2018

Coloring problems in graph theory

Kacy Messerschmidt
Iowa State University

Coloring problems in graph theory

by

Kacy Messerschmidt

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mathematics

Program of Study Committee:
Bernard Lidický, Major Professor
Steve Butler
Ryan Martin
James Rossmanith
Michael Young

The student author, whose presentation of the scholarship herein was approved by the program of study committee, is solely responsible for the content of this dissertation. The Graduate College will ensure this dissertation is globally accessible and will not permit alterations after a degree is conferred.

Iowa State University

Ames, Iowa

2018

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to thank Jen for gently pushing me to quit procrastinating and for believing in me unconditionally. I would like to thank Bernard for not-so-gently pushing me to quit procrastinating and for not giving up on me when graduating seemed like an impossible task.

ABSTRACT

In this thesis, we focus on variants of the coloring problem on graphs. A *coloring* of a graph G is an assignment of colors to the vertices. A coloring is *proper* if no two adjacent vertices are assigned the same color. Colorings are a central part of graph theory and over time many variants of proper colorings have been introduced. The variants we study are packing colorings, improper colorings, and facial unique-maximum colorings.

A *packing coloring* of a graph G is an assignment of colors $1, \ldots, k$ to the vertices of G such that the distance between any two vertices that receive color i is greater than i. A $(d_1, \ldots, d_k)$-*coloring* of G is an assignment of colors $1, \ldots, k$ to the vertices of G such that the distance between any two vertices that receive color i is greater than d_i. We study packing colorings of multi-layer hexagonal lattices, improving a result of Fiala, Klavžar, and Lidický [14], and find the packing chromatic number of the truncated square lattice. We also prove that subcubic planar graphs are $(1, 1, 2, 2, 2)$-colorable.

A *facial unique-maximum coloring* of G is an assignment of colors $1, \ldots, k$ to the vertices of G such that no two adjacent vertices receive the same color and the maximum color on a face appears only once on that face. We disprove a conjecture of Fabrici and Göring [12] that plane graphs are facial unique-maximum 4-colorable. Inspired by this result, we also provide sufficient conditions for the facial unique-maximum 4-colorability of a plane graph.

A $\{0, p\}$-*coloring* of G is an assignment of colors 0 and p to the vertices of G such that the vertices that receive color 0 form an independent set and the vertices that receive color p form a linear forest. We will explore $\{0, p\}$-colorings, an offshoot of improper colorings, and prove that subcubic planar K_4-free graphs are $\{0, p\}$-colorable. This result is a corollary of a theorem by Borodin, Kostochka, and Toft [6], a fact that we failed to realize before the completion of our proof.

CHAPTER 1. INTRODUCTION

Graph theory is an area of discrete mathematics with applications in a wide range of fields, including computer science, sociology, and chemistry. The concept of graph coloring was introduced in order to solve the problem of coloring countries on a map so that no two countries that shared a border received the same color. This most basic variant of graph coloring, known as a *proper coloring*, is a key concept in modern graph theory. Indeed, the cornerstone of the theory of proper graph colorings, the Four Color Theorem [2], is one of the most famous results in all of graph theory. Proper colorings have been studied extensively, and with good reason. There is no shortage of applications of proper colorings, including scheduling, mobile networks, and Sudoku.

There are many other variants of graph coloring that have arisen from various applications. We will present new results in packing colorings, improper colorings, and facial unique-maximum colorings.

In the next chapter, we will review basic terminology from graph theory and define the three graph coloring variants mentioned above.

In Chapter 3, we will establish methods used to prove results in later chapters. These include discharging, a well-known proof technique in graph coloring; linear/integer programming, an optimization modeling technique commonly used in industry and applied mathematics; boolean satisfiability, a modeling framework commonly used in computer science; and Thomassen's precoloring method, which he used to prove that all planar graphs are 5-choosable [23].

In Chapters 4 and 5, we discuss packing colorings. Chapter 4 contains results on packing colorings of infinite lattices, in particular the hexagonal lattice and the truncated square lattice. This chapter is based on the manuscript *Packing chromatic numbers of multi-layer lattices* by Messerschmidt [20].

In Chapter 5, we study $(1, 1, 2, 2, 2)$-colorings, a variant of packing colorings. In Chapter 6, we deal with a type of improper coloring called a $\{0, p\}$-coloring. In particular, we prove that planar subcubic graphs are $(1, 1, 2, 2, 2)$-colorable and planar subcubic K_4-free graphs are $\{0, p\}$-colorable.

In Chapter 7, we will study facial unique-maximum colorings. We will disprove a conjecture on 4-color facial unique-maximum colorings and provide a condition for when a graph has a 4-color facial unique-maximum coloring. This chapter is based on the published paper *A counterexample to a conjecture on facial unique-maximal colorings* and the manuscript *On facial unique-maximum colorings of plane graphs* by Lidický, Messerschmidt, and Škrekovski ([19] and [18])

Many of the proofs in this thesis make use of computer programs. These programs are available in the Appendix so that the interested reader can verify our results.

CHAPTER 2. DEFINITIONS

In this chapter, we will review graph theory terminology that is relevant to the topics in this thesis.

2.1 Basics

We denote the set of real numbers by $\mathbb{R}$ and the set of integers by $\mathbb{Z}$. We denote the set $\{k \in \mathbb{Z} : 1 \leq k \leq n\}$ by $[n]$.

2.2 Graph theory

A *graph* G is an ordered pair (V, E), where elements of V are called *vertices* and elements of E, called *edges*, are two-element subsets of V. We often use $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G. We also use the notation uv as a shorthand for the edge $\{u, v\}$. A graph H is said to be a subgraph of G, written $H \subseteq G$, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. A subgraph H of G is an *induced* subgraph of G if $E(H) = \{uv \in E(G) : u, v \in V(H)\}$. A subgraph H of G is a *proper* subgraph of G if $V(H)$ is a proper subset of $V(G)$ or $E(H)$ is a proper subset of $E(G)$. Given $V' \subseteq V(G)$, the unique induced subgraph of G with vertex set V' is denoted $G[V']$. A *simple graph* G is a graph without *loops* (edges from a vertex to itself) or *multiedges* (multiple edges between the same two vertices).

Two vertices $u, v \in V(G)$ are *adjacent* if $uv \in E(G)$. The set $N(v) = \{u \in V(G) : uv \in E(G)\}$ is called the *neighborhood* of v and the elements of $N(v)$ are the *neighbors* of v. A vertex v and an edge e are *incident* if $v \in e$. We also say that two edges e_1 and e_2 are

incident if $e_1 \cap e_2 \neq \emptyset$. The *degree* of a vertex v, denoted $\deg(v)$, is the number of vertices that are adjacent to v. We say v is a *k-vertex* if $\deg(v) = k$, a *k^--vertex* if $\deg(v) \leq k$, and a *k^+-vertex* if $\deg(v) \geq k$. A graph G is *k-regular* if each vertex is a k-vertex. In this thesis, we will often refer to a graph as *cubic* if each vertex is a 3-vertex or *subcubic* if each vertex is a 3^--vertex. An *independent set* is a subset S of the vertices such that no two vertices in S are adjacent.

A graph G is *connected* if, for any vertices $u, v \in V(G)$, there exists a path $P \subseteq G$ that contains u and v. If G is not connected, it is *disconnected*. A *component* of a graph G is a connected subgraph of G that is not a proper subgraph of any connected subgraph of G. In a connected graph G, a vertex v is a *cut-vertex* if $G[V(G)\backslash\{v\}]$ is disconnected. The *distance* between two vertices $u, v \in V(G)$, denoted $\text{dist}(u, v)$, is the length of the shortest path $P \subseteq G$ that contains u and v. We define the distance between two vertices in different components to be ∞.

A *walk* is a sequence $v_1 v_2 \cdots v_k$ of (not necessarily distinct) vertices in a graph such that v_i is adjacent to v_{i+1} for $1 \leq i < k$. A *path* is a walk with no repeated vertices. A *cycle* is a path with an extra edge between the endpoints. A *chord* of a cycle C is an edge not in $E(C)$ that connects two vertices in $V(C)$. A graph is *complete* if each vertex is adjacent to every other vertex. We denote the path on n vertices by P_n, the cycle on n vertices by C_n, and the complete graph on n vertices by K_n. See Figure 2.1 for examples of these basic graphs. A *tree* is a connected graph that contains no cycles as subgraphs. A *forest* is a graph whose components are trees. A *linear forest* is a graph whose components are paths. Given a graph G, the *subdivision* of G, denoted $S(G)$, is formed by replacing each edge $uv \in E(G)$ with a new vertex w and two new edges uw and vw. Given two graphs G_1 and G_2, the *Cartesian product* of G_1 and G_2, denoted $G_1 \square G_2$, is the graph with vertex set $\{(v_1, v_2) : v_1 \in V(G_1), v_2 \in V(G_2)\}$ and edge set $\{(u_1, u_2)(u_1, v_2) : u_2 v_2 \in E(G_2)\} \cup \{(u_1, u_2)(v_1, u_2) : u_1 v_1 \in E(G_1)\}$.

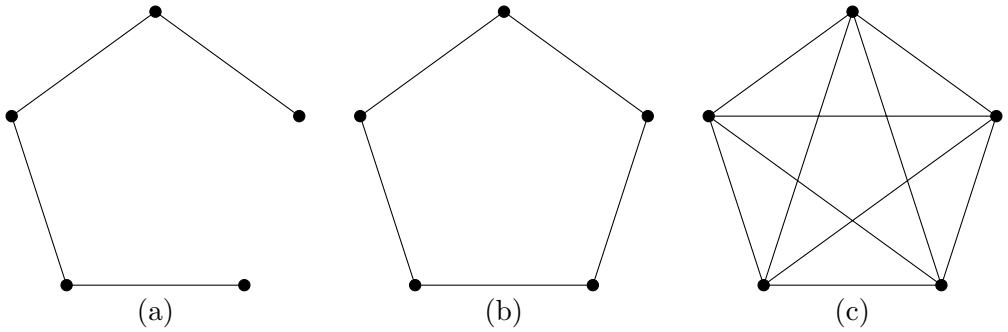

Figure 2.1 Examples of basic graphs with five vertices: (a) a path (P_5), (b) a cycle (C_5), and (c) a complete graph (K_5).

A graph is *planar* if it can be drawn in the plane without any edges crossing and a planar graph is *plane* if it is drawn in the plane in this manner. The edges of a plane graph partition the plane into contiguous regions called *faces*. A face f is said to be *incident* with the edges and vertices that form the boundary of f. Two faces are *adjacent* if they share an incident edge. The *length* of a face f, denoted $\ell(f)$, is the number of edges that separate f from another face plus twice the number of edges around f that do not separate it from another face. The unbounded face of a plane graph is known as the *outer face*. We use $F(G)$ to denote the set of faces of a plane graph G.

2.3 Graph coloring

Given a color set C, a *coloring* $c : V(G) \to C$ of a graph G is an assignment of colors to the vertices of G. The set of colors used in a graph coloring is usually $[k]$. We do not directly study proper colorings in this thesis, but the concept is central to the understanding of other coloring variants. A coloring is *proper* if any two adjacent vertices receive different colors. A *k-coloring* of a graph G is a proper coloring using k colors and G is *k-colorable* if it has a k-coloring. Figure 2.2(a) shows an example of a 2-coloring of $P_3 \square P_3$. The *chromatic number* of G, denoted $\chi(G)$, is the smallest k such that G is k-colorable.

A *list assignment* of a graph G is a function $L : V(G) \to 2^C$ that assigns to each vertex a list of colors from C. An *L-coloring* of G is a proper coloring of G where each vertex v receives a color from its list $L(v)$. A graph G is *L-colorable* if it has an L-coloring for a particular list assignment L. Figure 2.2(b) shows an example of a list assignment L on $P_3 \square P_3$ and an L-coloring of that graph. We say that G is *k-choosable* if it is L-colorable for any list assignment L such that $|L(v)| \geq k$ for all $v \in V(G)$. The *choosability* of G, denoted $\mathrm{ch}(G)$, is the smallest k such that G is k-choosable. If a graph is k-choosable, it is k-colorable. After all, a proper coloring is just a list coloring where each vertex receives the same list. On the other hand, $K_{2,4}$ is an example of a graph that is 2-colorable but not 2-choosable.

2.3.1 Packing coloring

Given a graph G, an *i-packing* is a set $X \subseteq V(G)$ such that $\mathrm{dist}(u, v) > i$ for any $u, v \in X$. A *packing k-coloring* is a partition of $V(G)$ into disjoint color classes $X_1, \ldots, X_k$ such that X_i is an i-packing for $1 \leq i \leq k$. Figure 2.2(c) shows an example of a packing 4-coloring of $P_3 \square P_3$. The *packing chromatic number* of G, denoted $\chi_\rho(G)$, is the smallest k such that G has a packing k-coloring. This type of coloring was originally inspired by the frequency assignment problem, which seeks to optimally assign broadcast frequencies to radio stations in order to limit interference between signals with the same frequency. It was introduced by Goddard et al. [16] under the name *broadcast chromatic number*, which underscored the original motivation. The term *packing chromatic number* was first used by Brešar, Klavžar, and Rall in [7]. We also study a generalization of packing colorings. A *$(d_1, \ldots, d_k)$-coloring* $\phi : V(G) \to \{\phi_i\}_{i \in [k]}$ of a graph G is a coloring such that the set of vertices that receive color ϕ_i forms a d_i-packing. By this definition packing k-coloring is equivalent to a $(1, 2, \ldots, k)$-coloring. If $d_i = d$, we refer to ϕ_i as a *d-color*.

2.3.2 Improper coloring

A $(d_1, \ldots, d_k)$-*improper-coloring* of a graph G is a partition of $V(G)$ into disjoint color classes $X_1, \ldots, X_k$ such that each vertex in X_i has at most d_i neighbors in X_i. In the literature (see Choi and Esperet [10]), these colorings are simply referred to as $(d_1, \ldots, d_k)$-colorings, or more generally as improper colorings. However, we use the term $(d_1, \ldots, d_k)$-improper-coloring to distinguish this type of coloring from the packing coloring variant. A $(0, 2)$-improper-coloring of a graph G would partition $V(G)$ into sets V_0 and V_2 such that V_0 is an independent set and $G[V_2]$ consists of paths and cycles. We define a slightly more restrictive variation on the $(0, 2)$-improper-coloring. A $\{0, p\}$-*coloring* of a graph G is a partition of $V(G)$ into sets V_0 and V_p such that V_0 is an independent set and $G[V_p]$ is a linear forest. Figure 2.2(d) shows an example of a $\{0, p\}$-coloring of $P_3 \square P_3$.

2.3.3 Facial unique-maximum coloring

A *facial unique-maximum k-coloring* of a plane graph G is a proper coloring $c : V(G) \to [k]$ such that, for each face $f \in F(G)$, there exists a vertex v on F such that $c(v) > c(u)$ for all vertices $u \neq v$ on f. We often use *FUM-coloring* as an abbreviation of the term *facial unique-maximum coloring*. Figure 2.2(e) shows an example of a FUM 4-coloring of $P_3 \square P_3$. The *FUM-chromatic number* of G, denoted $\chi_{\text{fum}}(G)$, is the smallest k such that G has a FUM k-coloring. This concept was introduced by Fabrici and Göring [12], who conjectured a stronger version of the Four Color Theorem.

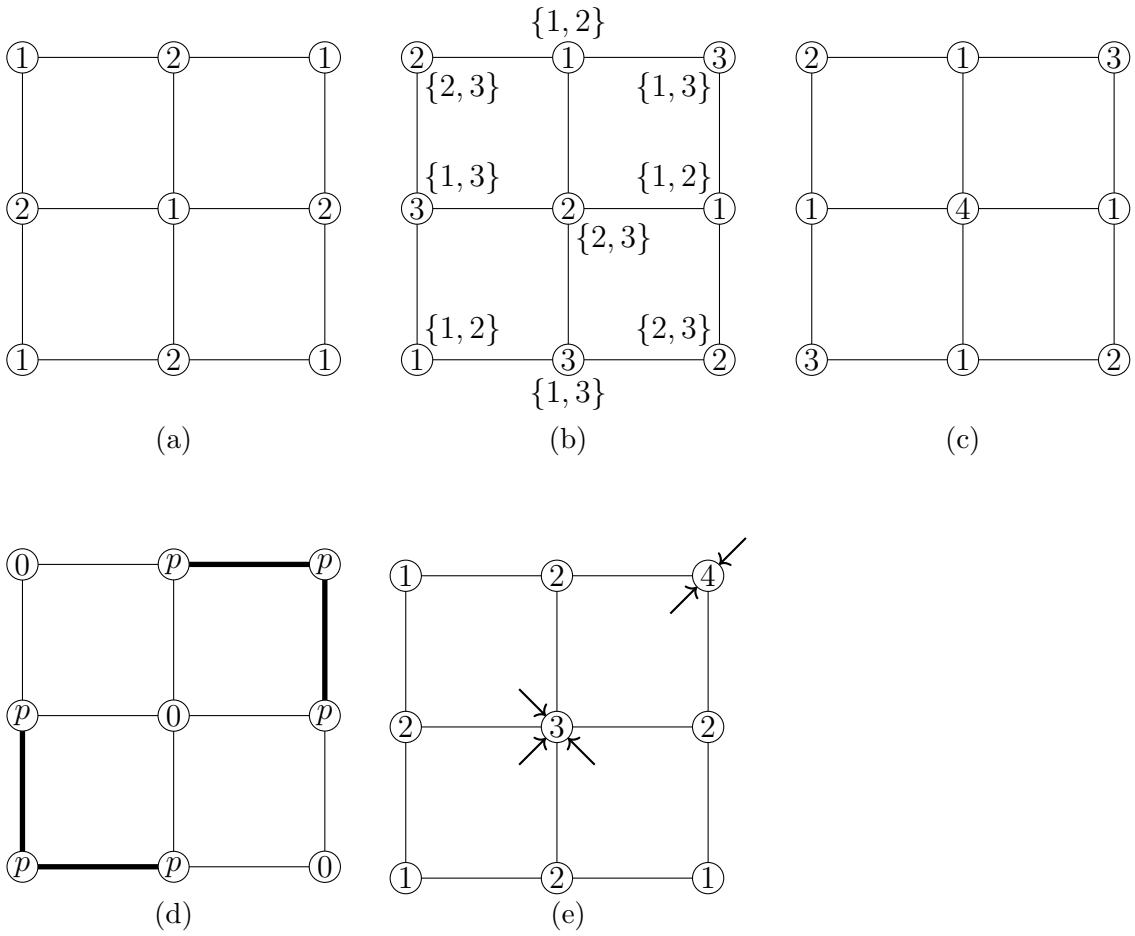

Figure 2.2 Various colorings of $P_3 \square P_3$: (a) a proper coloring, (b) a list coloring, (c) a packing coloring, (d) a $\{0, p\}$-coloring, and (e) a FUM-coloring. In (d), the thick edges represent edges in $G[V_p]$. In (e), the arrows point from each face to the vertex on its boundary receiving the unique-maximum color.

CHAPTER 3. METHODS

In this chapter, we outline the various methods used to prove results in the subsequent chapters: discharging, linear and integer programming, boolean satisfiability, and Thomassen's precoloring method. We will then demonstrate each method by proving a simple theorem.

3.1 Discharging

Discharging was developed by Wernicke in 1904 [25] in an attempt to prove the Four Color Theorem. Discharging seeks to quantify the general fact that planar graphs cannot have too many faces with high length and too many vertices with high degree.

A discharging proof is usually a proof by contradiction. We assume that our result is false and take G to be a minimum counterexample; that is, among all graphs which contradict the result, G has the smallest possible number of vertices.

We apply charge to the vertices and faces of G using the function $\mu : V(G) \cup F(G) \to \mathbb{R}$ defined by $\mu(v) = a\deg(v) - b$ for all $v \in V(G)$ and $\mu(f) = (-\frac{1}{2}b - a)\ell(f) - b$ for all $f \in F(G)$, where $a, b > 0$. Recall three well-known formulas in graph theory:

- $|V(G)| - |E(G)| + |F(G)| = 2$ (Euler's Formula)
- $\sum_{v \in V(G)} \deg(v) = 2|E(G)|$ (Handshaking Lemma)
- $\sum_{f \in F(G)} \ell(f) = 2|F(G)|$ (Facial Handshaking Lemma)

Using these formulas, we can show that the total charge on the vertices and faces of G is

$$\begin{aligned}\sum_{v\in V(G)}\mu(v)+\sum_{f\in F(G)}\mu(f) &= \sum_{v\in V(G)}\big(a\deg(v)-b\big)+\sum_{f\in F(G)}\left[\left(-\frac{1}{2}b-a\right)\ell(f)-b\right]\\ &= 2a|E(G)|-b|V(G)|+2\left(-\frac{1}{2}b-a\right)|E(G)|-b|F(G)|\\ &= -b\big(|V(G)|-|E(G)|+|F(G)|\big)\\ &= -2b<0.\end{aligned}$$

We then construct rules which preserve charge while redistributing it so that each vertex and face has nonnegative charge. To make this easier, we often rule out certain subgraphs of G called *reducible configurations* using our assumptions about G. Since total charge was initially negative but becomes nonnegative after redistribution, we have a contradiction.

We demonstrate the discharging method by proving the theorem for which Wernicke initially developed the method.

Theorem 1 (Wernicke [25])**.** *If G is a planar graph with minimum degree 5, then there exists an edge $uv\in E(G)$ such that $\deg(u)=5$ and $5\le\deg(v)\le 6$.*

Proof. Suppose that the statement is false. Let G be a counterexample with the smallest possible number of vertices. Arbitrarily add as many edges as possible to G while maintaining its planarity. This will result in a new graph G' whose faces are all length 3. An edge is *light* if one endpoint is a 5-vertex and the other is a 5- or 6-vertex. Each vertex in G is a 5^+-vertex, so any vertex incident to a new edge is a 6^+-vertex in G'. Hence, if G' contains a light edge, that edge must have been in G.

Apply charge $\mu(v) = \deg(v) - 6$ to each vertex $v \in V(G')$ and $\mu(f) = 2\ell(f) - 6$ to each face $f \in F(G')$. We verify that this charge function will result in a negative initial charge:

$$\begin{aligned}\sum_{v\in V(G')} \mu(v) + \sum_{f\in F(G')} \mu(f) &= \sum_{v\in V(G')} \big(\deg(v) - 6\big) + \sum_{f\in F(G')} \big(2\ell(f) - 6\big) \\ &= 2|E(G')| - 6|V(G')| + 4|E(G')| - 6|F(G')| \\ &= -6\big(|V(G')| - |E(G')| + |F(G')|\big) \\ &= -12.\end{aligned}$$

We now proceed to the discharging phase of the proof. Redistribute charge throughout G' according to the following rule:

(R1) Each 5-vertex takes charge $\frac{1}{5}$ from each of its neighbors.

Finally, we show that each vertex and face in G' has nonnegative final charge, thus deriving our contradiction. Each face in G' is a triangle, so each face has charge 0. Each 5-vertex began with charge -1 and took charge $\frac{1}{5}$ from each of its neighbors. Since two 5-vertices cannot be adjacent, each 5-vertex has final charge 0. Let v be a 6^+-vertex. Since v had initial charge $\deg(v) - 6$ and gave charge $\frac{1}{5}$ to each degree 5 neighbor, the final charge of v is at least $\frac{4}{5}\deg(v) - 6$. We immediately see that if v is an 8^+-vertex, it must have positive charge. If v is a 6-vertex, then it cannot be adjacent to a 5-vertex, thus v has final charge 0. If v is a 7-vertex, then it would need to be adjacent to at least six 5-vertices to have a negative final charge. This would require that two of these vertices be incident to the same face, and since each face of G' is a triangle, those two 5-vertices would have to be adjacent, which is impossible.

We have thus shown that, while the total initial charge on G' was negative and no charge was created or destroyed, each face and vertex has nonnegative final charge. This is a contradiction, therefore no counterexample exists. □

3.2 Linear and integer programming

Linear programming is a method of modeling an optimization problem using linear relationships. It was developed by George Dantzig in 1947 for the U.S. Air Force and has since been used extensively in industry. Given a system with parameters $X_1, \ldots, X_n$, we seek to maximize or minimize the linear objective function $\sum_{i=1}^{n} c_i X_i$ subject to a set of linear constraints of the form $\sum_{j=1}^{n} a_{i,j} X_j \leq b_i$. Each linear constraint represents a half-space in $\mathbb{R}^n$ and the set of feasible solutions is the intersection of these half-spaces, which is a convex polyhedron. An optimal solution, if one exists, will occur at a vertex of the polyhedron.

An integer program is a linear program with a single added constraint: $X_i \in \mathbb{Z}$ for all i. While all linear programs are solvable in polynomial time, solving an integer program is NP-hard, even when $X_i \in \{0, 1\}$ for all i.

Below, we use integer programs to produce a simple result on the packing chromatic number of $P_3 \square P_3$.

Theorem 2. $\chi_\rho(P_3 \square P_3) = 4$.

Proof. The proof consists of two integer programs: one exhibiting a packing 4-coloring of $P_3 \square P_3$, shown in Figure 3.1, and one showing that no packing 3-coloring of $P_3 \square P_3$ exists.

First, we will show that no packing 3-coloring exists. Specifically, we will show that if three colors are used, we can cover at most eight vertices. Let X_{ijk} be a binary variable representing whether vertex v_{ij} can receive color k. We require that each vertex can receive at most one color. For each $1 \leq i, j \leq 3$, we add the constraint

$$X_{ij1} + X_{ij2} + X_{ij3} \leq 1.$$

We also need to enforce the distance restriction. For each color k and each pair of vertices u, v such that $\text{dist}(u, v) \leq k$, at most one of u and v can receive color k.

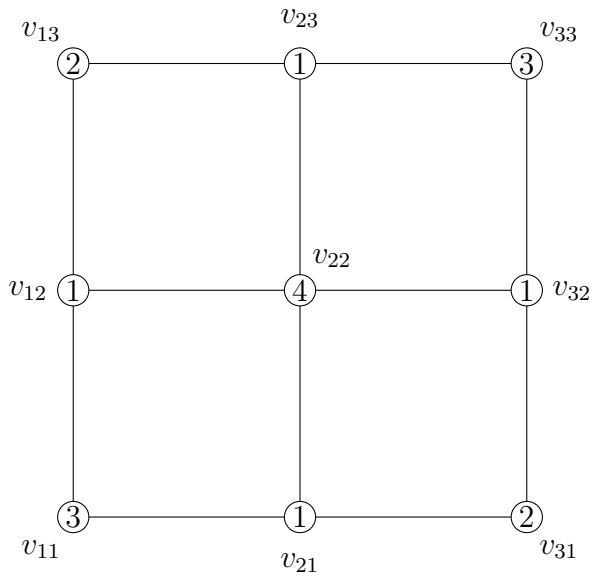

Figure 3.1 $P_3 \square P_3$ with an example of an optimal packing.

For v_{11} for example, we have the following constraints:

$$
\begin{array}{lll}
X_{111} + X_{121} \leq 1 & X_{112} + X_{122} \leq 1 & X_{113} + X_{123} \leq 1 \\
X_{111} + X_{211} \leq 1 & X_{112} + X_{212} \leq 1 & X_{113} + X_{213} \leq 1 \\
 & X_{112} + X_{132} \leq 1 & X_{113} + X_{133} \leq 1 \\
 & X_{112} + X_{222} \leq 1 & X_{113} + X_{23} \leq 1 \\
 & X_{112} + X_{312} \leq 1 & X_{113} + X_{313} \leq 1 \\
 & & X_{113} + X_{233} \leq 1 \\
 & & X_{113} + X_{323} \leq 1
\end{array}
$$

The final integer program is

$$\begin{aligned}
\text{Maximize} \quad & \sum_{i,j,k\in\{1,2,3\}} X_{ijk} \\
\text{Subject to} \quad & X_{111} + X_{112} + X_{113} \leq 1 \\
& \vdots \\
& X_{331} + X_{332} + X_{333} \leq 1 \\
& X_{111} + X_{121} \leq 1 \\
& \vdots \\
& X_{323} + X_{333} \leq 1 \\
& X_{ijk} \in \{0,1\} \text{ for } i,j,k \in \{1,2,3\}
\end{aligned}$$

We enter this integer program into Sage's `MixedIntegerLinearProgram` function and solve it using Sage's built-in GLPK solver. The result is that the maximum value of the objective function is 8. Since $P_3 \square P_3$ has nine vertices, no packing 3-coloring exists. Next, we construct a similar integer program to try and find a packing 4-coloring of $P_3 \square P_3$:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{\substack{i,j\in\{1,2,3\} \\ k\in\{1,2,3,4\}}} X_{ijk} \\
\text{Subject to} \quad & X_{111} + X_{112} + X_{113} + X_{114} \leq 1 \\
& \vdots \\
& X_{331} + X_{332} + X_{333} + X_{334} \leq 1 \\
& X_{111} + X_{121} \leq 1 \\
& \vdots \\
& X_{324} + X_{334} \leq 1 \\
& X_{ijk} \in \{0,1\} \text{ for } i,j \in \{1,2,3\}, k \in \{1,2,3,4\}
\end{aligned}$$

The objective function achieves a maximum of 9 with the following solution:

$$
\begin{array}{llll}
X_{111}=0 & X_{112}=0 & X_{113}=1 & X_{114}=0 \\
X_{121}=1 & X_{122}=0 & X_{123}=0 & X_{124}=0 \\
X_{131}=0 & X_{132}=1 & X_{133}=0 & X_{134}=0 \\
X_{211}=1 & X_{212}=0 & X_{213}=0 & X_{214}=0 \\
X_{221}=0 & X_{222}=0 & X_{223}=0 & X_{224}=1 \\
X_{231}=1 & X_{232}=0 & X_{233}=0 & X_{234}=0 \\
X_{311}=0 & X_{312}=1 & X_{313}=0 & X_{314}=0 \\
X_{321}=1 & X_{322}=0 & X_{323}=0 & X_{324}=0 \\
X_{331}=0 & X_{332}=0 & X_{333}=1 & X_{334}=0
\end{array}
$$

These variable assignments can be interpreted as the packing coloring shown in Figure 3.1. Since $P_3 \square P_3$ has a packing 4-coloring but no packing 3-coloring, $\chi_\rho(P_3 \square P_3) = 4$. □

3.3 Boolean satisfiability

A boolean satisfiability problem is a logical statement consisting of boolean variables. If there exist values for each variable such that the entire statement evaluates to true, the problem is *satisfiable*. If no such set of values exists, then the problem is *unsatisfiable*. A *literal* is a single boolean variable or its negation. A *clause* is a statement consisting of literals separated by logical ORs. A boolean satisfiability problem is in *conjunctive normal form* if it consists of clauses separated by logical ANDs. The boolean satisfiability problem is one of the classic NP-complete problems, however many efficient heuristics have been developed to solve boolean satisfiability problems in conjunctive normal form.

Inspired by Chen, Martin, Martin, and Raimondi [9], we provide another proof of Theorem 2 by constructing boolean satisfiability problems.

Proof of Theorem 2. We proceed by proving that no packing 3-coloring exists and finding a packing 4-coloring, as in the previous proof. Let X_{ijk} be a binary variable representing

whether vertex v_{ij} can receive color k. First, we require that each vertex receive a color. For each vertex v_{ij}, we add the clause $X_{ij1} \vee X_{ij2} \vee X_{ij3}$. Then, for each color k and each pair of vertices u, v such that $\text{dist}(u, v) \leq k$, if one of these vertices receives color k, the other does not. For vertex v_{11} for example, we add the following clauses:

$$\begin{array}{lll}
\neg X_{111} \vee \neg X_{121} & \neg X_{112} \vee \neg X_{122} & \neg X_{113} \vee \neg X_{123} \\
\neg X_{111} \vee \neg X_{211} & \neg X_{112} \vee \neg X_{212} & \neg X_{113} \vee \neg X_{213} \\
 & \neg X_{112} \vee \neg X_{132} & \neg X_{113} \vee \neg X_{133} \\
 & \neg X_{112} \vee \neg X_{222} & \neg X_{113} \vee \neg X_{23} \\
 & \neg X_{112} \vee \neg X_{312} & \neg X_{113} \vee \neg X_{313} \\
 & & \neg X_{113} \vee \neg X_{233} \\
 & & \neg X_{113} \vee \neg X_{323}
\end{array}$$

The final boolean satisfiability problem is

$$\begin{aligned}
&(X_{111} \vee X_{112} \vee X_{113}) \wedge \cdots \wedge \\
&(X_{331} \vee X_{332} \vee X_{333}) \wedge \\
&\quad (\neg X_{111} \vee \neg X_{121}) \wedge \\
&\quad (\neg X_{112} \vee \neg X_{122}) \wedge \cdots \wedge \\
&\quad (\neg X_{323} \vee \neg X_{333}).
\end{aligned}$$

Entering this formula into the Glucose SAT solver [3] reveals that the problem is unsatisfiable, thus no packing 3-coloring of $P_3 \square P_3$ exists.

Next, we construct a similar boolean satisfiability problem to try and find a packing

4-coloring of $P_3 \square P_3$:

$$(X_{111} \vee X_{112} \vee X_{113} \vee X_{114}) \wedge \cdots \wedge$$
$$(X_{331} \vee X_{332} \vee X_{333} \vee X_{334}) \wedge$$
$$(\neg X_{111} \vee \neg X_{121}) \wedge$$
$$(\neg X_{112} \vee \neg X_{122}) \wedge \cdots \wedge$$
$$(\neg X_{324} \vee \neg X_{334})$$

Glucose determines that this problem is satisfiable and provides a solution:

$$\begin{array}{llll} X_{111} = 0 & X_{112} = 0 & X_{113} = 1 & X_{114} = 0 \\ X_{121} = 1 & X_{122} = 0 & X_{123} = 0 & X_{124} = 0 \\ X_{131} = 0 & X_{132} = 1 & X_{133} = 0 & X_{134} = 0 \\ X_{211} = 1 & X_{212} = 0 & X_{213} = 0 & X_{214} = 0 \\ X_{221} = 0 & X_{222} = 0 & X_{223} = 0 & X_{224} = 1 \\ X_{231} = 1 & X_{232} = 0 & X_{233} = 0 & X_{234} = 0 \\ X_{311} = 0 & X_{312} = 1 & X_{313} = 0 & X_{314} = 0 \\ X_{321} = 1 & X_{322} = 0 & X_{323} = 0 & X_{324} = 0 \\ X_{331} = 0 & X_{332} = 0 & X_{333} = 1 & X_{334} = 0 \end{array}$$

These variable assignments can be interpreted as the packing coloring shown in Figure 3.1. Since $P_3 \square P_3$ has a packing 4-coloring but no packing 3-coloring, $\chi_\rho(P_3 \square P_3) = 4$. □

3.4 Thomassen's precoloring method

This technique was developed by Thomassen [23] in order to prove the following result.

Theorem 3 (Thomassen [23])**.** *All planar graphs are* 5*-choosable.*

Theorem 3 is a corollary of the slightly stronger result in Theorem 4. Thomassen's method is best demonstrated by proving Theorem 4, which it was originally developed for.

Theorem 4 (Thomassen [23])**.** *Let G be a graph with list assignment L such that*

- *there is a path P with at most two vertices on the outer face such that the vertices in P have distinct lists of size 1,*
- *every other vertex on the outer face has at least three colors in its list, and*
- *every vertex not on the outer face has at least five colors in its list.*

Then G is L-colorable.

Proof. Suppose for contradiction that G is a counterexample with the minimum number of vertices and L is its corresponding list assignment. We refer to the vertices in P as *precolored* vertices since we have no choice in which color they receive. We first prove that the outer face of G is bounded by an induced cycle.

We may assume that G is connected, otherwise by the minimality of G we could color each component separately. Suppose G has a cut-vertex v. Let G_1 and G_2 be induced subgraphs of G such that $V(G_1) \cup V(G_2) = V(G)$ and $V(G_1) \cap V(G_2) = \{v\}$. Assume without loss of generality that $a, b \in V(G_1)$. By the minimality of G, there exists an L-coloring c_1 of G_1. Let L' be a list assignment on G_2 such that $L'(v) = \{c_1(v)\}$ and L' is equal to L on $V(G_2)\backslash\{v\}$. Since L' meets the criteria of a list assignment laid out in the theorem (v is the precolored vertex), we can find an L'-coloring c_2 of G_2 that agrees with c_1 on v. Extending the colorings c_1 and c_2 to the entire graph yields an L-coloring. This is a contradiction, thus G cannot have a cut-vertex.

Since G is connected and has no cut-vertices, the outer face of G must be bounded by a cycle C. Suppose that C is not an induced cycle; that is, C has a chord uv. Let G_1 and G_2 be induced subgraphs of G such that $V(G_1) \cup V(G_2) = V(G)$ and $V(G_1) \cap V(G_2) = \{u, v\}$. Assume without loss of generality that $a, b \in V(G_1)$. We can again find an L-coloring c_1 of G_1 by the minimality of G. Let L' be a list assignment on G_2 such that $L'(u) = \{c_1(u)\}$, $L'(v) = \{c_1(v)\}$, and L' is equal to L on $V(G_2)\backslash\{u, v\}$. By the minimality of G, we can find

an L'-coloring c_2 of G_2 that agrees with c_1 on u and v. We extend c_1 and c_2 to the entire graph to again get a contradiction, thus C must be an induced cycle.

If P contains no vertices, arbitrarily pick a vertex and reduce its list to one color, then redefine P to contain that vertex. Let a be one of the vertices in P. Let x and y be adjacent vertices on the outer face such that x is adjacent to a and $x \notin V(P)$. Let C be a set consisting of two colors from $L(x)$ not in $L(a)$. Let $G' = G[V(G)\backslash\{x\}]$, $N = N(x)\backslash\{a, y\}$, and L' be a list assignment on G' such that $L'(v) = L(v)\backslash C$ if $v \in N$ and $L'(v) = L(v)$ otherwise. By the minimality of G, there exists an L'-coloring c of G'. Since y is the only neighbor of x that could have received one of the two colors in C, we can extend c to x, thus coloring the entire graph. This is a contradiction, therefore no counterexample exists.

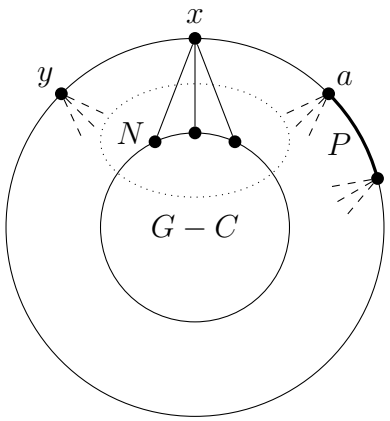

Figure 3.2 An example of a graph G with outer face bounded by an induced cycle. Note that $y \notin V(P)$ in this example, which need not be the case.

□

CHAPTER 4. PACKING COLORING OF GRIDS

This chapter is based on *Packing chromatic numbers of multi-layer lattices* [20], a paper in preparation for submission.

4.1 Introduction

The packing chromatic number of the square lattice $\mathbb{Z}^2$ is a major problem and has already been studied extensively. Goddard, Harris, Hedetniemi, Hedetniemi, and Rall [16] showed that the packing chromatic number of $\mathbb{Z}^2$ is between 9 and 23. Later, Fiala, Klavžar, and Lidický [14] improved the lower bound to 10 while Holub and Soukal [17] improved the upper bound to 17. Finally, Chen, Martin, Martin, and Raimondi [9] improved the lower bound further to 13 and the upper bound to 15, which is where both bounds remain. Finbow and Rall [15] showed that $\mathbb{Z}^3$ (or $\mathbb{Z}^2\Box\mathbb{Z}$) does not have a finite packing. This brings about a natural question: for which $n \in \mathbb{N} \cup \{\infty\}$ does $\chi_\rho(\mathbb{Z}^2\Box P_n)$ jump to infinity? Fiala, Klavžar, and Lidický [14] showed that the answer to this question is 2.

In [14], Fiala, Klavžar, and Lidický also derived two results on the hexagonal lattice $\mathcal{H}$. They showed that $\chi_\rho(\mathcal{H}) = 7$ and $\chi_\rho(\mathcal{H}\Box P_n) = \infty$ for all $n \geq 6$. Left as an open question was for which n does $\chi_\rho(\mathcal{H}\Box P_n)$ make the jump to infinity. We respond with the following conjecture.

Conjecture 1. *The smallest value of n such that $\chi_\rho(\mathcal{H}\Box P_n) = \infty$ is 3.*

Recently, Moss [21] proved that $\chi_\rho(\mathcal{H}\Box P_2) \leq 205$. Our main result is the following theorem.

Theorem 5. *For $n \geq 4$, $\chi_\rho(\mathcal{H}\square P_n) = \infty$, where $\mathcal{H}$ is the infinite hexagonal lattice.*

We also provide a lower bound for $\chi_\rho(\mathcal{H}\square P_3)$.

Theorem 6. $\chi_\rho(\mathcal{H}\square P_3) > 346$.

While we could not prove that $\mathcal{H}\square P_3$ does not have a finite packing, we believe that Theorem 6 strongly suggests that Conjecture 1 is true.

Inspired by the hexagonal lattice, we studied another cubic lattice, the truncated square lattice $\mathcal{T}$, shown in Figure 4.5. We determined its packing chromatic number exactly.

Theorem 7. $\chi_\rho(\mathcal{T}) = 7$, *where $\mathcal{T}$ is the truncated square lattice.*

The proof of Theorem 5 will introduce a new method involving integer and linear programs for determining packing chromatic numbers. Theorem 7 is proved using SAT solvers, a technique developed by Chen, Martin, Martin, and Raimondi [9].

In the next section, we prove Theorem 5. Section 4.3 describes the proof of Theorem 7.

4.2 Hexagonal lattices

4.2.1 4 layers

The main topic of this section is the proof of Theorem 5. In order to prove Theorem 5, we will first need to establish some concepts related to packing density.

Let X_i be the set of vertices in a graph receiving color i. In a finite graph G, the density of X_i, denoted $d(X_i)$, would simply be defined as $\frac{|X_i|}{|V(G)|}$. Since $|V(\mathcal{H}\square P_4)| = \infty$, we need to tweak the definition slightly. Let $\mathcal{H}_i$ be the induced subgraph of $\mathcal{H}$ on the vertex set $\{u \in V(\mathcal{H}) : \text{dist}(u, v) \leq i\}$. Since $\mathcal{H}$ is vertex transitive, the choice of v in this definition is irrelevant. For the purpose of packing colorings on $\mathcal{H}\square P_4$, define the density of X_i as $\limsup_{i\to\infty} \frac{|X_i \cap V(\mathcal{H}_i\square P_4)|}{|V(\mathcal{H}_i\square P_4)|}$.

Lemma 1. *Given a subgraph G of $\mathcal{H}$, if M is an upper bound for $d_{G\square P_4}(X_i)$, then M is also an upper bound for $d_{\mathcal{H}\square P_4}(X_i)$.*

Proof. Suppose $d_{\mathcal{H}\square P_4}(X_i) > M$. Since each vertex in $\mathcal{H}\square P_4$ appears in an equal number of copies of $G\square P_4$, $d_{\mathcal{H}\square P_4}(X_i)$ is bounded above by the average of the density of X_i over all isomorphic copies of $G\square P_4$ in $\mathcal{H}\square P_4$. This would require that the density of X_i in at least one copy of $G\square P_4$ be greater than M, which is a contradiction. □

We use Lemma 1 to establish density bounds in the next two lemmas.

Lemma 2. $d(X_3) \leq \frac{1}{8}$.

Proof. Consider the subgraph $C_6\square P_4$ (see Figure 4.1). If a layer of this subgraph contains a vertex from X_3, there is only one location in the adjacent layer that can contain a vertex from X_3 (the opposite corner of the hex). However, three consecutive layers cannot contain a vertex from X_3. It is therefore only possible to have three vertices from X_3 in $C_6\square P_4$, for a maximum density of $\frac{1}{8}$.

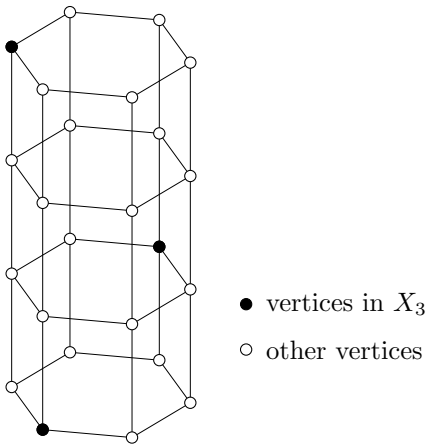

Figure 4.1 An example of a packing X_3 of $C_6\square P_4$ with density $\frac{1}{8}$.

□

We will now use an integer programming method to bound the density of colors 1, 2, and 4.

Lemma 3. $d(X_1 \cup X_2 \cup X_4) \leq \frac{2}{3}$.

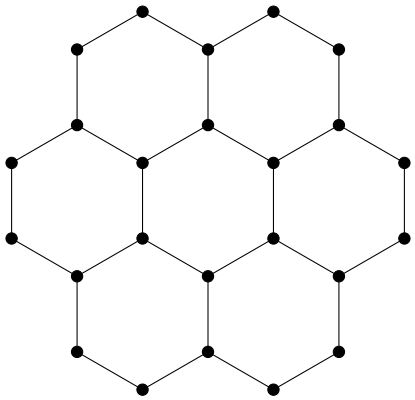

Figure 4.2 A subgraph $H \subseteq \mathcal{H}$ used to bound $d(X_1 \cup X_2 \cup X_4)$.

Proof. Let H be the subgraph of $\mathcal{H}$ shown in Figure 4.2. Consider the graph $H \square P_4$. For each $v \in V(H \square P_4)$ and $i \in \{1, 2, 4\}$, let $Y_{v,i}$ be a binary variable that represents whether v can receive color i. Each vertex v can only receive one color, giving us the constraint

$$Y_{v,1} + Y_{v,2} + Y_{v,4} \leq 1 \text{ for all } v \in V(H \square P_4).$$

The distance restriction gives us the following three constraints:

$$Y_{v,1} + Y_{u,1} \leq 1 \text{ for all } u, v \in V(H \square P_4) \text{ s.t. } \operatorname{dist}(u, v) = 1$$

$$Y_{v,2} + Y_{u,2} \leq 1 \text{ for all } u, v \in V(H \square P_4) \text{ s.t. } 1 \leq \operatorname{dist}(u, v) \leq 2$$

$$Y_{v,4} + Y_{u,4} \leq 1 \text{ for all } u, v \in V(H \square P_4) \text{ s.t. } 1 \leq \operatorname{dist}(u, v) \leq 4.$$

Hence, the largest packing of $H\square P_4$ with colors 1, 2, and 4 can be found by solving the following integer program:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{\substack{v\in V(H\square P_4)\\ i\in\{1,2,4\}}} Y_{v,i} \\
\text{Subject to} \quad & Y_{v,i} \in \{0,1\} \text{ for all } v \in V(H\square P_4), i \in \{1,2,4\} \\
& Y_{v,1} + Y_{u,1} \leq 1 \text{ for all } u,v \in V(H\square P_4) \text{ s.t. } \operatorname{dist}(u,v) = 1 \\
& Y_{v,2} + Y_{u,2} \leq 1 \text{ for all } u,v \in V(H\square P_4) \text{ s.t. } 1 \leq \operatorname{dist}(u,v) \leq 2 \\
& Y_{v,4} + Y_{u,4} \leq 1 \text{ for all } u,v \in V(H\square P_4) \text{ s.t. } 1 \leq \operatorname{dist}(u,v) \leq 4 \\
& \sum_{i\in\{1,2,4\}} Y_{v,i} \leq 1 \text{ for all } v \in V(H\square P_4).
\end{aligned}$$

The solution to this integer program is 64. Since $H\square P_4$ has 96 vertices, this works out to a density of $\frac{2}{3}$, which is an upper bound for $d(X_1 \cup X_2 \cup X_4)$ by Lemma 1. □

We use weight functions to find bounds on density for the remaining colors. Given a vertex $v \in X_i$, a *weight function* $w_v : V \to [0,1]$ assigns a weight to the vertices around v in the hope that no vertex receives total weight greater than 1 from all vertices in X_i. Given a weight function w_v, the *area* A covered by v is $\sum_{u\in V} w_v(u)$.

The bounds for colors 5 and 6 are calculated using a linear programming method that takes into account interaction between vertices in different layers that share the same color.

Lemma 4. $d(X_6) \leq 0.0273$

Proof. Given a vertex v, let

$$w_v(u) = \begin{cases} 1, & \operatorname{dist}(u,v) \leq 3 \\ 0, & \text{otherwise.} \end{cases}$$

Notice that a packing X_6 is valid only if for each $u \in \mathcal{H}\square P_4$, $\sum_{v \in X_6} w_v(u) \leq 1$. Let d_i be the density of color 6 in layer i and let $A_{i,j}$ be area in layer i covered by a vertex in layer j. Finding an upper bound for $d(X_6)$ is equivalent to maximizing $\frac{1}{4}(d_1 + d_2 + d_3 + d_4)$. We can bound d_i above by

$$\frac{1}{A_{i,i}} \left(1 - \sum_{\substack{1 \leq j \leq 4 \\ j \neq i}} A_{i,j} d_j \right).$$

Using the formula

$$A_{i,j} = \begin{cases} 19, & i = j \\ 10, & |i-j| = 1 \\ 4, & |i-j| = 2 \\ 1, & |i-j| = 3 \end{cases}$$

for color 6, we have the following linear program:

$$\begin{aligned} \text{Maximize} \quad & \tfrac{1}{4}(d_1 + d_2 + d_3 + d_4) \\ \text{Subject to} \quad & d_i \in [0,1] \text{ for all } i \in \{1,2,3,4\} \\ & d_1 \leq \tfrac{1}{19}(1 - 10d_2 - 4d_3 - d_4) \\ & d_2 \leq \tfrac{1}{19}(1 - 10d_1 - 10d_3 - 4d_4) \\ & d_3 \leq \tfrac{1}{19}(1 - 4d_1 - 10d_2 - 10d_4) \\ & d_4 \leq \tfrac{1}{19}(1 - d_1 - 4d_2 - 10d_3) \end{aligned}$$

The solution to this linear program gives us $d(X_6) \leq 0.0273$. □

Lemma 5. $d(X_5) \leq 0.0417$.

Proof. We again wish to construct a weight function that captures the area covered by color 5. Constructing this function for color 5 presents a greater challenge than for color 6. Our goal is to maximize the total weight given to each vertex while maintaining the property that each vertex receives a total weight no greater than 1. We could use a binary weight function like the one used in Lemma 4, but we can do better if we deal with distance 3 vertices as a special case.

Let u be a vertex in layer 1 or 4. There exist configurations in which there are four vertices from X_5 that are distance 3 from u. However, there also exists configurations in which there are three vertices from X_5 that are distance 3 from u and a fourth cannot be added. To account for this, we define a new function to help us maximize the total weight given to the vertices.

Given vertices u and v with $\text{dist}(u, v) = d$, let $n_v(u)$ be the number of neighbors x of v such that there is a path of length d between v and u that passes through x. Let u_v be the vertex in the same layer as v that is closest to u (where $u_v = u$ if u is in the same layer as v) and define $n_v^\ell(u) := n_v(u_v)$. Let $w_v : V(\mathcal{H}\square P_4) \to [0, 1]$ be the function defined as follows:

- if $\text{dist}(v, u) < 3$, $w_v(u) = 1$
- if $\text{dist}(v, u) = 3$ and u is in layer 2 or 3, then $w_v(u) = \frac{1}{3} n_v^\ell(u)$
- if $\text{dist}(v, u) = 3$ and u is in layer 1 or 4, then $w_v(u) = \frac{1}{4} n_v(u)$
- if $\text{dist}(v, u) > 3$, $w_v(u) = 0$

Claim 1. *If X_5 is a valid packing, then $\sum_{v \in X_5} w_v(u) \leq 1$ for all $u \in V$.*

Proof. Let u and v be vertices in $\mathcal{H}\square P_4$, where $v \in X_5$. If $\text{dist}(u, v) < 3$, then we must have $\text{dist}(u, v') > 3$ for any $v' \in X_5 \backslash \{v\}$, in which case v' contributes no weight to u. Assume then that $\text{dist}(u, v) = 3$.

Suppose there exists a neighbor x of u such that x lies on a path of length 3 connecting u and v and a path of length 3 connecting u and some $v' \in X_5 \backslash \{v\}$. Then there is a path of length 2 connecting v and x and a path of length 2 connecting v' and x. This gives $\text{dist}(v, v') \leq 4$, which is a contradiction. Hence, for each neighbor x of u, there must be only one vertex in X_5 which is connected to u by a path of length 3 that passes through x. Consequently, $\deg(u)$ is an upper bound on the number of distance 3 neighbors of u from X_5.

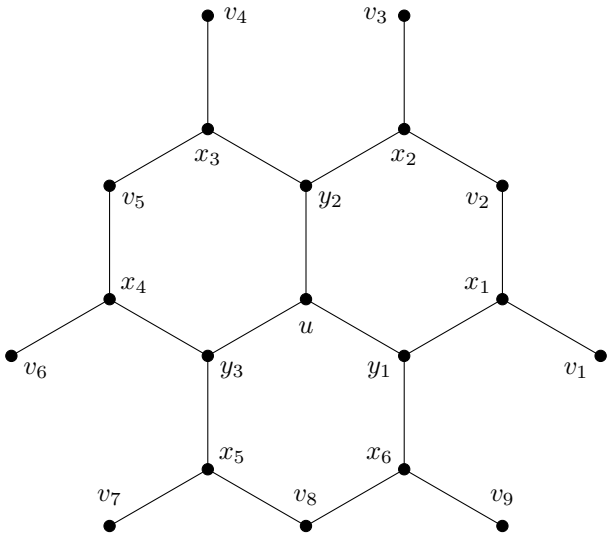

Figure 4.3 Since x_1 and x_2 both lie on length 3 paths between v_2 and u, $n_{v_2}(u) = 2$. On the other hand, x_1 is the only neighbor of v_1 that lies on a length 3 path between v_1 and u, so $n_{v_1}(u) = 1$. Notice that v_1, v_4, and v_7 could all be in X_5, but if instead $v_2 \in X_5$, then only one other v_i could be in X_5.

If u is in layer 1 or 4, $\deg(u) = 4$. For each $v \in X_5$ at distance 3 from u with $n_v(u)$ we reduce by one the maximum number of vertices from X_5 that can be distance 3 from u. Hence, u can receive $\frac{1}{2}$ weight from two vertices, or $\frac{1}{2}$ from one vertex and $\frac{1}{4}$ from up to two vertices, or $\frac{1}{4}$ from up to four vertices.

If u is in layer 2 or 3, $\deg(u) = 5$. If v is not in the same layer as u, the only way that $n_v(u) = 1$ is if v is three layers away from u, which is impossible. We can thus have at most three vertices in X_5 at distance 3 from u: one for each neighbor of u in the same layer as u. We ignore the neighbors of u in adjacent layers and use the function n_v^ℓ to determine the contribution of v. If $n_v^\ell(u) = 2$, then there can be at most one more vertex from X_5 at distance 3 from u, so u receives $\frac{2}{3}$ weight from v and at most $\frac{1}{3}$ additional weight. If $n_{v'}^\ell(u) = 1$ for all $v' \in X_5$ that are distance 3 from u, then u receives $\frac{1}{3}$ from at most three vertices. In any case, u receives total weight no greater than 1. □

Let d_i be the density of color 5 in layer i. Each vertex in layer 2 that receives color 5 covers an area of 7 in layer 1, while each such vertex in layer 3 covers $\frac{5}{2}$ in layer 1 and each such vertex in layer 4 covers $\frac{1}{4}$ in layer 1. In total, the area in layer 1 covered by vertices outside of layer 1 is $7d_2 + \frac{5}{2}d_3 + \frac{1}{4}d_4$. Each vertex in layer 1 that receives color 5 covers an area of 13 in layer 1, thus $d_1 \leq \frac{1}{13}\left(1 - 7d_2 - \frac{5}{2}d_3 - \frac{1}{4}d_4\right)$. Using similar bounds for d_2, d_3, and d_4, we construct the following linear program:

$$\begin{aligned} \text{Maximize} \quad & \tfrac{1}{4}(d_1 + d_2 + d_3 + d_4) \\ \text{Subject to} \quad & d_i \in [0,1] \text{ for all } i \in \{1,2,3,4\} \\ & d_1 \leq \tfrac{1}{13}(1 - 7d_2 - \tfrac{5}{2}d_3 - \tfrac{1}{4}d_4) \\ & d_2 \leq \tfrac{1}{14}(1 - 6d_1 - 6d_3 - 2d_4) \\ & d_3 \leq \tfrac{1}{14}(1 - 2d_1 - 6d_2 - 6d_4) \\ & d_4 \leq \tfrac{1}{13}(1 - \tfrac{1}{4}d_1 - \tfrac{5}{2}d_2 - 7d_3) \end{aligned}$$

The solution to this linear program gives us $d(X_5) \leq 0.0417$. □

Lemma 6. *For $k \geq 4$, the density of X_{2k} is bounded above by $\frac{1}{6k^2-12k+16}$.*

Proof. Given a vertex v in $\mathcal{H}$, the number of vertices at distance k from v is $3k$, hence the number of vertices at distance at most k, including v, is

$$1 + \sum_{i=1}^{k} 3i = 1 + 3\frac{(k+1)k}{2}.$$

Summing over four layers, we have

$$A(v) = \sum_{i=k-3}^{k} \left(1 + 3\frac{(i+1)i}{2}\right) = 6k^2 - 12k + 16,$$

therefore $d(X_{2k}) \leq \frac{1}{6k^2-12k+16}$. □

We now have all of the bounds necessary to prove Theorem 5.

Proof of Theorem 5. Since $d(X_i)$ decreases monotonically with i, we can use $d(X_6)$ as an upper bound for $d(X_7)$ and $d(X_{2k})$ as an upper bound for $d(X_{2k+1})$. Thus, we have

$$\begin{aligned}
\sum_{k=8}^{\infty} d(X_k) &\leq 2\sum_{k=4}^{\infty} d(X_{2k}) \\
&\leq 2\sum_{k=4}^{\infty} \frac{1}{6k^2 - 12k + 16} \\
&\leq 2\sum_{k=4}^{\infty} \frac{1}{6k^2} \\
&\leq \frac{1}{3}\int_3^{\infty} \frac{1}{k^2} dk \\
&= -\frac{1}{3}k^{-1}\Big|_3^{\infty} \\
&= \frac{1}{9}.
\end{aligned}$$

Given any packing coloring $X_1, \ldots, X_n$, the following inequality holds:

$$\begin{aligned}
\sum_{k=1}^{n} d(X_k) &\leq \sum_{k=1}^{\infty} d(X_k) \\
&\leq d(X_1 \cup X_2 \cup X_4) + d(X_3) + d(X_5) + d(X_6) + d(X_7) + \sum_{k=8}^{\infty} d(X_k) \\
&\leq \frac{2}{3} + \frac{1}{8} + 0.0417 + 2 \cdot 0.0273 + \frac{1}{9} \\
&< 0.9992 < 1.
\end{aligned}$$

Hence, no finite packing of $\mathcal{H}\square P_4$ exists. For any $n > 4$, the existence of a finite packing of $\mathcal{H}\square P_n$ would imply that a complete finite packing of $\mathcal{H}\square P_4$. Therefore, for any $n \geq 4$, no finite packing of $\mathcal{H}\square P_n$ exists. □

4.2.2 3 layers

We also provide a lower bound for $\chi_\rho(\mathcal{H}\square P_3)$ using methods similar to those used to prove Theorem 5.

Lemma 7. $d(X_3) \leq \frac{1}{9}$.

Proof. Consider the subgraph $C_6 \square P_3$. No layer can contain two vertices from X_3 and only two layers of $C_6 \square P_3$ can contain one vertex from X_3, hence $d(X_3) \leq \frac{1}{9}$. □

Lemma 8. $d(X_1 \cup X_2 \cup X_4 \cup X_8) \leq \frac{49}{72}$.

Proof. Let H be the subgraph of $\mathcal{H}$ in Figure 4.2. Consider the graph $H \square P_3$. We can find the largest packing of $H \square P_3$ with colors 1, 2, 4, and 8 by solving an integer program similar to the one in the proof of Lemma 3:

$$\begin{aligned}
\text{Maximize} \quad & \sum_{\substack{v \in V(H \square P_3) \\ i \in \{1,2,4,8\}}} Y_{v,i} \\
\text{Subject to} \quad & Y_{v,i} \in \{0,1\} \text{ for all } v \in V(H \square P_3), i \in \{1,2,4,8\} \\
& Y_{v,1} + Y_{u,1} \leq 1 \text{ for all } u, v \in V(H \square P_3) \text{ s.t. } \operatorname{dist}(u,v) = 1 \\
& Y_{v,2} + Y_{u,2} \leq 1 \text{ for all } u, v \in V(H \square P_3) \text{ s.t. } 1 \leq \operatorname{dist}(u,v) \leq 2 \\
& Y_{v,4} + Y_{u,4} \leq 1 \text{ for all } u, v \in V(H \square P_3) \text{ s.t. } 1 \leq \operatorname{dist}(u,v) \leq 4 \\
& Y_{v,8} + Y_{u,4} \leq 1 \text{ for all } u, v \in V(H \square P_3) \text{ s.t. } 1 \leq \operatorname{dist}(u,v) \leq 8 \\
& \sum_{i \in \{1,2,4,8\}} Y_{v,i} \leq 1 \text{ for all } v \in V(H \square P_3).
\end{aligned}$$

The solution to this integer program is 49, giving us a density of $\frac{49}{72}$. □

Lemma 9. *Given a vertex v in $\mathcal{H}$, the number of vertices u with $\operatorname{dist}(u,v) = d$ and $n_v(u) = 2$ is $3\lfloor \frac{d-1}{2} \rfloor$.*

Proof. To make the hexagonal lattice easier to work with, we flatten it so that it more closely resembles a subgraph of the square lattice (see Figure 4.4). The vertices of $\mathcal{H}$ can thus be expressed as elements of $\mathbb{Z}^2$: $V(\mathcal{H}) = \{(i,j) : i, j \in \mathbb{Z}\}$.

Consider the vertex v from Figure 4.4. Let $v = (0,0)$. Given a vertex $u = (k, \ell)$, it is clear that $n_v(u) = 1$ if $\ell = 0$, so we consider the cases when $\ell < 0$ and $\ell > 0$. Assume by symmetry that $k \geq 0$.

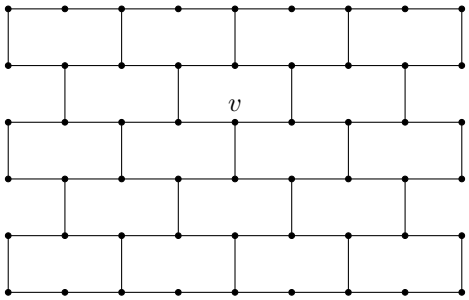

Figure 4.4 A more convenient drawing of the hexagonal lattice.

Claim 2. *If $\ell < 0$, then $n_v(u) = 2$ if and only if $|\ell| - k \leq -1$.*

Proof. Clearly $\text{dist}(u, v) = \text{dist}_{\mathbb{Z}^2}(u, v) = |\ell| + k$ if and only if $|\ell| - k \leq 1$. If $|\ell| - k > 1$, then any path from v to u in $\mathbb{Z}^2$ of length $\text{dist}_{\mathbb{Z}^2}(u, v)$ must contain a subpath $(i, j)(i, j-1)(i, j-2)$. Since the path $(i, j)(i, j-1)(i, j-2)$ does not exist in $\mathcal{H}$, it must be replaced by either $(i, j)(i+1, j)(i+1, j-1)(i, j-1)(i, j-2)$ or $(i, j)(i, j-1)(i+1, j-1)(i+1, j-2)(i, j-2)$, depending on the parity of i and j. The minimum number of edge-disjoint subpaths of the form $(i, j)(i, j-1)(i, j-2)$ in a uv-path in $\mathbb{Z}^2$ is $\lfloor \frac{|\ell|-k}{2} \rfloor$, each of which increases $\text{dist}_{\mathcal{H}}(u, v)$ by 2. Hence, $\text{dist}(u, v) = |\ell| + k + \max\left\{0, 2\left\lfloor \frac{|\ell|-k}{2} \right\rfloor\right\}$. Note that this formula is only valid because u is below v and v is adjacent to the vertex directly below it.

If $|\ell| - k \leq -1$, then we can easily find two uv-paths of length $\text{dist}(u, v)$, one of which passes through $(1, 0)$ and one of which passes through $(0, -1)$. Suppose that $|\ell| - k > -1$. We will show that any uv-path P that passes through $(1, 0)$ does not have length $\text{dist}(u, v)$. Assume that P begins with $(0, 0)(1, 0)(2, 0)$. If $u = (0, -1)$, then clearly P does not have length $\text{dist}(u, v)$; otherwise, $|\ell| - |k - 2| > -1$. Since $\text{dist}\big((2, 0), (k, \ell)\big) = |\ell| + |k - 2| + \max\left\{0, 2\left\lfloor \frac{|\ell|-|k-2|}{2} \right\rfloor\right\}$, the length of P is $2 + |\ell| + |k - 2| + \max\left\{0, 2\left\lfloor \frac{|\ell|-|k-2|}{2} \right\rfloor\right\}$.

If $k < 2$, then

$$\begin{aligned}
|E(P)| &= |\ell| + 4 - k + \max\left\{0, 2\left\lfloor \frac{|\ell| - 2 + k}{2} \right\rfloor\right\} \\
&= |\ell| + 2 + \max\left\{2 - k, 2\left\lfloor \frac{|\ell| - k}{2} \right\rfloor + k\right\} \\
&> |\ell| + k + \max\left\{0, 2\left\lfloor \frac{|\ell| - k}{2} \right\rfloor\right\} \\
&= \mathrm{dist}(u, v).
\end{aligned}$$

If $k \geq 2$, then

$$\begin{aligned}
|E(P)| &= |\ell| + k + \max\left\{0, 2\left\lfloor \frac{|\ell| - k + 2}{2} \right\rfloor\right\} \\
&> |\ell| + k + \max\left\{0, 2\left\lfloor \frac{|\ell| - k}{2} \right\rfloor\right\} \\
&= \mathrm{dist}(u, v).
\end{aligned}$$

□

By Claim 2, for any vertex $u = (k, \ell)$ with $n_v(u) = 2$ and $\ell < 0$, we have $|k| + \ell \geq 1$ and $\mathrm{dist}(u, v) = |k| - \ell$. Given a fixed $d = \mathrm{dist}(u, v)$,

$$\begin{aligned}
\ell &\geq 1 - |k| \\
&\geq 1 - (\ell + d) \\
&\geq \frac{1 - d}{2}.
\end{aligned}$$

This allows for $\lfloor \frac{d-1}{2} \rfloor$ values of ℓ. Since $k \neq 0$, each value of ℓ corresponds to two possible values of k, hence there are $2\lfloor \frac{d-1}{2} \rfloor$ vertices u with $n_v(u) = 2$ and $\ell < 0$.

Claim 3. *If $\ell > 0$, then $n_v(u) = 2$ if and only if $k < \ell$.*

Proof. Since $\mathrm{dist}(u, v) = \ell + k$ if and only if $\ell - k \leq 0$, we can show that $\mathrm{dist}(u, v) = \ell + k + \max\left\{0, 2\left\lfloor \frac{\ell - k + 1}{2} \right\rfloor\right\}$ using an argument similar to that in Claim 2. If $\ell < k$, then $\mathrm{dist}(u, v) = \ell + k$ and clearly no uv-path that passes through $(-1, 0)$ or $(0, -1)$ has length $\mathrm{dist}(u, v)$. Suppose $\ell \geq k$. The paths $P_1 = (0,0)(-1,0)(-1,1)(0,1)$ and $P_2 = (0,0)(1,0)(1,1)(0,1)$

are two uv-paths of length $3 = \operatorname{dist}\big((0,0),(0,1)\big)$ that pass through different neighbors of v. Consider a $(0,1)(k,\ell)$-path of length $\operatorname{dist}\big((0,1),(k,\ell)\big)$. We have

$$\begin{aligned}\operatorname{dist}\big((0,1),(k,\ell)\big) &= \ell - 1 + k + \max\left\{0, 2\left\lfloor\frac{\ell-1-k}{2}\right\rfloor\right\} \\ &= \ell + k + \max\left\{0, 2\left\lfloor\frac{\ell+1-k}{2}\right\rfloor\right\} - 3 \\ &= \operatorname{dist}(u,v) - 3,\end{aligned}$$

hence we can construct uv-paths $P_1 \cup P$ and $P_2 \cup P$ of length $\operatorname{dist}(u,v)$ that pass through different neighbors of v. □

By Claim 3, for any vertex $u = (k,\ell)$ with $n_v(u) = 2$ and $\ell > 0$, we have $|k| < \ell$ and $\operatorname{dist}(u,v) = \ell + |k| + 2\left\lfloor\frac{\ell-|k|+1}{2}\right\rfloor$. If $d = \operatorname{dist}(u,v)$ is even, then $d = 2\ell$. Since $|k| + \ell$ must be even and $|k| < \ell$, there are $d/2 - 1$ possible values of k. If d is odd, then $d = 2\ell + 1$. In this case $k + \ell$ must be odd, so there are $\frac{d-1}{2}$ possible values of k. Regardless of the parity of d, there is only one possible value of ℓ and $\lfloor\frac{d-1}{2}\rfloor$ possible values of k. Therefore, there are $3\lfloor\frac{d-1}{2}\rfloor$ vertices u with $n_v(u) = 2$ in total.

□

Proof of Theorem 6. Let v be a vertex in $\mathcal{H}\square P_3$ in layer i. For each color $n \geq 5$, we estimate the density using a generalized weight function based on the parity of n. If $n = 2k$, we use the weight function

$$w_v(u) = \begin{cases} 1, & \operatorname{dist}(u,v) \leq k \\ 0, & \text{otherwise} \end{cases}$$

to calculate the area function

$$A_{i,j} = 1 + 3\frac{(k - |i-j|)(k - |i-j| + 1)}{2}.$$

If $n = 2k + 1$, we use the weight function

$$w_v(u) = \begin{cases} 1, & \text{dist}(u, v) < k + 1 \\ \frac{1}{3} n_{v,i}(u), & \text{dist}(u, v) = k + 1 \\ 0, & \text{otherwise.} \end{cases}$$

to calculate the area function

$$A_{i,j} = 3\frac{(k - |i - j|)(k - |i - j| + 1)}{2} + \left\lfloor \frac{3(k - |i - j|)}{2} \right\rfloor + 2.$$

The vertices u in layer j with $\text{dist}(u, v) < k+1$ contribute $1+3\frac{(k-|i-j|)(k-|i-j|+1)}{2}$ to $A_{i,j}$. By Lemma 9, there are $3\lfloor\frac{k-|i-j|}{2}\rfloor$ vertices u in layer j with $\text{dist}(u, v) = k + 1$ and $n_{v,i}(u) = 2$, each contributing $\frac{2}{3}$ to $A_{i,j}$. This leaves $3(k - |i - j| + 1) - 3\lfloor\frac{k-|i-j|}{2}\rfloor$ vertices contributing $\frac{1}{3}$. The total contribution of vertices in layer j at distance $k + 1$ is

$$\begin{aligned} &\frac{2}{3}\left(3\left\lfloor\frac{k - |i - j|}{2}\right\rfloor\right) + \frac{1}{3}\left[3(k - |i - j| + 1) - 3\left\lfloor\frac{k - |i - j|}{2}\right\rfloor\right] \\ &\qquad = 2\left\lfloor\frac{k - |i - j|}{2}\right\rfloor + k - |i - j| + 1 - \left\lfloor\frac{k - |i - j|}{2}\right\rfloor \\ &\qquad = \left\lfloor\frac{k - |i - j|}{2}\right\rfloor + k - |i - j| + 1 \\ &\qquad = \left\lfloor\frac{3(k - |i - j|)}{2}\right\rfloor + 1. \end{aligned}$$

Using these values of $A_{i,j}$, we construct the linear program

$$\begin{aligned} \text{Maximize} \quad & \tfrac{1}{3}(d_1 + d_2 + d_3) \\ \text{Subject to} \quad & d_i \in [0, 1] \text{ for all } i \in \{1, 2, 3\} \\ & d_1 \le \tfrac{1}{A_{1,1}}(1 - A_{1,2}d_2 - A_{1,3}d_3) \\ & d_2 \le \tfrac{1}{A_{2,2}}(1 - A_{2,1}d_1 - A_{2,3}d_3) \\ & d_3 \le \tfrac{1}{A_{3,3}}(1 - A_{3,1}d_1 - A_{3,2}d_2), \end{aligned}$$

providing the upper bounds for colors 5, 6, and 7, as well as colors 9 through 346.

These bounds and the bounds from Lemmas 7 and 8 give us

$$\begin{aligned}\sum_{k=1}^{347} d(X_k) &\leq d(X_1 \cup X_2 \cup X_4 \cup X_8) + d(X_3) + \sum_{k=5}^{7} d(X_k) + \sum_{k=9}^{346} d(X_k) \\ &< \frac{49}{72} + \frac{1}{9} + 0.095754 + 0.112577 \\ &< 0.999996 < 1.\end{aligned}$$

□

4.3 Truncated square lattices

Another infinite cubic graph that we studied was the truncated square lattice, $\mathcal{T}$, shown in Figure 4.5. Our goal is to prove Theorem 7, which states that $\chi_\rho(\mathcal{T}) = 7$.

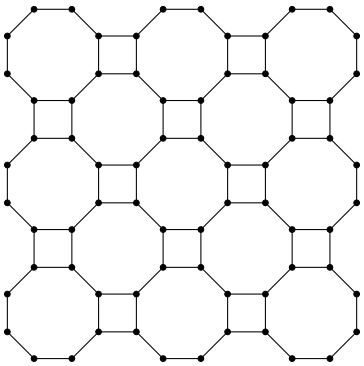

Figure 4.5 The truncated square lattice, $\mathcal{T}$.

We flatten $\mathcal{T}$ in the same fashion as $\mathcal{H}$ (see Figure 4.6) so that its vertices can be expressed as elements of $\mathbb{Z}^2$.

Let m and n be positive integers such that $4 \mid m$ and $2 \mid n$ and let $V_{m,n} = \{(i,j) : 1 \leq i \leq m, 1 \leq j \leq n\}$, where vertex $(1,1)$ is adjacent to $(1,2)$. Take $G_{m,n} = \mathcal{T}[V_{m,n}]$. If $\chi_\rho(\mathcal{T}) \leq M$, then there must be a subgraph isomorphic to G that has a packing M-coloring. Conversely, if G does not have a packing M-coloring, then $\chi_\rho(\mathcal{T}) > M$. Hence, our strategy for finding a lower bound for $\chi_\rho(\mathcal{T})$ is to find a large value of M such that $\chi_\rho(G_{m,n}) > M$ for some m, n.

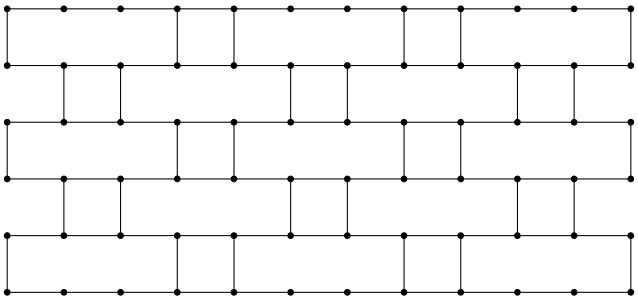

Figure 4.6 A more convenient drawing of the truncated square lattice.

To find an upper bound for $\chi_\rho(\mathcal{T})$, we use use a similar strategy on a slightly modified version of $G_{m,n}$. Let $G'_{m,n}$ be a graph with vertex set $V(G_{m,n})$ and edge set

$$E \cup \{(1,j)(m,j) : 1 \leq j \leq n\} \cup \{(i,1)(i,n) : 1 < i < m, \deg_{G_{m,n}}(i,1) = 2\}.$$

Notice that this graph is designed to preserve adjacencies lost from cutting G out of the lattice. If $G'_{m,n}$ has a packing M-coloring, then $\mathcal{T}$ can be partitioned into copies of $G_{m,n}$, each receiving the same packing coloring. Finding such a coloring, which we refer to as a *tiling*, would be sufficient to prove that $\chi_\rho(G_{m,n}) < M$. Hence, our strategy for finding an upper bound for $\chi_\rho(\mathcal{T})$ is to find a small value of M such that $\chi_\rho(G_{m,n}) \leq M$ for some m, n.

To find these bounds, we make use of a SAT solver to solve a boolean satisfiability problem. A *boolean satisfiability problem* is a logical statement made up of clauses separated by logical And ($\wedge$) operators. Each clause consists of literals – either boolean variables or their negations – separated by logical Or ($\vee$) operators. We will reformulate the problem of calculating the packing chromatic number of an infinite graph as a satisfiability problem.

Let $X_{i,j,k}$ be a boolean variable that represents whether vertex (i,j) can receive color k. The problem of finding a packing ℓ-coloring requires two constraints:

- Each vertex must be able to receive at least one color: $\bigvee_{(i,j)\in V_{m,n}} X_{i,j,k}$, $1 \le k \le \ell$.
- Two vertices can only receive color k if they are at distance greater than k:
  $\neg X_{i_1,j_1,k} \vee \neg X_{1_2,j_2,k}$, $(i_1,j_1),(i_2,j_2) \in V_{m,n}$, $1 \le k \le \ell$, $\operatorname{dist}\big((i_1,j_1),(i_2,j_2)\big) \le k$.

According to Chen, Martin, Martin, and Raimondi [9], the SAT solver program will run more efficiently if the large clauses from the first constraint are broken up into several smaller clauses. A large clause can be broken up into $k+1$ smaller clauses using k new variables known as *commander variables*. For example, the clause $X_{i,j,1} \vee X_{i,j,2} \vee \cdots \vee X_{i,j,m}$ can be broken up using the commander variables $C_{i,j,1}, C_{i,j,2}, \ldots, C_{i,j,k}$ as follows:

$$\begin{gathered}
\big(X_{i,j,1} \vee \cdots \vee X_{i,j,\lfloor m/k \rfloor} \vee \neg C_{i,j,1}\big) \wedge \\
\big(X_{i,j,\lfloor m/k \rfloor+1} \vee \cdots \vee X_{i,j,\lfloor 2m/k \rfloor} \vee \neg C_{i,j,2}\big) \wedge \\
\vdots \\
\big(X_{i,j,\lfloor (k-1)m/k \rfloor+1} \vee \cdots \vee X_{i,j,m} \vee \neg C_{i,j,m}\big) \wedge \\
(C_{i,j,1} \vee \cdots \vee C_{i,j,k})
\end{gathered}$$

Notice that if no $X_{i,j,k}$ is true, then each $\neg C_{i,j,k}$ must be true and thus the last clause cannot be satisfied. It is not clear how best to break up large clauses, so we decided that a large clause containing m literals would be broken up into $\lfloor \sqrt{m} \rfloor + 1$ clauses using $\lfloor \sqrt{m} \rfloor$ commander variables.

Since $\mathcal{T}$ is vertex-transitive, we can precolor a single vertex of $G'_{m,n}$ when calculating an upper bound for $\chi_\rho(\mathcal{T})$. If we are looking for a packing ℓ-coloring of $G'_{m,n}$, then we precolor an arbitrary vertex with color ℓ in order to reduce the number of clauses as much as possible. If a coloring using fewer than ℓ colors exists, then we can recolor any single vertex with ℓ. If a coloring using exactly ℓ colors exists, then there exists such a coloring where any single arbitrary vertex receives color ℓ.

We are now ready to prove Theorem 7.

Proof of Theorem 7. Using the satisfiability problem outlined above, we can obtain the following two results:

- $\chi_\rho(G_{12,12}) > 6$
- $\chi_\rho(G'_{8,6}) \leq 7$

The first result implies that $\chi_\rho(\mathcal{T}) \geq 7$ while the second asserts the existence of a packing 7-coloring of $\mathcal{T}$ (see Figure 4.7). Therefore, $\chi_\rho(\mathcal{T}) = 7$. □

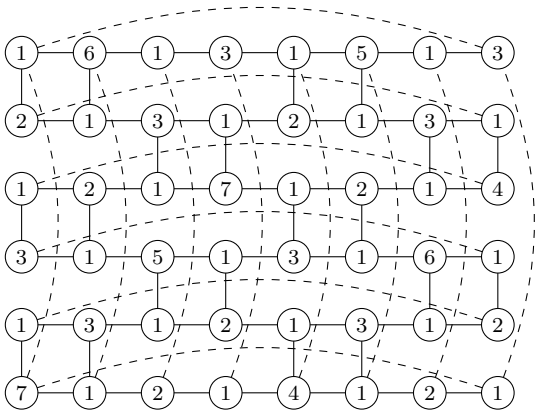

Figure 4.7 A packing 7-coloring of $G'_{8,6}$, which can be used to tile $\mathcal{T}$.

CHAPTER 5. PACKING COLORING OF SUBCUBIC PLANAR GRAPHS

This chapter is the result of joint work with Bernard Lidický.

In [22], Sloper found that there are 4-regular graphs with arbitrarily high packing chromatic number. In particular, Sloper found that the infinite 4-regular tree does not have a finite packing while the infinite cubic tree has packing chromatic number 7. Until recently, it was unknown whether there was a bound on the packing chromatic number of subcubic graphs. Balogh, Kostochka, and Liu [4] showed there are indeed subcubic graphs with arbitrarily high chromatic number, but this question has nonetheless inspired much interest in packing colorings of subcubic graphs.

Recall that the subdivision of a graph G, $S(G)$, is formed by replacing each edge uv with a vertex w and two new edges uw and vw. In [8], Brešar, Klavžar, Rall, and Wash make the following conjecture.

Conjecture 2 (Brešar, Klavžar, Rall, Wash [8])**.** *If G is a subcubic graph, then $\chi_\rho(S(G)) \leq 5$.*

The following proposition is provided as a step towards a possible proof of Conjecture 2.

Proposition 1 (Brešar, Klavžar, Rall, Wash [8])**.** *If a graph G is $(1,1,2,2)$-colorable, then $\chi_\rho(S(G)) \leq 5$.*

The proof of Proposition 1 is fairly straightforward. A 1-packing in G corresponds to a 3-packing in $S(G)$ and a 2-packing in G corresponds to a 5-packing in $S(G)$. If we use color

1 on all of the vertices in $S(G)$ added during the subdivision, then a $(1,1,2,2)$-coloring of G corresponds to a $(1,3,3,5,5)$-coloring of $S(G)$ A $(1,3,3,5,5)$-coloring also qualifies as a $(1,2,3,4,5)$-coloring, which is a packing 5-coloring.

Balogh, Kostochka, and Liu [5] have a result very similar to Conjecture 2.

Theorem 8 (Balogh, Kostochka, Liu [5])**.** *If G is a connected subcubic graph, then $S(G)$ has a packing 8-coloring such that color 8 is used at most once.*

This is proven using a method similar to that proposed in [8].

Theorem 9 (Balogh, Kostochka, Liu [5])**.** *Every connected cubic graph has a $(1,1,2,2,3,3,4)$-coloring such that color 4 is used at most once.*

Inspired by Conjecture 2 and Theorem 8, we proved the following result.

Theorem 10. *All subcubic planar graphs are $(1,1,2,2,2)$-colorable.*

Proof. We prove the result using the discharging method. Suppose for contradiction that the claim is false. Let G be a counterexample with the minimum possible number of vertices. Throughout the proof, we will use a and b as the two 1-colors and c, d, and e as the three 2-colors. We begin by generating a list of reducible configurations.

Claim 4. *G does not contain any 2-vertices.*

Proof. We prove an equivalent statement: configurations $C_{2,1}$ and $C_{2,2}$ (see Figure 5.1) are reducible.

Suppose for contradiction that G contains the configuration $C_{2,1}$. Let $U = \{u_1, u_2\}$ and $W = \{w_{11}, w_{12}, w_{21}, w_{22}\}$. Let G' be the graph constructed from G by removing v and adding the edge u_1u_2. Since G' has fewer vertices than G, by the minimality of G there exists a coloring ϕ of G. We will now try to extend the coloring ϕ to v. For $X \subseteq V(G)$, let $\phi(X) = \{\phi(v) : v \in X\}$. We may assume that $\phi(U) = \{a, b\}$, otherwise we can color v with the 1-color missing from $\phi(U)$. Assume without loss of generality that $\phi(u_1) = a$ and $\phi(u_2) = b$. If we can recolor u_1 to b or u_2 to a then we can color v with the 1-color we

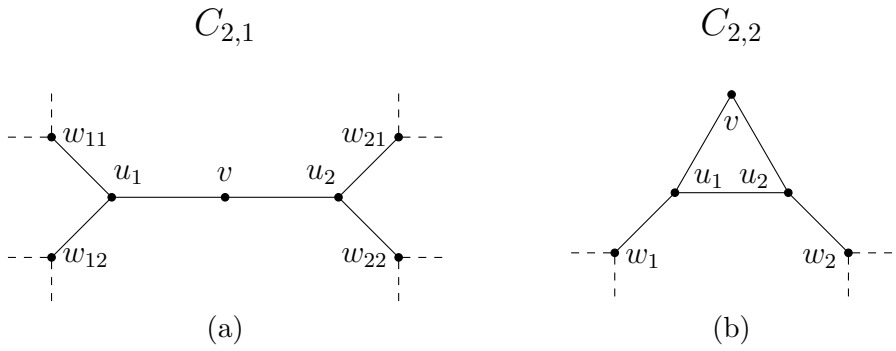

Figure 5.1 The two possible configurations of a 2-vertex.

freed up, so assume without loss of generality that $\phi(w_{11}) = b$ and $\phi(w_{21}) = a$. Since $\phi(W)$ contains at most two 2-colors, we can use the third to color v. This completes the coloring of G, hence $C_{2,1}$ is reducible.

Suppose now that G contains the configuration $C_{2,2}$. Let $U = \{u_1, u_2\}$ and $W = \{w_1, w_2\}$. Let ϕ be a coloring of $G - v$. We can again assume that $\phi(u_1) = a$, $\phi(u_2) = b$, $\phi(w_1) = b$, and $\phi(w_2) = a$. This allows us to color v with any 2-color, completing the coloring of G, hence $C_{2,2}$ is also reducible. □

For the remaining reducible configurations (see Figure 5.2), we use a computer program.

Claim 5. *The configurations $C_{3,1}$, $C_{3,2}$, $C_{3,3}$, $C_{3,4}$, C_4, C_5, C_6, C_7, and C_8 are reducible.*

Proof. Consider the configuration $C_{3,1}$ from Figure 5.2(a). Our objective is to prove that any precoloring of the vertices outside of $C_{3,1}$ can be extended to $C_{3,1}$. To each 2-vertex, we append the configuration in Figure 5.3, which we call a 4-*cluster*. This gives us the graph $C^1_{3,1}$ in Figure 5.4(a). We must also consider two other possibilities. If the two 2-vertices are adjacent to each other, then we have the graph $C^2_{3,1}$ in Figure 5.4(b). If the two 2-vertices in $C_{3,1}$ are adjacent to the same vertex, then we append a single 4-cluster, which gives us the graph $C^3_{3,1}$ in Figure 5.4(c).

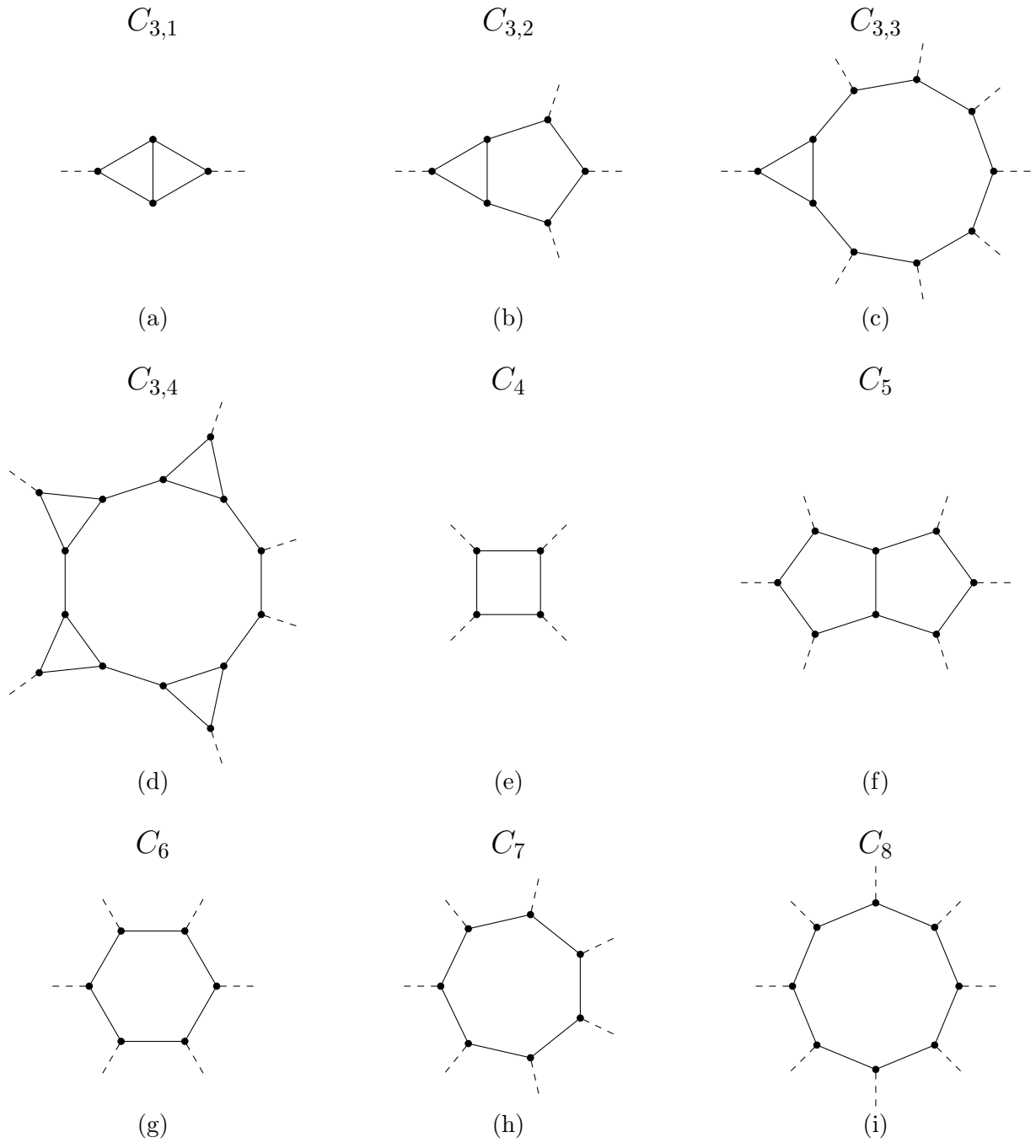

Figure 5.2 More reducible configurations.

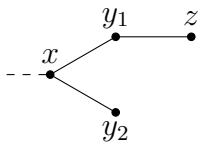

Figure 5.3 Precolored vertices added to reducible configurations.

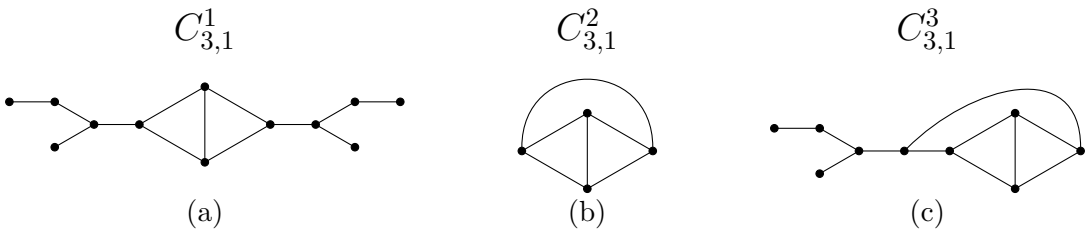

Figure 5.4 Partial precolorings on these graphs are extended to prove that $C_{3,1}$ is reducible.

Each 4-cluster receives receives a cluster precoloring $\big(\phi(x), \phi(y_1), \phi(y_2), \phi(z)\big)$ from the following list, where the color 0 represents leaving x uncolored:

(a, b, c, a)	(b, a, c, a)	(c, a, b, a)	$(0, c, d, e)$
(a, b, d, a)	(b, a, d, a)	(d, a, b, a)	$(0, c, e, d)$
(a, b, e, a)	(b, a, e, a)	(e, a, b, a)	$(0, d, e, c)$

For each combination of cluster precolorings on the two clusters in $C^1_{3,1}$, the computer program will attempt to extend the configuration precoloring to all of $C^1_{3,1}$. The program will likewise try to extend each cluster precoloring of the cluster in $C^3_{3,1}$ to all of $C^3_{3,1}$. Since $C^2_{3,1}$ does not have any clusters to precolor, the program simply tries to find any $(1, 1, 2, 2, 2)$-coloring. We argue that if all of these configuration precolorings can be extended, then $C_{3,1}$ is reducible. To do this we make a much simpler argument. If we replaced each 4-cluster with a 3-cluster consisting only of x, y_1, and y_2, then we could prove that $C_{3,1}$ is reducible by iterating through all combinations of proper cluster precolorings of each 3-cluster and extending the configuration precolorings to all of $C_{3,1}$. However, we will

show that checking the 12 listed 4-cluster precolorings is equivalent to checking all possible 3-cluster precolorings.

First, we make three observations.

Observation 1. *Given a cluster precoloring ϕ, swapping $\phi(y_1)$ and $\phi(y_2)$ has no effect on our ability to extend ϕ to $C_{3,1}$. This means, for example, that the 3-cluster precoloring (a, b, c) is equivalent to (a, c, b).*

Observation 2. *If we can extend a precoloring ϕ with $\phi(y_1) \neq \phi(y_2)$, then we can extend the precoloring ϕ' with $\phi'(y_2) = \phi(y_1)$ and $\phi' = \phi$ elsewhere. Thus, if we can extend (a, b, c), there is no need to check (a, b, b).*

Observation 3. *If ϕ is a precoloring with $\phi(x) \in \{c, d, e\}$ and $|\{\phi(y_1), \phi(y_2) \cap \{a, b\}\}| = 1$, we can change $\phi(x)$ to the available color from $\{a, b\}$ before trying to extend ϕ to $C_{3,1}$. So, instead of trying to extend the coloring (c, a, d), we could try to extend (b, a, d).*

We conclude that checking the 4-cluster precolorings (a, b, c, a), (a, b, d, a), (a, b, e, a), (b, a, c, a), (b, a, d, a), and (b, a, e, a) is equivalent to checking all 3-cluster precolorings ϕ such that either

- $\phi(x) = a$ and $b \in \{\phi(y_1), \phi(y_2)\}$,
- $\phi(x) = b$ and $a \in \{\phi(y_1), \phi(y_2)\}$, or
- $\phi(x) \in \{c, d, e\}$ and $|\{\phi(y_1), \phi(y_2) \cap \{a, b\}\}| = 1$.

We also conclude that checking (c, a, b, a), (d, a, b, a), and (e, a, b, a) is equivalent to checking all 3-cluster precolorings ϕ such that $\{\phi(y_1), \phi(y_2)\} = \{a, b\}$. All that is left to consider for a 3-cluster coloring ϕ is the case where $\{\phi(y_1), \phi(y_2)\} \subseteq \{c, d, e\}$. We make one final observation.

Observation 4. *If we can extend a 4-cluster precoloring ϕ with $\{\phi(y_1), \phi(y_2), \phi(z)\} = \{c, d, e\}$, then we can extend the precoloring ϕ' with $\phi'(z) \in \{a, b, \phi(y_2)\}$ and $\phi' = \phi$ elsewhere. Hence, if we can extend $(0, c, d, e)$, we can also extend $(0, c, d, a)$, $(0, c, d, b)$, and $(0, c, d, d)$.*

Thus, checking $(0, c, d, e)$, $(0, c, e, d)$, and $(0, d, e, c)$ is equivalent to checking all 3-cluster precolorings ϕ such that $\{\phi(y_1), \phi(y_2)\} \subseteq \{c, d, e\}$. With computer assistance, we iterate through all combinations of the cluster precolorings of the two 4-clusters in $C^1_{3,1}$ and all cluster precolorings of the 4-cluster in $C^3_{3,1}$. We find that each configuration precoloring is extendable, thus $C^1_{3,1}$ and $C^3_{3,1}$ are reducible. We also find that $C^2_{3,1}$ is $(1, 1, 2, 2, 2)$-colorable. As a result, $C_{3,1}$ is reducible.

For the remaining configurations in Figure 5.2, we follow a similar procedure. For each pair of 2-vertices, the program adds a common neighbor or an edge between them. It then repeats this procedure for each configuration that is generated in this way, discarding nonplanar configurations, until it can no longer add vertices or edges without generating nonplanar configurations. For every valid configuration that is generated, including the original configuration in Figure 5.2, the program appends a 4-cluster to each 2-vertex. Then, for each configuration precoloring, it tries to extend the precoloring to the entire configuration. Each case of each of the configurations in Figure 5.2 is extendable, therefore $C_{3,2}$, $C_{3,3}$, $C_{3,4}$, C_5, C_6, C_7, and C_8 are reducible. □

We now proceed to the discharging portion of the proof. Apply charge to the vertices of G according to the function $\mu(v) = 2\deg(v) - 6$ and to the faces of G according to the

function $\mu(f) = \ell(f) - 6$. The total initial charge on G is

$$\begin{aligned}\sum_{v\in V(G)} \mu(v) + \sum_{f\in F(G)} \mu(f) &= \sum_{v\in V(G)} \big(2\deg(v) - 6\big) + \sum_{f\in F(G)} \big(\ell(f) - 6\big) \\ &= 4|E(G)| - 6|V(G)| + 2|E(G)| - 6|F(G)| \\ &= -6\big(|V(G)| - |E(G)| + |F(G)|\big) \\ &= -12.\end{aligned}$$

Redistribute charge according to the following rules:

(R1) Each face f gives charge 1 to each 3-face adjacent to f.

(R2) Each face f gives charge $\frac{1}{5}$ to each 5-face adjacent to f.

Since all vertices in G received an initial charge of 0 and the rules did not affect vertices, all vertices have a final charge of 0.

Let f be a face and let $\mu'(f)$ be the charge on f after applying (R1) and (R2). By configurations C_4, C_6, C_7, and C_8, f cannot be a 4-, 6-, 7-, or 8-face.

If $\ell(f) = 3$, then $\mu(f) = -3$ and f receives charge 1 from each of its three adjacent faces by (R1). By configurations $C_{3,1}$ and $C_{3,2}$, f does not have any adjacent 3- or 5-faces to give charge to by rules (R1) and (R2), thus $\mu'(f) = 0$.

If $\ell(f) = 5$, then $\mu(f) = -1$ and f receives charge $\frac{1}{5}$ from each of its five adjacent faces by (R2). By configurations $C_{3,2}$ and C_5, f does not have any adjacent 3- or 5-faces to give charge to by rules (R1) and (R2), thus $\mu'(f) = 0$.

If $\ell(f) = 9$, then $\mu(f) = 3$. By configuration $C_{3,3}$, f cannot be adjacent to any 3-faces. By (R2), f gives up charge of at most $\frac{1}{5}$ to each neighboring face, thus $\mu'(f) \geq \frac{6}{5}$.

If $\ell(f) = 10$, then $\mu(f) = 4$. Since all vertices have degree 3, any two consecutive neighboring faces of f must be adjacent to each other. Hence by configurations $C_{3,1}$, $C_{3,2}$, and C_5, f gives up charge to at most five of its neighboring faces. The only way that $\mu'(f) < 0$ is if f gives up charge to five of its neighboring faces and at least four of those

are 3-faces. This would require that G contains configuration $C_{3,4}$, which is reducible, thus $\mu'(f) \geq 0$.

If $\ell(f) \geq 11$, then f loses charge at most 1 to at most $\lfloor \ell(f)/2 \rfloor$ neighboring faces. The final charge of f is

$$\begin{aligned}
\mu'(f) &\geq \mu(f) - \left\lfloor \frac{\ell(f)}{2} \right\rfloor \\
&= \ell(f) - 6 - \left\lfloor \frac{\ell(f)}{2} \right\rfloor \\
&= \left\lceil \frac{\ell(f)}{2} \right\rceil - 6 \\
&\geq 0,
\end{aligned}$$

thus f has nonnegative final charge.

Since all vertices and faces have nonnegative final charge, the total final charge of G is nonnegative. However, the total initial charge of G was -12 and no charge was created or destroyed. This provides our contradiction. □

CHAPTER 6. IMPROPER COLORING

This chapter is the result of joint work with Ilkyoo Choi and Bernard Lidický.

Recall that a $\{0, p\}$-coloring is a partition of the vertices into sets V_0 and V_p such that V_0 is an independent set and $G[V_p]$ is a linear forest In this chapter, we prove the following result.

Theorem 11. *All subcubic K_4-free planar graphs have a $\{0, p\}$-coloring.*

It should be noted, however, that after we completed our proof we discovered the following result by Borodin, Kostochka, and Toft [6].

Theorem 12 (Borodin, Kostochka, Toft [6])**.** *Let G be a connected graph with maximum degree $\Delta \geq 3$ and not the complete graph $K_{\Delta+1}$. Let also $k_1, \ldots, k_s$ be positive integers, $s \geq 2$, such that $k_1 + \cdots + k_s \geq \Delta$. Then $V(G)$ can be covered by induced subgraphs $G_1, \ldots, G_s$ such that $\mathrm{col}(G_i) \leq k_i$ whenever $1 \leq i \leq s$.*

Note that the *coloring number* of G, denoted $\mathrm{col}(G)$, is the minimum k such that in every subgraph G' of G, there is a vertex v such that $\deg_{G'}(v) < k$. Theorem 11 is a corollary of Theorem 12 with $\Delta = 3$, $k_1 = 1$, and $k_2 = 2$. Regardless, we include our proof below.

Proof of Theorem 11. Suppose by way of contradiction that the statement is false. Let G be a counterexample with the smallest number of vertices. By minimality, G must be connected, otherwise we could color each of the components of G. We begin by proving that every vertex in G has degree 3.

Claim 6. *G contains no 1-vertices.*

Proof. Suppose there exists a 1-vertex v. Let v_1 be the neighbor of v. Color $G - v$. No matter which color v_1 receives, we can use the other color for v. □

Claim 7. *G contains no 2-vertices.*

Proof. Suppose there exists a 2-vertex v. Let v_1 and v_2 be the neighbors of v. Color $G - v$. If v_1 and v_2 receive the same color, we can use the other color for v. Assume that $c(v_1) = 0$ and $c(v_2) = p$. If $\deg(v_2) = 2$, then we can color v with p, so assume $\deg(v_2) = 3$. Let v_3 and v_4 be the other neighbors of v_2. Color v with p. This creates a conflict only if $c(v_3) = c(v_4) = p$, in which case we simply recolor v_2 with 0. □

Since $\Delta(G) = 3$, by Claims 6 and 7 G must be cubic.

Claim 8. *Let $C = v_1 v_2 \cdots v_n$ be an induced cycle. For $i = 1, \ldots, n$, let v_i' be the neighbor of v_i outside of C. If c is a proper $\{0, p\}$-coloring for $G - C$, then c is extendable to G unless $c(v_i') = 0$ for $i = 1, \ldots, n$.*

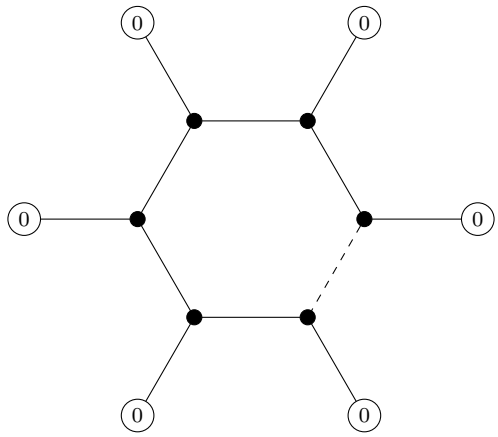

Figure 6.1 The only coloring of $G - C$ that cannot be extended to G.

Proof. Assume that at least one of the vertices $v_1', \ldots, v_n'$ has color p. Consider the following cases.

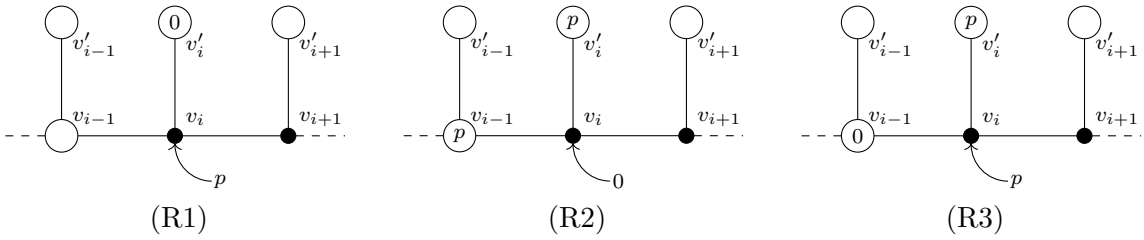

Figure 6.2 Rules for coloring the cycle C.

Case 1: At least one of the vertices $v'_1, \cdots, v'_n$ has color 0.

In particular, this means that there exists $i \in \{1, \ldots, n\}$ such that $c(v'_i) = 0$ and $c(v'_{i+1}) = p$ (where $v'_{n+1} = v'_1$). By symmetry, we may assume that $c(v'_n) = 0$ and $c(v'_1) = p$. Color v_1 with 0. Color v_2, …, v_n in order according to the following rules:

(R1) If $c(v'_i) = 0$, then color v_i with p.

(R2) If $c(v'_i) = c(v_{i-1}) = p$, then color v_i with 0.

(R3) If $c(v'_i) = p$ and $c(v_{i-1}) = 0$, then color v_i with p.

Suppose these rules do not result in a proper $\{0, p\}$-coloring of G. Then G contains either a p-cycle (a cycle consisting of vertices colored with p), a p-claw (a vertex and its three neighbors, all colored with p), or two neighbors colored with 0. It is clear from the rules that we did not introduce a pair of neighbors colored with 0. If a p-claw centered at vertex v was introduced, then we can simply recolor v with 0 to make the coloring proper. Since v_1 received color 0, C cannot be a p-cycle. If a p-cycle was introduced elsewhere, it must contain the path $v'_i v_i v_{i+1} \cdots v_{j-1} v_j v'_j$ for some i and j with $1 < i < j < n$. Since $c(v'_j) = p$, (R3) must have been applied to v_j. This means that $c(v_{j-1}) = 0$, a contradiction, hence no p-cycle exists and we must have a proper $\{0, p\}$-coloring.

Case 2: $c(v'_i) = p$ for $i = 1, 2, \ldots, n$.

If there exists a vertex v'_i which does not have a neighbor that is precolored with 0, then we can recolor v'_i with 0. Since G is K_4-free, there must be at least two distinct vertices

among $v'_1, \ldots, v'_n$, which means that one of the vertices $v'_1, \ldots, v'_n$ is colored with 0 and at least one is colored with p, hence the new coloring extends to C by the above argument. If no such v'_i exists and n is even, then we can simply color odd-indexed vertices of C with p and even-indexed vertices with 0. Assume then that n is odd and for $i = 1, \ldots, n$, v'_i has at least one neighbor precolored with 0. There does not exist a p-path that contains v'_n, v'_1, and v'_2, otherwise one of those vertices would have two neighbors in the p-path and one uncolored neighbor. We can assume by symmetry that there is not a p-path that contains v'_n and v'_1. In this case, we can again color odd-indexed vertices with p and even-indexed vertices with 0. □

Claim 9. *Let $C_1 = v_1 v_2 \cdots v_n$ and $C_2 = v_1 v_2 \cdots v_k u_{k+1} u_{k+2} \cdots u_m$ be two induced cycles that share the path $P = v_1 v_2 \cdots v_k$ such that v_i is not adjacent to u_j for $k+1 \leq i \leq n$ and $k+1 \leq j \leq m$. The configuration consisting of these two cycles is reducible.*

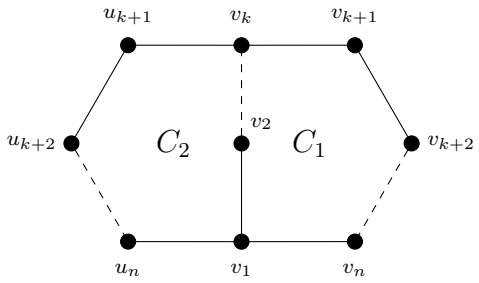

Figure 6.3 Two cycles C_1 and C_2 sharing a path, where $u_i v_j \notin E(G)$ for any $k+1 \leq i \leq n$ and $k+1 \leq j \leq n$.

Proof. By minimality, there exists a proper $\{0, p\}$-coloring c of $G - C_1$. By Claim 8, c is extendable to G unless each precolored vertex adjacent to C_1 has color 0. If this is the case, let c' be a coloring on $G - C_2$ such that $c' = c$ on $G - (C_1 \cup C_2)$ and $c'(v_i) = p$ for $i = k+1, k+2, \ldots, n$. Since the neighbors of v_1 and v_k outside of C_2 (v_n and v_{k+1}, respectively) have color p, by Claim 8 c' is extendable to G. □

Claim 10. *No two cycles share a path.*

Proof. Suppose the converse is true. Let $C = v_1 v_2 \cdots v_n$ be the smallest cycle in G that shares a path with another cycle. For each pair of vertices $v_i, v_j \in C$, find the shortest path between v_i and v_j that does not intersect C except at its endpoints, if one exists. Among all such paths, let $P = v_k p_1 p_2 \cdots p_m v_\ell$ be the shortest. Note that we are guaranteed at least one such path: along the cycle that C shares a path with. The path P cannot consist of a single edge, otherwise there exists a cycle shorter than C which shares a path with another cycle. The graph $C \cup P$ forms the reducible configuration from Claim 9 unless there exists an edge $p_i v_j$, where $j \neq k, \ell$. If $2 \leq i \leq m-1$, then $v_k p_1 \cdots p_i v_j$ is shorter than P, which is a contradiction. By symmetry, we can assume that $i = 1$. The path P cannot consist of a single edge and the path $v_k p_1 v_j$ cannot be shorter than P, so $P = v_k p_1 v_\ell$. The cycles $v_k v_{k+1} \cdots v_\ell p_1$ and $v_\ell v_{\ell+1} \cdots v_k p_1$ cannot be shorter than C, so $n \leq 4$. Since C consists of at most four vertices, v_j must be adjacent to either v_k or v_ℓ. One of the cycles $v_k p_1 v_j$ or $v_\ell p_1 v_j$ exists, so we must have $n = 3$, otherwise there would be a cycle shorter than C that shares a path with another cycle. Therefore, this configuration only consists of four vertices, v_k, v_ℓ, v_j, and p_1. These vertices form a K_4, a contradiction. □

We are now ready to prove Theorem 11. Let $P = v_1 v_2 \cdots v_n$ be a path of maximum length in G. Since $\deg(v_1) = 3$, v_1 must have two neighbors u and w besides v_2. If u or w are not both on P, then there exists a path longer than P, so $u, w \in \{v_3, \ldots, v_n\}$. This forms two cycles which share a path, contradicting Claim 10. □

CHAPTER 7. FACIAL UNIQUE-MAXIMAL COLORING

This chapter is based on *A counterexample to a conjecture on facial unique-maximal colorings* [19], in *Discrete Applied Mathematics*, and *On facial unique-maximum colorings of plane graphs* [18], a paper in preparation for submission.

7.1 Introduction

Facial unique-maximum colorings were first studied by Fabrici and Göring [12]. In the same paper, Fabrici and Göring proved that for a plane graph G, $\chi_{\text{fum}}(G) \leq 6$. This result was improved upon by Wendland [24] and later by Andova, Lidický, Lužar, and Škrekovski [1]. The edge variant of the problem was studied by Fabrici, Jendrol', and Vrbjarová [13]. More information on these facially constrained colorings can be found in a survey by Czap and Jendrol' [11].

One of the most important theorems in graph theory is the Four Color Theorem, which states that any planar graph has a proper coloring using at most four colors [2]. Fabrici and Göring [12] proposed the following strengthening of the Four Color Theorem.

Conjecture 3 (Fabrici, Göring [12])**.** *Given any plane graph G, $\chi_{\text{fum}}(G) \leq 4$.*

In Section 7.2, we show that Conjecture 3 is false. In Section 7.3, we provide sufficient conditions for a graph G to have $\chi_{\text{fum}}(G) \leq 4$, improving a result of Andova, Lidický, Lužar, and Škrekovski [1].

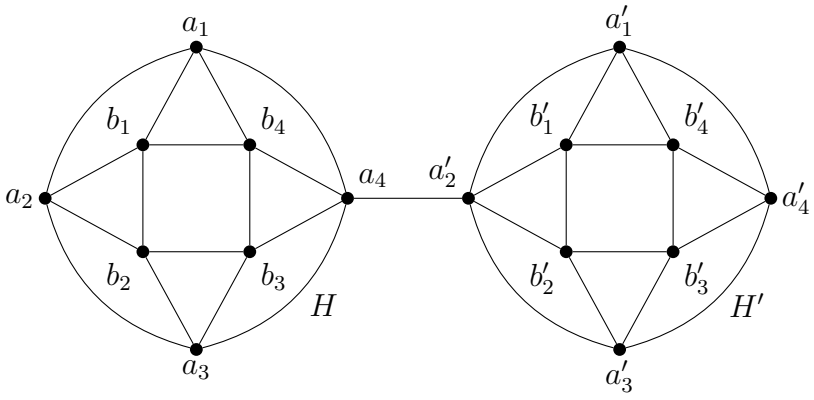

Figure 7.1 A counterexample to Conjecture 3.

7.2 Counterexample to Fabrici and Göring's conjecture

In [19], we provide a counterexample to Conjecture 3.

Proposition 2. *There exists a plane graph G with $\chi_{\text{fum}}(G) > 4$.*

Proof. Let G be the graph depicted in Figure 7.1. It consists of the induced graph H on the vertex set $\{a_1, \ldots, a_4, b_1, \ldots b_4\}$ and the edge set $\{a_i a_{i+1}, b_i b_{i+1}, a_i b_i, a_{i+1} b_i : 1 \leq i \leq 4 \text{ with } a_5 = a_1 \text{ and } b_5 = b_1\}$, H' (an isomorphic copy of H), and the edge $a_4 a'_2$ connecting them. Suppose for contradiction that G has a FUM-coloring with colors in $\{1, 2, 3, 4\}$. The color 4 is assigned to at most one vertex in the outer face of G, so by symmetry we may assume that a_1, a_2, a_3, and a_4 have colors in $\{1, 2, 3\}$. Next we proceed only with H to obtain the contradiction.

By symmetry, assume b_4 is the unique vertex in H that (possibly) has color 4. Without loss of generality, we assume a_1, b_1, and a_2 are colored by x, y, and z, respectively, where $\{x, y, z\} = \{1, 2, 3\}$. This forces b_2 to be colored with x, a_3 to be colored with y, and b_3 to be colored with z. Since a_4 is adjacent to vertices with colors x, y, and z, it must have color 4, a contradiction. □

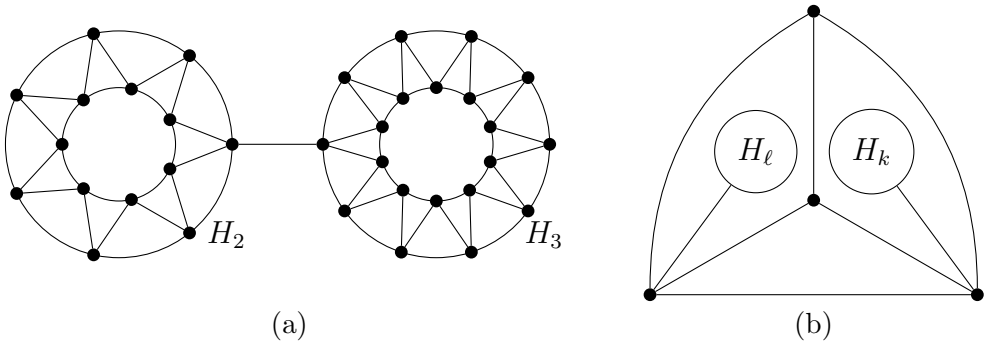

Figure 7.2 More counterexamples to Conjecture 3.

The contradiction in Proposition 2 is produced from the property of H that every coloring of H by colors $\{1, 2, 3, 4\}$, where every interior face has a unique-maximum color, has a vertex in the outer face colored by 4. We can generalize the counterexample in Figure 7.1 by constructing an infinite family of graphs $\mathcal{H} = \{H_k\}_{k\geq 1}$ that can take the place of H or H'. We construct a graph H_k on $6k+2$ vertices by first embedding the cycle $b_1 b_2 \cdots b_{3k+1}$ inside the cycle $a_1 a_2 \cdots a_{3k+1}$. For $1 \leq i \leq 3k$, add edges $a_i b_i$ and $b_i a_{i+1}$, then add the edges $a_{3k+1} b_{3k+1}$ and $b_{3k+1} a_1$. By this definition, the graph H is equivalent to H_1. See Figure 7.2(a) for an example of a generalization of the counterexample.

It is possible to construct more diverse counterexamples by embedding copies of members of $\mathcal{H}$ inside the faces of any 4-chromatic graph G and adding an edge from each copy to some vertex on the face it belongs to. It suffices to embed the graphs from $\mathcal{H}$ into a set of faces K such that in every 4-coloring of G, there is at least one face in K incident with a vertex of G colored by 4. An example of this with G being K_4 is given in Figure 7.2(b).

We now introduce a variation of Conjecture 3 with maximum degree and connectivity conditions added.

Conjecture 4. *If G is a connected plane graph with maximum degree 4, then $\chi_{\text{fum}}(G) \leq 4$.*

Notice that we constructed a counterexample of maximum degree 5. Moreover, removing the edge $a_4 a_2'$ from the graph in Figure 7.1 gives a disconnected graph with maximum degree

4 that does not have a FUM-coloring with colors in $\{1, 2, 3, 4\}$. Recall that Andova, Lidický, Lužar, and Škrekovski [1] showed that maximum degree 3 suffices.

7.3 Improving the χ_{fum} bound for subcubic plane graphs

Fabrici and Göring [12] proved that for any plane graph G, $\chi_{\text{fum}}(G) \leq 6$, while Wendland [24] improved the upper bound to 5. Andova, Lidický, Lužar, and Škrekovski [1] proved that if G is a subcubic or outerplane graph, $\chi_{\text{fum}}(G) \leq 4$. In [18], we improve on the subcubic result with the following theorem.

Theorem 13. *Let G be a plane graph and $X = \{v \in V(G) : \mathrm{d}(v) \geq 4\}$. If $G[X]$ is a matching, then $\chi_{\text{fum}}(G) \leq 4$.*

Theorem 13 is proven as a corollary of a slightly stronger result stated in Lemma 10.

Lemma 10. *Let G be a plane graph with a path P on the outer face with at most two vertices. Let $X = \{v \in V(G) : \mathrm{d}(v) \geq 4\} \cup \{v \in V(P) : \mathrm{d}(v) = 3\}$ such that $G[X]$ is a matching. Given any proper coloring c of the vertices in P using colors $\{1, 2, 3\}$, there exists an extension of c to G using colors $\{1, 2, 3, 4\}$ such that color 4 does not appear on the outer face and all internal faces have a unique maximum color.*

Proof. Suppose for contradiction that G is a counterexample of minimum order. Let F be the outer face of G.

First we introduce a claim, that we repeatedly use when using the minimality of G.

Claim 11. *Let H be a proper subgraph of G and P' be a path on the outer face of H with at most two vertices such that $\mathrm{d}_H(v) < \mathrm{d}_G(v)$ for all $v \in V(P') \setminus V(P)$. Then any proper coloring c' of the vertices in P' using colors $\{1, 2, 3\}$ can be extended to a coloring of H using colors $\{1, 2, 3, 4\}$ such that color 4 does not appear on the outer face and all internal faces have a unique maximum color.*

Proof. Let $X' = \{v \in V(H) : \mathrm{d}_H(v) \geq 4\} \cup \{v \in V(P') : \mathrm{d}_H(v) = 3\}$. If X' is a matching, then H can be colored by the minimality of G. Suppose for contradiction that X' is not a

matching. Then there must be a vertex v that is in X' but not in X. Since $\mathrm{d}_H(v) \leq \mathrm{d}_G(v)$, we cannot have $\mathrm{d}_H(v) \geq 4$, otherwise $\mathrm{d}_G(v) \geq 4$ and thus v would be in X. So v must be in the set $\{v \in V(P') : \mathrm{d}_H(v) = 3\}$. This implies that either $\mathrm{d}_G(v) \geq 4$ or $v \in P$. In either case $v \in X$, which is a contradiction.

□

Claim 12. *F is bounded by a cycle.*

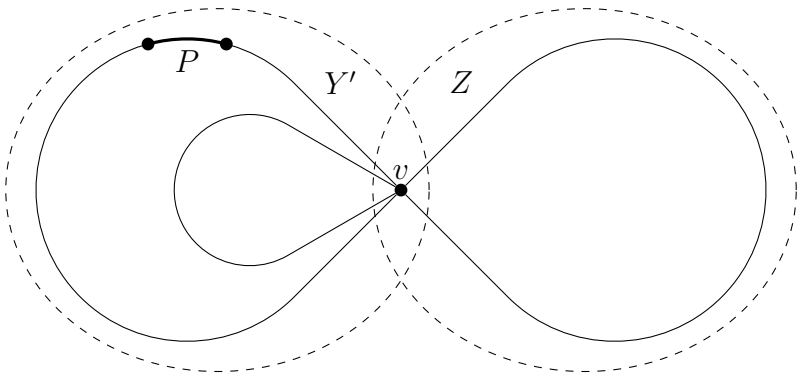

Figure 7.3 An example of a cut vertex v incident with F where one component of $G - v$ is drawn inside another.

Proof. First note that G has only one connected component incident to F, otherwise Claim 11 could be used on each such component of G separately. Also note that if G has no internal faces, then it is a tree and any proper coloring using $\{1, 2, 3\}$ works.

If every vertex in the outer face is incident to exactly two edges of the outer face, F is a cycle. So suppose for contradiction that G has a vertex v incident with at least three edges in the outer face. Notice that v is a cut vertex. Let Y be the set of vertices consisting of v and the vertices of the connected component of $G - v$ that intersects P, if such a component exists. If no component of $G - v$ intersects P, pick an arbitrary one. Let Y' be the union of Y and the set of all vertices that are drawn in the interior faces of $G[Y]$ in G; see Figure 7.3. By the minimality of G, there exists a coloring $c_{Y'}$ of $G[Y']$. Let $Z = (V(G) \setminus Y') \cup \{v\}$. Since $\mathrm{d}_{G[Z]}(v) < \mathrm{d}_G(v)$, by Claim 11 a precoloring of v with $c_{Y'}(v)$ can be extended to a

coloring c_Z of $G[Z]$. Since v has the same color in $c_{Y'}$ and c_Z, we can combine the two colorings into a coloring c of G, a contradiction. □

Let C be the cycle that bounds F.

Claim 13. *C does not have any chords.*

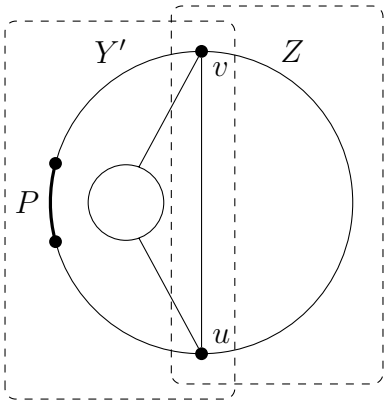

Figure 7.4 An example of a chord uv on C where one component of $G - \{u, v\}$ is drawn inside another.

Proof. Suppose for contradiction that C has a chord uv. Let Y be the set of vertices consisting of u and v and the vertices of the connected component of $G - \{u, v\}$ that intersects P, if such a component exists. If no component of $G - v$ intersects P, pick an arbitrary one. Let Y' be the union of Y and the set of all vertices that are drawn in the interior faces of $G[Y]$ in G; see Figure 7.4. By the minimality of G, there exists a coloring $c_{Y'}$ of $G[Y']$. Let $Z = (V(G) \setminus Y') \cup \{u, v\}$. Since $\mathrm{d}_{G[Z]}(v) < \mathrm{d}_G(v)$ and $\mathrm{d}_{G[Z]}(u) < \mathrm{d}_G(u)$, by Claim 11 a precoloring of u with $c_{Y'}(u)$ and v with $c_{Y'}(v)$ can be extended to a coloring c_Z of $G[Z]$. Since u and v each have the same color in $c_{Y'}$ and c_Z, we can combine the two colorings into a coloring c of G, a contradiction. Therefore, C does not have any chords. □

Claim 14. *G is not a cycle.*

Proof. Suppose for contradiction that G is a cycle. If a vertex in P receives color 3, then color the rest of G alternately with colors 1 and 2. Otherwise, pick a vertex not in P to receive color 3, then color the rest of the vertices alternately with 1 and 2. The interior face has only one vertex that receives color 3, hence all interior faces have a unique maximum color, which is a contradiction. □

Claim 15. *$V(C) \setminus V(P)$ does not contain a 2-vertex.*

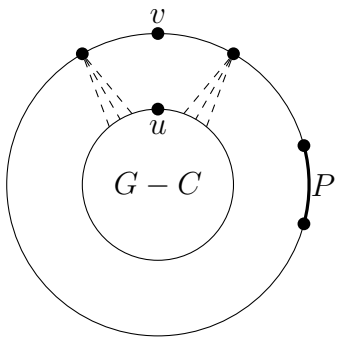

Figure 7.5 A non-precolored 2-vertex v on the outer face of G.

Proof. Suppose for contradiction that $V(C) \setminus V(P)$ contains a 2-vertex v. By Claim 13 and Claim 14, there must be a vertex u that is on the outer face of $G - v$ but not in C, as shown in Figure 7.5. Let $Y = V(G) \setminus \{u, v\}$. By the minimality of G, we can color Y such that every internal face of $G[Y]$ has a unique maximum color and the vertices in the outer face receive colors from $\{1, 2, 3\}$. Coloring u with 4 gives a unique maximum color to every face incident with u. We now color v with the color from $\{1, 2, 3\}$ not used on either of its two neighbors to complete the coloring and arrive at a contradiction. □

Claim 16. *$V(C) \setminus V(P)$ does not have a 3-vertex.*

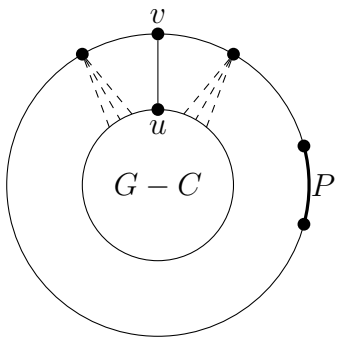

Figure 7.6 A non-precolored 3-vertex v on the outer face of G.

Proof. Suppose for contradiction that $V(C) \setminus V(P)$ has a 3-vertex v with neighbor u not in C; see Figure 7.6. Let $Y = V(G) \setminus \{u, v\}$. By the minimality of G, we can color Y such that every internal face of $G[Y]$ has a unique maximum color. Coloring u with 4 gives a unique maximum color to every face incident with u. We now color v with the color from $\{1, 2, 3\}$ not used on either of its two neighbors from C to complete the coloring and arrive at a contradiction. □

By Claim 12, C cannot have any 1-vertices. By Claims 15 and 16, $V(C) \setminus V(P)$ cannot have 2- or 3-vertices either. Hence each vertex in $V(C) \setminus V(P)$ must have degree at least 4.

Claim 17. *P does not consist of two 2-vertices.*

Proof. Suppose for contradiction that P consists of two 2-vertices v_1 and v_2. Let u_1 and u_2 be the neighbors of v_1 and v_2, respectively, in $V(C) \setminus V(P)$. Note that it is possible that $|V(C)| = 3$, in which case $u_1 = u_2$. Since all vertices in $V(C) \setminus V(P)$ have degree at least 4, either $u_1 = u_2$ or u_1 and u_2 are the only vertices in $V(C) \setminus V(P)$ and are therefore adjacent along F. By Claim 13 and Claim 14, there must exist a vertex w that is on the outer face of $G - \{v_1, v_2\}$ but not in C, as shown in Figure 7.7. Let $Y = V(G) \setminus \{v_1, v_2, w\}$. Color u_1 with a color from $\{1, 2, 3\}$ not used on v_1 and color u_2 with a color from $\{1, 2, 3\}$ not used

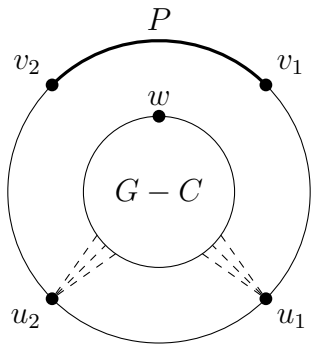

Figure 7.7 A precolored path P consisting of two 2-vertices.

on either v_2 or u_1. By Claim 11, we can extend the precoloring on $\{u_1, u_2\}$ to all of $G[Y]$ such that every internal face of $G[Y]$ has a unique maximum color. Coloring w with 4 gives a unique maximum color to every face incident with w, thus completing the coloring of G and producing a contradiction. □

Claim 18. *P does not have a 2-vertex.*

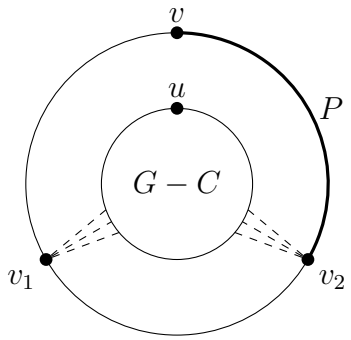

Figure 7.8 A precolored 2-vertex v on the outer face of G.

Proof. Suppose for contradiction that P has a 2-vertex v. Let v_1 and v_2 be the neighbors of v in C. By symmetry, we can assume that v_1 is not in P. If $|V(P)| = 1$, then $v_2 \in V(C) \setminus V(P)$ and thus has degree at least 4. Since X is a matching, v_1 and v_2 must be the only vertices in $V(C) \setminus V(P)$. If $|V(P)| = 2$, then $v_2 \in V(P)$ and $\mathrm{d}(v_2) \geq 3$. Since $\mathrm{d}(v_1) \geq 4$, it must

be the only vertex in $V(C) \setminus V(P)$. In either case, v_1 and v_2 must be adjacent along F. If v_2 is not in $V(P)$, color v_2 with a color from $\{1, 2, 3\}$ not used on v. Next, color v_1 with a color from $\{1, 2, 3\}$ not used on either v or v_2. By Claim 13 and Claim 14, there must exist a vertex u that is on the outer face of $G - v$ but not in C, as shown in Figure 7.8. Let $Y = V(G) \setminus \{u, v\}$. By Claim 11, we can extend the precoloring on $\{v_1, v_2\}$ to all of $G[Y]$ such that every internal face of $G[Y]$ has a unique maximum color. Coloring w with 4 gives a unique maximum color to every face of G incident with w, thus completing the coloring and producing a contradiction. □

By Claim 18, each vertex in P must have degree at least 3. Furthermore, by Claim 15 and Claim 16 each vertex in $V(C) \setminus V(P)$ must have degree at least 4. This means that every vertex in C also belongs to X, a contradiction since X is a matching and C is a cycle. □

We are now ready to prove Theorem 13.

Proof of Theorem 13. Let G be a plane graph and $X = \{v \in V(G) : \mathrm{d}(v) \geq 4\}$ such that $G[X]$ is a matching. Pick a vertex v on the outer face of G and apply Lemma 10 to $G - v$. Color v with 4 to complete the coloring. □

We end with a conjecture that would extend the result in Theorem 13.

Conjecture 5. *Let G be a plane graph and $X = \{v \in V(G) : \mathrm{d}(v) \geq 4\}$. If $G[X]$ has maximum degree 3, then $\chi_{\mathrm{fum}}(G) \leq 4$.*

BIBLIOGRAPHY

[1] Andova, V., Lidický, B., Lužar, B., and Škrekovski, R. (2018). On facial unique-maximum (edge-)coloring. *Discrete Appl. Math.*, 237:26–32.

[2] Appel, K. and Haken, W. (1977). The solution of the four-color map problem. *Sci. Amer.*, 237:108–121.

[3] Audemard, G. and Simon, L. Glucose SAT. http://www.labri.fr/perso/lsimon/glucose/.

[4] Balogh, J., Kostochka, A., and Liu, X. (2018a). Packing chromatic number of cubic graphs. *Discrete Math.*, 341:474–483.

[5] Balogh, J., Kostochka, A., and Liu, X. (2018b). Packing chromatic number of subdivision of cubic graphs. Arxiv e-print 1803.02537.

[6] Borodin, O., Kostochka, A., and Toft, B. (2000). Variable degeneracy: extensions of Brooks' and Gallai's theorems. *Discrete Math.*, 214:101–112.

[7] Brešar, B., Klavžar, S., and Rall, D. (2007). On the packing chromatic number of Cartesian products, hexagonal lattices, and trees. *Discrete Appl. Math.*, 155:2303–2311.

[8] Brešar, B., Klavžar, S., Rall, D., and Wash, K. (2017). Packing chromatic number, $(1, 1, 2, 2)$-colorings, and characterizing the Petersen graph. *Aequat. Math.*, 91:169–184.

[9] Chen, T., Martin, B., Martin, J., and Raimondi, F. (2017). The packing chromatic number of the infinite square lattice is between 13 and 15. *Discrete Appl. Math.*, 225:136–142.

[10] Choi, I. and Esperet, L. (2016). Improper coloring of graphs on surfaces. Arxiv e-print 1603.02841.

[11] Czap, J. and Jendrol', S. (2017). Facially-constrained colorings of plane graphs: a survey. *Discrete Math.*, 340:2691–2703.

[12] Fabrici, I. and Göring, F. (2016). Unique-maximum coloring of plane graphs. *Discuss. Math. Graph Theory*, 36:95–102.

[13] Fabrici, I., Jendrol', S., and Vrbjarová, M. (2015). Unique-maximum edge-colouring of

plane graphs with respect to faces. *Discrete Appl. Math.*, 185:239–243.

[14] Fiala, J., Klavžar, S., and Lidický, B. (2009). The packing chromatic number of infinite product graphs. *European J. Combin.*, 30:1101–1113.

[15] Finbow, S. and Rall, D. (2010). On the packing chromatic number of some lattices. *Discrete Appl. Math.*, 158:1224–1228.

[16] Goddard, W., Harris, J. M., Hedetniemi, S. M., Hedetniemi, S. T., and Rall, D. F. (2008). Broadcast chromatic numbers of graphs. *Ars Combin.*, 86:33–49.

[17] Holub, P. and Soukal, R. (2010). A note on packing chromatic number of the square lattice. *Electron. J. Combin.*, 17:#N17.

[18] Lidický, B., Messerschmidt, K., and Škrekovski, R. On facial unique-maximum colorings of plane graphs. Manuscript.

[19] Lidický, B., Messerschmidt, K., and Škrekovski, R. (2018). A counterexample to a conjecture on facial unique-maximal colorings. *Discrete Appl. Math.*, 237:123–125.

[20] Messerschmidt, K. Packing chromatic numbers of multi-layer lattices. Manuscript.

[21] Moss, K. (2017). *Coloring problems in graph theory.* PhD thesis, Iowa State University.

[22] Sloper, C. (2004). An eccentric coloring of trees. *Australas. J. Combin.*, 23:309–321.

[23] Thomassen, C. (1994). Every planar graph is 5-choosable. *J. Combin. Theory Ser. B*, 62:180–181.

[24] Wendland, A. (2016). Coloring of plane graphs with unique maximal colors on faces. *J. Graph Theory*, 83:359–371.

[25] Wernicke, P. (1904). Über den kartographischen Vierfarbensatz. *Math. Ann. (in German)*, 58:413–426.

APPENDIX. ADDITIONAL CODE

The following Sage code is used to calculate the maximum density of a packing coloring of $\mathcal{H}\Box P_4$.

```
        # Upper bound for density of color 5

        p = MixedIntegerLinearProgram(maximization = True, solver = "GLPK")
        x = p.new_variable(integer = False, nonnegative = True)

        p.add_constraint(x[0] <= (1.0/13)*(1 - 7*x[1] - 2.5*x[2] - .25*x[3]))
        p.add_constraint(x[1] <= (1.0/14)*(1 - 6*x[0] - 6*x[2] - 2*x[3]))
        p.add_constraint(x[2] <= (1.0/14)*(1 - 2*x[0] - 6*x[1] - 6*x[3]))
        p.add_constraint(x[3] <= (1.0/13)*(1 - .25*x[0] - 2.5*x[1] - 7*x[2]))
        p.set_objective(.25*(x[0] + x[1] + x[2] + x[3]))

        d5 = p.solve()
        print "Color 5:            ", d5

        ##################################################

        # Upper bound for density of color 6 (also used as upper bound for
    color 7)

        p = MixedIntegerLinearProgram(maximization = True, solver = "GLPK")
        x = p.new_variable(integer = False, nonnegative = True)

        p.add_constraint(x[0] <= (1.0/19)*(1 - 10*x[1] - 4*x[2] - x[3]))
        p.add_constraint(x[1] <= (1.0/19)*(1 - 10*x[0] - 10*x[2] - 4*x[3]))
        p.add_constraint(x[2] <= (1.0/19)*(1 - 4*x[0] - 10*x[1] - 10*x[3]))
        p.add_constraint(x[3] <= (1.0/19)*(1 - x[0] - 4*x[1] - 10*x[2]))
        p.set_objective(.25*(x[0] + x[1] + x[2] + x[3]))

        d6 = p.solve()
        print "Color 6:            ", d6

        ##################################################

        # Upper bound for density of colors 1, 2, and 4

        d124 = 2.0/3
        print "Colors 1, 2, and 4:", d124

        # Upper bound for density of color 3

        d3 = 1.0/8
        print "Color 3:            ", d3

        # Upper bound for density of colors 8, 9, 10...
```

```
d_rest = 1.0/9
print "Colors >6:          ", d_rest

################################################

# Upper bound for all packings

print "All colors:        ", d124 + d3 + d5 + 2*d6 + d_rest
```

The following Sage code is used to calculate the maximum density of a packing 347-coloring of $\mathcal{H}\square P_3$.

```
# Generates upper bound for the density of any color greater than 3

def LPMaxDensity(n):
    if (n < 4):
        print "Enter n >= 4"
        return

    p = MixedIntegerLinearProgram(maximization=True, solver = "GLPK")
    x = p.new_variable(integer=False, nonnegative=True)

    if (n % 2 == 0):
        k = n/2
        c0 = 1 + 3*k*(k + 1)/2
        c1 = 1 + 3*(k - 1)*k/2
        c2 = 1 + 3*(k - 2)*(k - 1)/2
    else:
        k = (n - 1)/2
        c0 = 3*k*(k + 1)/2 + math.floor(3.0*k/2) + 2
        c1 = 3*(k - 1)*k/2 + math.floor(3.0*(k - 1)/2) + 2
        c2 = 3*(k - 2)*(k - 1)/2 + math.floor(3.0*(k - 2)/2) + 2

    p.add_constraint(x[0] <= (1.0/c0)*(1 - c1*x[1] - c2*x[2]))
    p.add_constraint(x[1] <= (1.0/c0)*(1 - c1*x[0] - c1*x[2]))
    p.add_constraint(x[2] <= (1.0/c0)*(1 - c2*x[0] - c1*x[1]))
    p.set_objective((1.0/3)*(x[0] + x[1] + x[2]))
    return p.solve()

################################################

# Sums upper bounds for colors in range [start, stop)

def LPMaxDensitySum(start, stop):
    if (start < 4):
        print "Enter start >= 4"
        return

    sum = 0
    for n in range(start, stop):
        sum += LPMaxDensity(n)

    return sum

################################################
```

```
# Upper bound for density of colors 1, 2, 4, and 8

d1248 = 49.0/72

# Upper bound for density of color 3

d3 = 1.0/9

#################################################

# Upper bound for all packings on 346 colors

d1248 + d3 + LPMaxDensitySum(5, 8) + LPMaxDensitySum(9, 347)
```

The following Sage program is used to generate the input file for the Glucose SAT solver [3] in order to determine upper or lower bounds for $\chi_\rho(\mathcal{T})$.

To prove $\chi_\rho(\mathcal{T}) > 6$, we showed that $\chi_\rho(G_{12,12}) > 6$. This can be done by running `Glucose_Input` with `height = 12`, `width = 12`, `num_colors = 6`, and `wrap = 0`. The resulting string can be entered into Glucose, which concludes that the underlying SAT problem is unsatisfiable.

To prove $\chi_\rho(\mathcal{T}) \leq 7$, we showed that $\chi_\rho(G'_{8,6}) \leq 7$. This can be done by running `Glucose_Input` with `height = 6`, `width = 8`, `num_colors = 7`, and `wrap = 1`. The resulting string can be entered into Glucose, which concludes that the underlying SAT problem is satisfiable. The Glucose output string can be interpreted as a packing coloring using the `Glucose_Output` program below.

```
# Generates distance matrix for section of truncated square grid
# Assume throughout that height is even, width is divisible by 4,
#     and the top left of the subgrid is a square
# Example with height = 4, width = 12:
#
#     o-o-o-o-o-o-o-o-o-o-o-o
#     | |     | |     | |
#     o-o-o-o-o-o-o-o-o-o-o-o
#         | |     | |     | |
#     o-o-o-o-o-o-o-o-o-o-o-o
#     | |     | |     | |
#     o-o-o-o-o-o-o-o-o-o-o-o

def Generate_Dist(height, width, wrap):

    n = width*height
    dist = []
    for i in range(n):
        dist.append([0]*n)

    # First determine neighbors
    for i in range(n):
        r1 = i//width
        c1 = i % width

        for j in range(i + 1, n):
            r2 = j//width
            c2 = j % width
```

```
                if ((r1 == r2) and (c1 + 1 == c2)):
                    dist[i][j] = 1

                if ((c1 == c2) and (r1 + 1 == r2) and ((2*r1 + c1) % 4 <=
1)):
                    dist[i][j] = 1

                if (wrap == True):
                    if ((r1 == r2) and (c1 == 0) and (c2 == width - 1)):
                        dist[i][j] = 1

                    if ((c1 == c2) and (r2 - r1 == height - 1) and ((2*r1
+ c1) % 4 > 1)):
                        dist[i][j] = 1

                if (dist[i][j] == 1):
                    dist[j][i] = 1

        d = 1
        done = False

        # dist[i][j] == d if there is a vertex k such that
        #     dist[i][k] == d - 1 and dist[j][k] == 1
        while (done == False):
            d += 1
            for i in range(n):
                for j in range(i + 1, n):
                    if (dist[i][j] == 0):
                        for k in range(n):
                            if ((dist[i][k] == d - 1) and (dist[j][k] ==
1)):
                                dist[i][j] = d
                                dist[j][i] = d
                                break

            # If all non-diagonal entries are nonzero, done
            done = True
            for i in range(n):
                for j in range(i + 1, n):
                    if (dist[i][j] == 0):
                        done = False
                        break
                if (done == False):
                    break

        return dist

    ##################################################

    def PVar(v, clr, num_colors):
        return v*num_colors + clr

    def CVar(v, n, num_colors, i):
        return n*num_colors + v*int(math.floor(sqrt(num_colors))) + i + 1
```

```
    ###################################################

    # Generates input file for glucose_static SAT solver program
    # Format for input file:
    #
    #   p cnf (number of variables) (number of clauses)
    #   (clause 1)
    #   (clause 2)
    #   ...
    #
    # Each variable is numbered 1 through num_variables
    # In the input file, each clause consists of a series of integers
separated by spaces
    #     and terminated by 0
    # We put i in the clause to represent variable i and -i to represent
its negation
    # If looking for a lower bound, set wrap to False
    # If looking for an upper bound, set wrap to True

    def Glucose_Input(height, width, num_colors, wrap):

        # Number of vertices
        n = height*width

        # Each vertex will have sqrt_num_colors commander variables, which
 will be used to
        #   split the large clauses into (sqrt_num_colors + 1) smaller
clauses
        sqrt_num_colors = int(math.floor(sqrt(num_colors)))

        # Each vertex gets one variables for each color,
        #   plus sqrt_num_colors commander variables
        num_variables = n*(num_colors + sqrt_num_colors)

        # Counts the number of clauses manually
        num_clauses = 0

        # Distance matrix
        dist = Generate_Dist(height, width, wrap)

        # Let P_{v, clr} represent vertex v receiving color clr
        # Variable P_{v, clr} is reassigned to the singly-indexed variable
        #     X_{v*num_colors + clr}
        # Let C_{v, i} be the i^th commander variable for vertex v
        # Variable C_{v, i} is reassigned to the singly-indexed variable
        #     X_{n*num_colors + v*sqrt_num_colors + i + 1}

        clauses = ""

        # Each vertex must receive at least one color
        for v in range(n):
            for i in range(sqrt_num_colors):
                for clr in range((i*num_colors)//sqrt_num_colors + 1, ((i
+ 1)*num_colors)//sqrt_num_colors + 1):
```

```
                    clauses += "%d " % PVar(v, clr, num_colors)

                clauses += "%d 0\n" % CVar(v, n, num_colors, i)
                num_clauses += 1

            for i in range(sqrt_num_colors):
                clauses += "-%d " % CVar(v, n, num_colors, i)

            clauses += "0\n"
            num_clauses += 1


        # Precolor vertex (height//2, width//2) with color num_colors
        precolored = (height//2)*width + width//2
        clauses += "%d 0\n" % PVar(precolored, num_colors, num_colors)
        num_clauses += 1

        # Conflicts with precolored vertex
        for v in range(n):
            if ((v != precolored) and (dist[precolored][v] <= num_colors))
:
                clauses += "-%d 0\n" % PVar(v, num_colors, num_colors)
                num_clauses += 1

        # Other conflicts
        for clr in range(1, num_colors + 1):
            for v1 in range(n):
                for v2 in range(v1 + 1, n):
                    if (dist[v1][v2] <= clr):
                        clauses += "-%d -%d 0\n" % (PVar(v1, clr,
num_colors), PVar(v2, clr, num_colors))
                        num_clauses += 1

        for i in range(n):
            for j in range(n):
                print dist[i][j],
            print
        print

        # Print output
        print "p cnf", num_variables, num_clauses, "\n", clauses
```

The following Sage program is used to interpret the Glucose output string as a packing coloring.

```
    # Reads output from glucose_static SAT solver program
    def Glucose_Output(height, width, num_colors, output_string):

        # Number of vertices
        n = height*width

        # Split output_string at spaces
        str = output_string.split()

        # Cut off commander variables
        str = str[:n*num_colors]
```

```
        # Convert strings to ints
        for i in range(n*num_colors):
            str[i] = int(str[i])

        grid = []
        for row in range(height):
            grid.append([0]*width)

        for row in range(height):
            for col in range(width):
                for clr in range(1, num_colors + 1):
                    if (str[(row*width + col)*num_colors + clr - 1] > 0):
                        grid[row][col] = clr
                        break

        print grid
```

The following Sage code generates input for a C++ program that tests the reducibility of configurations from Figure 5.2. The program will test the graphs in Figure 5.2 as well as graphs generated from these by adding common neighbors. This code was written by Bernard Lidický and is included with his explicit permission. It is included for interested reader who wants to verify the correctness.

```
    # Starts with basic configurations and then it tries to add common
    #     neighbors or add edges to the outside.

    import itertools

    def draw_G_with_external_neighbors(G, with_external_neighbors,name):

        # Drawing of a graph G with with_external_neighbors
        # The external neighbors are indicated by blue edges
        G = G.copy()
        external_neighbors=[0]*G.order()

        for d in with_external_neighbors:
            external_neighbors[d] = 1

        external_edges=dict()
        external_edges["blue"]=[]
        for v in G.vertices():
            for i in range(external_neighbors[v]):
                u = G.add_vertex()
                G.add_edge(u,v)
                external_edges["blue"].append([u,v])
        G.graphplot(edge_colors=external_edges).plot(title="{} with
external".format(name)).show()

    ##################################################

    def str_G(G):

        # Converting a graph G to the format that can read later
        bstr = str(G.order())+" "
```

```
        for i in range(G.order()):
            for j in range(i+1,G.order()):
                if G.has_edge(i,j):
                    bstr += " 2"
                else:
                    bstr += " 1"
            bstr += " "
        return bstr

    ######################################################

    def process_graph_11222(G, with_external_neighbors, name):

        # Preparing graph for 11222 packing coloring
        G=G.copy()
        print name
        print len(with_external_neighbors)
        for v in with_external_neighbors:
            a = G.add_vertex()
            b = G.add_vertex()
            c = G.add_vertex()
            d = G.add_vertex()
            G.add_edges([[v,a],[a,b],[a,c],[c,d]])
            print a,b,c,d
        # G.add_vertex?
        print str_G(G)

    ######################################################

    def process_graph_112233(G, with_external_neighbors, name):

        # Preparing graph for 112233 packing coloring - not used
        print name
        print len(with_external_neighbors)
        for v in with_external_neighbors:
            a = G.add_vertex()
            b = G.add_vertex()
            c = G.add_vertex()
            d = G.add_vertex()
            e = G.add_vertex()
            G.add_edges([[v,a],[a,b],[a,c],[c,d],[c,e]])
            print a,b,c,d,e
        #G.add_vertex?
        print str_G(G)

    ######################################################

    def process_graph(G, with_external_neighbors, name):

        # Preparing graph output
        #draw_G_with_external_neighbors(G, with_external_neighbors,name)
        process_graph_11222(G, with_external_neighbors, name)
        #process_graph_11223(G, with_external_neighbors, name)

    # List of graphs that was already tested
```

```
    # Used to reduce testing the same graphs that differs by isomorphism
many times

    tested_for_adding_common_neighbors=[]
    tested_for_adding_extra_edges=[]

    ###################################################

    def is_planar_with_faces(G, faces):

        # Tests if G is planar subject to having faces being faces
        # faces is a list of lists i.e. [[1,2,3],[3,4,5,6]] and the test
        #     is done by adding an extra vertex adjacent to all vertices
of the face
        # Note it makes no sense to do this for triangular face

        Gtest = G.copy()
        #return Gtest.is_planar()
        for f in faces:
            v=Gtest.add_vertex()
            for u in f:
                Gtest.add_edge(u,v)
        return Gtest.is_planar()

    ###################################################

    def try_adding_extra_edges(G,with_external_neighbor,name,faces):
        global tested_for_adding_extra_edges
        for extra_edge in itertools.combinations(with_external_neighbor,
2):
            if G.distance(extra_edge[0], extra_edge[1]) < 2:
                continue
            Gadd = G.copy()
            Gadd.add_edge(extra_edge)
            if is_planar_with_faces(Gadd,faces) == False:
                continue
            #Gadd.show()
            Gadd_with_external_neighbor = copy(with_external_neighbor)
            Gadd_with_external_neighbor.remove(extra_edge[0])
            Gadd_with_external_neighbor.remove(extra_edge[1])
            #print Gadd_with_external_neighbor
            already_tested=False
            for Gdone in tested_for_adding_extra_edges:
                if Gdone.is_isomorphic(Gadd):
                    already_tested = True
                    break
            if already_tested:
                continue
            tested_for_adding_extra_edges.append(Gadd.copy())

            nameadd=name+"+{}E{}".format(extra_edge[0],extra_edge[1])
            process_graph(Gadd,Gadd_with_external_neighbor,nameadd)
            try_adding_extra_edges(Gadd,Gadd_with_external_neighbor,
nameadd,faces)
```

```
    ##################################################

    def try_adding_common_neighbor(G, with_external_neighbor ,name, faces ,
maxv=None):
        global tested_for_adding_common_neighbors
        global tested_for_adding_extra_edges
        if maxv == None:
            maxv = G.order()
        #for extra_edge in itertools.combinations([2,9], 2):
        for extra_edge in itertools.combinations(with_external_neighbor ,
2):
            if extra_edge[0] > maxv or extra_edge[1] > maxv:
                continue
            if G.distance(extra_edge[0], extra_edge[1]) < 1:
                continue
            Gadd = G.copy()
            v=Gadd.add_vertex()
            Gadd.add_edge(v,extra_edge[0])
            Gadd.add_edge(v,extra_edge[1])
            if is_planar_with_faces(Gadd,faces) == False:
                continue
            Gadd_with_external_neighbor = copy(with_external_neighbor)
            Gadd_with_external_neighbor.remove(extra_edge[0])
            Gadd_with_external_neighbor.remove(extra_edge[1])
            Gadd_with_external_neighbor.append(v)
            already_tested=False
            for Gdone in tested_for_adding_common_neighbors:
                if Gdone.is_isomorphic(Gadd):
                    already_tested = True
                    break
            if already_tested:
                continue
            tested_for_adding_common_neighbors.append(Gadd.copy())

            nameadd=name+"+{}V{}".format(extra_edge[0],extra_edge[1])
            process_graph(Gadd,Gadd_with_external_neighbor ,nameadd)
            try_adding_common_neighbor(Gadd, Gadd_with_external_neighbor ,
nameadd, faces ,None) # None used to be maxv
            tested_for_adding_extra_edges = []
            try_adding_extra_edges(Gadd,Gadd_with_external_neighbor ,
nameadd,faces)

    ##################################################

    def try_everything(G, with_external_neighbor ,name,faces=[]):
        global tested_for_adding_common_neighbors
        global tested_for_adding_extra_edges

        #draw_G_with_external_neighbors(G,with_external_neighbor ,name)
        #return

        #Reset the testings...
        tested_for_adding_common_neighbors=[]
        tested_for_adding_extra_edges=[]
        process_graph(G,with_external_neighbor ,name)
```

```
            try_adding_extra_edges(G, with_external_neighbor ,name,  faces)
            try_adding_common_neighbor(G, with_external_neighbor ,name, faces)

        ################################################

        G=Graph()
        G.add_edges([[0,1],[0,2],[1,2],[1,3],[2,3]])
        try_everything(G,  [0,3], "C3|C3    ")

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,0],[0,5],[1,5]])
        try_everything(G,  [2,3,4,5], "C5|C3    ",[[0,1,2,3,4]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8], [8,0],
[9,0], [9,1]])
        try_everything(G,  [2,3,4,5,6,7,8,9], "C9|C3", [[0,1,2,3,4,5,6,7,8]])

        # With identification of edges
        G=Graph()
        G.add_edges
([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,3],[0,2],[4,7],[5,7]])
        try_everything(G,  [0,1,6,7], "C9|C3X    ",[[0,1,2,3,4,5,6]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,8], [8,13],
[13,0], [9,0],[9,1], [2,10], [3,10], [4,11], [5,11], [6,12],[7,12] ])
        try_everything(G,  [8,9,10,11,12,13], "C10|4*C3",
[[0,1,2,3,4,5,6,7,8,13]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[0,3]])
        try_everything(G,  [0,1,2,3], "C4",[[0,1,2,3]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,0],[3,7]])
        try_everything(G,  [0,1,2,4,5,6], "C5|C5",[[0,1,2,3,7],[3,4,5,6,7]])

        # With identification of edges
        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[2,4],[1,5],[0,5],[5,3]])
        try_everything(G,  [0,4], "C5|C5X    ",[[0,1,2,3,5],[1,2,4,3,5]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,0]])
        try_everything(G,  [0,1,2,3,4,5], "C6    ",[[0,1,2,3,4,5]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,0]])
        try_everything(G,  [0,1,2,3,4,5,6], "C7    ",[[0,1,2,3,4,5,6]])

        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[4,5],[5,6],[6,7],[7,0]])
        try_everything(G,  [0,1,2,3,4,5,6,7], "C8    ",[[0,1,2,3,4,5,6,7]])
```

```
        # With identification of edges
        G=Graph()
        G.add_edges([[0,1],[1,2],[2,3],[3,4],[2,4],[1,5],[0,5]])
        try_everything(G, [0,5,3,4], "C8X    ",[[0,1,2,3,4,5]])
```

The following C++ program tests the reducibility of a graph based on input generated by the previous Sage program. This code was written by Bernard Lidický and is included with his explicit permission.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <cassert>
#include <string>

using namespace std;

const int V = 100;

#define COLORING_11222
//#define COLORING_11223
//#define COLORING_112233


/*
 Colors
0 .. uncolored
1 .. distance 1
2 .. distance 1
3 .. distance 2
4 .. distance 2
5 .. distance 2 (or 3)?
*/

#ifdef COLORING_11222
const int COLORS = 5;
const int MAX_DIST = 2;  // Largest distance
const int PENDANT_VERTICES = 4;
const int outer_colors[][PENDANT_VERTICES]= {
    {0,3,4,5}, {1,2,3,1}, {3,1,2,5},
    {0,3,5,4}, {0,4,5,3},  {1,2,4,1}, {1,2,5,1}, {2,1,3,5}, {2,1,4,5},
   {2,1,5,2},  {4,1,2,5}, {5,1,2,3} };  // In 'triples' of colors....
const int outer_cnt = 12;
const int outer_cnt_first = 3;
const int coldist[]={0,0,0,1,1,1};  // Index of color to conflict graph
const char* filename="input.txt";
#endif


#ifdef COLORING_11223
const int COLORS = 5;
const int MAX_DIST = 3;  // Largest distance
const int PENDANT_VERTICES = 4;
const int outer_colors[][PENDANT_VERTICES]= {
      {0,3,4,5}, {0,3,5,4},  {1,2,3,5},  {1,2,5,1},  {2,1,5,2},  {3,1,2,5},
```

```
    {5,1,2,3}, {0,4,5,3},
      {1,2,4,5}, {2,1,3,5}, {2,1,4,5}, {4,1,2,5},};  // In 'triples' of colors
    ....
const int outer_cnt = 12;
const int outer_cnt_first = 8;
const int coldist[]={0,0,0,1,1,2};  // Index of color to conflict graph
const char* filename="input.txt";
#endif

#ifdef COLORING_112233
//              123456
const int COLORS = 6;
const int MAX_DIST = 3;  // Largest distance
const int PENDANT_VERTICES = 5;
const int outer_colors[][PENDANT_VERTICES]= {
      {0,3,4,5,6}, {0,3,5,4,6}, {0,3,6,4,5}, {0,4,5,3,6}, {0,4,6,3,5},
    {0,5,6,3,4},
      {1,2,3,5,6}, {1,2,4,5,6}, {1,2,5,6,1}, {1,2,6,5,1},
      {2,1,3,5,6}, {2,1,4,5,6}, {2,1,5,6,1}, {2,1,6,5,1},
      {3,1,2,5,6}, {4,1,2,5,6}, {5,1,2,6,1}, {6,1,2,5,1} };  // In 'triples'
    of colors....
const int outer_cnt = 17;
const int outer_cnt_first = 18;
const int coldist[]={0,0,0,1,1,2,2};  // Index of color to conflict graph
const char* filename="input112233.txt";
#endif



vector<vector<int> > tripples;


// Adjacency matrix
int G[MAX_DIST][V][V]; // Index 0 and index 1 are G and G^2
int v = 0; // number of vertices


// List of neighbors
int deg[MAX_DIST][V];
int N[MAX_DIST][V][V]; // Index 0 and index 1 are G and G^2


int coloring[V];


void print_coloring()
{
    for (int x = 0; x < v; x++)
    {
        cout << coloring[x] << " ";
    }
}


bool is_coloring_extendable(int x)
```

```
{
    // We colored all vertices
    if (x >= v)
    {
        //print_coloring();
        //cout << endl;
        return true;
    }

    // Already colored
    if (coloring[x] != 0) return is_coloring_extendable(x+1);

    // Now coloring
    for (int c = 1; c <= COLORS; c++)
    {
        // Test if C is possible
        bool found_conflict = false;

        for (int i = 0; i < deg[coldist[c]][x]; i++)
        {
            if (coloring[ N[coldist[c]][x][i] ] == c)
            {
                found_conflict = true;
                break;
            }
        }
        if (found_conflict) continue;

        coloring[x] = c;
        if (is_coloring_extendable(x+1))
        {
            coloring[x] = 0;
            return true;
        }

    }
    coloring[x] = 0;

    return false;
}

bool generate_precolorings(int to_color)
{
    if (to_color >= (int)tripples.size())
    {
        //cout << "Precoloring ";
        //print_coloring();

        if (!is_coloring_extendable(0))
        {
            cout << "Precoloring ";
            print_coloring();
            cout << " does not extend" <<  endl;
            return false;
        }
```

```
        return true;
    }

    for (int oc = 0; oc < outer_cnt; oc++)
    {
        for (int i = 0; i < PENDANT_VERTICES; i++)
        {
            coloring[tripples[to_color][i]] = outer_colors[oc][i];
        }

        if (!generate_precolorings(to_color+1)) return false;
    }

    return true;
}

bool generate_precolorings_first()
{
    if (tripples.size() > 0)
    {
        for (int oc = 0; oc < outer_cnt_first; oc++)
        {
            for (int i = 0; i < PENDANT_VERTICES; i++)
            {
                coloring[tripples[0][i]] = outer_colors[oc][i];
            }

            if (!generate_precolorings(1)) return false;
        }
    }
    else
    {
        if (!generate_precolorings(0)) return false;
    }

    return true;
}

int main(int argc, char* argv[])
{
    ifstream input;
    input.open (filename);

    cerr << "Opening file: " << filename << endl;

    while(input.good())
    {
        string name;
        while (input.good() && (name.length()==0 || name == " " || name == "
    " || name == "    "))
        {
            std::getline(input,name);
            //cerr  << name << endl;
        }
```

```
if (!input.good()) break;
cout << "'" << name << "'  ";


int triples_count = 0;
input >> triples_count;
//cout << "Loading " << triples_count << " triples" << endl;
tripples.clear();
for (int i = 0; i < triples_count; i++)
{
    vector<int> abc;
    for (int i = 0; i < PENDANT_VERTICES; i++)
    {
        int x;
        input >> x;
        abc.push_back(x);
    }
    tripples.push_back(abc);
}

input >> v;
//cout << "Loading graph on " << v << " vertices" << endl;
assert(v < V);
for (int x = 0; x < v; x++)
{
    G[0][x][x] = 0;
    for (int y = x+1; y < v; y++)
    {
        int e;
        input >> e;
        G[0][x][y] = e-1;
        G[0][y][x] = e-1;
    }
}

// Generate other distances of G
for (int g = 1; g < MAX_DIST; g++)
{
    for (int x = 0; x < v; x++)
    {
        G[g][x][x] = 0;
        for (int y = x+1; y < v; y++)
        {
            G[g][y][x] = G[g][x][y] = G[g-1][x][y];
            for (int z = 0; z < v; z++)
            {
                if (G[g-1][x][z] == 1 && G[0][y][z] == 1)
                {
                    G[g][x][y] = 1;
                    G[g][y][x] = 1;
                }
            }
        }
    }
}
```

```
        // Create list of neighbors
        for (int g = 0; g < MAX_DIST; g++)
        {
            for (int x = 0; x < v; x++)
            {
                deg[g][x] = 0;
                for (int y = 0; y < v; y++)
                {
                    if (y == x || G[g][x][y] == 0) continue;
                    N[g][x][ deg[g][x]++ ] = y;
                }
            }
        }

        // Uncolor everything
        for (int x = 0; x < v; x++) coloring[x] = 0;


        //cout << "Listing non-extendable colorings:" << endl;
        if (generate_precolorings_first())
        {
            cout << "Reducible" << endl;
        }
        //cout << "Done" << endl;
    }

    input.close();
    return 0;
}
```