta} \geq 5000$, so we can use the simplified form of Chernoff's bound (Lemma 3 in [75])

$$\Pr(Y \geq \hat{\mu}(1+\delta)) \leq e^{-\hat{\mu}\delta^2/3}.$$

Simple analysis again shows that for $\hat{\Delta} \geq 5000$, this is at most $e^{-66\log\hat{\Delta}}$. Hence, overall, the bad event B_v has probability at most $\hat{\Delta}^{-66}$. Thus $p = \hat{\Delta}^{-66}$ will suffice.

Let $p = \hat{\Delta}^{-66}$ and $d = 3\hat{\Delta}^3$. Then $epd \leq 1$ and so the parameters p and d are valid for use in Theorem 2.9. Hence the Moser-Tardos algorithm generates, in expected polynomial time, a configuration avoiding all such bad events B_v. This configuration is a vertex set V' such that $G[V']$ has $\Delta(G[V']) \leq t$. Furthermore, by construction the vertex classes restricted to the vertices in V' all have size exactly b'.

By Theorem 2.10, for any vertex class V_i and fixed set $U_i \subseteq V_i$, the probability of an event $X_i = U_i$ in the MT-distribution is at most e^{epr} times its probability in the original probability space Ω, where r is the number of bad-events affecting event $X_i = U_i$. The original sampling probability is $\frac{1}{\binom{b}{b'}}$ from the uniform distribution. The number of events affecting X_i is at most $r = 3\hat{\Delta}^3$. Thus,

$$\Pr(X_i = U_i) \leq \frac{e^{3\hat{\Delta}^3 \cdot e\hat{\Delta}^{-66}}}{\binom{b}{b'}}.$$

Simple analysis shows that for $\hat{\Delta} \geq 5000$, we have $e^{3\hat{\Delta}^3 \cdot e\hat{\Delta}^{-66}} \leq 1 + \hat{\Delta}^{-11}$. Since $\mathrm{E}[w(X_i)] \leq w(V_i)$ with probability one, we therefore have

$$\begin{aligned}
\mathrm{E}[w(X_i)] &= w(V_i) - \sum_{\substack{U_i \subseteq V_i \\ |U_i| = b'}} \Pr(X_i = U_i) w(V_i \setminus U_i) \\
&\geq w(V_i) - \frac{1 + \hat{\Delta}^{-11}}{\binom{b}{b'}} \sum_{\substack{U_i \subseteq V_i \\ |U_i| = b'}} w(V_i \setminus U_i) \\
&= w(V_i) - \frac{1 + \hat{\Delta}^{-11}}{\binom{b}{b'}} \sum_{u \in V_i} w(u) \sum_{\substack{U_i \subseteq V_i \\ |U_i| = b' \\ u \notin U_i}} 1 \\
&= w(V_i) - \frac{1 + \hat{\Delta}^{-11}}{\binom{b}{b'}} w(V_i) \tbinom{b-1}{b'} \\
&= w(V_i) \left(1 - (1 - \tfrac{b'}{b})(1 + \hat{\Delta}^{-11})\right) \\
&\geq w(V_i)(\tfrac{b'}{b} - \hat{\Delta}^{-11}).
\end{aligned}$$

Summing over all vertex classes V_i (and noting that $\frac{b'}{b} \geq \frac{1}{2}$), this gives

$$\mathrm{E}[w(V')] \geq w(V(G))(\tfrac{b}{b'} - \hat{\Delta}^{-11}) \geq \frac{b' w(V(G))}{b}(1 - 2\hat{\Delta}^{-11}).$$

Since $w(V') \leq w(V(G))$, the number of iterations of the while loop in $\mathcal{A}_3$ is $O(\hat{\Delta}^{10}) = \mathrm{poly}(|V(G)|)$. Initialising each variable and running the Moser-Tardos algorithm clearly takes expected runtime $\mathrm{poly}(|V(G)|)$, and so the statement holds. $\square$

The next lemma removes the runtime dependence on $\hat{\Delta}$ in Lemma 6.3, but requires slightly stronger conditions.

Lemma 6.4. *There is a randomised algorithm $\mathcal{A}_4$ that takes as input a parameter $0 < \epsilon < 1$, a graph G, a vertex partition $(V_1, \ldots, V_m)$ such that for some $b \geq (2+\epsilon)\Delta(G)$ $|V_i| = b$ for each i, and a weight function $w : V(G) \to \mathbb{R}^{\geq 0}$. It generates an IT M of G with*

$$w(M) \geq \frac{w(V(G))}{b}(1 - \epsilon^2).$$

For fixed ϵ, the expected runtime is $\mathrm{poly}(|V(G)|)$.

6.2.2 $\mathcal{A}_4$

Input: A parameter $0 < \epsilon < 1$, graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b \geq (2+\epsilon)\Delta(G)$ for each i, and weight function $w : V(G) \to \mathbb{R}^{\geq 0}$.

Output: An IT M of weight at least $\frac{w(V(G))}{b}(1-\epsilon^2)$.

```
 1: function $\mathcal{A}_4(G; \epsilon; w; V_1, \dots, V_m)$
 2:     $\hat{\Delta} := \frac{b}{2+\epsilon}$
 3:     $C :=$ a sufficiently large constant
 4:     $t := \max\left\{0, \left\lfloor \log_2 \frac{\hat{\Delta}\epsilon^9}{C} \right\rfloor\right\}$
 5:     Initialise $G_0 := G$, $\hat{\Delta}_0 := \hat{\Delta}$, and $b_0 := b$.
 6:     for $i = 1, \dots, t$ do
 7:         $V_i' := \mathcal{A}_3(G_{i-1}; \hat{\Delta}_{i-1}; w; V_1, \dots, V_m)$
 8:         $\mathcal{V}_i :=$ the vertex partition induced by restricting the vertex classes to vertices
    in $V_i'$.
 9:         $G_i := G[V_i']$
10:         $b_i := \left\lceil \frac{b_{i-1}}{2} \right\rceil$
11:         $\hat{\Delta}_i := \frac{\hat{\Delta}_i}{2} + 10\sqrt{\hat{\Delta}_i \log \hat{\Delta}_i}$
12:     $M := \mathcal{A}_2(G_t; \frac{\epsilon^2}{2}; w; \mathcal{V}_t)$
13:     return $M$
```

Proof: Let the algorithm $\mathcal{A}_4$ be as defined in 6.2.2.

Note that $\mathcal{A}_4$ repeatedly applies $\mathcal{A}_3$ for $t = \max\left\{0, \left\lfloor \log_2 \frac{\hat{\Delta}\epsilon^9}{C} \right\rfloor\right\}$ rounds, where C is a constant to be specified later. This generates a series of graphs G_i for $i = 0, \ldots, t$. After this process completes, it applies $\mathcal{A}_2$ to G_t to get the desired IT M. We therefore must check the preconditions are satisfied for every iteration of $\mathcal{A}_3$ as well as for the application of $\mathcal{A}_2$.

As we apply $\mathcal{A}_3$, we must ensure that each graph G_i has vertex classes of size exactly b_i and maximum degree bounded by a parameter $\hat{\Delta}_i$. Let $W_i = w(V(G_i))$. The parameters b_i, $\hat{\Delta}_i$, and W_i are given initially by

$$b_0 = b, \quad \hat{\Delta}_0 = \hat{\Delta} = \tfrac{b}{2+\epsilon}, \quad W_0 = w(V(G)),$$

and, assuming the preconditions of Lemma 6.3 remained satisfied, later by

$$b_{i+1} = \left\lceil \tfrac{b_i}{2} \right\rceil, \quad \hat{\Delta}_{i+1} = \tfrac{\hat{\Delta}_i}{2} + 10\sqrt{\hat{\Delta}_i \log \hat{\Delta}_i}, \quad \tfrac{W_{i+1}}{b_{i+1}} \geq \tfrac{W_i}{b_i}(1 - \hat{\Delta}_i^{-10}).$$

We now verify that the preconditions are satisfied, starting with estimating $\hat{\Delta}_i$.

Claim 6.5. *$\frac{C}{\epsilon^9} \leq \hat{\Delta}_i \leq \hat{\Delta}2^{-i} + (\hat{\Delta}2^{-i})^{2/3}$ for all i.*

Proof: It is clear that $\hat{\Delta}_i \geq \hat{\Delta}2^{-i}$. By our choice of t, we therefore have that $\hat{\Delta}_i \geq \frac{C}{\epsilon^9}$ for all i. It remains to show the upper bound. We do so by induction on i.

For $i = 0$, it is clearly true that $\hat{\Delta} \leq \hat{\Delta} + \hat{\Delta}^{2/3}$. For the inductive step, by definition we have

$$\hat{\Delta}_{i+1} = \frac{\hat{\Delta}_i}{2} + 10\sqrt{\hat{\Delta}_i \log \hat{\Delta}_i}.$$

As $\hat{\Delta}_i \geq \hat{\Delta}2^{-i} \geq C$ (since $0 < \epsilon < 1$), for C sufficiently large we have $\hat{\Delta}_i \leq 2^{1-i}\hat{\Delta}$. Thus, by the induction hypothesis,

$$\begin{aligned}
\hat{\Delta}_{i+1} &\leq \hat{\Delta}2^{-i-1} + \frac{(\hat{\Delta}2^{-i})^{2/3}}{2} + 10\sqrt{2^{1-i}\hat{\Delta}\log(2^{1-i}\hat{\Delta})} \\
&\leq \hat{\Delta}2^{-i-1} + (\hat{\Delta}2^{-i-1})^{2/3} - 0.1(\hat{\Delta}2^{-i})^{2/3} + 10\sqrt{2^{1-i}\hat{\Delta}\log(2^{1-i}\hat{\Delta})}.
\end{aligned}$$

Since $\hat{\Delta}2^{-i} \geq C$ for all i, for C sufficiently large, the above bound is at most $\hat{\Delta}2^{-i-1} + (\hat{\Delta}2^{-i-1})^{2/3}$. Hence the upper bound holds. □

By Claim 6.5, $\hat{\Delta}_i \geq \frac{C}{\epsilon^9} \geq 2$ for all i. It remains to show that $2\hat{\Delta}_i \leq b_i \leq 3\hat{\Delta}_i$ for $i < t$.

Claim 6.6. $2\hat{\Delta}_i < b_i \leq 3\hat{\Delta}_i$ *for all* $i < t$.

Proof: We show that the upper bound and lower bound are maintained separately. The proof of the upper bound is by induction on i. For $i = 0$, the bound holds by the choice of $\hat{\Delta}_0$.

Note that

$$\frac{b_{i+1}}{\hat{\Delta}_{i+1}} \leq \frac{\frac{b_i+1}{2}}{\frac{\hat{\Delta}_i}{2} + 10\sqrt{\hat{\Delta}_i \log \hat{\Delta}_i}} = \frac{b_i + 1}{\hat{\Delta}_i + 20\sqrt{\hat{\Delta}_i \log \hat{\Delta}_i}} \leq \frac{b_i}{\hat{\Delta}_i} \leq 3,$$

where the second inequality holds because $\hat{\Delta}_i \geq C$ and C is sufficiently large. Hence $b_{i+1} \leq 3\hat{\Delta}_{i+1}$.

For the lower bound, note that $b_i \geq 2^{-i}b$ for all i. Since $i \leq t$, we have $\hat{\Delta}2^{-i} \geq \hat{\Delta}2^{-t} \geq \frac{C}{\epsilon^9}$. Thus the upper bound in Claim 6.5 implies that $\hat{\Delta}_i \leq \hat{\Delta}_i 2^{-1}\left(1 + \frac{C}{\epsilon^9}\right)^{-1/3}$. Hence,

$$b_i - 2\hat{\Delta}_i \geq 2^{-i}b - 2\left(2^{-i}\hat{\Delta}\left(1 + \frac{\epsilon^3}{C^{1/3}}\right)\right) = 2^{-i}\left(b - 2\hat{\Delta} - \frac{2\epsilon^3\hat{\Delta}}{C^{1/3}}\right) = 2^{-i}\left(\epsilon^3\hat{\Delta} - \frac{\epsilon^3\hat{\Delta}}{C^{1/3}}\right).$$

This is positive as $C > 1$ and so $b_i > 2\hat{\Delta}_i$ for all i. □

Since $\hat{\Delta}_i \geq \frac{C}{\epsilon^9} \geq C$ for $i \leq t$, by choosing $C \geq 5000$ we also ensure that $\hat{\Delta}_i \geq 5000$ as well. Thus by Claims 6.5 and 6.6, the preconditions of $\mathcal{A}_3$ are satisfied for all t iterations and for fixed ϵ, each iteration takes expected time poly$(|V(G)|)$.

We now show that for $\eta = \frac{\epsilon^2}{2}$ and G_t, the preconditions of $\mathcal{A}_2$ are met. Let $G' = G_t$. Note that $\hat{\Delta}_t \geq \Delta(G')$. As $b_t > 2\hat{\Delta}_t$, we therefore have that $b_t > 2\Delta(G')$. By Claim 6.5 and our choice of t, we have

$$\Delta(G') \leq \hat{\Delta}_t \leq 2^{1-t}\hat{\Delta} \leq \frac{4C}{\epsilon^9}.$$

As $b_t \leq 3\hat{\Delta}_t$, we have $b_t \leq \text{poly}(\frac{1}{\epsilon})$. By our choice of η, for fixed ϵ we have that the runtime of $\mathcal{A}_2$ is poly$(|V(G')|)$, which is poly$(|V(G)|)$.

It remains to analyse the weight $W_t = w(V(G'))$. We have

$$\frac{W_t}{b_t} \geq \frac{w(V(G))}{b}\prod_{i=0}^{t}(1 - \hat{\Delta}_i^{-10}) \geq \frac{w(V(G))}{b}\prod_{i=0}^{t} e^{-\hat{\Delta}_i^{-10}/2} = \frac{w(V(G))}{b} e^{-\frac{\sum_{i=0}^{t}\hat{\Delta}_i^{-10}}{2}}.$$

As $\hat{\Delta}_i \geq \hat{\Delta}2^{-i}$ for each i, the sum is bounded by $2(2^{-t}\hat{\Delta})^{-10}$. By the choice of t, this is at most $2(\frac{C}{\epsilon^9})^{-10}$, which is at most $\frac{\epsilon^2}{10}$ for $C \geq 2$. Since $\epsilon < 1$, we have

$$\frac{W_t}{b_t} \geq \frac{w(V(G))}{b} e^{-\epsilon^2/10} \geq \frac{w(V(G))}{b}\left(1 - \frac{\epsilon^2}{2}\right).$$

Thus, by Lemma 6.2, $\mathcal{A}_2$ on inputs G' and η returns an IT M of weight

$$w(M) \geq \frac{w(V(G_t))}{b_t}(1-\eta) \geq \frac{w(V(G))}{b}\left(1 - \frac{\epsilon^2}{2}\right)\left(1 - \frac{\epsilon^2}{2}\right) \geq \frac{w(V(G))}{b}\left(1 - \epsilon^2\right).$$

This concludes the proof as $\mathcal{A}_4$ returns this M. □

With Lemma 6.4, we are able prove some algorithmic versions of Theorem 5.2, including Theorem 6.10, in Section 6.3.

6.3 The Randomised Algorithm FindWeightIT

In this section, we give a randomised algorithm, called FindWeightIT, that finds ITs in vertex-weighted graphs G in expected polynomial time, where this runtime does not depend on $\Delta(G)$ (Theorem 6.10). This allows us to find ITs in graphs with unbounded maximum degree, unlike FindWeightPIT and $\mathcal{A}_2$ which, for graphs with maximum degree Δ, both have runtimes that depend on Δ being fixed (recall that b in Lemma 6.2 satisfies $b > 2\Delta(G) = 2\Delta$).

We begin by defining another LP. Let G be a graph, $(V_1, \ldots, V_m)$ a vertex partition of G, $w : V(G) \to \mathbb{R}$ a weight function, and $\delta \in \mathbb{R}^{\geq 0}$ a parameter. We define $\mathcal{P}_w^\delta$ to be the following LP and τ_w^δ to be the largest objective value to $\mathcal{P}_w^\delta$.

$$\begin{array}{rrcll}
\text{max} & \sum_{v \in V(G)} w(v)\gamma_v & & & \\
\text{subject to} & \sum_{u \in N(v)} \gamma_u & \leq & \delta & \forall v \in V(G) \\
& \sum_{v \in V_i} \gamma_v & = & 1 & \forall i \in \{1, \ldots, m\} \\
& 0 \leq \gamma_v & \leq & 1 & \forall v \in v(G).
\end{array}$$

If this LP is not feasible, we define $\mathcal{P}_w^\delta = \emptyset$ and $\tau_w^\delta = -\infty$.

Note that $\mathcal{P}_w^\delta$ is very similar to $\mathcal{P}_w$ and $\mathcal{P}_w^{r,\epsilon}$ from Section 5.2. However, the bound on $\sum\limits_{u \in N(v)} \gamma_u$ is determined by a parameter δ instead of being a constant $\frac{1}{2}$ (see $\mathcal{P}_w$) or determined by parameters r and ϵ (see $\mathcal{P}_w^{r,\epsilon}$). Moreover, $\mathcal{P}_w^\delta$ requires $\sum\limits_{v \in V_i} \gamma_v$ to be 1 rather than at most 1.

Lemma 6.4 leads to the following result, which is an algorithmic version of Theorem 5.2.

Proposition 6.7. *There exists a randomised algorithm $\mathcal{A}_5$ which takes as inputs a parameter $0 < \epsilon < \frac{1}{2}$, a graph G, a vertex partition $(V_1, \dots, V_m)$, a weight function $w : V(G) \to \mathbb{R}^{\geq 0}$, and a vector γ in the polytope $\mathcal{P}_w^\delta$ where $\delta = \frac{1}{2} - \epsilon$, and returns an IT M of G with*

$$w(M) \geq (1 - 2\epsilon^2) \sum_{v \in V(G)} \gamma_v w(v).$$

For fixed ϵ, the expected runtime is poly$(|V(G)|)$.

Before proving Proposition 6.7, we discuss how it compares to Theorem 5.4 and Theorem 5.5 from Chapter 5. (Recall Theorem 5.4 is the LP formulation of Theorem 5.2.) Recall that Theorem 5.4 states that there exists a PIT of weight at least $\tau_w = \sum\limits_{v \in V(G)} \gamma_v w(v)$ and $\mathcal{P}_w$ has the condition $\sum\limits_{u \in N(v)} \gamma_u \leq \frac{1}{2}$ for all $v \in V(G)$. Thus, Proposition 6.7 shows that we can find (in expected polynomial time for fixed ϵ) an IT of G at some cost to the weight of the IT. However, $\mathcal{A}_5$ is more applicable than the deterministic algorithm FindWeightPIT of Theorem 5.5 as the conditions of $\mathcal{P}_w^\delta$ do not depend on r. Hence there is no requirement that the graphs G be r-claw-free.

We now prove Proposition 6.7.

Proof of 6.7: Let the algorithm $\mathcal{A}_5$ be as defined in 6.3.1.

The subroutine BlowUp takes as inputs a graph G, vector γ, and parameter ϵ and "blows-up" G with independent sets into a graph G' as follows. For each vertex $v \in V(G)$, BlowUp creates $\lceil t\gamma_v \rceil$ new vertices in $V(G')$ where $t = \left\lceil \frac{|V(G)|}{\epsilon^3} \right\rceil$. The map $f : V(G') \to V(G)$ is defined by setting $f(u)$ to be the unique vertex $v \in V(G)$ such that u is created from $v \in V(G)$. The edge set $E(G')$ is the set $\{u_1u_2 : f(u_1)f(u_2) \in E(G)\}$. For $U \subseteq V(G')$, we write $f(U) = \{f(u) : u \in U\}$. Thus $(V_1', \dots, V_m')$ is the vertex partition of G' where $V_i' = f^{-1}(V_i)$ and w' is the weight function defined by $w'(u) = w(f(u))$.

6.3.1 $\mathcal{A}_5$

Input: A parameter $0 < \epsilon < \frac{1}{2}$, graph G, vertex partition $(V_1, \ldots, V_m)$, weight function $w : V(G) \to \mathbb{R}^{\geq 0}$, and vector γ in the polytope $\mathcal{P}_w^\delta$ where $\delta = \frac{1}{2} - \epsilon$.
Output: An IT M of weight at least $(1 - 2\epsilon^2) \sum\limits_{v \in V(G)} \gamma_v w(v)$.

```
1: function $\mathcal{A}_5(G; \epsilon; \gamma; w; V_1, \ldots, V_m)$
2:     $\delta := \frac{1}{2} - \epsilon$
3:     $(G', f) :=$ BlowUp$(G; \gamma; \epsilon)$
4:     for $v \in V(G')$ do
5:         $w'(v) := w(f(v))$
6:     for $i = 1, \ldots, m$ do
7:         $V_i' := \{v \in V(G') : f(v) \in V_i\}$
8:         $V_i^* :=$ the set of $t$ vertices of $V_i'$ of largest weight
9:     $G^* := G\left[\bigcup_{i=1}^{m} V_i^*\right]$
10:    $\epsilon' := \frac{\epsilon}{1-\epsilon}$
11:    $M := \mathcal{A}_4(G^*; \epsilon'; w'; V_1^*, \ldots, V_m^*)$
12:    return $f(M)$
```

Consider some vertex $u \in V(G')$ with $f(u) = v \in V(G)$. Since γ satisfies $\mathcal{P}_w^\delta$, we have

$$\deg(u) = \sum_{x \in N(v)} \lceil t\gamma_x \rceil \leq \sum_{x \in N(v)} (t\gamma_x + 1) \leq |V(G)| + t \sum_{x \in N(v)} \gamma_x \leq |V(G)| + t\delta.$$

Similarly, the size of each vertex class V_i' satisfies

$$|V_i'| = \sum_{x \in V_i} \lceil t\gamma_x \rceil \leq \sum_{x \in V_i} (t\gamma_x + 1) \leq |V(G)| + t,$$

and

$$|V_i'| = \sum_{x \in V_i} \lceil t\gamma_x \rceil \geq \sum_{x \in V_i} t\gamma_x = t.$$

Thus discarding the lowest-weight $|V_i'| - t$ vertices from each vertex class V_i' is well-defined and the resulting vertex classes V_i^* have size exactly $b = t$. Hence G^* is well-defined and has vertex partition $(V_1^*, \ldots, V_m^*)$ with $|V_i^*| = b$ for each i. Moreover, since vertices were only discarded, we have $\Delta(G^*) \leq \Delta(G') \leq |V(G)| + t\delta$.

For $\epsilon' = \frac{\epsilon}{1-\epsilon} > 0$ and $\epsilon < \frac{1}{2}$, we have

$$\frac{b}{\Delta(G^*)} \geq \frac{t}{|V(G)| + t\delta} = \frac{1}{\frac{|V(G)|}{t} + \delta} \geq \frac{1}{\epsilon^3 + (\frac{1}{2} - \epsilon)} \geq \frac{1}{\frac{1}{2} - \frac{\epsilon}{2}} = 2(1+\epsilon').$$

By applying $\mathcal{A}_4$ on inputs ϵ', G^*, $(V_1^*, \dots, V_m^*)$, and w', we get an IT M of G^* with $w'(M^*) \geq \frac{w'(V(G^*))}{b}(1 - (\epsilon')^2)$. The corresponding set $f(M)$ is an IT of G with weight $w(f(M)) = w'(M)$.

Because the vertex classes V_i' originally had size at most $|V(G)| + t$, discarding the excess vertices from each V_i' reduces the weight by a factor of at most $\frac{|V(G)|}{|V(G)|+t}$, and so $w'(V(G^*)) \geq \left(\frac{t}{|V(G)|+t}\right) w'(V(G'))$. Thus,

$$w'(V(G')) = \sum_{u \in V(G')} w'(u) = \sum_{v \in V(G)} \lceil t\gamma_v \rceil w(v) \geq t \sum_{v \in V(G)} \gamma_v w(v).$$

We therefore have

$$w(M) \geq \frac{w'(V(G^*))}{b}(1 - (\epsilon')^2) \geq \frac{t}{|V(G)| + t}(1 - (\epsilon')^2) \sum_{v \in V(G)} \gamma_v w(v).$$

By our choice of t, we have $\frac{|V(G)|}{t} \leq \epsilon^3$. Simple calculations then show that the above is at least $(1 - 2\epsilon^2) \sum\limits_{v \in V(G)} \gamma_v w(v)$.

For fixed ϵ, we know $t \leq \text{poly}(|V(G)|)$ and so $|V(G^*)| \leq \text{poly}(|V(G)|)$. Furthermore, ϵ' is fixed. Hence the runtime of $\mathcal{A}_4$ and of the overall algorithm $\mathcal{A}_5$, is $\text{poly}(|V(G)|)$. $\square$

We finish by removing the undesired factor of $(1 - 2\epsilon^2)$.

Theorem 6.8. *There exists a randomised algorithm $\mathcal{A}_6$ which takes as inputs a parameter $0 < \delta < \frac{1}{2}$, a graph G, vertex partition $(V_1, \dots, V_m)$, and weight function $w : V(G) \to \mathbb{R}$ where $\mathcal{P}_w^\delta \neq \emptyset$ and returns an IT M of G with $w(M) \geq \tau_w^\delta$. For fixed δ, the expected runtime is* $\text{poly}(|V(G)|)$.

Proof: Let the algorithm $\mathcal{A}_6$ be as defined in 6.3.2.

The algorithm $\mathcal{A}_6$ aims to apply $\mathcal{A}_5$ to a subgraph of G under a different weight function. It begins by initialising ϵ finding a $\gamma \in \mathcal{P}_w^\delta$ with $\sum\limits_{v \in V(G)} \gamma_v w(v) = \tau_w^\delta$. Note that this

6.3.2 $\mathcal{A}_6$

```
Input: A parameter $0 < \delta < \frac{1}{2}$, graph $G$, vertex partition $(V_1, \dots, V_m)$, weight function
    $w : V(G) \to \mathbb{R}$ where $\mathcal{P}_w^\delta \neq \emptyset$.
Output: An IT $M$ of weight at least $\tau_w^\delta$.
 1: function $\mathcal{A}_6(G; \delta; w; V_1, \dots, V_m)$
 2:     $\epsilon := \frac{1}{2} - \delta$
 3:     $\gamma :=$ an optimal solution to $\mathcal{P}_w^\delta$
 4:     for $i = 1, \dots, m$ do
 5:         Sort the vertices of $V_i$ by non-increasing order of weight and label them
            $v_{i,1}, \dots, v_{i,t_i}$ for $t_i = |V_i|$.
 6:         $s_i :=$ smallest index such that $\sum_{k=1}^{s_i} \gamma_{i,k} \geq 1 - \epsilon$ (where $\gamma_{i,k}$ is the entry for $v_{i,k}$)
 7:         $V_i' := \{v_{i,1}, \dots, v_{i,s_i}\}$
 8:     $G' := G\left[\bigcup_{i=1}^{m} V_i'\right]$
 9:     for all $v \in V(G')$ do
10:         $w'(v) := w(v) - w(v_{i,s_i})$ where $v \in V_i'$
11:         $\gamma'_{i,j} := \begin{cases} \frac{\gamma_{i,j}}{1-\epsilon} & \text{if } j < s_i \\ 1 - \frac{\sum_{k=1}^{s_i-1} \gamma_{i,k}}{1-\epsilon} & \text{if } j = s_i \end{cases}$ (where $\gamma'_{i,j}$ is the entry for $v = v_{i,j}$ in $\gamma'$)
12:     $\epsilon' := \frac{\epsilon}{2-2\epsilon}$
13:     $M := \mathcal{A}_5(G'; \epsilon'; \gamma'; w'; V_1', \dots, V_m')$
14:     return $M$
```

takes poly($|V(G)|$) time since $\mathcal{P}_w^\delta$ has poly($|V(G)|$) constraints. It then sorts the vertices in each vertex class in non-increasing order of weight and labels the vertices in V_i with $v_{i,1}, v_{i,2}, \ldots, v_{i,t_i}$ so that $w(v_{i,1}) \geq w(v_{i,2}) \geq \cdots \geq w(v_{i,t_i})$. Also, we use $\gamma_{i,j}$ to denote the entry for vertex $v_{i,j}$ in γ.

Since $\sum\limits_{v \in V_i} \gamma_v = 1$, each V_i has a smallest index s_i such that

$$\sum_{k=1}^{s_i} \gamma_{i,k} \geq 1 - \epsilon.$$

Thus the induced subgraph G' of G obtained by removing from each V_i all of the vertices $v_{i,j}$ with $j > s_i$ is well defined, as are the subsets $V_i' = V_i \cap V(G')$ and weight function $w' : V(G') \to \mathbb{R}^{\geq 0}$ defined by $w'(v) = w(v) - w(v_{i,s_i})$ where $v \in V_i'$. (By the sorted order of the vertices, we have $w(v) \geq w(v_{i,s_i})$ for each vertex $v \in V(G')$ with $v \in V_i'$.)

Let $\gamma' \in [0,1]^{V(G')}$ be the vector given by

$$\gamma'_{i,j} = \begin{cases} \frac{\gamma_{i,j}}{1-\epsilon} & \text{if } j < s_i \\ 1 - \frac{\sum\limits_{k=1}^{s_i-1} \gamma_{i,k}}{1-\epsilon} & \text{if } j = s_i, \end{cases}$$

where $\gamma'_{i,j}$ denotes the entry for vertex $v_{i,j}$ in γ'. Note that $\gamma'_{i,s_i} \geq 0$ by the choice of s_i.

Claim 6.9. *$\gamma' \in \mathcal{P}_{w'}^{\delta'}$, where $\delta' = \frac{\frac{1}{2}-\epsilon}{1-\epsilon}$.*

Proof: The constraint $\sum\limits_{k=1}^{s_i} \gamma'_{i,k} = 1$ for each $i \in \{1, \ldots, m\}$ follows from the definition of γ'_{i,s_i}. For the constraint on the neighbourhood of a vertex u, note that for all i, j, we have

$$\gamma'_{i,j} \leq \frac{\gamma_{i,j}}{1-\epsilon}.$$

This is clear for $j < s_i$. To see it holds for $j = s_i$, note that

$$\begin{aligned}
\frac{\gamma'_{i,j}}{\gamma_{i,j}} &= \frac{1-\epsilon-\sum\limits_{k=1}^{s_i-1}\gamma_{i,k}}{(1-\epsilon)\gamma_{i,s_i}} \\
&= \frac{1-\epsilon-\sum\limits_{k=1}^{s_i}\gamma_{i,k}+\gamma_{i,s_i}}{(1-\epsilon)\gamma_{i,s_i}} \\
&\leq \frac{\gamma_{i,s_i}}{(1-\epsilon)\gamma_{i,s_i}} \\
&= \frac{1}{1-\epsilon}.
\end{aligned}$$

Since $\gamma'_{i,j} \leq \frac{\gamma_{i,j}}{1-\epsilon}$ and $\gamma \in \mathcal{P}_w^\delta$, we therefore have that $\gamma' \in \mathcal{P}_{w'}^{\delta'}$. □

Simple analysis shows that $\delta' = \frac{1}{2} - \epsilon'$ for $\epsilon' = \frac{\epsilon}{2-2\epsilon}$ and that $0 < \epsilon' < \frac{1}{2}$ (recall $\epsilon < \frac{1}{2}$). Thus by Proposition 6.7, $\mathcal{A}_5$ on inputs ϵ', G', $(V'_1, \dots, V'_m)$, γ', and w' returns an IT M of G' with

$$w'(M) \geq (1-2(\epsilon')^2)\sum_{v\in V(G')}\gamma'_v w'(v) = \frac{1-2(\epsilon')^2}{1-\epsilon}\sum_{i=1}^{m}\sum_{j=1}^{s_i-1}\gamma_{i,j}(w(v_{i,j})-w(v_{i,s_i})),$$

where we omit the summand $j = s_i$ since $w'(v_{i,s_i}) = 0$. For fixed δ, we know δ' is fixed and so $\mathcal{A}_5$ runs in expected time poly$(|V(G'|)$, which is poly$(|V(G)|)$.

Since $\epsilon < \frac{1}{2}$, we have $\frac{1-2(\epsilon')^2}{1-\epsilon} = \frac{2-4\epsilon+\epsilon^2}{2(1-\epsilon)^3} \geq 1$, and so $w'(M) \geq \sum\limits_{i=1}^{m}\sum\limits_{j=1}^{s_i-1}\gamma_{i,j}(w(v_{i,j})-w(v_{i,s_i}))$. Since M is an IT of G', M is an IT of G and

$$\begin{aligned}
w(M) &= \sum_{i=1}^{m}\sum_{v\in M\cap V_i} w'(v) + w(v_{i,s_i}) \\
&= w'(M) + \sum_{i=1}^{m} w(v_{i,s_i}) \\
&\geq \sum_{i=1}^{m}\left(w(v_{i,s_i}) + \sum_{j=1}^{s_i-1}\gamma_{i,j}(w(v_{i,j})-w(v_{i,s_i}))\right) \\
&= \sum_{i=1}^{m}\left(w(v_{i,s_i}) + \sum_{j=1}^{t_i}\gamma_{i,j}(w(v_{i,j})-w(v_{i,s_i})) - \sum_{j=s_i}^{t_i}\gamma_{i,j}(w(v_{i,j})-w(v_{i,s_i}))\right).
\end{aligned}$$

Since $\sum_{j=1}^{t_i} \gamma_{i,j} = 1$ and $\sum_{i=1}^{m}\sum_{j=1}^{t_i} \gamma_{i,j} w(v_{i,j}) = \tau_w^\delta$, this is equal to

$$\tau_w^\delta - \sum_{i=1}^{m}\sum_{j=s_i}^{t_i} \gamma_{i,j}(w(v_{i,j}) - w(v_{i,s_i})).$$

As $w(v_{i,j}) \le w(v_{i,s_i})$ for $j > s_i$, this is at least τ_w^δ. Since each step of $\mathcal{A}_6$ runs in time poly($|V(G)|$) or expected time poly($|V(G)|$) for fixed δ, the entire algorithm runs in expected time poly($|V(G)|$) for fixed δ. □

As a simple corollary of Theorem 6.8, we obtain Theorem 6.10.

Theorem 6.10. *There exists a randomised algorithm* FindWeightIT *which takes as inputs a parameter* $0 < \epsilon < 1$*, a graph* G*, a vertex partition* $(V_1, \dots, V_m)$ *such that* $|V_i| = b \ge (2+\epsilon)\Delta(G)$ *for each* i*, and a weight function* $w : V(G) \to \mathbb{R}$*, and finds an IT in* G *with weight at least* $\frac{w(V(G))}{b}$*. For fixed* ϵ*, the expected runtime is* poly($|V(G)|$).

Proof: Let the algorithm FindWeightIT be as defined in 6.3.3.

6.3.3 FindWeightIT

Input: A parameter $0 < \epsilon < 1$, graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| = b \ge (2+\epsilon)\Delta(G)$ for each i, and a weight function $w : V(G) \to \mathbb{R}$.
Output: An IT M of weight at least $\frac{w(V(G))}{b}$.

```
1: function FindWeightIT($G; \epsilon; w; V_1, \dots, V_m$)
2:     $M := \mathcal{A}_6\left(G; \frac{1}{2+\epsilon}; w; V_1, \dots, V_m\right)$
3:     return $M$
```

The algorithm FindWeightIT simply applies $\mathcal{A}_6$ to G, $(V_1, \dots, V_m)$, w, and $\delta = \frac{1}{2+\epsilon}$. Clearly $0 < \delta < \frac{1}{2}$, so the statement holds by Theorem 6.8 assuming $\tau_w^\delta \ge \frac{w(V(G))}{b}$. It therefore suffices to show this.

Note that $\gamma_v = \frac{1}{b}$ is a solution to $\mathcal{P}_w^\delta$ since for every vertex v,

$$\sum_{u \in N(v)} \gamma_u = \sum_{u \in N(v)} \frac{1}{b} \le \frac{\Delta(G)}{b} \le \frac{1}{2+\epsilon} = \delta.$$

Also, for each V_i,

$$\sum_{v \in V_i} \gamma_v = \sum_{v \in V_i} \frac{1}{b} = 1.$$

Thus $\tau_w^\delta \geq \sum\limits_v \gamma_v w(v) = \frac{w(V(G))}{b}$. □

As a corollary of Theorem 6.10, we can also find a randomised algorithm that takes as inputs a parameter $0 < \delta < \frac{1}{2}$, a graph G, vertex partition $(V_1, \ldots, V_m)$, and weight function $w : V(G) \to \mathbb{R}$ and finds a PIT in G with weight at least τ_w^δ. For fixed δ, the expected runtime is $\text{poly}(|V(G)|)$. This shows another analogue of Theorem 5.2.

6.4 Fractional Strong Colourings

In this section, we discuss an application of FindWeightIT from Theorem 6.10. We begin by discussing the relationship between weighted ITs and *fractional strong colourings*. This relationship was also studied by Aharoni, Berger, and Ziv [3], and we will discuss their results in this section. First, we formally define fractional strong colourings.

Definition 6.11. Let G be a graph and $\mathcal{V}$ a vertex partition of G with classes of size b. A *fractional strong colouring (of G with respect to $\mathcal{V}$)* is a function f that assigns to each IT M of G a real number $0 \leq f(M) \leq 1$, such that:

1. for every $v \in V(G)$, we have $\sum\limits_{M \ni v} f(M) \geq 1$, and
2. $\sum\limits_{M} f(M) \leq b$.

Note that if such an f exists, both inequalities in Definition 6.11 must hold with equality. Furthermore, if $f(M) \in \{0, 1\}$ for each IT M, then f is a strong colouring of G with respect to $(V_1, \ldots, V_m)$ (see Definition 6.12 in Section 4.4).

Definition 6.12. A graph G is *fractionally strongly b-colourable* if for every vertex partition $\mathcal{V}$ of G into classes of size b, G has a fractional strong colouring with respect to $\mathcal{V}$.

As mentioned in the introduction of this chapter (as well as in Section 4.4), it has been conjectured (see [3]) that the strong chromatic number is bounded above by $\Delta(G)$ for every graph G. In [3], Aharoni, Berger, and Ziv proved the fractional version of the strong colouring conjecture, which we state in Theorem 6.13.

Theorem 6.13 ([3])**.** *Every graph G is fractionally strongly $2\Delta(G)$-colourable.*

However, the proof of Theorem 6.13 is not fully algorithmic. In this section, we prove the following algorithmic statement.

Theorem 6.14. *There exists a randomised algorithm* FindManyITs *that takes as input parameters $0 < \epsilon, \eta < 1$, a graph G, and a vertex partition $(V_1, \ldots, V_m)$ such that $|V_i| = b \geq (2+\epsilon)\Delta(G)$ for all i. It returns a collection $\mathcal{M}$ of ITs of G such that each $v \in V(G)$ is in at least $\frac{(1-\eta)|\mathcal{M}|}{b}$ and at most $\frac{(1+\eta)|\mathcal{M}|}{b}$ of the elements of $\mathcal{M}$. For fixed ϵ, the expected runtime is* $\text{poly}(|V(G)|, \frac{1}{\eta})$.

Theorem 6.14 can be viewed as an algorithmic approximation to fractional strong colouring as follows. Let $0 < \epsilon, \eta < 1$, G be a graph, and $(V_1, \ldots, V_m)$ be a vertex partition such that $|V_i| = b \geq (2+\epsilon)\Delta(G)$ for each i. Apply FindManyITs to obtain a collection $\mathcal{M}$ of ITs of G. Let $f(M) = \frac{b}{|\mathcal{M}|}$ for each $M \in \mathcal{M}$ and $f(M) = 0$ for all other ITs M. This would give:

1. for every $v \in V(G)$, $1 - \eta \leq \sum\limits_{M \ni v} f(M) \leq 1 + \eta$, and
2. $\sum\limits_{M} f(M) = b$.

Hence f is an approximation of a fractional strong colouring.

We now prove Theorem 6.14, using FindWeightIT from Theorem 6.10.

Proof of Theorem 6.14: Let the algorithm FindManyITs be as defined in 6.4.1.

By Theorem 6.10, for fixed ϵ each iteration of this procedure runs in expected time $\text{poly}(|V(G)|)$ and the number of iterations is $t \leq \text{poly}(|V(G)|, \frac{1}{\eta})$.

Fix $v \in V(G)$ and let j denote the number of distinct i for which $v \in M_i$. We need to verify that j is close to $\frac{t}{b}$. To see this, note that $w_{t+1}(V(G)) \geq w_{t+1}(v) \geq (1+\alpha)^{t-j}$. However, we have $w_1(V(G)) = bm$ and for each $1 \leq i \leq t$, we have

$$\begin{aligned} w_{i+1}(V(G)) &= (1+\alpha)w_i(V(G)) - \alpha w_i(M_i) \\ &\leq (1+\alpha)w_i(V(G)) - \alpha \frac{w_i(V(G))}{b} \\ &= w_i(V(G))\left(1 + \alpha - \frac{\alpha}{b}\right). \end{aligned}$$

Thus $w_{t+1}(V(G)) \leq |V(G)| \left(1 + \alpha - \frac{\alpha}{b}\right)^t$. Hence

$$(1+\alpha)^{t-j} \leq w_{t+1}(V(G)) \leq |V(G)| \left(1 + \alpha - \frac{\alpha}{b}\right)^t.$$

6.4.1 FindManyITs

```
Input: A graph $G$, parameters $0 < \epsilon, \eta < 1$, and vertex partition $(V_1, \dots, V_m)$ such that
    $|V_i| = b \geq (2+\epsilon)\Delta(G)$ for each $i$.
Output: A collection $\mathcal{M}$ of ITs of $G$.
1: function FindManyITs($G; \epsilon; \eta; V_1, \dots, V_m$)
2:     $\alpha := \frac{\eta}{2b}$
3:     $t := \left\lceil \frac{2b^2 \log |V(G)|}{\alpha\eta} \right\rceil$
4:     Initialise $w_1(v) := 1$ for each $v \in V(G)$.
5:     for $i = 1, \dots, t$ do
6:         $M_i :=$ FindWeightIT($G; \epsilon; w_i; V_1, \dots, V_m$)
7:         for all $v \in V(G)$ do
8:             $w_{i+1}(v) := \begin{cases} w_i(v) & \text{if } v \in M_i \\ (1+\alpha)w_i(v) & \text{if } v \notin M_i. \end{cases}$
9:     return $\mathcal{M} = \{M_1, \dots, M_t\}$
```

After cross-multiplying and using the bound $(1+x) \leq e^x$, we have that

$$1 \leq (1+\alpha)^j |V(G)| \left(1 - \frac{\alpha}{b(1+\alpha)}\right)^t \leq |V(G)| e^{\alpha j - \frac{\alpha t}{b(1+\alpha)}}.$$

Taking logarithms, this implies that $\frac{j}{t} \geq \frac{1}{b(1+\alpha)} - \frac{\log |V(G)|}{\alpha t}$. By choice of α and t, we have $\frac{1}{1+\alpha} \geq 1 - \alpha = 1 - \frac{\eta}{2b}$ and $\frac{\log |V(G)|}{\alpha t} \leq \frac{\eta}{2b^2}$. Therefore,

$$\frac{j}{t} \geq \frac{1 - \frac{\eta}{2b}}{b} - \frac{\eta}{2b^2} = \frac{1 - \frac{\eta}{b}}{b} \geq \frac{1-\eta}{b},$$

as claimed.

To show the upper bound on j, note that the other $b-1$ vertices in Vclass(v) appear in total $t-j$ times, which implies that one of the vertices appears at most $\frac{t-j}{b-1}$ times. However, we have already shown that each vertex appears at least $\frac{t(1-\eta/b)}{b}$ times. Thus

$$\frac{t-j}{b-1} \geq \frac{t(1-\eta/b)}{b},$$

which further implies that

$$\frac{j}{t} \leq \frac{b\eta + b - \eta}{b^2} \leq \frac{1+\eta}{b}.$$ □

6.5 Strong Colourings

In this section, we discuss the implications of Theorem 6.10 to strong colourings. To review the definition of strong colouring and discussion of results related to strong colouring, we refer the reader to Section 4.4. We include only a brief review of the necessary results of [3] here.

The proof of Aharoni, Berger, and Ziv in [3] showed that the strong chromatic number $s\chi(G)$ satisfies $s\chi(G) \leq 3\Delta(G)$ using Theorem 4.35, a modification of Theorem 2.3. Recall that Theorem 4.35 gives a sufficient condition for the existence of an IT containing a specified vertex. In Section 4.4, we proved Corollary 4.36, an algorithmic version of Theorem 4.35, using Theorem 3.1. We now use Theorem 6.10 to prove a different algorithmic version of Theorem 4.35, stated as Corollary 6.15.

Corollary 6.15. *There exists a randomised algorithm* RandITv *that takes as input a parameter* $0 < \epsilon < 1$*, graph* G*, vertex partition* $(V_1, \ldots, V_m)$ *with* $|V_i| \geq (2+\epsilon)\Delta(G)$ *for each* i*, and an arbitrary* $v \in V(G)$*, and finds an IT in* G *that contains* v*. For fixed* ϵ*, the expected runtime of* RandITv *is* $\mathrm{poly}(|V(G)|)$*.*

Proof: Let the algorithm RandITv be as defined in 6.5.1.

The algorithm RandITv uses FindWeightIT on a subgraph of G using a special weight function to ensure v is in the set M returned by FindWeightIT. We begin by verifying that G', ϵ, w, and $(V_1', \ldots, V_m')$ are valid inputs for FindWeightIT before explaining why v is necessarily in M.

Note that for each vertex class V_i, V_i' is a subset of V_i such that $|V_i'| = b$. Moreover, $v \in V_j'$ for $V_j = \mathrm{Vclass}(v)$.

For the weight function w on $V(G')$ given by $w(v) = 1$ and $w(u) = 0$ for all $u \neq v$, we claim that $\tau_w^\delta > 0$ for $\delta = \frac{1}{2+\epsilon}$. To see this, note that setting $\gamma_u = \frac{1}{b}$ for each $u \in V_i'$ defines a feasible solution to $\mathcal{P}_w^\delta$ and has $\sum\limits_{u \in V(G')} \gamma_u w(u) = \gamma_v = \frac{1}{b}$.

Thus by Theorem 6.10, FindWeightIT on inputs ϵ, G', $(V_1', \ldots, V_m')$, and w generates an IT M with $w(M) \geq \tau_w^\delta > 0$ in expected time $\mathrm{poly}(|V(G)|)$ for fixed ϵ. Since v is the only vertex with non-zero weight, we therefore have $v \in M$. □

As seen in Section 4.4, Corollary 4.36 led to an algorithmic result for strong colouring. Corollary 6.15 also gives an algorithmic result for strong colouring, which is the following.

6.5.1 RandITv

Input: A parameter $0 < \epsilon < 1$, graph G, vertex partition $(V_1, \dots, V_m)$ such that $|V_i| \geq (2+\epsilon)\Delta(G)$ for each i, and vertex v.

Output: An IT of G that contains v.

```
1:  **function** RandITv($G; \epsilon; v; V_1, \dots, V_m$)
2:      $b := \lceil (2+\epsilon)\Delta \rceil$
3:      **for** $i = 1, \dots, m$ **do**
4:          **if** $v \in V_i$ **then**
5:              $V_i'$ := a subset of $V_i$ containing $v$ with size $b$
6:          **else**
7:              $V_i'$:= a subset of $V_i$ of size $b$
8:      $G' := \left[\bigcup_{i=1}^{m} V_i'\right]$
9:      **for** $u \in V(G')$ **do**
10:         **if** $u = v$ **then**
11:             $w(u) := 1$
12:         **else**
13:             $w(u) := 0$
14:     $M$ := FindWeightIT $(G'; \epsilon; w; V_1', \dots, V_m')$
15:     **return** $M$
```

Corollary 6.16. *There exists a randomised algorithm* RandStrongColour *that takes as input a parameter* $0 < \epsilon < 1$*, a graph* G *and a vertex partition* $(V_1, \ldots, V_m)$ *such that* $|V_i| = b \geq (3+\epsilon)\Delta(G)$*, and returns a strong* b*-colouring of* G *with respect to* $(V_1, \ldots, V_m)$*. For fixed* ϵ*, the expected runtime of* RandStrongColour *is* $\text{poly}(|V(G)|)$*.*

Similar to the proof of Corollary 4.34, the proof of Theorem 6.16 uses the same technique as the proof of Theorem 4.33 from [3] with Corollary 6.15 instead of Theorem 4.35. Since we give a detailed description of this colouring technique in Section 4.4, we omit the description here.

Proof: Let RandStrongColour be as defined in 6.5.2.

6.5.2 RandStrongColour

Input: A graph G, parameter $0 < \epsilon < 1$, and a vertex partition $(V_1, \ldots, V_m)$ where $|V_i| = b \geq (3+\epsilon)\Delta(G)$ for each i.

Output: A strong b-colouring of G with respect to $(V_1, \ldots, V_m)$.

```
1:  function RandStrongColour($G; \epsilon; V_1, \ldots, V_m$)
2:      for all $v \in V(G)$ do $c(v) := 0$
3:      while there is a vertex $v$ such that $c(v) = 0$ do
4:          Choose a vertex $v$ such that $c(v) = 0$.
5:          Choose a colour $\alpha$ missing on Vclass($v$).
6:          $(W, V'_1, \ldots, V'_m) :=$ RemoveColoured($G; c; \alpha; V_1, \ldots, V_m$)
7:          $G' := G\left[\bigcup_{i=1}^{m} V'_i\right]$
8:          $M :=$ RandITv($G'; \epsilon; v; V'_1, \ldots, V'_m$)
9:          $c :=$ ReColour($c; \alpha; M; W; V_1, \ldots, V_m$)
10:     return $c$
```

Note that RandStrongColour is the same algorithm as StrongColour in 4.4.1, but with colours in $\{1, \ldots, b\}$, $G' = G\left[\bigcup_{i=1}^{m} V'_i\right]$, and $M :=$ RandITv$(G'; \epsilon; v; V'_1, \ldots, V'_m)$ instead of colours in $\{1, \ldots, 3\Delta+1\}$, $G' = G\left[\{v\} \cup \left(\bigcup_{j \neq i} V'_j\right)\right]$, and $M :=$ FindITorBD$(G'; \{v\}, V'_1, \ldots, V'_{i-1}, V'_{i+1}, \ldots, V_m)$. All other steps are the same.

Note that this change to G' does not drastically affect the number of operations needed to find G' (still $O(|V(G)|^2)$). Thus it suffices to show that G' and $(V'_1, \ldots, V'_m)$ satisfy the conditions for RandITv.

Recall that RemoveColoured in StrongColour (and therefore in RandStrongColour) removes at most $\Delta(G)$ vertices from each vertex class (namely the vertices sharing a colour with the neighbours of the vertex coloured α in the class). As the vertex classes of the vertex partition of G have size $b \geq (3+\epsilon)\Delta(G)$, we have that the resulting vertex classes have size $b - \Delta(G) \geq (2+\epsilon)\Delta(G)$. By taking the graph induced on these vertex classes (G') and the vertex v, we have that RandITv returns an IT containing v. For fixed ϵ, the expected runtime is poly($|V(G)|$).

As these are the only changes, RandStrongColour returns a strong b-colouring of G in expected time poly($|V(G)|$) for fixed ϵ. □

Chapter 7

Concluding Remarks and Future Work

In this thesis, we introduced three new algorithms that find ITs (or PITs) efficiently in large classes of graphs. The first algorithm, called FindITorBD, was used to prove Theorem 3.1 and Corollary 3.2. These two results are algorithmic versions of the domination (Theorem 2.4) and maximum degree (Theorem 2.3) existence results of [48, 49], which are known to be best possible existence results of their type. Moreover, Theorem 3.1 and Corollary 3.2 require only a slight strengthening of the hypotheses of Theorems 2.4 and 2.3.

While the runtime of FindITorBD is polynomial in the number of vertices, it is exponential in the parameters r and ϵ used to define FindITorBD. Thus FindITorBD is only efficient when r and ϵ are fixed. The dependence on ϵ seems unavoidable, however we are not certain of the nature or even the necessity of the dependence on r. The requirement that the input graph G be r-claw-free was needed because of our choice of signature vector. It is an interesting open question whether this condition is essential. If the dependence on r could be avoided or if the condition on G being r-claw-free could be substantially weakened, then algorithmic versions of more applications of Theorems 2.4 and 2.3 would be possible (see e.g. [44]).

The second algorithm, called FindWeightPIT, efficiently finds PITs of weight at least $\tau_w^{r,\epsilon}$, where $\tau_w^{r,\epsilon}$ is a parameter of vertex-weighted graphs (see Section 5.2). FindWeightPIT was used to prove Theorem 5.5, an algorithmic version of a result of Aharoni, Berger, and Ziv from [3] (Theorem 5.2). Theorem 5.2 generalises Theorem 2.4 to vertex-weighted graphs, similar to the way the theory of weighted matchings generalising Hall's theorem (see Claim 5.3 in Section 5.1). The main application of FindWeightPIT given in this thesis was to the strong colouring problem (via FindWeightIT), but we expect there are other applications of FindWeightPIT to other problems for vertex-weighted graphs.

The last algorithm, called FindWeightIT, finds ITs of weight at least $\frac{w(V(G))}{b}$ in vertex-weighted graphs G that are partitioned into classes of size $b \geq (2+\epsilon)\Delta(G)$. FindWeightIT can also be used to find PITs with weight at least $(\tau')^{\delta}_{w}$, where $(\tau')^{\delta}_{w}$ is the largest objective value to $(\mathcal{P}')^{\delta}_{w}$, the LP obtained by replacing the constraint $\sum\limits_{v \in V_i} \gamma_v = 1$ in $\mathcal{P}^{\delta}_{w}$ with $\sum\limits_{v \in V_i} \gamma_v \leq 1$. This proves a different algorithmic version of Theorem 5.2.

Unlike FindITorBD and FindWeightPIT, the algorithm FindWeightIT is a randomised algorithm. Hence FindWeightIT runs in expected polynomial time. The benefit of the randomisation in FindWeightIT is that we are able to remove the dependence of the run-time on $\Delta(G)$ from FindWeightPIT by using an algorithmic version of the LLL. Thus FindWeightIT can be applied to graphs where $\Delta(G)$ is unbounded, whereas the deterministic algorithms FindITorBD and FindWeightPIT cannot.

The fact that FindWeightIT can find both weighted ITs and PITs algorithmically opens the door to randomised algorithms for various applications of Theorem 2.3 in weighted graphs. However, FindWeightIT cannot be applied to applications that use Theorem 2.4 and not Theorem 2.3 (such as those in Section 4.3). It would be interesting to explore if other randomisation techniques can be used to develop a randomised algorithm for Theorem 2.4. This would in turn create randomised algorithmic results for more applications of FindITorBD.

In Chapters 4 and 6, we presented algorithmic proofs of some applications of Theorems 2.4 and 2.3. In particular, we proved algorithmic versions of results for matchings in bipartite hypergraphs (Section 4.1), circular edge-colourings (Section 4.2), hitting sets for maximum cliques (Section 4.3), strong colourings (Sections 4.4 and 6.5), fractional strong colourings (Section 6.4), and colourings with bounded sized monochromatic components (Section 4.5). However, Theorems 4 and 6 have been used in many other areas, and we expect there will be many more interesting applications.

A longer-term goal would be to find an algorithm based on the topological proofs of Theorems 2.4 and 2.3. These topological proofs use the notion of *topological connectedness* (see [2]). There are other criteria that guarantee the existence of an IT for which only a topological proof is known (see [4, 2]). These other criteria also have many applications, so an algorithm based on the topological proof of Theorems 2.4 and 2.3 may also be useful. This, however, would require a very different approach than the work done in this thesis.

References

[1] D. Achlioptas and F. Iliopoulos. Random walks that find perfect objects and the Lovász local lemma. *Journal of the ACM*, 63(3):22:1–22:29, 2016.

[2] R. Aharoni and E. Berger. The intersection of a matroid and a simplicial complex. *Transactions of the American Mathematical Society*, 358(11):4895–4917, 2006.

[3] R. Aharoni, E. Berger, and R. Ziv. Independent systems of representatives in weighted graphs. *Combinatorica*, 27(3):253–267, 2007.

[4] R. Aharoni and P. Haxell. Hall's theorem for hypergraphs. *Journal of Graph Theory*, 35(2):83–88, 2000.

[5] N. Alon. The linear arboricity of graphs. *Israel Journal of Mathematics*, 62(3):311–325, 1988.

[6] N. Alon. The strong chromatic number of a graph. *Random Structures and Algorithms*, 3(1):1–7, 1992.

[7] N. Alon. Problems and results in extremal combinatorics–I. *Discrete Mathematics*, 273(1-3):31–53, 2003.

[8] N. Alon and V. Asodi. Edge colouring with delays. *Combinatorics, Probability and Computing*, 16(2):173191, 2007.

[9] N. Alon, G. Ding, B. Oporowski, and D. Vertigan. Partitioning into graphs with only small components. *Journal of Combinatorial Theory, Series B*, 87(2):231–243, 2003.

[10] N. Alon and J. Spencer. *The Probabilistic Method*. Wiley, 3rd edition, 2008.

[11] C. Annamalai. Finding perfect matchings in bipartite hypergraphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, pages 1814–1823, 2016.

[12] C. Annamalai. *Algorithmic advances in allocation and scheduling.* PhD thesis, ETH Zurich, 2017.

[13] C. Annamalai. Finding perfect matchings in bipartite hypergraphs. *Combinatorica*, 38(6):1285–1307, 2018.

[14] A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, volume 5171 of *Lecture Notes in Computer Science*, pages 10–20, 2008.

[15] A. Asadpour, U. Feige, and A. Saberi. Santa claus meets hypergraph matchings. *ACM Transactions on Algorithms*, 8(3):24:1–24:9, 2012.

[16] M. Axenovich and R. Martin. On the strong chromatic number of graphs. *SIAM Journal of Discrete Mathematics*, 20(3):741–747, 2006.

[17] J. Beck. An algorithmic approach to the Lovász local lemma. I. *Random Structures and Algorithms*, 2(4):343–365, 1991.

[18] R. Berke. *Coloring and Transversals of Graphs.* PhD thesis, ETH Zurich, 2008.

[19] R. Berke and T. Szabó. Relaxed two-coloring of cubic graphs. *Journal of Combinatorial Theory, Series B*, 97(4):652–668, 2007.

[20] R. Bissacot, R. Fernández, A. Procacci, and B. Scoppola. An improvement of the Lovász local lemma via cluster expansion. *Combinatorics, Probability and Computing*, 20(5):709–719, 2011.

[21] B. Bollobás, P. Erdős, and E. Szemerédi. On complete subgraphs of r-chromatic graphs. *Discrete Mathematics*, 13(2):97–107, 1975.

[22] J. Bondy and P. Hell. A note on the star chromatic number. *Journal of Graph Theory*, 14(4):479–482, 1990.

[23] J. Britnell, A. Evseev, R. Guralnick, P. Holmes, and A. Maróti. Sets of elements that pairwise generate a linear group. *Journal of Combinatorial Theory, Series A*, 115(3):442 – 465, 2008.

[24] R. Brooks. On colouring the nodes of a network. *Mathematical Proceedings of the Cambridge Philosophical Society*, 37(2):194–197, 1941.

[25] K. Chandrasekaran, N. Goyal, and B. Haeupler. Deterministic algorithms for the Lovász local lemma. *SIAM Journal on Computing*, 42(6):2132–2155, 2013.

[26] D. Christofides, K. Edwards, and A. King. A note on hitting maximum and maximal cliques with a stable set. *Journal of Graph Theory*, 73(3):354–360, 2013.

[27] A. Czumaj and C. Scheideler. Coloring non-uniform hypergraphs: A new algorithmic approach to the general Lovász local lemma. *Random Structures and Algorithms*, 17(3-4):213–237, 2000.

[28] R. Diestel. *Graph Theory*. Springer, 4th edition, 2010.

[29] K. Diks and P. Stanczyk. Perfect matching for biconnected cubic graphs in $O(n \log^2 n)$ time. In *SOFSEM 2010: Theory and Practice of Computer Science*, volume 5901 of *Lecture Notes in Computer Science*, pages 321–333, 2010.

[30] K. Edwards and G. Farr. On monochromatic component size for improper colourings. *Discrete Applied Mathematics*, 148(1):89–105, 2005.

[31] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *ISAAC 2010: Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 403–414, 2010.

[32] P. Erdős and L. Lovász. Problems and results on 3-chromatic hypergraphs and some related questions. *Colloquia Mathematica Societatis János Bolyai*, 10:609–627, 1975.

[33] L. Esperet and G. Joret. Colouring planar graphs with three colours and no large monochromatic components. *Combinatorics, Probability and Computing*, 23(4):551–570, 2014.

[34] M. Fellows. Transversals of vertex partitions in graphs. *SIAM Journal of Discrete Mathematics*, 3(2):206–215, 1990.

[35] M. Fischer and M. Ghaffari. Sublogarithmic distributed algorithms for Lovász local lemma, and the complexity hierarchy. In *31st International Symposium on Distributed Computing*, DISC '17, pages 18:1–18:16, 2017.

[36] H. Fleischner and M. Stiebitz. A solution to a colouring problem of P. Erdős. *Discrete Mathematics*, 101(1-3):39–48, 1992.

[37] A. Graf, D. Harris, and P. Haxell. Algorithms for weighted independent transversals and strong colouring. in progress.

[38] A. Graf and P. Haxell. Finding independent transversals efficiently. arXiv: 1811.02687, 2018.

[39] B. Haeupler, B. Saha, and A. Srinivasan. New constructive aspects of the Lovász local lemma. *Journal of the ACM*, 58(6):28:1–28:28, 2011.

[40] A. Hajnal. A theorem on k-saturated graphs. *Canadian Journal of Mathematics*, 17:720–724, 1965.

[41] D. Harris. private communication.

[42] D. Harris. Lopsidependency in the Moser-Tardos framework: Beyond the lopsided Lovász local lemma. *ACM Transactions on Algorithms*, 13(1):17:1–17:26, 2016.

[43] D. Harris. Derandomizing the Lovász local lemma via log-space statistical tests. arXiv: 1807.06672, 2018.

[44] D. Harris and A. Srinivasan. Constraint satisfaction, packet routing, and the Lovász local lemma. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 685–694, 2013.

[45] D. Harris and A. Srinivasan. A constructive algorithm for the Lovász local lemma on permutations. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pages 907–925, 2014.

[46] D. Harris and A. Srinivasan. Algorithmic and enumerative aspects of the Moser-Tardos distribution. *ACM Transactions on Algorithms*, 13(3):33:1–33:40, 2017.

[47] D. Harris and A. Srinivasan. A constructive Lovász local lemma for permutations. *Theory of Computing*, 13(17):1–41, 2017.

[48] P. Haxell. A condition for matchability in hypergraphs. *Graphs and Combinatorics*, 11(3):245–248, 1995.

[49] P. Haxell. A note on vertex list colouring. *Combinatorics, Probability and Computing*, 10(4):345–347, 2001.

[50] P. Haxell. On the strong chromatic number. *Combinatorics, Probability, and Computing*, 13(6):857–865, 2004.

[51] P. Haxell. An improved bound for the strong chromatic number. *Journal of Graph Theory*, 58(2):148–158, 2008.

[52] P. Haxell. On forming committees. *The American Mathematical Monthly*, 118(9):777–788, 2011.

[53] P. Haxell, T. Szabó, and G. Tardos. Bounded size components–partitions and transversals. *Journal of Combinatorial Theory, Series B*, 88(2):281–297, 2003.

[54] C. Hierholzer and C. Wiener. Ueber die möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren. *Mathematische Annalen*, 6(1):30–32, 1873.

[55] F. Jaeger and T. Swart. Conjecture 1. In *Combinatorics 79, Part II*, volume 9 of *Annals of Discrete Mathematics*, page 305. 1980.

[56] G. Jin. Complete subgraphs of r-partite graphs. *Combinatorics, Probability and Computing*, 1(3):241–250, 1992.

[57] T. Kaiser, D. Král, and R. Škrekovski. A revival of the girth conjecture. *Journal of Combinatorial Theory, Series B*, 92(1):41–53, 2004.

[58] T. Kaiser, D. Král', R. Škrekovski, and X. Zhu. The circular chromatic index of graphs of high girth. *Journal of Combinatorial Theory, Series B*, 97(1):1 – 13, 2007.

[59] R. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Complexity of Computer Computations. Springer, 1972.

[60] A. King. *Claw-free graphs and two conjectures on ω, Δ, and χ*. PhD thesis, McGill University, 2009.

[61] A. King. Hitting all maximum cliques with a stable set using lopsided independent transversals. *Journal of Graph Theory*, 67(4):300–305, 2011.

[62] A. King and B. Reed. Bounding χ in terms of ω and δ for quasi-line graphs. *Journal of Graph Theory*, 59(3):215–228, 2008.

[63] A. King, B. Reed, and A. Vetta. An upper bound for the chromatic number of line graphs. *European Journal of Combinatorics*, 28(8):2182–2187, 2007.

[64] M. Kochol. Snarks without small cycles. *Journal of Combinatorial Theory, Series B*, 67(1):34–47, 1996.

[65] K. Kolipaka and M. Szegedy. Moser and Tardos meet Lovász. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing*, STOC '11, pages 235–244, 2011.

[66] K. Kolipaka, M. Szegedy, and Y. Xu. A sharper local lemma with improved applications. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 7408 of *Lecture Notes in Computer Science*, pages 603–614, 2012.

[67] A. Kostochka. Degree, density, and chromatic number of graphs. *Metody Diskretnogo Analiza*, 35:45–70, 1980.

[68] D. Král, E. Máčajová, J. Mazák, and J.-S. Sereni. Circular edge-colorings of cubic graphs with girth six. *Journal of Combinatorial Theory, Series B*, 100(4):351–358, 2010.

[69] Michael Krivelevich. Almost perfect matchings in random uniform hypergraphs. *Discrete Mathematics*, 170(1):259 – 263, 1997.

[70] N. Linial, J. Matoušek, O. Sheffet, and G. Tardos. Graph coloring with no large monochromatic components. *Electronic Notes in Discrete Mathematics*, 29:115–122, 2007.

[71] C. Liu and S. Oum. Partitioning H-minor free graphs into three subgraphs with no large components. *Journal of Combinatorial Theory, Series B*, 128:114–133, 2018.

[72] P. Loh and B. Sudakov. On the strong chromatic number of random graphs. *Combinatorics, Probability and Computing*, 17(2):271–286, 2008.

[73] L. Lovász. On decomposition of graphs. *Studia Scientiarum Mathematicarum Hungarica*, 1:237–238, 1966.

[74] B. Mohar, B. Reed, and D. Wood. Colourings with bounded monochromatic components in graphs of given circumference. *Australasian Journal of Combinatorics*, 69(2):236–242, 2017.

[75] M. Molloy. The list chromatic number of graphs with small clique number. *Journal of Combinatorial Theory, Series B*, 134:264–284, 2019.

[76] M. Molloy and B. Reed. Further algorithmic aspects of the local lemma. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 524–529, 1998.

[77] M. Molloy and B. Reed. *The Strong Chromatic Number*, pages 61–65. Graph Colouring and the Probabilistic Method. Springer, 2002.

[78] R. Moser. A constructive proof of the Lovász local lemma. In *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, STOC '09, pages 343–350, 2009.

[79] R. Moser and G. Tardos. A constructive proof of the general Lovász local lemma. *Journal of the ACM*, 57(2):11:1–11:15, 2010.

[80] A. Panconesi and A. Srinivasan. Randomized distributed edge coloring via an extension of the chernoff–hoeffding bounds. *SIAM Journal on Computing*, 26(2):350–368, 1997.

[81] W. Pegden. An extension of the Moser–Tardos algorithmic local lemma. *SIAM Journal on Discrete Mathematics*, 28(02):911–917, 2014.

[82] J. Petersen. Die theorie der regulären graphs. *Acta Mathematica*, 15:193–220, 1891.

[83] L. Rabern. A note on Reed's conjecture. *SIAM Journal of Discrete Mathematics*, 22(2):820–827, 2008.

[84] L. Rabern. On hitting all maximum cliques with an independent set. *Journal of Graph Theory*, 66(1):32–37, 2011.

[85] J. Ramírez-Alfonsín and B. Reed, editors. *Perfect graphs.* Wiley, 2001.

[86] B. Reed. ω, δ, and χ. *Journal of Graph Theory*, 27(4):177–212, 1998.

[87] A. Schrijver. Bipartite edge coloring in $o(\delta m)$ time. *SIAM Journal on Computing*, 28(3):841–846, 1998.

[88] J. Spencer. Asymptotic lower bounds for Ramsey functions. *Discrete Mathematics*, 20:69–76, 1977.

[89] A. Srinivasan. Improved algorithmic versions of the Lovász local lemma. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '08, pages 611–620, 2008.

[90] M. Stiebitz, D. Scheide, B. Toft, and L. Favrholdt. *Graph Edge Coloring: Vizing's Theorem and Goldberg's Conjecture.* Wiley, 2012.

[91] T. Szabó and G. Tardos. Extremal problems for transversals in graphs with bounded degree. *Combinatorica*, 26(3):333–351, 2006.

[92] A. Vince. Star chromatic number. *Journal of Graph Theory*, 12(4):551–559, 1988.

[93] V. Vizing. On an estimate of the chromatic class of a p-graph. *Diskret. Analiz.*, 3:25–30, 1964.

[94] D. Wood. Defective and clustered graph colouring. *The Electronic Journal of Combinatorics*, #DS23, 2018.

[95] R. Yuster. Independent transversals in r-partite graphs. *Discrete Mathematics*, 176(1–3):255–261, 1997.

[96] X. Zhu. Circular chromatic number: a survey. *Discrete Mathematics*, 229(1–3):371–410, 2001.

[97] X. Zhu. Recent developments in circular colourings of graphs. In *Topics in Discrete Mathematics*, volume 26 of *Algorithms and Combinatorics*, pages 497–550. 2006.

Appendix A

Circular Colouring Details

In this appendix, we include some of the details omitted from Section 4.2.1. In particular, we provide the colouring of an octopus (due to Kaiser, Král, and Škrekovski) from [57] in Appendix A.1. We also provide a more detailed analysis of the runtime of the various tasks of CircularP in Appendix A.2. We refer the reader to Section 4.2.1.1 for all definitions and terms used in this appendix.

A.1 An Outline of the Colouring

In this section, we provide Kaiser, Král, and Škrekovski's method to colour the blocks of an octopus. As we are only concerned with making Kaiser, Král, and Škrekovski's result fully algorithmic, we include only the actual colouring technique. The proof of the correctness of this colouring can be found in [57].

Assume that G is a cubic bridgeless graph with girth $g \geq f(p)$ for $f(p)$ as in Corollary 4.9, F is a fixed 1-factor of G, and o is an octopus with a $2p$-segment as its head.

Claim A.1 ([57])**.** *Suppose a block B of an octopus contains at least $4k+1$ vertices and has its input edge coloured k or $3p-k$ for some $0 \leq k \leq \lceil \frac{p}{2} \rceil - 1$. Then any partial colouring that colours one contact edge adjacent to B with p and the other contact edge with $2p$ can be extended to the inner edges and output edges of B.*

The extension of the partial colouring of Lemma A.1 is as follows. If $k = 0$, start at the contact edge coloured p and colour the inner edges of B by alternating between $2p$ and p, starting with the colour $2p$. Next, colour every output edge of B with 0.

If $k \geq 1$, let Q be the path on $4k+1$ vertices in B whose centre vertex is incident with the input edge of B. If $Q \neq B$, colour the inner edges of B between the contact edge coloured $2p$ and Q by alternating between p and $2p$, starting with the edge adjacent to the contact edge and colouring it p. Then, colour the inner edges of B between the contact edge coloured p and Q by alternating between p and $2p$, starting with the edge adjacent to the contact edge and colouring it $2p$. Next, colour the output edges of B that are not incident with a vertex in Q with 0.

We now colour Q and the output edges incident with vertices in Q in one of two ways. If the input edge of B is coloured k, colour the edges of Q starting with the edge adjacent to the edge coloured p as follows:

$$2p+1, p+1, 2p+2, p+2, \ldots, 2p+k, p+k,$$

$$2p+k, p+(k-1), 2p+(k-1), p+(k-2), \ldots, 2p+1, p.$$

Starting with the output edge that is incident with the end of Q that is incident with edges coloured p and $2p+1$, colour the output edges of B incident with vertices in Q as follows:

$$0, 0, 1, 1, \ldots, k-1, k-1,$$

$$k-1, k-1, k-2, k-2, \ldots, 0, 0.$$

Figure A.1 depicts this colouring.

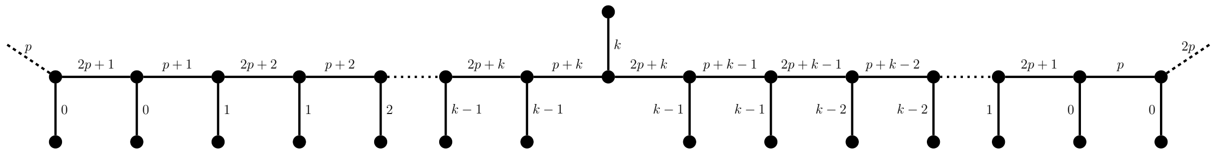

Figure A.1: The colouring of block B when the input edge is coloured k.

If the input edge of B is coloured $3p-k$, colour the edges of Q starting with the edge adjacent to the edge coloured $2p$ as follows:

$$p-1, 2p-1, p-2, 2p-2, \ldots, p-k, 2p-k,$$

$$p-k, 2p-(k-1), p-(k-1), 2p-(k-2), \ldots, p-1, 2p.$$

Starting with the output edge that is incident with the end of Q that is incident with edges coloured $2p$ and $p-1$, colour the output edges of B incident with vertices in Q as follows:

$$3p, 3p, 3p-1, 3p-1, \ldots, 3p-(k-1), 3p-(k-1),$$

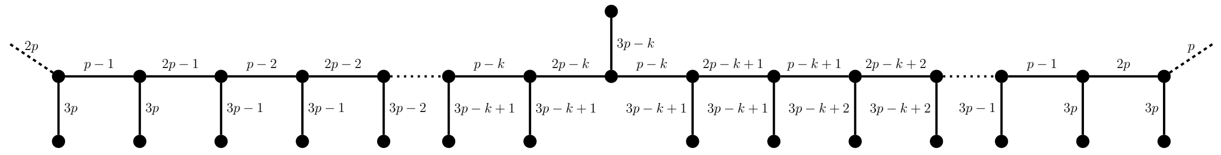

Figure A.2: The colouring of block B when the input edge is coloured $3p - k$.

$$3p-(k-1), 3p-(k-1), 3p-(k-2), 3p-(k-2), \dots, 3p, 3p.$$

Figure A.2 depicts this colouring.

It is easy to check that the above colourings extend the partial colouring to a proper circular $(3p+1)/p$-edge-colouring of B for any $k \in \{0, \dots, \lceil \frac{p}{2} \rceil - 1\}$. Also, it is clear that this extension avoids the colours $k, \dots, 3p-k$ if $k \geq 1$ and avoids the colours $1, \dots, 3p-1$ if $k = 0$ on the output edges of B.

Note that for $\ell \geq 2$, if p is even, then

$$2p + 5 - 4\ell = 4\left(\tfrac{p}{2} + 1 - \ell\right) + 1,$$

and if p is odd, then

$$2p + 7 - 4\ell = 4\left(\tfrac{p+1}{2} + 1 - \ell\right) + 1.$$

Hence for each $\ell \geq 2$, the blocks in $\mathcal{P}(\ell)$ contain $4\left(\lceil \frac{p}{2} \rceil + 1 - \ell\right) + 1$ vertices. We now use this to show that the colouring of Claim A.1 will extend the partial colouring to all of the inner edges of octopus o, thus proving Lemma 4.14 (restated in Lemma A.2).

Lemma A.2 ([57])**.** *Suppose c is a partial proper circular $(3p+1)/p$-edge-colouring of G and o is an octopus in G such that c colours the contact edges of o by p and $2p$ so that each pair of contact edges adjacent to the same block receive opposite colours and c does not colour any of the inner edges of o. Then c can be extended to a proper circular $(3p+1)/p$-edge-colouring of o.*

The extension of the partial colouring is as follows. We first extend c to the head of o and then continue by, for each $2 \leq \ell \leq \lceil \frac{p}{2} \rceil + 1$, extending the colouring of all blocks of $\mathcal{P}(\ell - 1)$ to a colouring of all blocks of $\mathcal{P}(\ell)$ using the colouring of Claim A.1.

Starting with the edge adjacent to the contact edge coloured p, colour the inner edges of the head as follows:

$$2p+1, p+1, 2p+2, p+2, \dots, \left\lfloor \tfrac{5p}{2} \right\rfloor, \left\lfloor \tfrac{3p}{2} \right\rfloor,$$

$$\left\lfloor \tfrac{p}{2} \right\rfloor, \left\lfloor \tfrac{3p}{2} \right\rfloor + 1, \left\lfloor \tfrac{p}{2} \right\rfloor + 1, \left\lfloor \tfrac{3p}{2} \right\rfloor + 2, \left\lfloor \tfrac{p}{2} \right\rfloor + 2, \dots, p-2, 2p-1, p-1.$$

Then, starting with the edge of F incident with the end of the head that is incident with edges coloured p and $2p+1$, colour the $2p$ edges of F incident with the vertices of the head as follows:

$$0, 0, 1, 1, \dots, \left\lfloor \frac{p}{2} \right\rfloor - 1, \left\lfloor \frac{p}{2} \right\rfloor - 1,$$
$$\left\lfloor \frac{5p}{2} \right\rfloor + 1, \left\lfloor \frac{5p}{2} \right\rfloor + 1, \left\lfloor \frac{5p}{2} \right\rfloor + 2, \left\lfloor \frac{5p}{2} \right\rfloor + 2, \dots, 3p, 3p.$$

Figure A.3 depicts this colouring.

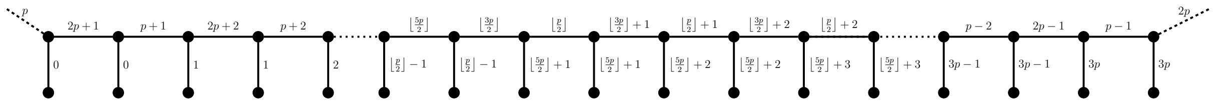
p
$2p+1$
$p+1$
$2p+2$
$p+2$
$\lfloor \frac{5p}{2} \rfloor$
$\lfloor \frac{3p}{2} \rfloor$
$\lfloor \frac{p}{2} \rfloor$
$\lfloor \frac{3p}{2} \rfloor + 1$
$\lfloor \frac{p}{2} \rfloor + 1$
$\lfloor \frac{3p}{2} \rfloor + 2$
$\lfloor \frac{p}{2} \rfloor + 2$
$p-2$
$2p-1$
$p-1$
$2p$
0
0
1
1
2
$\lfloor \frac{p}{2} \rfloor - 1$
$\lfloor \frac{p}{2} \rfloor - 1$
$\lfloor \frac{5p}{2} \rfloor + 1$
$\lfloor \frac{5p}{2} \rfloor + 1$
$\lfloor \frac{5p}{2} \rfloor + 2$
$\lfloor \frac{5p}{2} \rfloor + 2$
$\lfloor \frac{5p}{2} \rfloor + 3$
$\lfloor \frac{5p}{2} \rfloor + 3$
$3p-1$
$3p-1$
$3p$
$3p$

Figure A.3: The colouring of the head of octopus o.

For every block B of the octopus, c colours one contact edge adjacent to B with p and the other contact edge $2p$. Also, the blocks in $\mathcal{P}(2)$ contain $4\left(\left\lceil \frac{p}{2} \right\rceil - 1\right) + 1$ vertices and the above colouring colours the input edge of any block in $\mathcal{P}(2)$ either k or $3p-k$ for some $0 \le k \le \left\lceil \frac{p}{2} \right\rceil - 1$. Thus the colouring of Claim A.1 extends the colouring of the head to all of the blocks in $\mathcal{P}(2)$. The output edges of the blocks avoid colours $\left\lceil \frac{p}{2} \right\rceil - 1, \dots, \left\lfloor \frac{5p}{2} \right\rfloor + 1$ and so are coloured either k or $3p-k$ for some $0 \le k \le \left\lceil \frac{p}{2} \right\rceil - 2$.

For each $2 \le \ell \le \left\lceil \frac{p}{2} \right\rceil$, the colouring of the blocks in $\mathcal{P}(\ell)$ by Claim A.1 avoids the colours $\left\lceil \frac{p}{2} \right\rceil + 1 - \ell, \dots, \left\lfloor \frac{5p}{2} \right\rfloor - 1 + \ell$ on their output edges. These output edges are the input edges of the blocks in $\mathcal{P}(\ell+1)$ and the blocks in $\mathcal{P}(\ell+1)$ have $4\left(\left\lceil \frac{p}{2} \right\rceil - \ell\right) + 1$ vertices. Thus the colouring of Claim A.1 extends the current colouring to each block in $\mathcal{P}(\ell+1)$. By applying the colouring of Claim A.1 to every block of $\mathcal{P}(\ell)$ for every level $\ell \in \left\{2, \dots, \left\lceil \frac{p}{2} \right\rceil\right\}$ (in order of level), the result is an extension of c to include a proper circular $(3p+1)/p$-edge-colouring of o.

Thus a partial colouring of G can be extended to octopus o if it colours the contact edges of o with p and $2p$ so that contact edges adjacent to the same block receive distinct colours, which proves Lemma 4.14.

A.2 The Runtime of CircularP

In this section, we verify that for fixed $p \ge 2$, the algorithm CircularP of Section 4.2.1.3 runs in time poly$(V(G))$. We therefore assume p is fixed. Recall that CircularP is an algorithm that takes as input a cubic bridgeless graph G with girth $g \ge f(p)$ and performs the following steps:

1. Find a 1-factor F of G.

2. Form ordered lists of vertices V_i such that each V_i is a unique cycle in $G - F$ and the vertices are stored in V_i in cyclic order.

3. Find an IT M of G' with respect to the vertex partition associated with the vertex classes in S.

4. Find a 2-colouring c of G^* with colours p and $2p$.

5. Determine the set of edges in the octopi of $\mathcal{O}$ and remove the colours c assigned to these edges.

6. Colour the edges of the octopi in $\mathcal{O}$ as in Claim 4.14.

7. Colour the remaining uncoloured edges 0.

8. Return the colouring.

Note that G', G^*, S, and $\mathcal{O}$ are defined in Section 4.2.1.2.

In the next few claims, we analyse the runtime of various tasks performed by CircularP. We put these claims together to prove the runtime of CircularP in Lemma A.6, which completes the proof of Corollary 4.9. For algorithms whose procedures are simple routines, we omit the proofs of their runtimes. Instead, we give a brief outline of what these procedures entail so as to give some intuition for the provided runtimes. Since the runtime of FindITorBD will dominate the runtime of all other subroutines in CircularP, we make no effort to minimise the runtime of these other algorithms.

Claim A.3. *There is an algorithm* CyclePartition *that takes as inputs a cubic bridgeless graph G and a 1-factor F of G and returns, in time $O(|V(G)|)$, a set $\mathcal{V}$ of lists V_i such that:*

(i) each V_i is the vertex set of a unique cycle in $G - F$,

(ii) each cycle of $G - F$ has a list V_i associated with it, and

(iii) for each V_i, the vertices are stored in cyclic order.

Given the graph G and 1-factor F as adjacency lists, the adjacency list of $G - F$ can be found in $O(|V(G)|)$ operations. CyclePartition then chooses a vertex v not already assigned to a list and adds it to a new list V_i. It then appends the first vertex u of the adjacency list of v that is not already in V_i to V_i and repeats this appending operation with u as v until no such vertex exists in the adjacency list of v. Next, CyclePartition chooses a new vertex not already assigned to a list and adds it to a new list V_{i+1}. The process continues until every vertex is contained in some list. As this procedure is simple given the adjacency lists, it can be easily implemented in time $O(|V(G)|)$ and so we omit the details of its runtime.

Claim A.4. *There is an algorithm* MakeGFK *that takes as inputs a cubic bridgeless graph G with girth g, a 1-factor F of G, and an integer $k \in \{1, \dots, g\}$ and returns, in time $O(|V(G)|)$, the graph $G_{F,k}$.*

The algorithm MakeGFK is a bit technical. First, MakeGFK invokes CyclePartition on inputs G and F to obtain the set $\mathcal{V}$ of ordered lists of vertices in $G - F$. Then, MakeGFK creates an adjacency list E that stores u in the list for v if v is incident with the same edge in F as a vertex in the k-segment that starts with vertex u. (Note that the k-segment starting at u is easy to determine given the list of $\mathcal{V}$ containing u.) Next, MakeGFK creates an adjacency list E' that stores u in the list for v if u is in the list of w in E for some w in the k-segment starting at v. (Again, the k-segment starting at v is easy to determine.) This results in E' being an adjacency list where u is in the list of v if and only if two vertices in the k-segments starting at u and v are incident with the same edge in F.

Note that for each $V_i \in \mathcal{V}$, the ordering of the vertices provides a natural bijection f from the vertices in V_i to the k-segments of vertices in V_i. Namely, $V_i[j]$ maps to $(V_i[j], V_i[j+1], \dots, V_i[j+k-1])$ where arithmetic is modulo $|V_i|$. Thus given $\mathcal{V}$ and E', MakeGFK can find $V(G_{F,k}) = \{f(v) : v \in V(G)\}$ and $E(G_{F,k}) = \{f(u)f(v) : uv \in E'\}$ efficiently. All of the above steps can be implemented using adjacency lists in time $O(|V(G)|)$. Hence MakeGFK returns $G_{F,k}$ in time $O(|V(G)|)$ and we omit further details.

Claim A.5. *There is an algorithm* Octopus *that takes as input an integer $p \geq 2$, a cubic bridgeless graph G with girth g as in Corollary 4.9, a 1-factor F of G, and a $2p$-segment h and returns the unique octopus in G whose head is h. For fixed p, the runtime is $O(|V(G)|)$.*

Using the algorithmic definition of an octopus presented in Definition 4.13, it is easy to construct the octopus with head h if the cyclic order of the vertices in $G - F$ is known for all cycles in $G - F$. Thus CyclePartition is a subroutine of Octopus, and the rest of the steps performed by Octopus consist of adding the appropriate vertices and edges of G to o based on the level of the octopus. This requires knowing the neighbour of a vertex that is incident with the same edge of F, which can be found in time $O(1)$ from the adjacency list of F. Hence Octopus can be implemented in time $O(|V(G)|)$ (due to the runtime of CyclePartition), and so we omit the details.

We are now ready to prove Lemma A.6.

Lemma A.6. *For fixed p, the algorithm* CircularP *runs in time polynomial in $|V(G)|$.*

Proof: The steps performed by CircularP are outlined at the start of this section. Also, recall that the input for CircularP is a cubic bridgeless graph G with girth $g \geq f(p)$ as in Corollary 4.9.

CircularP begins by invoking a subroutine that takes as input a cubic bridgeless graph and returns a 1-factor of that graph. In particular, CircularP may use the algorithm of Diks and Stanczyk from [29] on input G, which returns a 1-factor F of G in time $O(|V(G)|[\log(|V(G)|)]^2)$. Next, CircularP uses CyclePartition to find the desired ordered lists of vertices $\mathcal{V}$ in time $O(|V(G)|)$ (Claim A.3).

To find the desired IT of G', CircularP must first construct G'. This can be done by first using MakeGFK to find the graph $G_{F,2p}$ in time $O(|V(G)|)$ (Claim A.4). Then, CircularP finds the graph H, which is either the $(p-1)^{\text{th}}$-power or p^{th}-power of $G_{F,2p}$ depending on the parity of p. By using a breadth first search on the vertices of $G_{F,2p}$, the graph $H = (G_{F,2p})^{p-1}$ or $H = (G_{F,2p})^p$ can be found from $G_{F,2p}$ in time $O(|V(G_{F,2p})||E(G_{F,2p})|) = O(|V(G)|^2)$. Next, CircularP finds the subset of $\mathcal{V}$ corresponding to cycles of odd length in $G - F$ and stores the lists of these cycle's $2p$-segments in a new partition $\mathcal{V}'$. This trivially takes $O(|V(G)|)$ operations to implement. The graph G' is then found by looking at the subgraph of H induced by the $2p$-segments in the lists in $\mathcal{V}'$, which can be determined using $O(|V(G)|^2)$ operations. CircularP then uses FindITorBD (for $r = \Delta(G') + 1$ and $\epsilon = \frac{1}{\Delta(G')}$) on inputs G' and $\mathcal{V}'$ (with $\mathcal{V}'$ interpreted as a vertex partition) to find an IT M of G' in time poly$(|V(G')|)$ (Claim 4.15). As $|V(G')| \leq |V(G)|$, this is poly$(|V(G)|)$.

CircularP continues by finding and 2-colouring the edges of G^*. Given the set of $2p$-segments in M, CircularP finds the set I_H of edges in these segments and defines $G^* = G - (F \cup I_H)$. This takes time $O(|V(G)|)$ using M and $\mathcal{V}$. CircularP then uses a subroutine to 2-colour the edges of G^* using the colours p and $2p$. Note that G^* is bipartite, so any 2-colouring algorithms for bipartite graphs can be used as this subroutine. We choose for each list in $\mathcal{V}$, to alternately colour the edges between consecutive vertices in the list p and $2p$ (starting with p) and ignoring the edges of I_H. This takes time $O(|V(G)|)$.

Next, CircularP finds the set of octopi $\mathcal{O}$ whose heads are in M using Octopus. As $|M| < |V(G)|$, by Claim A.5 the set of octopi in $\mathcal{O}$ can be found in time $O(|V(G)|)$.

Given these octopi, the colours assigned to the edges of octopi in $\mathcal{O}$ can be removed using $O(|V(G)|)$ operations (since the number of edges in the octopi in $\mathcal{O}$ is $O(|V(G)|)$). CircularP then uses the colouring of Lemma 4.14 to recolour the edges of the octopi in $\mathcal{O}$, extending the colouring to the edges of the octopi. This takes time $O(|V(G)|)$. Finally, all remaining uncoloured edges are assigned the colour 0 and the resulting colouring is returned. Both of these tasks require $O(|V(G)|)$ operations.

Hence for fixed p, CircularP terminates in time poly$(|V(G)|)$, with the majority of the runtime being used to find M in G'. □

This completes the proof of Corollary 4.9.

Appendix B

The Runtime of HitCliques

In this appendix, we prove that for fixed $\Delta = \Delta(G)$, the algorithm HitCliques runs in time poly($|V(G)|$). We therefore assume that Δ is fixed. We refer the reader to Section 4.3 for all definitions and terms used in this appendix.

Recall from Section 4.3 that HitCliques is the following algorithm.

B.0.1 HitCliques

```
Input: A graph $G$ with $\omega(G) > \frac{2}{3}(\Delta + 1)$.
Output: An independent set $M$ that meets every maximum clique in $G$.
1: function HitCliques($G$)
2:     $\mathcal{C}$ := MaximumCliques($G$)
3:     $G(\mathcal{C})$ := MakeCliqueGraph($G, \mathcal{C}$)
4:     $\mathcal{V}$ := CliqueIntersect($G, G(\mathcal{C})$)
5:     $G'$ := MakeG'($G, \mathcal{V}$)
6:     $k := \lceil \frac{\Delta+1}{3} \rceil$
7:     $M$ := FindITorBD($G'; \mathcal{V}$) for $r = k$ and $\epsilon = \frac{1}{k-1}$.
8:     return $M$
```

The subroutine MaximumCliques finds the set $\mathcal{C}$ of all maximum cliques in G. The subroutine MakeCliqueGraph uses G and $\mathcal{C}$ to construct the clique graph $G(\mathcal{C})$. CliqueIntersect is an algorithm that, given a graph G and its clique graph $G(\mathcal{C})$, returns a set $\mathcal{V}$ of lists V_i of vertices of G such that each component i of $G(\mathcal{C})$ has a unique V_i and each V_i contains all of the vertices that are in every clique $C \in \mathcal{C}_i$. Given this vertex partition $\mathcal{V}$ and G, MakeG' creates the graph G' induced by the vertices in the union of lists in $\mathcal{V}$. This choice of G' and $\mathcal{V}$ satisfy the conditions of Corollary 4.25 with $k = \lceil \frac{\Delta+1}{3} \rceil$. Thus FindITorBD for this k returns an independent set M that hits all maximum cliques in G.

In the next few claims, we analyse the runtimes of the subroutines used by HitCliques. These claims complete the proof of Corollary 4.23. For algorithms whose procedures are simple routines, we omit detailed proofs of their runtimes. However, we will give a brief outline of what these procedures entail to give some intuition for the provided runtimes. As the runtime of FindITorBD will dominate the runtime of all other subroutines of HitCliques, we make no effort to minimise the runtimes of these other algorithms.

Claim B.1. *For fixed Δ, the set $\mathcal{C}$ of all maximum cliques in G can be found in time $O(|V(G)|)$ and $|\mathcal{C}| = O(|V(G)|)$.*

To find the set of maximum cliques, we first find the set of all maximal cliques in G and then remove the cliques not of maximum size. We can do so by using an algorithm of Eppstein, Löffler, and Strash from [31] that finds the set of all maximal cliques in a graph H of *degeneracy* d in time $O(3^{d/3}d|V(H)|)$. The *degeneracy* of a graph H is the smallest number d such that every subgraph of H contains a vertex of degree at most d. As G is a graph with maximum degree Δ and Δ is a fixed, we clearly have that G is a graph of degeneracy $d \leq \Delta$. Hence the algorithm of [31] will find the set of all maximal (and therefore maximum) cliques in G in time $O(3^{\Delta/3}\Delta|V(G)|)$, which is linear in $|V(G)|$ since Δ is fixed.

Furthermore, Eppstein, Löffler, and Strash [31] showed that in the worst case, there are at most $(|V(H)| - d)3^{d/3}$ maximal cliques in graphs H of degeneracy d. Thus the number of maximal (and therefore maximum) cliques in G is also $O(|V(G)|)$ since $d = \Delta$ is fixed. Hence removing the maximal cliques that are not maximum cliques takes time $O(|V(G)|)$.

Claim B.2. *For fixed Δ, the clique graph $G(\mathcal{C})$ can be constructed from a graph G and its set $\mathcal{C}$ of maximum cliques in time $O(|V(G)|^3)$.*

Suppose we are given G as an adjacency matrix and $\mathcal{C}$ as a $|V(G)| \times |\mathcal{C}|$ matrix X such that column i is the characteristic vector of maximum clique $C_i \in \mathcal{C}$. We will return the clique graph $G(\mathcal{C})$ as an adjacency matrix A with the rows and columns indexed by elements in $\mathcal{C}$. For each entry A_{ij} of A, we define $A_{ij} = 1$ if $i \neq j$ and $X_{vi} = X_{vj} = 1$ for some $v \in V(G)$ and $A_{ij} = 0$ otherwise, i.e. $A_{ij} = 1$ if distinct maximum cliques C_i and C_j intersect in some vertex $v \in V(G)$ and $A_{ij} = 0$ otherwise. There are $|\mathcal{C}|^2$ entries of A and each entry requires at most $|V(G)|$ tests, so A can be determined in time $|\mathcal{C}|^2 \cdot |V(G)| = O(|V(G)|^3)$.

Claim B.3. *For fixed Δ, the sets V_i of vertices of G such that each V_i is the mutual intersection of the cliques in $\mathcal{C}_i$ can be found in time $O(|V(G)|^3)$.*

Suppose we are given G and $G(\mathcal{C})$ as adjacency matrices and $\mathcal{C}$ as a $|V(G)| \times |\mathcal{C}|$ matrix X such that column i is the characteristic vector of maximum clique $C_i \in \mathcal{C}$. Similar to CyclePartition in Section 4.2.1.3, we can form lists of cliques by choosing a clique $C \in \mathcal{C}$ not already assigned to a list and assigning it to a new list U_i. Then for each clique D such that v_C and v_D are adjacent in $G(\mathcal{C})$, add D to U_i. It then repeats this adding operation for each clique D in U_i until no more cliques can be added. We then start the next list U_{i+1} with some clique $C \in \mathcal{C}$ not already assigned to a list and continue until every clique in $\mathcal{C}$ appears in some list. This entire process takes time $O(|\mathcal{C}|)$. Note that we can store each list U_i as a $|\mathcal{C}| \times 1$ characteristic vector and there are at most $|\mathcal{C}|$ lists.

Given these lists U_i, we define the sets V_i as characteristic vectors as follows. For each $C \in U_i$ and each $v \in V(G)$, we check if $X_{vC} = 1$. If so, we set $(V_i)_v = 1$ and set $(V_i)_v = 0$ otherwise. The result is that the vectors V_i indicate the vertices of G that appear in every maximum clique in U_i. These tests take time $|\mathcal{C}| \cdot O(|V(G)| \cdot |\mathcal{C}|)$, so the total time to find the V_i is $O(|\mathcal{C}|) + O(|V(G)| \cdot |\mathcal{C}|^2) = O(|V(G)|^3)$.

Claim B.4. *For fixed Δ, the graph $G' = G\left[\bigcup_{V_i \in \mathcal{V}} V_i\right]$ can be constructed from G and $\mathcal{V}$ in time $O(|V(G)|^2)$.*

Suppose we are given G as an adjacency matrix and $\mathcal{V}$ as a $|V(G)| \times |\mathcal{V}|$ matrix Y such that column i is the characteristic vector of $V_i \in \mathcal{V}$. Then G' can be returned as an adjacency matrix A obtained by taking a copy of G and removing row v and column v from this copy if $Y_{vi} = 0$ for every column i in Y. The only rows and columns left are those of vertices in some $V_i \in \mathcal{V}$, and so the resulting matrix A is the adjacency matrix for G'. Testing each v for $Y_{vi} = 0$ for all columns i in Y takes time $|V(G)| \cdot |\mathcal{V}| = O(|V(G)|^2)$ since $|\mathcal{V}| \leq |\mathcal{C}|$.

By Claims B.1, B.2, B.3, and B.4, the time required to implement all steps of HitCliques besides FindITorBD is $O(|V(G)|^3)$ when Δ is fixed. Thus for fixed Δ, HitCliques runs in time poly$(|V(G)|)$, which completes the proof of Theorem 4.23.

Appendix C

The Runtime of StrongColour

In this appendix, we prove that fixed $\Delta = \Delta(G)$, the the algorithm StrongColour runs in time poly($|V(G)|$). We therefore assume that Δ is fixed. We refer the reader to Section 4.4 for all definitions and terms used in this appendix.

Recall from Section 4.4 that StrongColour is the following algorithm.

C.0.1 StrongColour

Input: A graph G with maximum degree Δ and a vertex partition $(V_1, \dots, V_m)$ where $|V_i| \leq 3\Delta + 1$ for each $1 \leq i \leq m$.

Output: A strong $(3\Delta + 1)$-colouring of G with respect to $(V_1, \dots, V_m)$.

```
1:  function StrongColour($G; V_1, \dots, V_m$)
2:      for all $v \in V(G)$ do $c(v) := 0$
3:      while there is a vertex $v$ such that $c(v) = 0$ do
4:          Choose a vertex $v$ such that $c(v) = 0$.
5:          Choose a colour $\alpha$ missing on Vclass($v$).
6:          $(W, V'_1, \dots, V'_m) :=$ RemoveColoured($G; c; \alpha; V_1, \dots, V_m$)
7:          $\mathcal{V} := \{V'_i : V_i \neq \text{Vclass}(v)\}$
8:          $G' := G\left[\{v\} \cup \left(\bigcup_{V'_j \in \mathcal{V}} V'_j\right)\right]$
9:          $M :=$ FindITorBD($G'; \{v\}, \mathcal{V}$) for $r = \Delta + 1$ and $\epsilon = \frac{1}{\Delta}$.
10:         $c :=$ ReColour($c; \alpha; M; W; V_1, \dots, V_m$)
11:     return $c$
```

The subroutine RemoveColoured returns the set W of vertices assigned colour α as well as subsets $V_i' \subseteq V_i$ of vertices $x \in V_i$ such that $c(x) = 0$ or $c(x) \neq c(y)$ for all $y \in N(w_i)$, where $\{w_i\} = W \cap V_i$ (if such a w_i exists). The subroutine ReColour reassigns $c(w_i)$ to be $c(u_i)$ for each $w_i \in W$ where $w_i \in V_i$ and $\{u_i\} = M \cap V_i$. It then reassigns $c(u_i)$ to be α for each $u_i \in M$ and returns the resulting colouring as c.

In the next few claims, we analyse the runtime of each of the subroutines of StrongColour. As the procedures are simple, we provide a brief outline of what these procedures entail but omit detailed proofs of their runtimes. Also, the runtime of FindITorBD will dominate the runtime of all other subroutines in StrongColour, so we make no effort to minimise the runtimes of these other algorithms.

Claim C.1. *For fixed Δ, finding a missing colour on a vertex class takes time $O(|V(G)|)$.*

Given a vertex class V_i as a characteristic vector and c as a $|V(G)| \times 1$ vector with entries in $\{0, \dots, 3\Delta + 1\}$, checking the entries of c corresponding to the vertices in V_i takes time $|V_i| \leq |V(G)|$. The set C of missing colours on V_i can be returned as a $(3\Delta + 1) \times 1$ characteristic vector where $C_\alpha = 0$ if $c_v = \alpha$ for some $v \in V_i$ and $C_\alpha = 1$ otherwise (note C does not contain an entry for "0" as "0" is not a colour). This clearly takes time $O(|V(G)|)$. Finding one non-zero entry in C takes time $O(\Delta) = O(1)$ since Δ is fixed. Hence finding a missing colour on V_i takes time $O(|V(G)|)$.

Claim C.2. *For fixed Δ,* RemoveColoured *runs in time $O(|V(G)|^2)$.*

Suppose G is given as an adjacency matrix, c as a $|V(G)| \times 1$ vector with entries in $\{0, \dots, 3\Delta + 1\}$, and $V_1, \dots, V_m$ are given as characteristic vectors. Determining the set W of vertices w such that $c_w = \alpha$ requires checking $|V(G)|$ entries of c. We assume W is stored as a characteristic vector of this set. For each $w \in W$, we do the following. We find the neighbours of w by checking row w of G, using $|V(G)|$ operations. Next, we determine c_u for each $u \in N(w)$, which takes $|N(w)| \leq \Delta$ operations. For V_i the vertex class containing w, we define V_i' to be a $|V(G)| \times 1$ vector by $(V_i')_x = 1$ if $c_x \neq c_u$ for all $u \in N(w)$ such that $c_u \neq 0$ and $(V_i')_x = 0$ otherwise. This requires another $O(|V(G)|)$ operations. Hence the vectors $W, V_1, \dots, V_m$ are returned in time $O(|V(G)|) + |W| \times O(|V(G)|) = O(|V(G)|^2)$.

Claim C.3. ReColour *returns the new colouring c in time $O(|V(G)|^2)$.*

Suppose G is given as an adjacency matrix, c as a $|V(G)| \times 1$ vector with entries in $\{0, \dots, 3\Delta + 1\}$, and M, W, $V_1, \dots V_m$ are all given as characteristic vectors. For each $w \in W$, we determine the V_i such that $w \in V_i$ and then find the vertex $u \in M \cap V_i$ and redefine $c_w = c_u$. This takes $O(|V(G)|)$ operations. Thus re-colouring the vertices of W

takes $|W| \cdot O(|V(G)| \leq O(|V(G)|^2)$ operations. We then define $c_u = \alpha$ for all $u \in M$, which takes an additional $m \leq |V(G)|$ operations. Thus the returned vector c is found in time $O(|V(G)|^2) + |V(G)| = O(|V(G)|^2)$.

With these claims, we now prove the remainder of Corollary 4.34.

Lemma C.4. *For fixed* Δ*,* StrongColour *completes in time* $\text{poly}(|V(G)|)$*.*

Proof: Let G be a graph with maximum degree Δ and vertex partition $(V_1, \ldots, V_m)$ where $|V_i| \leq 3\Delta + 1$ for each i. By adding isolated vertices to G and the vertex classes V_i, we assume $|V_i| = 3\Delta + 1$ for each i.

Suppose G is given as an adjacency matrix and $V_1, \ldots, V_m$ are given as characteristic vectors. Choosing a vertex v and determining $\text{Vclass}(v)$ takes time $O(|V(G)|)$. Choosing α takes time $3\Delta + 1 = O(1)$. By Claims C.1 and C.2, finding the missing colours and the sets $W, V_1', \ldots, V_m'$ can be accomplished in time $O(|V(G)|^2)$.

The graph G', stored as an adjacency matrix A, can be found by taking a copy of G and performing the following row and column deletions. Find each $u \neq v$ such that $u \in \text{Vclass}(v)$, remove the row and column indexed by u from the copy of G, which takes time $O(|V(G)|)$. Then for each $V_i \neq \text{Vclass}(v)$, remove the rows and columns indexed by w for each vertex w such that $(V_i)_w \neq (V_i')_w$, which takes an additional $m \cdot O(|V(G)|) = O(|V(G)|^2)$ operations. The resulting matrix A is therefore found in time $O(|V(G)|) + O(|V(G)|^2) = O(|V(G)|^2)$.

As FindITorBD is the algorithm that finds the IT in Corollary 4.36 and $|V(G')| \leq |V(G)|$, M is found in time $p(|V(G)|)$ for some polynomial p since Δ is fixed. By Claim C.3, the re-colouring can be implemented in time $O(|V(G)|^2)$. Hence each iteration of the while loop of StrongColour takes time $O(|V(G)|^2) + p(|V(G)|)$. As the number of uncoloured vertices decreases during each iteration, the while loop has at most $|V(G)|$ iterations. Initialising c takes time $|V(G)|$, so StrongColour completes in time $|V(G)| + |V(G)|(O(|V(G)|^2) + p(|V(G)|))$. Hence StrongColour runs in time $\text{poly}(|V(G)|)$ when Δ is fixed. □

Appendix D

Runtimes for the Monochromatic Component Algorithms

In this appendix, we prove that for fixed $d \geq 3$ and $r \geq 1$, the algorithms SmallComponents, BoundedVert, and BoundedEdge from Section 4.5 run in time poly$(|V(G)|)$. We do so by analysing the runtime of SmallComponents in Appendix D.1 the runtime of BoundedVert in Appendix D.2, and the runtime of BoundedEdge in Appendix D.3. Also, we assume throughout this appendix that r and d are fixed. We refer the reader to Section 4.5 for all definitions and terms used in this appendix.

D.1 The Runtime of SmallComponents

In this section, we prove that for fixed $d \geq 3$ and $r \geq 1$, the algorithm SmallComponents from Section 4.5 runs in time poly$(|V(G)|)$. Recall from Section 4.5 that SmallComponents is the algorithm in D.1.1.

We prove that for fixed d and r, SmallComponents runs in time poly$(|V(G)|)$ (Lemma D.4). To do so, we analyse the runtime of the subroutines of SmallComponents individually in the next few claims. For procedures that are simple, we provide a brief outline of what the procedures entail but omit the proofs of their runtimes. Furthermore, since the runtime of FindITorBD will dominate the runtime of all other subroutines of SmallComponents, we make no attempt to minimise the runtimes of the other algorithms.

Recall that Order arranges the vertices of B_i of degree 2 in $G[B_i]$ in the order of their appearance in the path or cycle of $G[B_i]$, returning this set as B_i'.

D.1.1 SmallComponents

Input: A graph G with maximum degree $\Delta(G) \leq d$, a vertex partition (A, B), and a partition $(B_1, \ldots, B_m)$ of B satisfying the conditions of Corollary 4.43 for some $d \geq 3$ and $r \geq 1$.

Output: A set $M \subseteq B$ such that every component of $G[B_i \setminus M]$ for each $i \in \{1, \ldots, m\}$ and every component of $G[A \cup M]$ has at most $(12r + 6)d - (18r + 24)$ vertices.

```
1:  function SmallComponents($G; A; B; B_1, \ldots, B_m$)
2:      $K := (12r + 6)d - (18r + 24)$
3:      $k := (2r + 1)d - (3r + 4)$
4:      for $i$ such that $|B_i| > K$ do
5:          $B'_i :=$ Order($G, B_i$)
6:          $(\mathcal{B}'_i, B''_i) :=$ SmallerParts($2k + 1, B'_i$)
7:      $V := \bigcup_{|B'_i| > K} B''_i$
8:      $\mathcal{B}' := \bigcup_{|B'_i| > K} \mathcal{B}'_i$
9:      $G' :=$ MakeG'($V; G; A; B_1, \ldots, B_m$)
10:     $M :=$ FindITorBD($G'; \mathcal{B}'$) for $r = k + 1$ and $\epsilon = \frac{1}{k}$.
11:     return $M$
```

Claim D.1. *For fixed $d \geq 3$ and $r \geq 1$,* $\mathrm{Order}(G, B_i)$ *runs in time $O(|V(G)|^2)$.*

Suppose G is given as an adjacency matrix and B_i as a characteristic vector. An ordering of the vertices of degree 2 in $G[B_i]$ that respects the cycle/path order of the vertices in $G[B_i]$ can be returned as follows. First, for each $v \in B_i$, compute the dot product of row v of G with B_i. If the dot product is 1, find the neighbour u_1 of v in B_i and add u_1 as the first vertex of the list B_i'. Find the other neighbour w_1 of u_1 in B_i and, if the dot product of row w_1 of G with B_i is 2, append w_1 to the end of list B_i'. Continue this process with $u_{j+1} = w_j$ until the dot product of row w_j of G with B_i is 1. Return the list B_i' (without appending w_j to it).

If the dot product of v of G with B_i is 2 for all $v \in B_i$, then choose a vertex $v \in B_i$ and add v as the first vertex of list B_i'. Find a neighbour u_1 of v in B_i and append u_1 to the end of list B_i'. Find the other neighbour w_1 of u_1 in B_i and append w_1 to the end of list B_i'. Continue this process with $u_{j+1} = w_j$ until the "other" neighbour of u_j is v. Return the list B_i' (without appending $w_j = v$ to it).

Computing the dot product of a row of G and B_i takes $O(|V(G)|)$ operations. Finding a neighbour $w \in B_i$ of a vertex $u \in B_i$ can be done by testing for each $x \in B_i$, if $G_{ux} = 1$. Hence finding a neighbour takes $O(|V(G)|)$ operations. Since there are $|B_i|$ dot product operations and $|B_i|$ neighbour-finding loops, the list B_i' is returned in time $|B_i| \cdot O(|V(G)|) + |B_i| \cdot O(|V(G)|) = O(|V(G)|^2)$.

The next subroutine, called SmallerParts, uses the ordering of B_i' (for B_i with $|B_i| > K$) to partition B_i' into ordered lists $B_{i,j}'$ of size exactly $2k+1$ (which are returned in the set $\mathcal{B}_i'$) and (a possibly empty) list B_i^* of size at most $2k$ (which is not included in $\mathcal{B}_i'$). The union of the elements in the $B_{i,j}'$ is returned as B_i''.

Claim D.2. *For fixed $d \geq 3$ and $r \geq 1$,* $\mathrm{SmallerParts}(2k+1, B_i')$ *returns a set $B_i'' \subseteq B_i'$ of vertices and a partition $\mathcal{B}_i'$ of B_i'' into classes $B_{i,j}'$ such that $G[B_{i,j}']$ is a path on $2k+1$ vertices for each $B_{i,j}' \in \mathcal{B}'$ in time $O(|V(G)|)$.*

Recall that the B_i' is an ordered list of the vertices of degree 2 in $G[B_i]$ where the order respects the cycle/path order of $G[B_i]$. Thus SmallerParts will look at segments $B_i'[j(2k+1)+1] \dots B_i'[j(2k+1)+2k+1]$ of length $2k+1$ in the list B_i'. For each $j \geq 0$ such that $j(2k+1) \leq |B_i'|$, SmallerParts adds the segment $B_i'[j(2k+1)+1] \dots B_i'[j(2k+1)+2k+1]$ as $B_{i,j}'$ to $\mathcal{B}_i'$ and $\{B_i'[j(2k+1)+1], \dots, B_i'[j(2k+1)+2k+1]\}$ to the set B_i''. This takes $O(1)$ operations for each j since k is constant (recall k is defined using d and r which are fixed constants). Hence the entire SmallerParts takes time $O(|V(G)|)$.

The next subroutine, called MakeG', creates the graph G' whose vertex set is V and whose edge set consists only of edges uv such that either $u, v \in B$ are neighbours of vertices in the same component of $G[A]$ or $uv \in E(G)$ with $\text{Vclass}(u) \neq \text{Vclass}(v)$.

Claim D.3. *For fixed $d \geq 3$ and $r \geq 1$,* MakeG'$(V; G; A; B_1, \ldots, B_m)$ *returns the desired graph G' in time $O(|V(G)|^3)$.*

Suppose G is given as an adjacency matrix and the sets V, A, $B_1, \ldots, B_m$ are all given as characteristic vectors. MakeG' will return the graph G' as a $|V| \times |V|$ adjacency matrix X as follows. The algorithm begins with X as a copy of G. For each $v \in V$, find $\text{Vclass}(v)$ and define the $|V(G)| \times 1$ vector b_v by $(b_v)_u = 0$ if $u \in \text{Vclass}(v) \setminus \{v\}$ and $(b_v)_u = 1$ otherwise. Replace each entry of row v by $X_{vu} \cdot (b_v)_u$. Next, remove all columns and rows corresponding to vertices $u \notin V$. All of these steps can be implemented in time $O(|V(G)|^2)$. The result is $|V| \times |V|$ matrix X such that $X_{uv} = 1$ if and only if $u, v \in V$, $uv \in E(G)$, and $\text{Vclass}(u) \neq \text{Vclass}(v)$, which accounts for all edges of type 2. It remains to account for the edges of type 1.

MakeG' starts by making another copy of G and removing all rows that do not correspond to a vertex in V and columns that do not correspond to a vertex in A. The resulting $|V| \times |A|$ matrix Y takes $O(|V(G)|^2)$ operations to form. Then, for each $X_{uv} = 0$ entry of X, we re-define X_{uv} for each $a \in A$ such that $Y_{va} = 1$ and $b \in A$ such that $Y_{ub} = 1$ to be $X_{uv} = 1$ if either $a = b$ or $G_{ab} = 1$. Note that by condition (i) of 4.43, each component of $G[A]$ has at most two vertices and by condition (iv), each $x \in V$ has neighbours in at most r components of $G[A]$. Hence there are at most $2r$ choices of a and $2r$ choices of b. Thus the testing takes time $O(|V(G)|)$ for each X_{uv}, so accounting for all edges of type 1 takes time $O(|V(G)|^3)$. Therefore G' returned as adjacency matrix X takes time $O(|V(G)|^2) + O(|V(G)|^3) = O(|V(G)|^3)$.

We are now ready to prove Lemma D.4.

Lemma D.4. *For fixed $d \geq 3$ and $r \geq 1$,* SmallComponents *runs in time* $\text{poly}(|V(G)|)$*.*

Proof: As k and K are constants that depend only on d and r, they can be computed in time $O(1)$. Determining whether $|B_i| > K$ for any class B_i takes $O(|V(G)|)$ calculations (dot product of the characteristic vector of B_i and the all ones vector). Similarly, Order and SmallerParts complete in times $O(|V(G)|^2)$ and $O(|V(G)|)$ respectively (Claims D.1 and D.2). As there are at most $m \leq |B| \leq |V(G)|$ sets B_i such that $|B_i| > K$, we have that the for loop in SmallComponents requires a total of $O(|V(G)|^3)$ operations to implement. Defining the sets V and $\mathcal{B}'$ takes time $O(|V(G)|)$. MakeG' returns the graph

G' in time $O(|V(G)|^3)$ by Claim D.3. Hence the steps other than FindITorBD take a total of $O(|V(G)|^3)$ operations to implement.

By Corollary 3.2, FindITorBD returns the IT M of G' in time poly($|V|$) (since k is fixed because d and r are fixed), which is poly($|V(G)|$). Let p be this polynomial. Thus SmallComponents runs in time $O(|V(G)|^3) + p(|V(G)|)$, which is poly($|V(G)|$). □

This completes the proof of Corollary 4.43.

D.2 The Runtime of BoundedVert

In this section, we prove that for fixed $\Delta(G) = \Delta \geq 3$, the algorithm BoundedVert from Section 4.5 runs in time poly($|V(G)|$). Recall from Section 4.5 that BoundedVert is the following algorithm.

D.2.1 BoundedVert

Input: A graph G with maximum degree $\Delta \geq 3$.
Output: A $\lceil \frac{\Delta+2}{3} \rceil$-vertex colouring of G such that the monochromatic components have size at most $K = 12\Delta^2 - 36\Delta + 12$.

```
1:  function BoundedVert($G$)
2:      $h := \lceil \frac{\Delta+2}{3} \rceil$
3:      $(A, B)$ := DegreePartition($G; 1, \Delta - 2$) for $\ell = 2$.
4:      $(B_1, \ldots, B_{h-1})$ := DegreePartition($G[B]; 2, 2, 2, ..., 2$) for $\ell = h - 1$.
5:      for $i = 1, \ldots, h - 1$ do
6:          $\mathcal{C}_i$ := Components($G; B_i$)
7:      $\mathcal{C} := \bigcup_{i=1}^{h-1} \mathcal{C}_i$
8:      $M$ := SmallComponents($G; A; B; \mathcal{C}$) for $d = \Delta$ and $r = \Delta - 2$.
9:      for $i = 1, \ldots, h - 1$ do
10:         for all $v \in B_i \setminus M$ do
11:             $c(v) := i$
12:     for all $v \in A \cup M$ do
13:         $c(v) := h$
14:     return $c$
```

As we saw in the proof of Lemma D.4 Appendix D.1, for fixed $d \geq 3$ and $r \geq 1$, SmallComponents takes time poly($|V(G)|$) to implement, with the majority of the time being used to implement FindITorBD as a subroutine. Thus, the runtime of SmallComponents dominates the runtime of the other steps in BoundedVert. We therefore make no attempt to optimise the runtime of the other subroutines of BoundedVert.

We begin analysing the runtime of BoundedVert by analysing the runtime of DegreePartition. Recall Theorem 4.52 due to Lovász [73], restated below for convenience.

Theorem D.5 ([73]). *Let G be a graph and let $n_1, \dots, n_\ell$ be nonnegative integers such that $n_1 + n_2 + \cdots + n_\ell \geq \Delta(G) - \ell + 1$. Then $V(G)$ can be partitioned into sets $V_1, \dots, V_\ell$ such that $\Delta(G[V_i]) \leq n_i$ for all $1 \leq i \leq \ell$.*

Proposition D.6 gives the original proof of Theorem 4.52, which shows that the vertex partition can be found in polynomial time for fixed Δ and ℓ.

Proposition D.6. *Let G be a graph with maximum degree Δ and let $n_1, \dots, n_\ell$ be nonnegative integers such that $n_1 + n_2 + \cdots + n_\ell \geq \Delta - \ell + 1$. Then for fixed Δ, $V(G)$ can be partitioned into (possibly empty) sets $V_1, \dots, V_\ell$ such that $\Delta(G[V_i]) \leq n_i$ for all $1 \leq i \leq \ell$ in time $O(|V(G)|^3)$.*

Proof: We may assume that $n_i \leq \Delta$ for each $1 \leq i \leq \ell$ as $\Delta(G[V_i]) \leq \Delta$ for every $V_i \subseteq V(G)$. Hence taking $V_i = V(G)$ for an i such that $n_i > \Delta$ and $V_j = \emptyset$ for all $j \neq i$ would satisfy the desired conditions. Also, we may assume $\ell \leq |V(G)|$ as we can ignore the $\ell - |V(G)|$ sets V_i that are guaranteed to be empty.

Consider the following simple algorithm. Take any vertex partition $(V_1, \dots, V_\ell)$ of G. While there is a vertex x such that $\deg_{G[V_i]}(x) > n_i$ (where $V_i = \text{Vclass}(x)$), move x to a vertex class $V_j \neq V_i$ such that $\deg_{G[V_j]}(x) \leq n_j$. This continues until there is no such vertex x. The resulting partition then satisfies $\Delta(G[V_i]) \leq n_i$ for all $1 \leq i \leq \ell$.

Claim D.7. *For any vertex x such that $\deg_{G[V_i]}(x) > n_i$ (where $V_i = \text{Vclass}(x)$), there exists a vertex class $V_j \neq V_i$ such that $\deg_{G[V_j]}(x) \leq n_j$.*

Proof: Let $x \in V_i$ be a vertex such that $\deg_{G[V_i]}(x) > n_i$. Suppose $\deg_{G[V_j]}(x) > n_j$ for all $j \neq i$. Since $\deg(x) \geq \sum_{k=1}^{\ell} \deg_{G[V_k]}(x)$, we have

$$\Delta \geq \deg(x) \geq \sum_{k=1}^{\ell} \deg_{G[V_k]}(x) \geq \sum_{k=1}^{\ell} (n_k + 1) \geq \ell + (\Delta - \ell + 1) = \Delta + 1,$$

which is a contradiction. Hence there must be such a vertex class V_j. □

Claim D.8. *For fixed Δ, the process completes after at most $O(|V(G)|^2)$ iterations of the while loop that moves vertices.*

Proof: Let $\mathcal{V}$ be a vertex partition of G. We define a potential function f by

$$f(\mathcal{V}) = \sum_{i=1}^{\ell}[|E(G[V_i])| - n_i|V_i|].$$

We claim that moving a vertex x such that $\deg_{G[V_i]}(x) > n_i$ (where $V_i = \text{Vclass}(x)$) from V_i to V_j decreases the value of $f(\mathcal{V})$. Let $V_i' = V_i - x$, $V_j' = V_j + x$, and $\mathcal{V}'$ be the resulting vertex partition of G. Then $|E(G[V_i'])| \leq |E(G[V_i])| - (n_i+1)$, $|E(G[V_j'])| \leq |E(G[V_j])| + n_j$, $|V_i'| = |V_i| - 1$, and $|V_j'| = |V_j| + 1$. Hence,

$$\begin{aligned}
f(\mathcal{V}') &= \sum_{k=1}^{\ell}[|E(G[V_k])| - n_k|V_k|] \\
&= [|E(G[V_i'])| - n_i|V_i'|] + [|E(G[V_j'])| - n_j|V_j'|] + \sum_{k\neq i,j}[|E(G[V_k])| - n_k|V_k|] \\
&\leq [|E(G[V_i])| - (n_i+1) - n_i(|V_i|-1)] + [|E(G[V_j])| + n_j - n_j(|V_j|+1)] \\
&+ \sum_{k\neq i,j}[|E(G[V_k])| - n_k|V_k|] \\
&= -1 + \sum_{k=1}^{\ell}[|E(G[V_k])| - n_k|V_k|] \\
&= f(\mathcal{V}) - 1.
\end{aligned}$$

Hence the potential function f always decreases when such a vertex x is moved. Clearly

$$f(\mathcal{V}) \geq \sum_{i=1}^{\ell}(-n_i)|V_i| \geq -|V(G)|\sum_{i=1}^{\ell} n_i,$$

and,

$$f(\mathcal{V}) \leq \sum_{i=1}^{\ell}|E(G)| = \ell|E(G)|.$$

As $n_i \leq \Delta$ for each $1 \leq i \leq \ell$, $\ell \leq |V(G)|$, and $|E(G)| \leq \Delta|V(G)|$, we have that the lower bound on $f(\mathcal{V})$ is $-\Delta|V(G)|^2$ and the upper bound is $\Delta|V(G)|^2$. Hence at most $2\Delta|V(G)|^2$ iterations of the while loop that moves vertices can be performed. □

Defining the initial partition can be completed in time $O(|V(G)|)$. Finding the vertex class V_j of Claim D.8 for each $x \in V(G)$ such that $\deg_{G[V_i]}(x) > n_i$ (where $V_i = \text{Vclass}(x)$) takes time $O(|V(G)|)$ as well. By Claim D.8, the process of moving such an x occurs at most $O(|V(G)|^2)$ times for fixed Δ. Hence the entire algorithm takes time
$O(|V(G)|) + O(|V(G)|^2) \cdot O(|V(G)|) = O(|V(G)|^3)$ for fixed Δ. □

We now analyse the runtime of Components. Recall that Components takes as input a graph G and vertex set S and returns the set of all vertex sets of the components of $G[S]$.

Claim D.9. *For fixed Δ,* Components(G, B_i) *runs in time* $O(|V(G)|^2)$.

Proof: Suppose G is given as an adjacency matrix and B_i as a characteristic vector. Components will return a set of characteristic vectors of the components of $G[B_i]$ as follows. Let $B_i' = B_i$ and $j = -1$. For each $v \in B_i'$, compute the dot product of row v of G with B_i'. If the dot product is 1, set $(B_i')_v = 0$, $j = j+1$, and $B_{i,j}$ to be a $|V(G)| \times 1$ vector with $(B_{i,j})_v = 1$ and $(B_{i,j})_u = 0$ for all $u \neq v$. Then find the neighbour u_1 of v in B_i' and set $(B_i')_{u_1} = 0$ and $(B_{i,j})_{u_1} = 1$. Find the other neighbour w_1 of u_1 in B_i' (recall $\Delta(G[B_i]) \leq 2$ and so u_1 has at most 2 neighbours) and set $(B_i')_{w_1} = 0$ and $(B_{i,j})_{w_1} = 1$. Continue this process with $u_{j+1} = w_j$ until u_{j+1} has no other neighbours in B_i'. Add $B_{i,j}$ to the set $\mathcal{C}_i$ of characteristic vectors of components of $G[B_i]$.

If the dot product of v of G with B_i' is 2 for all $v \in B_i'$, then choose a vertex $v \in B_i'$. Set $(B_i')_v = 0$, $j = j+1$, and $B_{i,j}$ to be a $|V| \times 1$ vector with $(B_{i,j})_v = 1$ and $(B_{i,j})_u = 0$ for all $u \neq v$. Find a neighbour u_1 of v in B_i' and set $(B_i')_{u_1} = 0$ and $(B_{i,j})_{u_1} = 1$. Find the other neighbour w_1 of u_1 in B_i' and set $(B_i')_{w_1} = 0$ and $(B_{i,j})_{w_1} = 1$. Continue this process with $u_{j+1} = w_j$ until u_{j+1} has no other neighbours in B_i'. Add $B_{i,j}$ to the set $\mathcal{C}_i$ of characteristic vectors of components of $G[B_i]$.

Computing the dot product of a row of G and B_i' takes $O(|V(G)|)$ operations. Finding a neighbour $w \in B_i'$ of a vertex $u \in B_i'$ can be done by testing for each $x \in B_i'$, if $G_{ux} = 1$. Hence finding a neighbour takes $O(|V(G)|)$ operations. Since there are $|B_i|$ dot product operations and $|B_i|$ neighbour-finding loops, the set of components $\mathcal{C}_i$ of $G[B_i]$ is returned in time $|B_i| \cdot O(|V(G)|) + |B_i| \cdot O(|V(G)|) = O(|V(G)|^2)$. □

We are now ready to prove that for fixed Δ, BoundedVert runs in time poly$(|V(G)|)$.

Claim D.10. *For fixed Δ,* BoundedVert *runs in time* poly$(|V(G)|)$.

Proof: By Proposition D.6, the vertex partitions (A, B) and $(B_1, \ldots, B_{h-1})$ can be found in time $O(|V(G)|^3)$ since Δ is fixed. By Claim D.9, the set $\mathcal{C}_i$ of vertex sets $B_{i,j}$ of the components of $G[B_i]$ can be found in time $O(|V(G)|^2)$ for each B_i. As there are $h-1 = O(\Delta)$ classes B_i, the set $\mathcal{C}$ of all the $B_{i,j}$ for all i can be found in time $O(|V(G)|^2)$ (since Δ is fixed).

By Corollary 4.43, SmallComponents runs in time poly$(|V(G)|)$ since Δ is fixed. Let p be this polynomial. As it takes 1 operation to assign the appropriate colour to each vertex in $V(G)$, colouring the vertices takes $|V(G)|$ operations to implement. Thus, for fixed Δ, the total runtime of BoundedVert is $O(|V(G)|^3) + p(|V(G)|)$, which is poly$(|V(G)|)$. □

D.3 The Runtime of BoundedEdge

In this section, we prove that for fixed $\Delta(G) = \Delta \geq 3$, the algorithm BoundedEdge from Section 4.5 runs in time poly$(|E(G)|)$. Recall from Section 4.5 that BoundedEdge is the following algorithm.

D.3.1 BoundedEdge

Input: A loopless graph G with maximum degree $\Delta \geq 3$.
Output: A $\lceil \frac{\Delta+1}{2} \rceil$-edge colouring of G such that the monochromatic components have size at most $K = 60\Delta - 60$.

```
function BoundedEdge($G$)
    $h := \lceil \frac{\Delta+1}{2} \rceil$
    $(A, B) :=$ TwoPartition($G$)
    $(B_1, \ldots, B_{h-1}) :=$ TwoFactor($G[B]; h-1$)
    $H :=$ LineGraph($G$)
    for $i = 1, \ldots, h-1$ do
        $\mathcal{C}_i :=$ Components($H; B_i$)
    $\mathcal{C} := \bigcup_{i=1}^{h-1} \mathcal{C}_i$
    $M :=$ SmallComponents($H; A; B; \mathcal{C}$) for $d = 4h-4$ and $r = 2$.
    for $i = 1, \ldots, h-1$ do
        for all $e \in B_i \setminus M$ do
            $c(e) := i$
    for all $e \in A \cup M$ do
        $c(e) := h$
    return $c$
```

Again, BoundedEdge uses SmallComponents as a subroutine. The runtime of

SmallComponents dominates the runtime of the other steps in BoundedEdge, so we again make no attempt to optimise the runtime of the other subroutines of BoundedEdge.

We begin by analysing the runtime of TwoFactor. Recall that TwoFactor is based on the following lemma from [9] that is a reformulation of a theorem due to Petersen [82].

Lemma D.11 ([9])**.** *Let ℓ be an integer and G be a loopless graph with $\Delta(G) \leq 2\ell$. Then there exists an edge partition $(E_1, \dots, E_\ell)$ such that $\Delta(G[E_i]) \leq 2$ for all i.*

TwoFactor is an algorithm that takes as input a parameter ℓ and a graph G with $\Delta(G) \leq 2\ell$ and returns an edge partition $(E_1, \dots, E_\ell)$ of G such that $\Delta(G[E_i]) \leq 2$ for all $1 \leq i \leq \ell$. We can find such a partition when ℓ is fixed in time $O(|E(G)|)$ as follows. For each $v \in V(G)$ with $\deg(v) < 2\ell$, add loops at v. Let G'_1 be the resulting multigraph.

For each $i \in \{1, \dots, \ell\}$, we repeat the following. By construction, G'_i is $2(\ell - i + 1)$-regular. Hence there exists an Euler tour in G'_i, which we can find in time $O(|E(G'_i)|) = O(|E(G)|)$ using Hierholzer's algorithm [54]. Let $v_0 e_0 v_1 \dots e_m v_0$ be this Euler tour. For each v_j and e_j, replace v_j by two vertices v_j^+ and v_j^- and $e_j = v_j v_{j+1}$ by $e_j = v_j^+ v_{j+1}^-$. Let G''_i be the resulting multigraph and note that G''_i is ℓ-regular and bipartite. Thus we can find a 1-factor M_i of G''_i in time $O(\ell|E(G''_i|) = O(|E(G)|)$ (see e.g. [87]). Identify each v_j^+ with v_j^- so that M_i becomes a 2-factor in G'_i. Let E'_i be this 2-factor and $G'_{i+1} = G'_i - E'_i$. Set $E_i = E'_i \cap E(G)$.

Since G'_1 is 2ℓ-regular, $G'_{\ell+1}$ has no edges. For fixed ℓ, each of the above steps takes time $O(|E(G)|)$ since $\Delta(G) \leq 2\ell$. Thus for fixed ℓ, the final edge partition $(E_1, \dots, E_\ell)$ is returned in time $O(|E(G)|)$. Hence for fixed ℓ, TwoFactor can be implemented in time $O(|E(G)|)$.

We now analyse the runtime of TwoPartition. Recall that TwoPartition is based on the following lemma from [9], which can be derived from Lemma 4.55 and is therefore also a reformulation of a theorem due to Petersen [82].

Lemma D.12 ([9])**.** *Every loopless graph G has a set $A \subseteq E(G)$ such that $\Delta(G - A) < \Delta(G)$ and each component of $G(A)$ is a path containing at most two edges.*

TwoPartition is an algorithm that takes as input a loopless graph G with $\Delta(G) = \Delta$ and returns an edge partition (A, B) of G such that $\Delta(G - A) < \Delta$ and the components of $G[A]$ are paths that contain at most two edges. For fixed Δ, we can find such a set A, and thus the edge partition (A, B), in time $O(|E(G)|)$ as follows. Let $\ell = \lceil \frac{\Delta}{2} \rceil$. For each $v \in V(G)$ with $\deg(v) < 2\ell$, add loops at v. Let G' be the resulting multigraph.

As G' is 2ℓ-regular, there exists an Euler tour in G', which we can find in time $O(|E(G')|) = O(|E(G)|)$ using Hierholzer's algorithm [54]. Let $v_0e_0v_1 \dots e_mv_0$ be this Euler tour. For each v_j and e_j, replace v_j by two vertices v_j^+ and v_j^- and $e_j = v_jv_{j+1}$ by $e_j = v_j^+v_{j+1}^-$. Let G'' be the resulting multigraph and note that G'' is ℓ-regular and bipartite. Thus we can find a 1-factor M of G'' in time $O(\ell|E(G''|) = O(|E(G)|)$ (see e.g. [87]). Identify each v_j^+ with v_j^- so that M becomes a 2-factor in G'. Let $E = M \cap E(G)$.

For each cycle C in $G[E]$, add every other edge of C to A. Note that if C is an odd cycle, this will result in the first and final edge added to A sharing a common endpoint. For each path P in $G[E]$, add every other edge of P to A and, if P has even length, add the edge incident with the other end of P to A.

By construction, the final set A is spanning and so $\Delta(G - A) < \Delta$. Furthermore, each component of $G[A]$ is a path that contains at most two edges. For fixed Δ, each of the above steps takes time $O(|E(G)|)$ since ℓ is a constant defined using Δ. Thus for fixed Δ, the final edge partition (A, B) satisfying the conditions on A is returned in time $O(|E(G)|)$. Hence for fixed Δ, TwoPartition can be implemented in time $O(|E(G)|)$.

Before proving Claim D.13, note that the subroutine Components is the same as in the algorithm BoundedVert. Also, the subroutine LineGraph simply finds the line graph $L(G)$ of the inputed graph G.

We are now ready to prove that for fixed $\Delta \geq 3$, BoundedEdge runs in time poly$(|E(G)|)$.

Claim D.13. *For fixed $\Delta \geq 3$,* BoundedEdge *runs in time polynomial in $|E(G)|$.*

Proof: From the discussions above, we know that TwoPartition and TwoFactor can both be implemented in time $O(|E(G)|)$ since Δ is fixed (recall h is a constant that depends on Δ). It can be easily seen that the line graph of a graph on n vertices can be determined in time $O(n^2)$. As Δ is fixed and $\Delta(G) = \Delta$, $|E(G)| = O(|V(G)|)$. Hence LineGraph finds the line graph H of G in time $O(|V(G)|^2) = O(|E(G)|^2)$.

By Claim D.9, Components can be used to find $\mathcal{C}$ in time $O(|V(H)|^3) = O(|E(G)|^3)$. Also, by Claim D.4, SmallComponents runs in time $p(|V(H)|) = p(|E(G)|)$ for some polynomial p since Δ is fixed. Finally, assigning a colour to an edge takes 1 operation, so the colouring takes exactly $|E(G)|$ operations.

Hence for fixed Δ, BoundedEdge runs in time $O(|E(G)|^3) + p(|E(G)|)$, which is poly$(|E(G)|)$. □