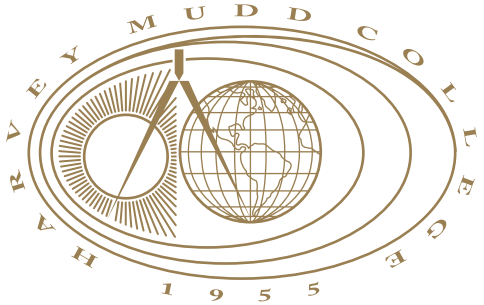
2001

Fractional Analogues in Graph Theory

Ari Nieh
Harvey Mudd College



In Search of the Fractional Four Color Theorem

by
Ari Nieh
Gregory Levin, Advisor

Advisor: _____

Second Reader: _____

(Arthur Benjamin)

May 2001

Department of Mathematics

HARVEY MUDD
COLLEGE

Abstract

In Search of the Fractional Four Color Theorem

by Ari Nieh

May 2001

Tait showed in 1878 that the Four Color Theorem is equivalent to being able to three-color the edges of any planar, three-regular, two-edge-connected graph. Not surprisingly, this equivalent problem proved to be equally difficult. We consider the problem of fractional colorings, which resemble ordinary colorings but allow for some degree of cheating. Happily, it is known that every planar three-regular, two-edge-connected graph is *fractionally* three-edge-colorable. Is there an analogue to Tait's Theorem which would allow us to derive the Fractional Four Color Theorem from this edge-coloring result?

Table of Contents

List of Figures **iii**

Chapter 1: Introduction **1**

Chapter 2: Background and Definitions **4**

2.1 Graph Theory Essentials 4

2.2 Proof of Tait's Theorem 5

2.3 Fractional Graph Theory 6

Chapter 3: Edge Cuts and Primitive Graphs **9**

3.1 Color Parity in Edge Cuts 9

3.2 Examples of Primitive Graphs 11

Chapter 4: Tait's Theorem Revisited **14**

4.1 Another Interpretation of Tait's Theorem 14

4.2 Generalization to $b = 2$ 17

Chapter 5: Conclusion **21**

Appendix A: Appendix **22**

A.1 The Subadditivity Lemma 22

A.2 Linear Programming 22

A.3 Matlab Code 23

Bibliography **33**

List of Figures

2.1 Proof of Tait's Theorem 6
2.2 A 2-fold Coloring of the 5-cycle 7

3.1 Our Graph Before and After Cutting and Rejoining Edges 10
3.2 Primitive Coloring of Petersen's Graph 12
3.3 Primitive Coloring of a Dodecahedron 13

4.1 Tait's Theorem 14
4.2 Another View of Tait's Theorem 15
4.3 Conditions for Consistency of f 16
4.4 Continuous Deformations of a Path in the Plane 17

Acknowledgments

I would like to thank my advisor, Professor Greg Levin, for his invaluable guidance in writing this thesis. I would also like to thank Professor Lesley Ward and Professor Art Benjamin for their feedback.

Chapter 1

Introduction

The Four Color Theorem, which confounded graph theorists for more than a century, also implies the result of interest herein, namely, the *Fractional* Four Color Theorem. However, the only known proofs of the Four Color Theorem are extremely complex and require heavy use of computers. Does a "simple" proof of the Fractional Four Color Theorem exist? We investigate this possibility by searching for a fractional analogue to Tait's Theorem. In Chapter 2, we will give background and definitions. Chapter 3 will consist of progress and results regarding edge cuts in relevant planar graphs, and Chapter 4 will focus on our characterization of primitive graphs. Finally, Chapter 5 will discuss our attempted generalization of Tait's Theorem.

The Four Color Problem

In 1852, while coloring a map of the counties of England, Francis Guthrie found that four colors were sufficient to ensure that adjacent counties were assigned different colors. Naturally, he wondered if it was necessarily true for all maps. In graph theory terms, this corresponds to coloring the faces of a planar graph such that no two faces that share an edge have the same color. Because the vertices of the planar dual of a graph correspond to the faces of the original graph, this problem is equivalent to showing that the chromatic number of a planar graph never exceeds four.

Graph theorists puzzled over this problem for years. Several, including Kempe and Tait, came up with faulty proofs. (In fact, the problem has remained a favorite of cranks to this day.) It was finally proven by Appel and Haken in 1976 through extensive use of computers. Unfortunately, many found the method of proof somewhat unsatisfying. Even today, although their approach has been simplified, no simple proof is known to exist.

Tait's Theorem

In 1873, Tait proved that coloring the edges of any 3-regular, 2-edge-connected planar graph with three colors such that incident edges did not share the same color was equivalent to properly four-coloring its faces. This implied that the Four Color Problem could be solved by finding a way to properly three-color the edges of all such graphs. Unfortunately, this problem proved no more tractable than the original.

Fractional Graph Theory

Fractional graph theory is a field of graph theory that defines rational-valued equivalents of normally integer-valued graph theory concepts. For example, the chromatic number of a graph, typically denoted $\chi(G)$, which represents the fewest number of colors necessary to color the vertices of a graph such that no two adjacent vertices are the same color, is necessarily an integer. The fractional chromatic number $\chi_f(G)$ denotes a similar concept, but can take on rational values. Our goal in this paper is a proof of the Fractional Four Color Theorem, namely, that every planar graph has $\chi_f(G) \leq 4$. Because it is always the case that $\chi_f \leq \chi$, the Fractional Four Color Theorem is implied by the normal Four Color Theorem.

However, to find an alternate way of getting at the Fractional Four Color Theorem, we might try to appeal to the fact that the *fractional* edge chromatic number

of any 3-regular, 2-edge-connected planar graph is known to be three. If there existed a fractional analogue of Tait's Theorem, which would (in an ideal world) take a 3-edge-coloring to a 4-face-coloring in some pleasantly fractional way, then the truth of the Fractional Four Color Theorem could be verified without appealing to the original, non-fractional problem.

Chapter 2

Background and Definitions

This chapter contains the mathematical background and terminology necessary for the remainder of the paper.

In Section 2.1 we review the basics of graph theory. In Section 2.2 we prove the relevant direction of Tait's Theorem. Section 2.3 introduces fractional graph theory and defines the specific notation and terms used in our research.

2.1 Graph Theory Essentials

A graph G consists of two sets- a vertex set V and an edge set E consisting of size 2 subsets of V.[1] Graphs are commonly represented visually, with vertices drawn as points, and edges as lines or curves between their two vertices. A *subgraph* G' of G is a graph with vertex set V' and edge set E' such that $V' \subseteq V$ and $E' \subseteq E$.

A *path* is a sequence of distinct adjacent vertices. A graph is *connected* if there exists a path between any two vertices. A *component* of G is a maximal connected subgraph. We denote the set of edges between two disjoint sets of vertices S and T by $[S, T]$. Such a set is an *edge cut* if S and T are nonempty and $S \cup T = V$. Note that deleting an edge cut necessarily disconnects a connected graph. An edge cut of size one is called a *cut-edge*. A graph is *k-edge-connected* if there does not exist an edge cut of size less than k.

A graph is called *planar* if it can be drawn in the plane without edge-crossings.

[1] This definition, which suffices for this paper, is actually the definition of a *simple graph*. A simple graph does not contain loops (edges from a vertex to itself) or multiple edges between two vertices.

Such a drawing is called a *crossing-free planar embedding*. A graph thus embedded is a *plane graph*. A planar embedding determines *faces* of a graph, which are the regions bounded by its edges.

Assigning a color to each vertex of a graph is called a *vertex coloring*. A coloring is called *proper* if no two adjacent vertices receive the same color. We will frequently denote colors with numbers for the sake of clarity. The *chromatic number* $\chi(G)$ of a graph is the smallest number of colors with which the vertices of G may be properly colored. *Edge colorings*, *proper edge colorings* and *edge chromatic number* are defined analogously. Because only proper colorings are of interest in this paper, we will henceforth abuse our terminology by omission of the qualifier "proper".

The number of edges incident to a vertex is the *degree* of the vertex. A graph is called *k-regular* if every vertex has degree k. A 1-regular graph (or subgraph) is called a *perfect matching*.

A *cycle* is a connected graph (or subgraph) in which each vertex has degree two. It is simplest to think of a cycle as a closed, non-intersecting loop of vertices and edges.

By definition, it is clear that that any 2-regular graph is the union of disjoint cycles.

2.2 Proof of Tait's Theorem

Theorem 1 (Tait 1878) *A 2-edge-connected 3-regular plane graph embedding is 4-face-colorable if and only if it is 3-edge-colorable.*

We will show only the backward direction, as the other direction is irrelevant to our research.

Proof:

We are given a 3-regular 2-edge-connected plane graph G. Assume that the edges have been properly colored with colors a, b, and c. Let E_a, E_b, and E_c be

the subgraphs formed by edges of their respective colors. Let $H_0 = E_a \cup E_b$ and $H_1 = E_a \cup E_c$. Both H_0 and H_1 are 2-regular, because they are the original graph with one color deleted, which removes one edge from each vertex. Therefore, both H_0 and H_1 are unions of disjoint cycles. As such, each can be used to assign binary strings of length two to the faces of G as shown.

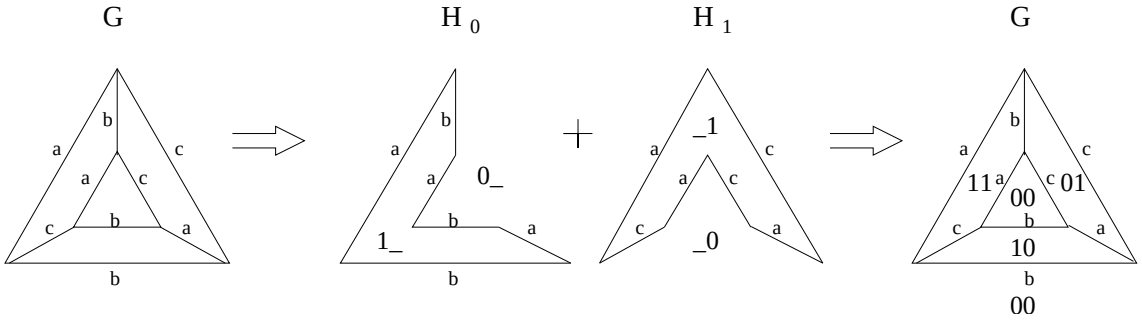


Figure 2.1: Proof of Tait's Theorem

To determine the first bit of each face string, examine H_0. Assign the first bit value one if it is contained in an odd number of cycles, and zero otherwise. (In our example, there are no nested cycles- so the number of cycles a region is contained in is always zero or one.) Similarly, using the subgraph H_1, let the second bit be one if it is contained in an odd number of cycles, and zero otherwise. Then, use the two bits assigned to each face to construct a 4-face-coloring with colors 00, 01, 10, and 11. This coloring is proper because each edge appears in at least one of H_0 and H_1, so adjacent faces must differ in at least one of the two bits. □

2.3 Fractional Graph Theory

In this section, we will define terms specific to fractional graph theory.

A *b-fold vertex coloring* of a graph G assigns to each vertex a set of b distinct colors. Such a coloring is *proper* if adjacent vertices receive disjoint color sets.

The *b-fold chromatic number* $\chi_b(G)$ of a graph is the minimum number of colors necessary to properly b-fold color the vertices of a graph.

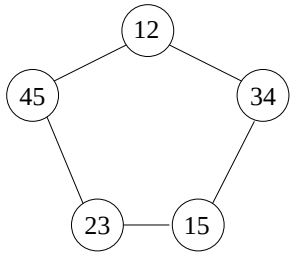


Figure 2.2: A 2-fold Coloring of the 5-cycle

Notice that $\chi_b(G) \leq b \cdot \chi(G)$ for all $b \in \mathbf{Z}$, because simply replicating an ordinary coloring will yield a b-fold coloring of $b \cdot \chi(G)$ colors.

The *fractional chromatic number* is defined by

$$\chi_f(G) = \lim_{b \to \infty} \frac{\chi_b(G)}{b} \tag{2.1}$$

This limit is guaranteed to exist by the subadditivity lemma, and is guaranteed to be achieved for some value of b due to results from linear programming[2]. It is also necessarily achieved for every integer multiple of the smallest such b.

The fractional quantities for edge and face colorings are defined in an analogous manner.

In trying to prove the Fractional Four Color Theorem, our goal is to find some way of transforming a b-fold $3b$-edge-coloring into a t-fold $4t$-face-coloring. Because Tait's theorem guarantees this for $b = 1$, we wish to consider b-fold $3b$-edge-colorings that do not directly "contain" ordinary (non-fractional) 3-edge-colorings. For this reason, it is necessary to define a class of b-fold colorings which are *fundamentally* many-fold, and not merely extensions of ordinary colorings.

[2]See appendix for more details

We call a b-fold $3b$-edge-coloring of a 3-regular graph *primitive* if every possible pair of colors appears on at least one edge. (Note that in such a coloring, each of the $3b$ colors appears on the edges around any given vertex exactly once; that is, all the edges containing any particular color are a perfect matching.) Any non-primitive b-fold $3b$-edge-coloring must contain an ordinary 3-edge-coloring in the following sense– given two colors a and b which never appear on the same edge, color the remaining edges with c. Because both a and b are incident to each vertex exactly once and on different edges, the remaining edges must form a perfect matching. Therefore, the graph is properly 3-edge-colored with a, b, and c, and our fractional coloring "contains" an ordinary coloring.

This definition characterizes the colorings which will be useful for our analogue. When we examine a non-primitive coloring, Tait's Theorem yields an obvious and somewhat unavoidable ordinary 4-face-coloring, preventing us from determining what kind of t-fold $4t$-face-coloring should be implied by our hypothetical analogue.

Call a 3-regular, 2-edge-connected planar graph *strongly primitive* if it has no non-primitive b-fold $3b$-edge-coloring for any positive integer b and hence no ordinary 3-edge-coloring (such a graph would be a counterexample to the Four Color Theorem, but it is convenient to define it nevertheless).

Chapter 3

Edge Cuts and Primitive Graphs

We were interested in examples to guide construction of smallest counterexamples of the Fractional Four Color Theorem. This motivated our study of properties of primitive graphs. To investigate these properties, we derived results regarding edge cuts in b-fold $3b$-edge-colored graphs. Define a *3b-graph* as a b-fold $3b$-edge-colored, 2-edge-connected, 3-regular graph.

The results in this chapter are not directly related to our conclusions, and are included for completeness.

3.1 Color Parity in Edge Cuts

Theorem 2 *In any edge cut of a 3b-graph, each color appears an equal number of times modulo 2.*

Proof: Let $S \subseteq V(G)$ define an edge cut $M = [S, S^c]$, and begin with $S = V(G)$ and $S^c = \{\phi\}$. Notice that initially, the parities of all colors appearing in M are vacuously equal. One at a time, move vertices from S to S^c until M is the edge cut in question. Because each of the $3b$ colors is incident to any vertex exactly once, each move either adds or subtracts one from the number of appearances of each color in M, which leaves their relative parities unchanged. Therefore, the edge cut must use every color with the same parity. □

Corollary: In any edge cut of size two in such a graph, both edges must be identically colored.

Proof: Each edge only uses b colors. Therefore, not every one of the $3b$ colors

can be used by two edges. Since at least one color is used zero times, all colors must be used an even number of times. Therefore, no color can appear in one edge's color set and not in the other's, and the two color sets must be identical.

We can now prove that the smallest counterexample to the Four Color Theorem is 3-edge-connected. (Unfortunately, Tait had gotten to this a century before we did using relatively simple methods of normal graph theory.)

Corollary: The smallest strongly primitive 3-regular 2-edge-connected planar graph must be 3-edge-connected.

Proof: Assume that there exists an edge cut of size two. Then the two edges are identically colored, and two smaller graphs may be formed by cutting both edges and joining them internally.

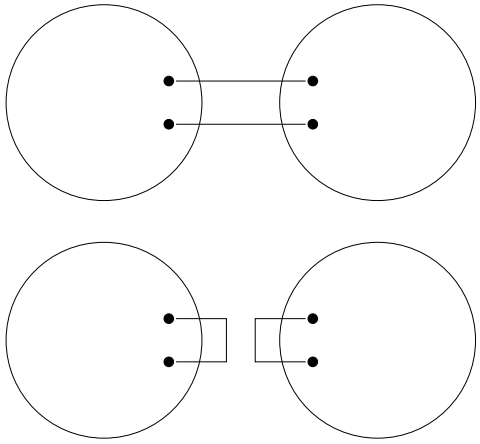


Figure 3.1: Our Graph Before and After Cutting and Rejoining Edges

Since the two smaller graphs formed cannot be strongly primitive, 3-color their edges. Then, after "synchronizing" the two colorings by appropriate permutations, cut and rejoin the edges to form the original graph, colored in a non-primitive way. This is a contradiction, so no such edge cut can exist. □

There are several nearly identical corollaries using similar arguments.

Corollary: The smallest strongly primitive 3-regular 2-edge-connected graph must be 3-edge-connected. □

Corollary: The smallest primitive 3-regular 2-edge-connected planar graph must be 3-edge-connected. □

Corollary: The smallest primitive 3-regular 2-edge-connected graph must be 3-edge-connected. □

Finally, since we have eliminated the possibility of size two edge cuts, we examine the next case.

Corollary: Given any edge cut of size three in such a graph, either each color is represented once, or half of the colors are represented twice and each edge shares a distinct half of its color set with each of the other two edges. □

If $b = 2$, then only the former case is possible. The proof is similar to that of Theorem 3.1, beginning with M as the given edge cut, and showing invariance of the parities of certain color combinations to eliminate the latter case.

3.2 Examples of Primitive Graphs

In an effort to find any example of a primitive 3b-graph, planar or not, we set $b = 2$ and looked for graphs that used every possible pair of colors on an edge. To find the smallest possible example, we looked at the 15 possible pairs of two colors.

Theorem 3 *Petersen's graph is the smallest primitive 3b-graph for* $b = 2$.

Proof: By our definition of primitivity, each possible color pair must appear together on some edge. Since there are $\binom{6}{2} = 15$ such pairs, any primitive graph for $b = 2$ must have at least 15 edges. Because 3b-graphs are 3-regular, this is equivalent to having at least 10 vertices. Petersen's graph satisfies these lower limits and is therefore minimum. Uniqueness follows by considering cases. □

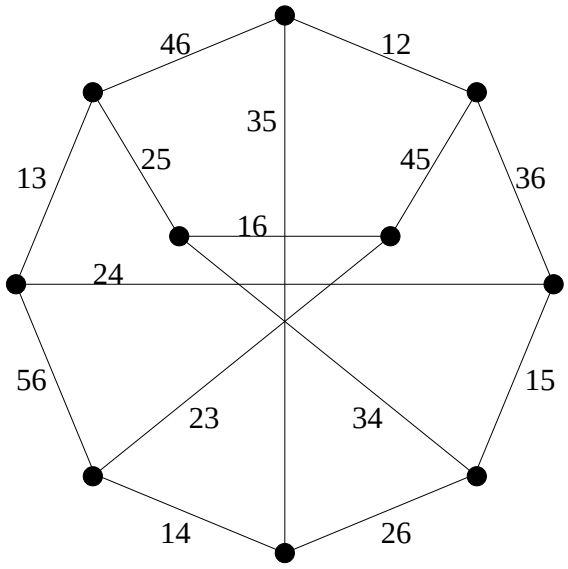


Figure 3.2: Primitive Coloring of Petersen's Graph

Unfortunately, Petersen's graph is not planar, so it is not a useful example for examining the relationship between the b-fold edge-coloring and any corresponding t-fold face-coloring.

After some mutation of Petersen's graph, we found that the dodecahedron is a planar $3b$-graph with a primitive 2-fold 6-edge-coloring.

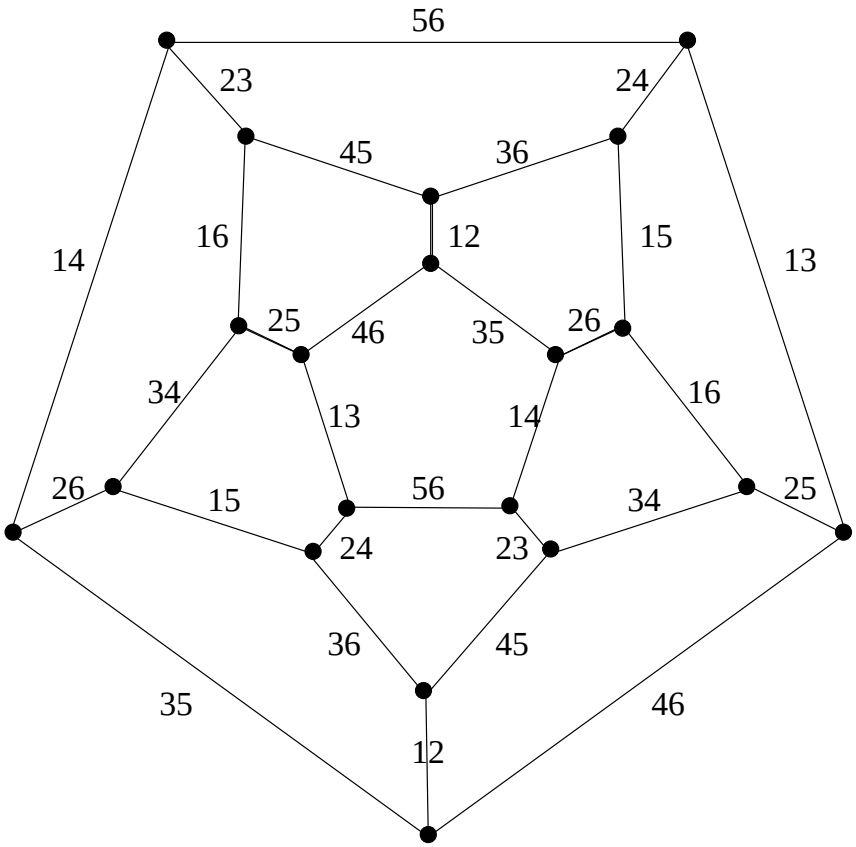


Figure 3.3: Primitive Coloring of a Dodecahedron

Chapter 4

Tait's Theorem Revisited

This chapter examines a more general interpretation of Tait's Theorem. In Section 4.1 we derive the alternate interpretation of Tait's Theorem. In Section 4.2 we generalize the logic behind the theorem. We then attempt to apply it to the case $b = 2$.

All graphs in this chapter are assumed to be planar unless otherwise specified.

4.1 Another Interpretation of Tait's Theorem

In our proof of Tait's Theorem, note that color a always appears on edges between faces whose binary strings differ in both bits. This is because the a edges appear in both H_0 and H_1. Similarly, edges with color b always appear between faces whose strings differ in only the first bit, because b is only in H_0. Lastly, edges with color c must appear between faces whose strings differ in only the second bit, because c is only in H_1.

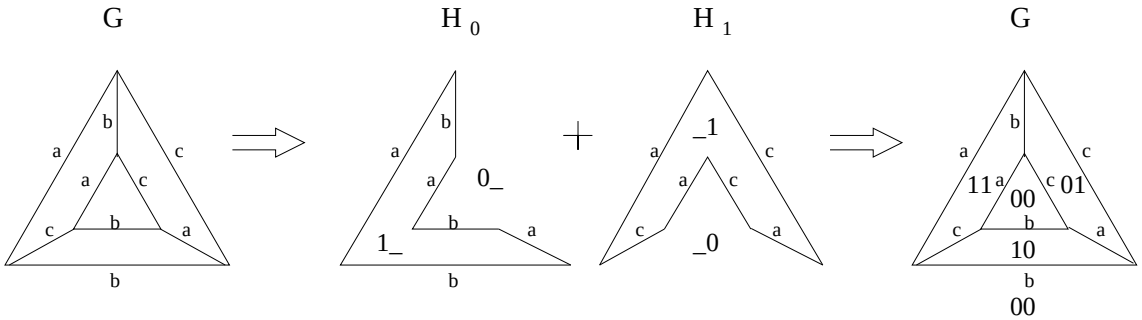


Figure 4.1: Tait's Theorem

In other words, the face strings generated by Tait's Theorem determine a unique function f from the edge color sets to binary edge strings. In this case, $f(a) = 11$, $f(b) = 10$, and $f(c) = 11$.

In fact, this function determines the binary face strings generated by Tait's Theorem in the following manner: use f to assign a binary string to each edge. Assign the unbounded face the zero string. Then, use the operation of binary XOR to fill in face strings. In other words, when "stepping over" an edge between faces F_1 and F_2 separated by edge color set C, give the string on F_1 value equal to the string on F_2 added bitwise modulo 2 to $f(C)$.

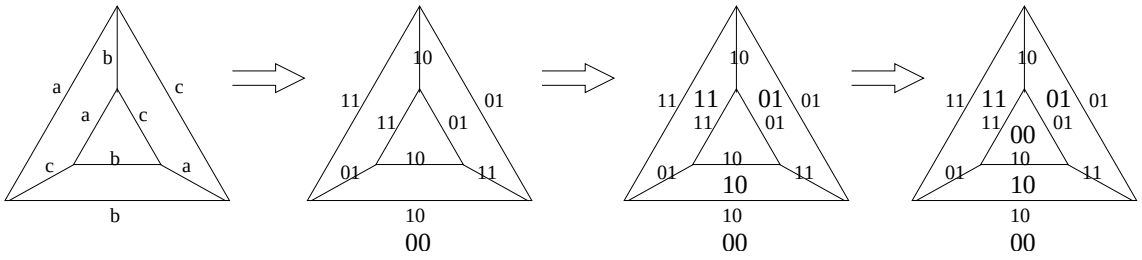


Figure 4.2: Another View of Tait's Theorem

While it is clear that a valid f results from the 4-face-coloring generated by Tait's theorem, we can make a more general claim about when a function f from all possible edge color sets into all binary strings of a certain length can be used to generate a t-fold $4t$-face-coloring. (Naturally, because we are attempting to find an analogue to Tait's Theorem, we would rather start with a $3b$-edge-coloring and devise an f which will generate face strings in this manner.) We call an assignment of binary face strings *proper* if adjacent faces receive different strings. We denote this assignment $h_f(F)$, a function mapping faces to binary strings as defined by the binary XORs generated by f. It is clear the h_f is proper if and only if it maps adjacent faces to different strings.

Theorem 4 *Given a function f that maps size b subsets of our $3b$ edge colors to binary*

strings of length l, h_f is proper *on any planar 3b-graph if and only if the following two conditions hold:*

(I) The zero string is not in the range of f.

(II) If A, B, and C are disjoint edge color sets each of size b, then the binary sum (or XOR) of $f(A)$, $f(B)$, and $f(C)$ is the zero string.

Note that in the second condition, $A \cup B \cup C$ is the set of all possible face colors, and A, B, and C are sets that could appear on the three edges incident to a single vertex.

Proof: It is fairly clear that these conditions are necessary. If there existed a color set C such that $f(C)$ was the zero string, then any graph with the color set C on an edge would have two adjacent faces F_1 and F_2 with identical binary strings. Similarly, if the second condition did not hold for some disjoint size b color sets A, B, and, C, any graph with a vertex v around which A, B, and C appeared could not have consistent face strings. If we were to start at D, one of the three faces adjacent to v and follow a closed loop around it passing through the three edges, we would end at D, and the binary difference between $h_f(D)$ and itself would be nonzero. This is impossible.

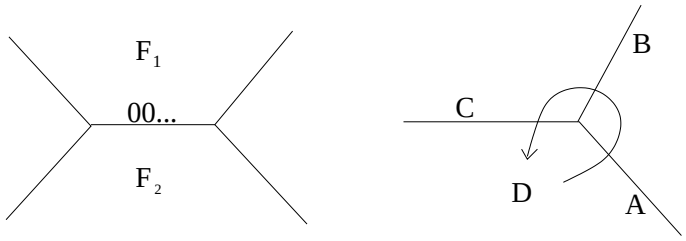


Figure 4.3: $h_f(F_1) = h_f(F_2)$, $h_f(D) = h_f(D) + f(A) + f(B) + f(C)$

Showing that these conditions are sufficient is slightly more involved. We must show that h_f is well-defined. That is, given a face F of a 3b-graph, suppose we

consider multiple paths in the plane between the unbounded face and F. How do we know that the sum of the binary strings on the edges through which a path passes is a constant for all paths? To prove this, we use condition (II). Because the plane is simply connected, all such paths can be continuously deformed into each other. Therefore, we can discuss what happens when a path is deformed through a vertex. Because the sum of the strings on the edges surrounding any vertex is the zero string, the binary sum of the edges through which a path passes is invariant under continuous deformations of that path, as shown. Therefore, h_f is well-defined. Finally, the first condition clearly suffices to ensure that adjacent face strings are different. □

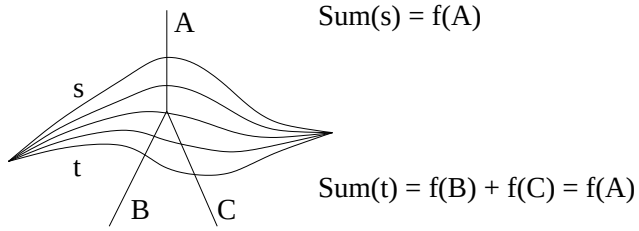


Figure 4.4: $f(A) + f(B) + f(C) = 00...$

4.2 Generalization to $b = 2$

Now that we know precisely what conditions on f generate consistent, proper face strings on any planar $3b$-graph, we can attempt to apply this to the case $b = 2$ and find an analogue of Tait's Theorem. If one does exist, proving it for $b = 2$ will likely shed some light on the general case. If one does not exist, then $b = 2$ might very well be the easiest counterexample.

Our plan of attack, then, is to find some f satisfying our properties, and then another function g mapping face strings to face color sets of size t chosen from $4t$

colors. Note that in our original proof of Tait's Theorem, g was somewhat trivial, as $t = 1$. Clearly, the necessary and sufficient condition on g for it to complete the analogue are that it maps potentially adjacent face strings (that is, face strings whose binary difference is in the range of f) to disjoint face color sets.

To find a possible f, we considered all 1-bit binary functions which satisfied condition (II) on f. (Clearly, only $f = 1$ could satisfy condition (I) for strings of length 1.) This set is an abelian group under binary XOR (because the XOR operation preserves condition (II)), and each of its 32 elements has order 2, so it is isomorphic to $Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2$. This set is easiest viewed as a matrix whose rows are the fifteen color pairs and whose columns are the functions. (Matlab code for generating this matrix is included in the appendix.)

$$\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}$$

It is not hard to see that all possible functions f fulfilling the desired conditions can be made by "smashing together" these 1-bit functions. In order to choose an f which will put sufficient structure on the face strings of our graph to enable us to find a valid g, we must pick some number of columns from our matrix. Linearly independent sets of columns are the only ones of interest, because linearly dependent columns will not provide additional structure on the face strings. Clearly,

fewer than three will result in violation of condition (I).

Picking three columns will not suffice, either. If we choose a submatrix of three columns with no row of zeroes (which is equivalent to the first condition), all possible nonzero binary strings of length three appear in the range of f. Clearly, there is no way to choose g such that face strings whose difference is in the range of f receive disjoint color sets, because the range of f is all of $\{0,1\}^3$.

Because our group is isomorphic to $Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2 \oplus Z_2$, the column space of any subset cannot have dimension greater than five. Therefore, if we seek an f that leads to a working analogue, we must pick strings of length four or five.

We have shown by computer that all sets of four linearly independent columns have at least eleven different rows. That is, the range of f has size eleven. We know that all possible face strings in $\{0,1\}^4$ could occur in some graph, because our submatrix has rank four. Therefore, given a face color string a, $g(a)$ must be disjoint from $g(b)$ for at least eleven strings $b \in \{0,1\}^4$. It follows that for any face color string a, $g(a)$ can overlap at most four other color sets.[1]

If g is one-to-one, we can model g as a hypergraph problem. A $hypergraph$ is a vertex set V and a $hyperedge$ set E composed of non-empty subsets of V. If g exists in this case, then there must exist a hypergraph with $4t$ vertices (representing the colors) and 16 t-hyperedges (that is, color sets of size t). It must also be the case that none of these 16 hyperedges is adjacent to more than four others. Such a hypergraph is a combinatorial impossibility.

Theorem 4.2.1 *There does not exist a hypergraph with $4t$ vertices and 16 t-hyperedges such that no hyperedge is adjacent to more than four others.*

Proof: Assume that such a hypergraph G exists. Choose a hyperedge a, and consider the subgraph G_1 induced by removing all vertices contained in a. G_1 has $3t$ vertices and at least 11 hyperedges, because a was adjacent to at most four

[1] $11 + 4 + 1 = 16$

others. Now pick a hyperedge b in G_1, and consider the subgraph G_2 induced by removing all vertices contained in b. By similar reasoning, G_2 has $2t$ vertices and at least six hyperedges.

I claim that G_2 is its own component in G. Consider a hyperedge e in G_2. Because e contains t vertices, only one of the other hyperedges in G_2 (namely, the one containing the other t vertices) can be disjoint from e. Therefore, e is adjacent to four other hyperedges in G_2, so none of the vertices in e can be in hyperedges from outside of G_2. Because e was selected without loss of generality, it follows that no vertex in G_2 can be part of an outside edge, so G_2 is its own component. It is also clear that G_2 must have exactly six hyperedges. If it had more, then e would have to be adjacent to more than four others.

This implies that the hypergraph $G-G_2$, which is well defined because no edges connect G_2 to the rest of G, has $2t$ vertices and ten hyperedges. This is clearly impossible, as we have just shown that a $2t$ vertex hypergraph with these conditions can have no more than six hyperedges. By contradiction, G does not exist. □

However, we do not know that g must be one-to-one. In fact, if our edge-coloring is not primitive, a g that is not one-to-one will properly four-color the faces of our graph! This leaves us with the question- if we have a primitive $3b$-edge-coloring, must g be one-to-one? Because non-primitive colorings yield obvious solutions via Tait's Theorem, this would eliminate the case of length four binary strings for $b = 2$ and allow us to focus on length five strings.

Chapter 5

Conclusion

Our research raises several unanswered questions worthy of further investigation.

-Is Petersen's graph the unique minimum primitive graph, independent of b?

-Does primitivity of a $3b$-edge-coloring imply that g must be one-to-one?

-Can an analogue of Tait's Theorem actually be found for $b = 2$, using $\{0, 1\}^5$ as our range of f and domain of g?

-If so, how can it be extended to all values of b?

-Is there a different way to generalize Tait's Theorem to the fractional case?

Appendix A

Appendix

A.1 The Subadditivity Lemma

A function f mapping the positive integers to R is $subadditive$ if $f(a) + f(b) \geq f(a+b)$ for all a, b.

The lemma itself states that if f is non-negative and subadditive, the limit as $n \to \infty$ of $\frac{f(n)}{n}$ exists and is equal to the infimum of $\frac{f(n)}{n}$ for all n.

A.2 Linear Programming

An alternate way to define the chromatic number of a graph is through linear programming. An *independent set* is a set of vertices, none of which have any edges between them. $\chi(G)$ is equal to the solution of the problem:

Minimize $\mathbf{c^T x}$ subject to $A\mathbf{x} \geq \mathbf{b}$ where $\mathbf{c}$ and $\mathbf{b}$ are appropriately sized vectors of all ones, A is the vertex-independence set adjacency matrix, and $\mathbf{x}$ is a vector of zeroes and ones. We can view the vector $\mathbf{x}$ as picking a set of independent sets such that every vertex is contained in at least one. A minimal such cover of the vertices is a minimal coloring, where one color is assigned to the vertices of each independent set. Therefore, the solution to this problem is the minimum number of colors needed to properly color the vertices of a graph- the chromatic number.

However, if we relax the requirement that entries of $\mathbf{x}$ are from $\{0, 1\}$, and instead allow them to be chosen from $[0, 1]$, the solution to our linear program is instead $\chi_f(G)$, the fractional chromatic number.

A.3 Matlab Code

The following matlab code, written by Professor Greg Levin, was instrumental in dealing with examples in our research.

```
% CHOOSE(n,k) returns "n choose k"

function m = choose(n,k)

%"out of range" input
if k>n
  m = 0;
  return;
end

%negative input
if (k<0) | (n<0)
  m = 0;
  return;
end

%non-integral input
if (k ~= floor(k)) | (n ~= floor(n))
  m = 0;
  return;
end
```

```
if k > (n/2)
  k = n-k;
end

m = prod((n-k+1):n)/prod(1:k);

% CHOOSE4   Iterates through all 21 C 4 col sets of G
%   This routine runs thru all size four sets of columns of the
%   matrix (group) G generated by MAKEG.M, and in each corresponding
%   15x4 submatrix, counts the number of *distinct* rows


counter = zeros(1,15);
foo = [0 0 0 0];

% Check each size four subset of G's columns, designated M
subset = [ones(4,1);zeros(27,1)];
while (subset ~= Inf)
  M = G(:,find(subset));
  count = distrows(M);
  % check for zero rows
  if ~all(sum(M,2))
    counter(1) = counter(1)+1;
  % report new row count
  else
    if counter(count) == 0
      count
    end
```

```
    counter(count) = counter(count)+1;
    if count == 7
      foo = [foo ; find(subset)'];
%      yeah = M;
%      return
    end
  end
  % report every 1000th subset
  if mod(sum(counter),1000)==0
    total = sum(counter)
  end
  subset = nextsub(subset);
end

% CHOOSE4   Iterates through all 21 C 4 col sets of G
%   This routine runs thru all size four sets of columns of the
%   matrix (group) G generated by MAKEG.M, and in each corresponding
%   15x4 submatrix, counts the number of *distinct* rows


counter = zeros(1,15);


% Generate the indicator strings for 31 C 4 (time consuming)
C31_4 = makecomb(31,4);


% Check each size four subset of G's columns, designated M
```

```
for subset=C31_4
  M = G(:,find(subset));
  count = distrows(M);
  % report new row count
  if counter(count) == 0
    count
  end
  counter(count) = counter(count)+1;
  % report every 1000th subset
  if mod(sum(counter),1000)==0
    total = sum(counter)
  end
  if count == 9
    yeah = M;
    return
  end
end

% DISTROWS(M)   Counts the distinct rows of the matrix M

function count = distrows(M)


rows = size(M,1);
count = rows;
i=1;

while (i<rows)
```

```
  for j=(i+1):rows
    if isequal( M(i,:) , M(j,:) )
      count = count-1;
      break
    end
  end
  i = i+1;
end

% MAKECOLD(N,K)   Returns an array representing N choose K
%   MAKECOLD(N,K) returns an N-by-(N choose K) array containing
%   every possible length N binary string with K ones.
%   Note that MAKECOMB is a faster, non-recursive routine
%   with the same functionality.

function C = makecold(n,k)



% We run recursively by putting 1s above makecomb(n-1,k-1)
% and 0s above makecomb(n-1,k).



% First check consistency
C = Inf;
if (fix(n) ~= n) | (fix(k) ~= k)
  return
end
if (n < 0 | k < 0 | n < k)
```

```
  return
end


% Next handle base cases
if (n==1)
  C = k;
  return
elseif (k==0)
  C = zeros(n,1);
  return
elseif (k==n)
  C = ones(n,1);
  return
end


% Now handle recursion
C1 = [ ones(1,choose(n-1,k-1)) ; makecomb(n-1,k-1)];
C2 = [zeros(1,choose(n-1,k))   ; makecomb(n-1,k)];
C = [C1 , C2];

% MAKECOMB(N,K)   Returns an array representing N choose K
%   MAKECOMB(N,K) returns an N-by-(N choose K) array containing
%   every possible length N binary string with K ones.
%   Unlike the recursive MAKECOLD, MAKECOMB uses the
%   NEXTSUB subroutine.
```

```
function C = makecomb(n,k)


% We run recursively by putting 1s above makecomb(n-1,k-1)
% and 0s above makecomb(n-1,k).


% First check consistency
C = Inf;
if (fix(n) ~= n) | (fix(k) ~= k)
  return
end
if (n < 0 | k < 0 | n < k)
  return
end


% initialize
nCk = choose(n,k);
C = zeros(n,nCk);
X = [ones(k,1);zeros(n-k,1)];
i = 0;


% fill in C
while X ~= Inf
  i = i+1;
  C(:,i) = X;
```

```
  X = nextsub(X);
end

% MAKEG   Generates the funky edge-color group ~ Z_2^5
%   This argumentless function returns the 15x31 matrix whose
%   columns are the binary vectors in the group of permissible
%   bit assignments to a 2-fold six coloring of a 3-regular graph.
%   It also creates G1, G2 and G3, the submatrices corresponding
%   to size 1, 2 and 3 color sets.

%function G = makeg()

G = zeros(15,31);
C62 = makecomb(6,2);


% Generate the 6 elements (columns) corresponding to the six colors
G1 = ones(15,6) - C62';


% Generate the 6 C 2 elements corresponding to color pairs
G2 = mod(G1*C62,2);


% Generate the (6 C 3)/2  elements corresponding to color triples
C63 = [ones(1,choose(5,2)) ; makecomb(5,2)];
G3 = mod(G1*C63,2);
```

```
G = [G1,G2,G3];

% NEXTSUB(X)   Returns the next equally sized subset
%   If X is a binary column vector with k ones (representing
%   a size k subset of an n set), the "next" size k subset
%   of an n set is returned, or Inf if X is the "last" subset

function Y = nextsub(X)

% check for column vector
if (size(X,2) > 1)
  Y = 'thats not a col vector, you dolt'
  return
end



% initialize values
n = size(X,1);
elts = find(X);
k = size(elts,1);
if k==0
  Y = Inf;
  return
end
```

```
% find which bit is free to move
move = k;
while elts(move)==(n-(k-move))
  move = move-1;
  if move==0
    Y = Inf;
    return;
  end
end


% move bit forward one, and put all following bits right after
next = elts(move)+1;
for i=move:k
  elts(i) = next+(i-move);
end


% fill in Y
Y = zeros(n,1);
Y(elts,:) = ones(k,1);
```

Bibliography

[1] Edward R. Scheinerman and Daniel H. Ullman. *Fractional Graph Theory*. John Wiley & Sons, Inc., 1997.

[2] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, 1996.