Markoff Numbers and Computations on the Markoff Graph

By

COLBY AUSTIN BROWN
DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Mathematics

in the

OFFICE OF GRADUATE STUDIES

of the

UNIVERSITY OF CALIFORNIA

DAVIS

Approved:

Elena Fuchs, Chair

Martin Luu

Junxian Li

Committee in Charge

2026

Dedicated to Mom and Dad.

Thank you for your endless and continuing support and encouraging me to pursue a lifetime of learning.

© Colby Austin Brown, 2026. All rights reserved.

Table of Contents

Abstract . . . . . **iv**

Chapter 1. The Markoff Equation

1.1 The Lagrange and Markoff Spectra 1

1.2 The Markoff Equation 4

1.3 The Markoff Graph 8

Chapter 2. The Markoff graph modulo p

2.1 The Markoff Equation modulo p 10

2.2 Fixed Points of the Vieta Involutions 14

2.3 The method of Bourgain, Gamburd, and Sarnak 18

Chapter 3. An algorithmic approach to BGS

3.1 The structure of the Markoff graph modulo p 24

3.2 Computational Machinery 29

3.3 Our Algorithm & Results 33

Chapter 4. Markoff Triples in Lean 4

4.1 The Lean 4 Type System 41

4.2 The Markoff Triples 49

4.3 A recursive principle for walks in the Markoff graph 57

4.4 Decidability and Computability 60

Chapter 5. Tree decompositions of Markoff graphs

5.1 Expander Families and the Cheeger Constant 62

5.2 Tree decompositions and treewidth 63

5.3 Decompositions of Markoff Graphs 67

5.4 Future work 72

Appendix: Lean code for Chapter 4 . . . . . **74**

Bibliography **96**

Abstract

This dissertation covers Markoff numbers and the Markoff equation, primarily from a computational point of view. The Markoff equation arises in many contexts, including the study of binary quadratic forms and approximations of irrational numbers by sequences of rationals.

In Chapter 1, we introduce the Markoff equation as

$$x^2 + y^2 + z^2 = 3xyz$$

and consider its positive integer solutions. We cover some of the history of the theory of these solutions and prove some of their basic number theoretic properties of use throughout the rest of this dissertation.

In Chapter 2, we consider the same equation (up to a constant scaling factor) in the mod p context for prime p. Central to this dissertation is Baragar's "Strong Approximation Conjecture" (1996), that all solutions to the Markoff equation mod p lift to solutions over the integers. Central to the study of the arithmetic of Markoff triples are a family of graphs known as the Markoff mod p graphs which we introduce. We also introduce the framework of Bourgain, Gamburd, and Sarnak (2016) for understanding these graphs and outline the current state of research towards the Strong Approximation Conjecture.

Chapter 3 describes a refinement of the Bourgain, Gamburd, and Sarnak algorithm suitable for explicit computations. We show that this algorithm runs in $\mathcal{O}(p^{1+\epsilon})$ time, and as a result of running this algorithm conclude that the Strong Approximation Conjecture holds for all primes less than $1,000,000$.

Chapter 4 translates the elementary properties of the Markoff equation into the language of computer verifiable proofs using the Lean 4 language and proof assistant. We cover the basics of type theory and use the Markoff equation as a case study in formalizing mathematical results.

Finally, in Chapter 5 we explore Markoff triples in the framework of tree decompositions, which capture many of the computational properties of graphs. We discuss how the problem of finding such decompositions relates to a conjecture on the expansion of Markoff graphs and outline ideas for future study in this direction.

Chapter 1

The Markoff Equation

In 1879 and 1880, Andrey Markoff published a pair of papers ([29] and [30]) on binary quadratic forms in which the equation $x^2 + y^2 + z^2 - xyz = 0$, now known as the Markoff equation, played a central role. In this chapter, we provide background on the Markoff equation and its solution set, culminating in a description of the Markoff graph. We begin in Section 1 with historical origins of the Markoff numbers and their connection with approximations of real numbers by rationals. In Section 2, we look at the Markoff equation and give a method for recursively enumerating all of its integer solutions. Finally, in Section 3, we define the Markoff tree and prove some of its basic properties. The material in this chapter is adapted from [12] and [1].

1.1 The Lagrange and Markoff Spectra

We begin by considering how well a real number $\alpha \in \mathbb{R} \setminus \mathbb{Q}$ may be approximated by rational numbers p/q where $|\alpha - p/q|$ is bounded above by a multiple of q^{-2}. For any irrational number $\alpha \in \mathbb{R} \setminus \mathbb{Q}$, define

$$L(\alpha) = \sup \left\{ L \in \mathbb{R} : \left| \alpha - \frac{p}{q} \right| < \frac{1}{Lq^2} \text{ for infinitely many } p/q \in \mathbb{Q} \right\} \tag{1.1}$$

which we may think of as a measure for how well α may be approxmiated by rational numbers. In this definition, we restricted α to be irrational; this is a necessary restriction since, as the next lemma will show, rational numbers cannot be approximated at all by other rational numbers in the sense of (1.1).

Lemma 1.1. *Let $\alpha = r/s$ for $r, s \in \mathbb{Z}$ and $L \in \mathbb{R}$. There are only finitely many rational numbers p, q such that*

$$\left| \alpha - \frac{p}{q} \right| < \frac{1}{Lq^2}.$$

Proof. Let $r/s \neq p/q$ satisfying the condition in (1.1). Cross multiplying the inequality, we see that $qr - sp \neq 0$ and in particular $|qr - sp| \geq 1$. We now have

$$\frac{1}{sq} \leq \left| \frac{rq - sp}{sq} \right| = \left| \frac{r}{s} - \frac{p}{q} \right| < \frac{2}{Lq^2}$$

and re-arranging $s < 2qL$. Therefore, there are only finitely many such values s, and for each of them, only finitely many r. □

Furthermore, the exponent of 2 is (1.1) is not arbitrary. Consideration of approximations of degree 2 is motivated by the following pair of theorems.

Theorem 1.2 ([26])**.** *Let $\alpha \in \mathbb{R} \setminus \mathbb{Q}$ be algebraic of degree d. Then, there is a constant $C > 0$ such that*

$$\frac{C}{q^d} < \left| \alpha - \frac{p}{q} \right|$$

for all $p/q \in \mathbb{Q}$.

Theorem 1.3 ([36])**.** *Let $\alpha \in \mathbb{R}$ and $\epsilon > 0$. If there are infinitely many $p/q \in \mathbb{Q}$ such that*

$$\left| \alpha - \frac{p}{q} \right| < \frac{1}{q^{2+\epsilon}},$$

then α is transcendental.

Theorem 1.2 tells us that algebraic numbers can be approximated to degree at least 2, and Theorem 1.3 tells us that 2 is the best possible degree of approximation.

Now, $\sqrt{5}$ is an element of the set in (1.1) for every $\alpha \in \mathbb{R} \setminus \mathbb{Q}$ as shown by Markoff in 1879 [29] and again by Hurwitz in 1891 [23]. We therefore have not only that $L(\alpha)$ exists for all irrational numbers but is bounded below by $\sqrt{5}$.

Theorem 1.4 ([23])**.** *$\sqrt{5} \leq L(\alpha)$ for all $\alpha \in \mathbb{R} \setminus \mathbb{Q}$.*

Definition 1.5. The *Lagrange spectrum* is the set of real numbers

$$\mathcal{L} = \{L(\alpha) \mid \alpha \in \mathbb{R} \setminus \mathbb{Q}\}.$$

Now let $f(x, y)$ be a real binary quadratic form

$$f(x, y) = ax^2 + bxy + cy^2$$

and consider the infimum (in absolute value) of f over all points on the integer lattice

$$m(f) = \inf_{\substack{(x,y)\in\mathbb{Z}^2 \\ (x,y)\neq(0,0)}} |f(x,y)|.$$

Also, recall that the discriminant of a binary form is given by $d(f) = b^2 - 4ac$.

Definition 1.6. The *Markoff spectrum* is the set of real numbers

$$\mathfrak{M} = \left\{ \frac{\sqrt{d(f)}}{m(f)} : d(f) > 0 \right\}$$

where f ranges over all real binary quadratic forms with positive discriminant.

The Lagrange spectrum is a proper subset of the Markoff spectrum, but below 3 the two spectra coincide. This was shown by Markoff in his 1879 and 1880 papers along with a characterization of the portions of the spectra below 3 in terms of the solutions to what is now called the Markoff equation.

Theorem 1.7 (Markoff's theorem [29, 30])**.** *The Lagrange spectrum below three is given by*

$$\mathfrak{M}_{<3} = \mathfrak{L}_{<3} = \left\{ \frac{\sqrt{9m^2-4}}{m} \right\}$$

where m ranges over all positive integers which appear as a coordinate in the integer solutions to

$$x^2 + y^2 + z^2 = 3xyz. \tag{1.2}$$

Below 3, both the Lagrange and Markoff spectra are discrete sets with 3 as the first limit point of either spectrum (we will show in Section 1.2 that there are arbitrarily large Markoff numbers). In 1977, Schecker [37] proved that every real number at least

$$\frac{2221564096 + 283748\sqrt{462}}{491993569} \approx 4.5278$$

lies in both $\mathfrak{L}$ and $\mathfrak{M}$; this number is known as Freiman's constant. Past this point, both spectra are therefore continuous; this portion of the spectra is known as Hall's Ray. Between 3 and Freiman's constant, $\mathfrak{L}$ is a proper subset of $\mathfrak{M}$, but both spectra contain uncountably many limit points and a fractal structure.

1.2 The Markoff Equation

Equation (1.2) is the *Markoff equation*, and the solution set will be denoted $\mathcal{M}$.

Clearly, $\mathcal{M}$ is closed under permutation of its three variables. It is also closed under the operation of flipping the sign of two of its coordinates, called a double sign change and indexed by the variable left alone:

$$\tau_1(x, y, z) = (x, -y, -z)$$
$$\tau_2(x, y, z) = (-x, y, -z)$$
$$\tau_3(x, y, z) = (-x, -y, z).$$

The Markoff equation has a trivial solution $(0, 0, 0)$ which we will often want to exclude; in these cases, we will denote the set of nontrivial solutions as $\mathcal{M}^\times$. Furthermore, 0 does not appear in any nontrivial solution.

Lemma 1.8. *The only solution to the Markoff equation with any coordinate 0 is the trivial solution.*

Proof. Without loss of generality let $a = 0$. Then the Markoff equation reads $b^2+c^2 = 0$, hence $b = c = 0$. □

Any positive integer appearing in a nontrivial solution to the Markoff equation is a *Markoff number*, and the set of Markoff numbers is denoted $x(\mathcal{M})$.

Lemma 1.9. *The only solution to the Markoff equation with repeat coordinates are* $(1, 1, 1)$ *and* $(2, 1, 1)$ *and its permutations.*

Proof. Assume $b = c$. Then the Markoff equation reads

$$a^2 + 2b^2 = 3ab^2 \tag{1.3}$$

and we deduce that $b^2 \mid a^2$, hence $b \mid a$. So, let $a = kb$. Substituting into the Markoff equation, we have $b^2(k^2 + 2) = 3kb^3$, hence $k|2$. In particular, $k = 1$ or $k = 2$.

In the first case $k = 1$, we have that $a = b = c$, and the Markoff equation becomes $3a^2 = 3a^3$. Hence, $a = 1$, and we have the solution $(1, 1, 1)$.

Otherwise, $k = 2$, and the Markoff equation becomes $6b^2 = 6b^3$ and $b = c = 1$, giving the solution $(2, 1, 1)$.

□

For any fixed $b, c \in \mathbb{Z}$, write the Markoff equation as a quadratic form

$$x^2 - 3bcx + (b^2 + c^2) = 0. \tag{1.4}$$

From the quadratic equation, we can solve this equation to obtain the solutions

$$a = \frac{3bc \pm \sqrt{9b^2c^2 - 4(b^2+c^2)}}{2}.$$

The sum of these solutions is

$$a_+ + a_- = 3bc.$$

Hence, (a, b, c) solves the Markoff equation if and only if $(3bc - a, b, c)$ does, too. The same applies to the other two coordinates by symmetry.

The operation of swapping one root for the other in any of the three coordinates in a Markoff triple is called a *Markoff-Vieta involution*. Written explicitly, these involutions are

$$\begin{aligned} V_1(a,b,c) &= (3bc - a, b, c) \\ V_2(a,b,c) &= (a, 3ac - b, c) \\ V_3(a,b,c) &= (a, b, 3ab - c). \end{aligned}$$

Let $(V_1(a,b,c))_1$ denote the first coordinate of the Markoff triple $V_1(a,b,c)$. Notice that for Markoff triples $(a,b,c) \in \mathcal{M}$,

$$a(V_1(a,b,c))_1 = 3abc - a^2 = (a^2+b^2+c^2) - a^2.$$

If (a,b,c) is a nontrivial Markoff triple, then $a \neq 0$ by Lemma 1.8, so we may divide both sides by a and obtain an equivalent description of the Markoff-Vieta involutions:

$$\begin{aligned} V_1(a,b,c) &= \left(\frac{b^2+c^2}{a},\ b,\ c\right) \\ V_2(a,b,c) &= \left(a,\ \frac{a^2+c^2}{b},\ c\right) \\ V_3(a,b,c) &= \left(a,\ b,\ \frac{a^2+b^2}{c}\right). \end{aligned} \tag{1.5}$$

Written this way, it becomes clear that the Vieta involutions map positive solutions to positive solutions and therefore fixes all of $\mathcal{M}^\times$.

Lemma 1.10. *The group Γ of automorphisms of $\mathbb{Z}^3$ generated by the Vieta involutions, double sign changes, and permutations of coordinates fixes $\mathcal{M}^\times$.*

Since the Vieta involutions change only one coordinate at a time, we will describe the operation $V_i(a,b,c)$ as a "Vieta involution acting on coordinate i" when there can be no confusion as to which of the 3 Vieta

involutions we mean.

Returning to the quadratic equation

$$a^2 - 3abc + (b^2 + c^2) = 0,$$

assume that a is not the largest of the coordinates. Then, if $a' = 3bc - a$, we have that a' is larger than any of a, b, or c. In other words, the Vieta involution applied to a non-maximum coordinate will turn that coordinate into the maximum of the triple. The key to unlocking the structure of the Markoff graph is in the observation that the converse is true too; specifically, that the Vieta involutions always turn the maximal coordinate into a non-maximal one.

Lemma 1.11. *If $(a, b, c) \in \mathcal{M}^\times$ and $c \leq b < a$, then $1 \leq 3bc - a < b$.*

Proof. Denote $a' = 3bc - a$ and consider the polynomial $f(x) = (x - a)(x - a')$. We can show that $f(b)$ lies between a and a' via the following.

$$\begin{aligned} f(b) &= (b - a)(b - (3bc - a)) \\ &= b^2(1 - 3c) + 3abc - a^2 \\ &= b^2(1 - 3c) + (a^2 + b^2 + c^2) - a^2 \\ &= b^2(2 - 3c) + c^2. \end{aligned}$$

By Lemma 1.8, $c > 0$. By assumption, $c^2 < b^2$, so $f(b) < 0$. Since $f(x)$ is an upward opening parabola, b must lie between the roots of the parabola. Therefore, $a' < b$. □

A first application of this lemma is in the following observation.

Lemma 1.12. *Every Markoff number a appears as the largest coordinate of some triple $(a, b, c) \in \mathcal{M}$.*

Proof. If a is not the largest coordinate of (a, b, c), then apply Vieta involutions on the maximum coordinates until a is the maximum. □

It is conjectured that every Markoff number appears as the largest coordinate of a unique triple $(a, b, c) \in \mathcal{M}$. This conjecture, called the Uniqueness Conjecture, was originally made in 1913 by Frobenius and has remained open despite appearing in equivalent forms across many disciplines.

Conjecture 1.13 (Uniqueness Conjecture [18])**.** *Every Markoff number appears as the largest coordinate in a unique Markoff triple (up to permutation).*

In 1996, Baragar showed in [2] that the uniqueness conjecture holds for Markoff numbers m such that m, $3m-2$, or $3m+2$ is kp for some $k \in \{1,2,4\}$ and p a prime.

Another application of Lemma 1.11 is in showing that all solutions to the Markoff equation are given by a series of Vieta involutions applied to $(1,1,1)$, which will allow us to argue about Markoff triples by induction.

Theorem 1.14. *All positive Markoff triples are accessible from* $(1,1,1)$ *via Vieta involutions, ordered by their largest coordinate.*

Proof. Since all entries of Markoff triples are positive by Lemma 1.8, the triple $(1,1,1)$ is minimal. For any Markoff triple other than $(1,1,1)$, there is a unique largest coordinate by Lemma 1.9. Apply the appropriate Vieta involution to reduce that coordinate, and repeat. The process must terminate in light of Lemma 1.10. □

Equipped with Theorem 1.14, we can prove the following lemma which will play an important role in Chapter 4.

Lemma 1.15. *The coordinates of every Markoff triple are pairwise coprime.*

Proof. We will prove this by induction on the largest coordinate of a Markoff triple (a,b,c) which we assume without loss of generality is a. The case $(1,1,1)$ is trivial, so assume $a \neq 1$. By our inductive hypothesis, we have that $(3bc-a,b,c)$ has pairwise coprime coordinates, and we can immediately deduce that if d divides b or c then $d \nmid a$. □

As a final note in this section, the choice of the coefficient on the right hand side of (1.2) being 3 is not arbitrary. There is one other choice of coefficient, namely 1, and furthermore the solution sets to these two forms of the Markoff equation are in bijection.

Lemma 1.16. *For the equation*

$$x^2+y^2+z^2 = kxyz, \tag{1.6}$$

the solution sets when $k=1$ *and* $k=3$ *are in bijection. There are no non-trivial integer solutions when* $k \notin \{1,3\}$.

Proof. We will first show that, when $k=1$, any solution (a,b,c) has three coordinates divisible by 3. Consider the equation (1.6) taken modulo 3. It must be that $xyz \equiv 0 \bmod 3$, since otherwise $x^2+y^2+z^2 \not\equiv 0 \bmod 3$, hence exactly one or two of x^2, y^2, and z^2 are congruent to 1 modulo 3. Specifically, at least one of these, without loss of generality let it be x^2, must be divisible by 3. But then x is divisible by 3, and the right hand side cannot be nonzero.

Since $xyz \equiv 0 \bmod 3$, all of x^2, y^2, and z^2 are congruent to 0 modulo 3, since they cannot all be congruent 1 modulo 3 and multiply to 0 and no square is congruent to 2 modulo 3.

Now a bijection between the solution sets for (1.6) when $k = 3$ and $k = 1$ is given by

$$(a, b, c) \mapsto (3a, 3b, 3c).$$

Finally, fix some $k > 0$, and let (a, b, c) be a solution to (1.6). Then,

$$\begin{aligned}(ka)^2 + (kb)^2 + (kc)^2 &= k^2(a^2 + b^2 + c^2) \\ &= k^2(kabc) \\ &= (ka)(kb)(kc).\end{aligned}$$

We therefore have (ka, kb, kc) is a solution to the Markoff equation with coefficient 1, hence $\gcd(ka, kb, kc) = 3$ by Lemma 1.15 and the bijection just introduced. But this is only possible if $k = 1$ or 3. $\square$

1.3 The Markoff Graph

Throughout the rest of this chapter, a graph is a tuple $G = (V, E)$ where V is a set of *vertices* and $E = R/\sim$ where R, the adjacency relation, is a symmetric antireflexive relation on V and $(u, v) \sim (v, u)$ identifies pairs with swapped entries. In other words, we only consider simple graphs, which are undirected and have no multiedges or loops.

A *walk* in a graph G is an ordered series of vertices $(v_1, \ldots, v_n)$ such that adjacent vertices in the walk are adjacent in G. We say the walk begins at v_1 and ends at v_n. If for every pair of vertices $u, v \in V(G)$ there is a walk between u and v, then G is said to be *connected*. A *path* is a walk with no repeating vertices. If $v_1 = v_n$, but there are no repeating vertices in $(v_1, \ldots, v_{n-1})$, then the walk is a *cycle*. A graph containing no cycles between any two vertices is *acyclic*, and a connected acyclic graph is called a *tree*.

Definition 1.17. The *Markoff graph* $\mathcal{G}^\times$ is the graph with vertex set $\mathcal{M}^\times$ and an edge between $x, y \in \mathcal{M}$ if $V_i(x) = y$ for some $i \in [3]$.

From now on, we will identify walks between vertices x and y in the Markoff graph with series of integers $\{i_j \in [3]\}_{j=1}^n$ such that

$$(V_{i_n} \circ \cdots \circ V_{i_1})(x) = y.$$

In the previous section, we showed that all Markoff triples are accessible via Vieta involutions. This gives

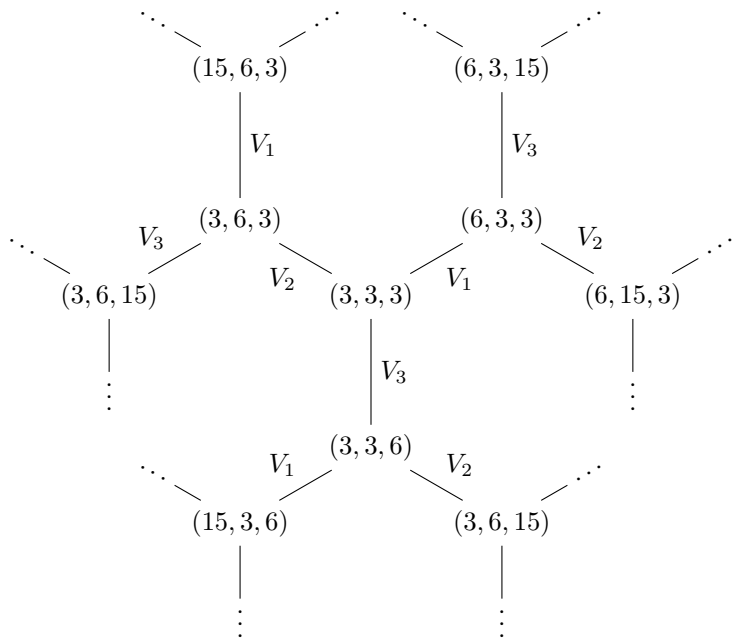


Figure 1.1: The Markoff tree $\mathcal{G}^{\times}$.

a path from any Markoff triple back to $(1, 1, 1)$ in the Markoff graph. Furthermore, these paths are unique.

Lemma 1.18. *For any Markoff triple (a, b, c), the path from (a, b, c) to $(1, 1, 1)$ given by applying Vieta involutions to the largest coordinate is the unique path from (a, b, c) to $(1, 1, 1)$.*

Proof. Any path P from (a, b, c) to $(1, 1, 1)$ must be strictly decreasing with respect to the largest coordinate of each triple. Otherwise, we would have a subwalk x, y, z with $\max(x) < \max(y)$ and $\max(z) < \max(y)$. Then, for some $i, j \in [3]$, we have $V_i(y) = x$ and $V_j(y) = z$. But there is a unique Vieta involution reducing the maximum coordinate of y. Therefore, $i = j$, so $x = z$ and our assumption that P is a path is contradicted. $\square$

Corallary 1.19. *The Markoff graph is acyclic.*

The Markoff graph is therefore an infinite 3-regular tree and we are justified in henceforth calling it the Markoff tree. We will often consider the Markoff tree to be rooted at $(1, 1, 1)$. Figure 1.1 depicts the first few vertices of the Markoff tree.

Chapter 2

The Markoff graph modulo p

In this chapter, we consider the Markoff equation taken modulo a prime $p > 2$

$$x^2 + y^2 + z^2 - xyz \equiv 0 \bmod p \tag{2.1}$$

whose solution set is denoted $\mathcal{M}(\mathbb{F}_p)$. Note that we have dropped the leading coefficient in front of the final term compared to (1.2).

In Section 1, we will derive the structure of a graph analogous to the Markoff tree of Section 1.3. In Section 2, we classify the fixed points of the Vieta involutions in the $\bmod p$ context, which will be obstructions for us throughout the rest of our work. Finally, in Section 3 we describe the method of Bourgain, Gamburd, and Sarnak [5] for understanding the structure of the Markoff graph modulo p.

2.1 The Markoff Equation modulo p

The triple $(3, 3, 3) \in (\mathbb{Z}/p\mathbb{Z})^3$ is a solution to (2.1) for all p, and the Markoff-Vieta involutions, permutations, and double sign changes still map solutions to solutions. Over $\mathbb{Z}$, no Vieta involution fixes any Markoff triple. In the mod p context, there are still no nontrivial points simultaneously fixed by all Vieta involutions, but there are points which are fixed by a single Vieta involution. We will characterize these points later, in Section 2.2.

Lemma 2.1. *For any $p > 3$, the only point simultaneously fixed by all three Markoff-Vieta involutions is $(0, 0, 0)$.*

Proof. Let (a, b, c) be fixed by all three Markoff-Vieta involutions, so that $bc = a$, $ac = b$, and $ab = c$.

Multiplying these three equations through by a, b, and c, respectively, we obtain

$$abc = a^2,$$
$$abc = b^2, \text{ and}$$
$$abc = c^2.$$

Hence, $a^2 = b^2 = c^2$. Let $k \in \mathbb{Z}/p\mathbb{Z}$ and $\sigma_a, \sigma_b, \sigma_c \in \{-1, 1\}$ such that $a = \sigma_a k$, $b = \sigma_b k$, and $c = \sigma_c k$. Substituting into the Markoff equation, we have

$$3k^2 = \sigma_a \sigma_b \sigma_c k.$$

When $k \neq 0$ we therefore have $k = \pm 3$. Up to permutation and double sign change, the only three cases are $(-3, 3, 3)$, which is not a solution to (2.1), along with $(3, 3, 3)$ and $(-3, -3, 3)$, neither of which is fixed by V_1. □

The set of numbers in $\mathbb{Z}/p\mathbb{Z}$ which may appear in a triple solution is denoted by $x(\mathcal{M}(\mathbb{F}_p))$ and is characterized by the following lemma.

Lemma 2.2. *Let $x(\mathcal{M}(\mathbb{F}_p)^\times)$ denote the set of mod p-integers which appear in the non-trivial solutions to* (2.1)*. Then,*

$$x(\mathcal{M}(\mathbb{F}_p)^\times) = \begin{cases} \mathbb{F}_p & p \equiv 1 \bmod 4, \\ \mathbb{F}_p \setminus \{-2, 2\} & p \equiv 3 \bmod 4. \end{cases}$$

Proof. We will first show that $2 \in x(\mathcal{M}(\mathbb{F}_p)^\times)$ if and only if $p \equiv 1 \bmod 4$. Since the Markoff triples are closed under double sign change, this will imply that $-2 \in x(\mathcal{M}(\mathbb{F}_p)^\times)$ if and only if $p \equiv 1 \bmod 4$ also. in the second step, we will show that $\mathbb{F}_p \setminus \{-2, 2\} \subseteq x(\mathcal{M}(\mathbb{F}_p)^\times)$ for all p.

Assume that $2 \in x(\mathcal{M}(\mathbb{F}_p)^\times)$. Without loss of generality in the permutation of the coordinates, let $(2, b, c)$ be a Markoff triple mod p. We then have

$$4 + b^2 + c^2 = 2bc,$$

a quadratic equation in b with discriminant

$$4c^2 - 4(4 + c^2) = (2b - 2c)^2.$$

Re-arranging, we have that $-16 = (2b - 2c)^2$. In particular, -1 is a quadratic residue, hence $p \equiv 1 \bmod 4$.

Conversely, if $p \equiv 1 \bmod 4$, then -1 is a quadratic residue mod p. Let $r^2 = -4$; then, $(2, r, 0)$ is a Markoff triple mod p.

We now show that every $a \neq \pm 2$ is also a Markoff number mod p. Consider the intersection

$$\{z^2 \mid z \in \mathbb{F}_p^\times\} \cap \{b^2(a^2 - 4) - 4a^2 \mid b \in \mathbb{F}_p^\times\}.$$

Both of these sets are defined by a map which is 2-to-1 on $\mathbb{F}_p^\times$ plus one additional element corresponding to $z = 0$ or $b = 0$, respectively. Therefore they both have size $(p-1)/2 + 1$. The sum of their sizes is $p + 1 > p = \mathbb{F}_p$, so their intersection is non-empty. We thus have some $z, b \in \mathbb{F}_p$ satisfying $z^2 = b^2(a^2 - 4) - 4a^2$. Then, $(a, b, (1/2)(ab + z))$ is a Markoff triple mod p since

$$\begin{aligned} a^2 + b^2 + \frac{(ab+z)^2}{4} &= a^2 + b^2 + \frac{a^2b^2 + 2abz + z^2}{4} \\ &= a^2 + b^2 + \frac{2a^2b^2 + 2abz - 4a^2 - 4b^2}{4} \\ &= \frac{a^2b^2 + abz}{2} \\ &= ab\left(\frac{ab+z}{2}\right). \end{aligned}$$

□

De Courcy-Ireland and Lee gave counts for the number of solutions to (2.1) in [13].

Lemma 2.3. *[13, Proposition 2.1] The number of solutions to the Markoff equation modulo p is given by*

$$|\mathcal{M}_p(\mathbb{F}_p)| = \begin{cases} p^2 + 3p + 1 & p \equiv 1 \bmod 4 \\ p^2 - 3p + 1 & p \equiv 3 \bmod 4. \end{cases}$$

Let the Markoff graph modulo p, denoted $\mathcal{G}_p$, be a graph with vertex set $\mathcal{M}_p$, and two vertices $u, v \in \mathcal{M}_p$ will have an edge between them if $V_i(u) = v$ for some $1 \leq i \leq 3$ and $u \neq v$. Unlike the $\mathbb{Z}$ case, we include the condition $u \neq v$ to prevent self loops. Then the Markoff graph modulo p, like the Markoff graph over $\mathbb{Z}$, is a simple graph with no self loops or multiple edges. The Markoff graph mod 5 is shown in Figure 2.1.

Solutions to the Markoff equation over $\mathbb{Z}$ clearly descend to solutions modulo p as

$$\phi : (a, b, c) \mapsto (a \bmod p, b \bmod p, c \bmod p). \tag{2.2}$$

Since this map also respects the Vieta involutions, it induces a graph homomorphism $\mathcal{G} \to \mathcal{G}_p$. In 1991, Baragar conjectured that this map is surjective. This conjecture is equivalent to the statement that $\mathcal{G}_p^\times$ is

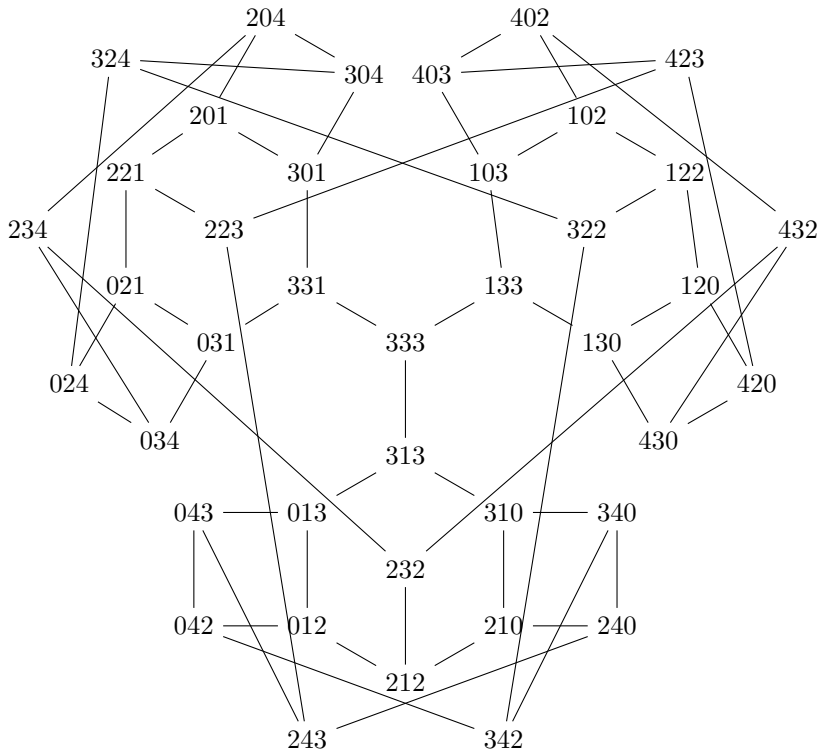


Figure 2.1: The Markoff graph $\mathcal{G}_5^\times$.

connected, since (2.2) being surjective would imply that the induced map on walks is also surjective. This is known as the *Strong Approximation Conjecture.*

Conjecture 2.4 ([3])**.** *For every prime p, the Markoff graph $\mathcal{G}_p^\times$ is connected.*

If the strong approximation conjecture is true, then the walks between nodes in $\mathcal{M}_p^\times$ would algorithmically give a method of lifting solutions to the Markoff equation mod p back to the integers.

Conjecture 2.5. *For every prime p, the canonical projection $\mathcal{M} \to \mathcal{M}(\mathbb{F}_p)$ is onto; equivalently, $\phi : \mathcal{G}^\times \to \mathcal{G}_p^\times$ is onto.*

Lemma 2.6. *Conjectures 2.4 and 2.5 are equivalent.*

Proof. If $\mathcal{G}_p^\times$ is connected, then for any $v \in \mathcal{M}(\mathbb{F}_p)$, there is a sequence $V_{i_1}, \ldots, V_{i_n}$ of Vieta involutions forming a path from $(1,1,1)$ to v. Taking the vertices in this path modulo p,

$$(V_{i_1} \circ \cdots \circ V_{i_n})(1,1,1) \equiv v \bmod p.$$

Conversely, assume $\mathcal{M} \to \mathcal{M}(\mathbb{F}_p)$ is onto. For $v \in \mathcal{M}(\mathbb{F}_p)$, let $w \in \mathcal{M}$ such that $w \equiv v \bmod p$. Since $\mathcal{G}^\times$ is

connected, there is a path S from $(1,1,1)$ to w. Now, S also forms a walk in $\mathcal{G}_p^\times$, connected $(1,1,1)$ to v. And since every vertex is connected to $(1,1,1)$, they are all connected. $\square$

Bourgain, Gamburd, and Sarnak use the property that connected Markoff graphs induce lifts $\mathcal{M}_p \to \mathcal{M}$ to show that almost all Markoff numbers are highly composite [5, Theorem 18]. The strong approximation conjecture is also the first step towards proving the much larger conjecture that $\mathcal{G}_p^\times$ forms an expander family, first proposed in [5]. Fuchs et. al. explored applications of the Markoff graph being connected to cryptography [19]. Silverman gave in [38] an algorithm for constructing paths in the Markoff graph between two given vertices in subexponential time with high probability.

Progress towards the strong approximation conjecture has relied heavily on the method of Bourgain, Gamburd, and Sarnak. In [5], they developed a three-tiered approach showing tha strong approximation holds for all primes p outside a zero density subset; this method will be covered in detail in see Section 4. This result was strengthened by Chen in [10] to strong approximation holding for all but a finite set of primes. A second, significantly shorter proof of this theorem was provided by Martin in [31]. Eddy et. al. proved in [15] an upper bound on p of approximately $3.448 \cdot 10^{392}$ for which $\mathcal{G}_p^\times$ may be disconnected. Their method, relying on what they call *maximal divisors* of $p \pm 1$, also proves that $\mathcal{G}_p^\times$ is connected for many specific primes, the smallest being $p = 1,327,363$, but is inconclusive for most primes less than 10^9. Maximal divisors will play an important role in the work we present in Chapter 4.

Even though the strong approximation conjecture does not hold for other Markoff-like equations of the form

$$x^2 + y^2 + z^2 - 3xyz = \kappa \tag{2.3}$$

for non-zero k, computational evidence suggests that these graphs, with the exception of the degenerative case $k = 2$ known as the "Cayley cubic", have connected components outside of a small number of exceptional orbits ([20] and [31]). Daniel Martin in [32] proved that all solutions to (2.3) lie in a single orbit outside of a small number of small exceptional orbits for primes not divisible by $24,504,480$.

2.2 Fixed Points of the Vieta Involutions

Many properties of the Markoff graph are consequences of the vertex degree being bound by 3. Even though only the trivial solution $(0,0,0)$ is fixed by all three Vieta involutions, there are a small number of solutions which are fixed by a single Vieta involution. For example, when $p = 5$ the triple $(5,5,6)$ is fixed by V_3. In this section, we characterize these fixed points and prove some basic properties about them.

Lemma 2.7. *No triple in $\mathcal{M}_p^\times$ fixed by any Vieta involution has any coordinate equal to* 0.

Proof. Let $(a, b, c) \in \mathcal{M}_p$ and assume without loss of generality that $V_1(a, b, c)(a, b, c)$. Then, $bc - a = a$, equivalently $bc = 2a$. If $b = 0$ or $c = 0$, then $a = 0$ too. Alternatively, if $a = 0$, then at least one of b and c must be 0. In any of these cases, at least two coordinates are 0; then it follows from (2.1) that the third must be zero, too. □

This result now allows us to characterize where the fixed points of the Vieta involutions are in terms of the alternative definition of the Vieta involution

$$V_1((a, b, c)) = \left(\frac{b^2 + c^2}{a}, b, c\right)$$

which still holds when $a \neq 0$.

Lemma 2.8. *A nontrivial Markoff mod p triple (a, b, c) is fixed by V_1 (resp. V_2, V_3) if and only if $a \neq 0$ and $a^2 = b^2 + c^2$ (resp. $b \neq 0$ and $b^2 = a^2 + c^2$, $c \neq 0$ and $c^2 = a^2 + b^2$).*

Proof. Assume $V_1((a, b, c)) = (a, b, c)$. Since $a \neq 0$ by Lemma 2.7, we may write the Vieta involution as $a = (b^2 + c^2)/a$ using the alternative definition of the Markoff-Vieta involution (1.5). Multiplying by a yields $a^2 = b^2 + c^2$. Beginning with $a^2 = b^2 + c^2$ and dividing by a yields the converse. □

Furthermore, a nontrivial solution can only be fixed by Vieta involution at a time.

Lemma 2.9. *The only Markoff triple mod p fixed by more than one Vieta involution is* $(0, 0, 0)$.

Proof. Assume without loss of generality that $V_1(a, b, c) = V_2(a, b, c) = (a, b, c)$. Then $bc = 2a$ and also $ac = 2b$. Substituting the first of these into the second yields

$$\frac{1}{2}bc^2 = 2b.$$

By Lemma 2.7 we can assume $b \neq 0$. Then, we have that $c = \pm 2$. In the case that $c = 2$, we have $a = b$ and the Markoff equation becomes

$$2a^2 + 4 = 2a^2,$$

which is absurd. Similarly, when $c = -2$ we have $a = -b$ and the Markoff equation becomes

$$2a^2 + 4 = -2a^2.$$

Hence $a^2 = b^2 = -1$ and $ab = 1$, so plugging back into the Markoff equation yields

$$-2 + c^2 = abc = c,$$

but $-2 + c^2 = c$ contradicts our previous statement that $c = -2$. $\square$

In terms of the Markoff graph $\mathcal{G}_p^\times$, this says that every vertex has degree either 2 or 3. Next, we would like to count the number of points fixed by any Vieta involution, i.e., the number of vertices of $\mathcal{G}_p^\times$ with degree 2.

Lemma 2.10. *If a triple $(a, b, c) \in (\mathbb{Z}/p\mathbb{Z})^3$ satisfies both $2a = bc$ and $a^2 = b^2 + c^2$, then $(a, b, c) \in \mathcal{M}_p$ and contains a self-loop at V_1.*

Proof. We may assume $a = 0$ since otherwise $b = c = 0$ as well. Lemma 2.8 guarantees that our triple is fixed by V_1, so we need only prove that $(a, b, c) \in \mathcal{M}_p$. By multiplying $a^2 = b^2 + c^2$ through by 2 and re-writing using the first equation, we obtain

$$\begin{aligned} 2a^2 &= 2b^2 + 2c^2 \\ abc &= 2b^2 + 2c^2 \\ abc &= a^2 + b^2 + c^2. \end{aligned}$$

$\square$

With these results in hand, we can now proceed to counting the number of self-loops in $\mathcal{M}_p$. From now on we will notate the Legendre symbol as

$$\left(\frac{a}{p}\right)_L = \begin{cases} 0 & a = 0, \\ 1 & a \text{ is a nonzero quadratic residue mod } p, \\ -1 & a\text{otherwise.} \end{cases}$$

Lemma 2.11. *The number of self-loops in $\mathcal{M}_p$ is*

$$\begin{cases} 3(p-5) & p \equiv 1 \bmod 4 \\ 3(p-3) & p \equiv 3 \bmod 4. \end{cases}$$

Proof. We will first show that each $b \in (\mathbb{Z}/p\mathbb{Z})^\times$ satisfying

$$\left(\frac{b^2 - 4}{p}\right)_L = 1$$

yields two distinct fixed points of V_1 distinct from all other choices of b. In the second step, we will count these fixed points.

Fix a nonzero b as above. Both $b^2 - 4$ and $4b^2$ are nonzero quadratic residues, so there are two distinct values of c such that

$$c^2 = \frac{4b^2}{b^2 - 4}. \tag{2.4}$$

For either of these values of c, let $a = (1/2)bc$. After rearranging, we have

$$b = \frac{2a}{c}. \tag{2.5}$$

From (2.4) we then have $b^2c^2 - 4c^2 = 4b^2$, and after substituting in (2.5) the equation becomes $4a^2 = 4b^2 + 4c^2$, hence $a^2 = b^2 + c^2$. Lemma 2.10 then guarantees this is a Markoff triple fixed by V_1. Conversely, Lemma 2.8 guarantees that every Markoff triple fixed by V_1 is of the above form.

Each b value satisfying the above restriction yields two fixed points of V_1 corresponding to the two choices of c. In the first line of the following, the leading -2 represents the subtract from the count the cases $b = \pm 2$, which correspond to $b^2 - 4 = 0$.

$$\begin{aligned}
-2 + \sum_{b=1}^{p-1} \left(1 + \left(\frac{b^2-4}{p}\right)_L\right) &= p - 2 + \sum_{b=0}^{p-1} \left(\frac{b^2-4}{p}\right)_L - \left(1 + \left(\frac{-4}{p}\right)_L\right) \\
&= p - 2 - \left(1 + \left(\frac{-1}{p}\right)_L\right) - \left(\frac{1}{p}\right)_L \\
&= p - 4 - \left(\frac{-1}{p}\right)_L,
\end{aligned}$$

where we have used the identity (when $p \nmid b^2 - 4ac$)

$$\sum_{x=0}^{p-1} \left(\frac{ax^2 + bx + c}{p}\right)_L = -\left(\frac{a}{p}\right)$$

in the second line. We therefore have

$$p - 4 - \left(\frac{-1}{p}\right)_L$$

fixed points of V_1; multiplying by three for the symmetry in the coordinates (where there is no double counting in light of Lemma 2.9) yields our claim. □

In the next section, we will present an alternate form of the Markoff graph which is loopless.

2.3 The method of Bourgain, Gamburd, and Sarnak

An alternative version of the Markoff graph which has proven extremely useful in analyzing the Markoff triples modulo p was given by Bourgain, Gamburd, and Sarnak in [5]. Instead of using the Vieta involution to determine the edge set of the graph, they use (directed) edges given by the so-called rotation maps. The rotation maps have the nice property that it is quite easy to calculate the size of an orbit of a Markoff triple under that rotation map. Furthermore, the Markoff triples may be partitioned according to the size of its largest orbit, and connectivity between the partitions is then considered. In particular, all triples of large enough orbit lie in a "large (connected) component". By counting the number of triples witnessed inside the large component and bounding the number of triples which may lie outside the large component, Bourgain, Gamburd, and Sarnak were able to prove that $\mathcal{G}_p^\times$ is connected for all but a zero-density subset of primes.

The BGS argument begins with the *rotation maps* which are compositions of a Markoff-Vieta involution with a transposition. If $1 \leq i \leq 3$, then

$$\mathrm{rot}_i = \tau_{i+1,\, i+2} \circ V_{i+1}$$

is the rotation map fixing coordinate i. Written explicitly,

$$\begin{aligned}
\mathrm{rot}_1 &: (a, b, c) \mapsto (a, c, ac - b), \\
\mathrm{rot}_2 &: (a, b, c) \mapsto (ab - c, b, a), \\
\mathrm{rot}_3 &: (a, b, c) \mapsto (b, bc - a, c).
\end{aligned}$$

Definition 2.12. The *BGS graph* $\mathcal{B}_p^\times$ is the directed graph with vertex set $\mathcal{M}_p^\times$ and an edge from x to y if $\mathrm{rot}_i(x) = y$ for some $1 \leq i \leq 3$.

Unlike $\mathcal{G}_p^\times$, this graph does not contain any self loops.

Lemma 2.13. *Let $x \in \mathcal{M}_p^\times$ and $1 \leq i \leq 3$. Then $\mathrm{rot}_i(x) \neq x$.*

Proof. Without loss of generality assume

$$\mathrm{rot}_1((a, b, c)) = (a, c, ac - b) = (a, b, c).$$

Then, $c = b$ and $ac - b = c$. Hence, $ac = 2c$. If $c = 0$, then $a = b = 0$ as well, so assume $c \neq 0$. Then we have $a = 2$, and plugging back into the Markoff equation we have $4 + 2b^2 = 2b^2$ which is absurd. $\square$

Fix $(a, b, c) \in \mathcal{M}(\mathbb{F}_p)$. Denote the rotation map rot_1 restricted to acting on triples with fixed first

coordinate a by

$$\begin{aligned}\mathrm{rot}'_a \begin{pmatrix} b \\ c \end{pmatrix} &= \begin{pmatrix} c \\ ac - b \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ -1 & a \end{pmatrix} \begin{pmatrix} b \\ c \end{pmatrix}.\end{aligned}$$

Let $a \neq \pm 2$. If $a = \chi + \chi^{-1}$ for $\chi \in \mathbb{F}_{p^2}$, then the matrix diagonalizes as

$$\begin{pmatrix} 0 & 1 \\ -1 & \chi + \chi^{-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \chi & \chi^{-1} \end{pmatrix} \begin{pmatrix} \chi & 0 \\ 0 & \chi^{-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ \chi & \chi^{-1} \end{pmatrix}^{-1}. \tag{2.6}$$

The size of the orbit of rot_1 acting on (a, b, c) is dependent on a only, and will be the same for any orbit with fixed first coordinate a. We therefore refer to the *order of the Markoff number a modulo p* as the size of any orbit with fixed first coordinate a. According to (2.6), this is equivalent to

$$\mathrm{ord}_p(a) = |\chi|$$

when $a \neq \pm 2$, where $|\cdot|$ is the multiplicative order in $\mathbb{F}_{p^2}^\times$.

If $a = \pm 2$, then $\chi = \pm 1$, and the matrices of (2.6) are non-invertible. In this case, we have the following lemma establishing the order of a.

Lemma 2.14 ([5])**.** *If $a = \pm 2$, then rot_1 acting on a Markoff triple (a, b, c) modulo p gives an orbit of size p or $2p$.*

The rotation orbits give us a convenient method of traversing the Markoff graph and by determining connectivity. Before we proceed, we establish the equivalence between connectivity of $\mathcal{M}_p^\times$ and $\mathcal{B}_p^\times$ via the following lemmas.

Lemma 2.15. *Every connected component of $\mathcal{B}_p^\times$ is strongly connected.*

Proof. Let $\mathrm{rot}_{i_1}, \ldots, \mathrm{rot}_{i_n}$ be a path from x to y in $\mathcal{B}_p^\times$. Then, $\mathrm{rot}_{i_n}^{|\mathrm{rot}_{i_n}|-1}, \ldots, \mathrm{rot}_{i_1}^{|\mathrm{rot}_{i_1}|-1}$ is a path from y to x, where $|\mathrm{rot}_{i_j}|$ is the order of the i_jth rotation map. □

Lemma 2.16. *The graph $\mathcal{M}_p^\times$ is connected if and only if $\mathcal{B}_p^\times$ is connected.*

Proof. Let the group S_3 act on $\mathcal{M}(\mathbb{F}_p)$ by permuting the coordinates of each Markoff triple. Since

$$V_i \circ \sigma = \sigma \circ V_{\sigma^{-1}(i)}$$

for all $\sigma \in S_3$, each path in $\mathcal{B}_p^\times$ may be re-written as a series of Markoff-Vieta involutions followed by a permutation of the coordinates, or vice-versa.

Assume $\mathcal{B}_p^\times$ is connected. Let $P_1 = \mathrm{rot}_{i_m} \circ \cdots \circ \mathrm{rot}_{i_1} \circ$ be a path from x to $(3,3,3)$ and $P_2 = \mathrm{rot}_{i_n} \circ \cdots \circ \mathrm{rot}_{i_{m+1}}$ a path from $(3,3,3)$ to y in $\mathcal{B}_p^\times$. Re-write $P_1 = \sigma_1 \circ V_{i'_m} \circ \cdots V_{i'_1} \circ$ and $P_2 = V_{i'_n} \circ \cdots \circ V_{i'_{m+1}} \circ \sigma_2$. Then,

$$\begin{aligned}
(V_{i'_n} \circ \cdots \circ V_{i'_1})x &= (P_2 \circ \sigma_2^{-1} \circ \sigma_1^{-1} \circ P_1)x \\
&= (P_2 \circ \sigma_2^{-1} \circ \sigma_1^{-1})(3,3,3) \\
&= P_2(3,3,3) \\
&= y,
\end{aligned}$$

and therefore $V_{i'_1} \circ \cdots V_{i'_n}$ is a path from x to y in $\mathcal{G}_p^\times$.

In the converse direction, assume $\mathcal{G}_p^\times$ is connected. Let $P = V_{i_m} \circ \cdots \circ V_{i_1}$ be a path from x to $(3,3,3)$ in $\mathcal{G}_p^\times$. Consider the paths in $\mathcal{B}_p^\times$

$$P'_k = \bigcirc_{j=1}^{k} \mathrm{rot}_{s_j(i_j)-1}$$

where $s_j \in S_3$ and is defined recursively as

$$\begin{aligned}
s_1 &= \mathbb{1}, \\
s_{j+1} &= \tau_{s_j(i_j), s_j(i_j)+1} \circ s_j.
\end{aligned}$$

We can now verify by induction that for all $1 \leq k \leq m$, the partial path P'_k connecting x to $(s_{k+1} \circ V_{i_k} \circ \cdots \circ V_{i_1})(x)$:

$$\begin{aligned}
P'_1 &= \mathrm{rot}_{s_1(i_1)-1} \\
&= \tau_{i_1, i_1+1} \circ V_{i_1}, \\
&= s_2 \circ V_{i_1}, \\
P'_\ell &= \mathrm{rot}_{s_\ell(i_\ell)-1} \circ P'_{1,\ell} \\
&= (\tau_{s_\ell(i_\ell), s_\ell(i_\ell)}) \circ V_{s_\ell(i_\ell)}) \circ (s_\ell \circ V_{i_\ell} \circ \cdots \circ V_1) \\
&= (\tau_{s_\ell(i_\ell), s_\ell(i_\ell)} \circ s_\ell) \circ (V_{s_\ell^{-1}(s_\ell(i_\ell))} \circ V_{i_\ell} \circ \cdots \circ V_1) \\
&= s_{\ell+1} \circ V_{i_l} \circ \cdots \circ V_1.
\end{aligned}$$

Therefore, P'_m is a path from x to $s_{m+1}(3,3,3) = (3,3,3)$ in $\mathcal{B}_p^\times$. Via an identical construction, there exists a path from y to $(3,3,3)$ in $\mathcal{B}_p^\times$, and thus according to Lemma 2.15, a path from $(3,3,3)$ to y. Then, the concatenation of these paths yields a path from x to y. □

Each triple in $\mathcal{B}_p^\times$ therefore lies inside of three orbits of finite order given by the three rotation maps. We classify each triple by the order of its largest orbit.

Definition 2.17. The *order of the triple* $(a,b,c) \in \mathcal{M}(\mathbb{F}_p)$ is

$$\mathrm{ord}_p((a,b,c)) = \max(\mathrm{ord}_p(a), \mathrm{ord}_p(b), \mathrm{ord}_p(c)).$$

If the order of a triple is $p \pm 1$, then it is called *a triple of maximal order*.

In the spirit of [5], we say that an element $a \in \mathbb{Z}/p\mathbb{Z}$ is *parabolic* if $a = \pm 2$, *hyperbolic* if $\mathrm{ord}_p(a)$ divides $p-1$, or *elliptic* if $\mathrm{ord}_p(a)$ divides $p+1$.

Remark 2.18. The orbits of triples with fixed parabolic coordinates have sizes $\mathrm{ord}_p(a) = p$ or $2p$. If a is parabolic and $a = \chi + \chi^{-1}$, then $\chi = \pm 1$. The matrix in (2.6) is not invertible, and $|\chi|$ is 1 or 2.

Bourgain, Gamburd, and Sarnak observed in [5] that all triples of maximum order (i.e., those of order $p \pm 1$) and parabolic triples belong to the same connected component $\mathcal{C}_p$, which they call the "large component". (An explicit proof of this is given in [4, Lemma 3.3].) Then, they iteratively lower the bound on the orders of triples which must be connected to $\mathcal{C}_p$. in the first step, the *endgame*, all triples of order at least $p^{\delta+1/2}$ are shown to be connected to $\mathcal{C}_p$, for some δ dependent on p. In the *middle game*, they show that there is an $\epsilon > 0$ dependent only on p for which $\mathrm{ord}_p(a) > p^\epsilon$ implies that every orbit of rot'_a contains a triple of order larger than $\mathrm{ord}_p(a)$. By successively walking from triple to triple of higher order, eventually a triple of order

at least $p^{\delta+1/2}$ is reached. Finally, in the *opening*, they bound the size of $|\mathcal{G}_p^\times \setminus \mathcal{C}_p|$ from below, and show that for primes p with $p \pm 1$ not having too many factors, there cannot be enough unaccounted for triples to constitute a connected component disconnected from $\mathcal{C}_p$. Specifically, Bourgain, Gamburd, and Sarnak proved in [5] a lower bound of $(\log p)^{1/3}$ on the size of any connected component. This bound was improved by Konyagin, Makarychev, Shparlinski, and Vyugin [25] to $(\log p)^{7/9}$, and again improved by Chen [10] to p.

Chapter 3

An algorithmic approach to BGS

In this chapter, we outline an algorithmic realization of the process introduced by Bourgain, Gamburd, and Sarnak and discussed in Chapter 2, along with an implementation of that algorithm in Rust. In particular, we constructively prove the following theorem.

Theorem 3.1. *There exists an algorithm with runtime $o(p^{1+\epsilon})$ for every $\epsilon > 0$ returning an affirmative or inconclusive response to the query, is the graph $\mathcal{G}_p^\times$ connected?*

Conjecture 3.2. *The algorithm exhibited here returns an affirmative response for all primes.*

Our algorithm is significantly better than a naive, flood fill-like approach with an $O(p^2)$ runtime, there being $p^2 + O(p)$ solutions to the Markoff equation modulo p [13, Proposition 2.1]. We demonstrate our implementation by testing the connectivity of $\mathcal{G}_p^\times$ for all primes less than one million, and a random sample of primes less than one hundred million.

Theorem 3.3. *The graph $\mathcal{G}_p^\times$ is connected for all primes p less than one million.*

In the general case of the Markoff equation (2.1) having an integer on the right hand side, the solutions are still fixed by the Markoff Vieta involutions and rotation maps, but the corresponding graphs are no longer connected. A partial classification of these graphs over the integers was carried out in [20], and Bourgain, Gamburd, and Sarnak announced in [5] a forthcoming article extending their process to these more general graphs. Connectivity was also shown for a large class of primes for these general Markoff graphs in [32]. Structures of the connected components for non-zero level sets over finite fields were studied in [13].

In Section 1, we outline the arguments of [5] and [15], elaborating on the features we will utilize in our algorithm. In Section 2, we construct the data structures and procedures at the core of our algorithm. Finally, in Section 3, we present the algorithm in total, along with our computational results for all primes

below one million, as well as auxiliary data useful for short circuiting the computations and improving real world performance.

The contents of this chapter were originally published as [6].

3.1 The structure of the Markoff graph modulo p

In this section, we review some of the structural properties of $\mathcal{G}_p^\times$. We begin our discussion with the orbits of the rotation maps, which will form useful walks along the graph.

Let $\mathcal{D}(n)$ be the set of positive divisors of n, and $\mathcal{D}(p \pm 1) = \mathcal{D}(p-1) \cup \mathcal{D}(p+1)$.

We will consider the group Γ generated by the *rotation maps* introduced in Chapter 2. Recall that the order of a Markoff number a, denoted $|a|$, is the size of any orbit under rot_1 with fixed first coordinate a. The following lemma guarantees that the order of χ always divides $p \pm 1$, a fact we will use in our classification of Markoff triples. Equivalently, either $\chi \in \mathbb{F}_p^\times$, or $\chi \in \mathbb{F}_{p^2}^\times$ with norm 1 to $\mathbb{F}_p$.

Lemma 3.4. *Let $\chi \in \mathbb{F}_{p^2}^\times$. Then $(\chi + \chi^{-1}) \in \mathbb{F}_p$ if and only if $|\chi| \in \mathcal{D}(p \pm 1)$.*

Proof. If $\chi \in \mathbb{F}_{p^2}^\times$ with $|\chi| \in \mathcal{D}(p \pm 1)$, then

$$\begin{aligned}\left(\chi + \frac{1}{\chi}\right)^p &= \chi^p + \frac{1}{\chi^p} \\ &= \chi^{\mp 1} + \frac{1}{\chi^{\mp 1}} \\ &= \chi + \frac{1}{\chi},\end{aligned}$$

and therefore $|\chi + \chi^{-1}| \in \mathcal{D}(p-1)$, equivalently $(\chi + \chi^{-1}) \in \mathbb{F}_p$.

Since $\mathbb{F}_{p^2}^\times$ is cyclic, there are $2p$ values $\chi \in \mathbb{F}_{p^2}^\times$ for which $|\chi|$ is divisible by $p-1$ or $p+1$. The map $\phi : \chi + \chi^{-1}$ is 2-to-1 on $\mathbb{F}_{p^2}^\times$, so the image of ϕ on the set $\{\chi : |\chi| \in \mathcal{D}(p \pm 1)\}$ is p values in $\mathbb{F}_p$. But, that is all of $\mathbb{F}_p$, so there must be no more values of $\chi \in \mathbb{F}_{p^2}^\times$ for which $\chi + \chi^{-1} \in \mathbb{F}_p$. □

The diagonalization (2.6) allows us to parameterize the Markoff triples in an orbit in terms of χ. Specifically, if $(a, b, c) \in \mathcal{M}(\mathbb{F}_p)$ and $a = \chi + \chi^{-1}$, then

$$\langle \mathrm{rot}'_a \rangle (b, c) = \left\{ k \left(\alpha \chi^\ell + \beta \frac{1}{\chi^\ell}, \alpha \chi^{\ell+1} + \beta \frac{1}{\chi^{\ell+1}} \right) : \ell \in \mathbb{Z} \right\},$$

where

$$k = \left(\chi - \frac{1}{\chi}\right)^{-1}, \ \alpha = \left(c - \frac{b}{\chi}\right), \text{ and } \beta = (\chi b - c).$$

A parameterization for arbitrary orbits is given in [15, Equation 3] as

$$\left\{\left(\chi+\frac{1}{\chi},\frac{\chi+\chi^{-1}}{\chi-\chi^{-1}}\left(r\chi^{\ell}+\frac{1}{r\chi^{\ell}}\right),\frac{\chi+\chi^{-1}}{\chi-\chi^{-1}}\left(r\chi^{\ell+1}+\frac{1}{r\chi^{\ell+1}}\right)\right):\ell\in\mathbb{Z}\right\}, \tag{3.1}$$

where any values $r,\chi\in\mathbb{F}_{p^2}^{\times}\backslash\{\pm1\}$ gives a set of triples solving the Markoff equation over $\mathbb{F}_{p^2}$ fixed by rot_1. For fixed $\chi\in\mathbb{F}_{p^2}^{\times}$, the orbits given by $r_1,r_2\in\mathbb{F}_{p^2}^{\times}$ according to (3.1) will be the same exactly when $r_1\langle\chi\rangle=r_2\langle\chi\rangle$. We therefore seek one representative r for each coset of $\langle\chi\rangle$ giving

$$\frac{r+r^{-1}}{\chi-\chi^{-1}}\in\mathbb{F}_p,$$

so that our expression (3.1) gives solutions to the Markoff equation in $\mathbb{F}_p$. The solutions are characterized by Lemma 3.5.

Lemma 3.5. *Let $\chi,r\in\mathbb{F}_{p^2}^{\times}$ and $\chi\neq\pm1$. Let $k\in\mathbb{F}_p$ be a quadratic nonresidue.*

(a) If $\chi\in\mathbb{F}_p^{\times}$, then $\chi-\chi^{-1}\in\mathbb{F}_p$.

(b) If $|\chi|\in\mathcal{D}(p+1)$, then $\chi-\chi^{-1}\in\sqrt{k}\mathbb{F}_p$.

(c) If $|r|\in\mathcal{D}(2(p+1))\setminus\mathcal{D}(p+1)$, then $r+r^{-1}\in\sqrt{k}\mathbb{F}_p$.

Proof. The proof of (a) is identical to the proof of Lemma 3.4 where $|\chi|\in\mathcal{D}(p-1)$. So, assume $|\chi|\in\mathcal{D}(p+1)$. Then,

$$\left(\chi-\frac{1}{\chi}\right)^p=\frac{1}{\chi}-\chi,$$

therefore $|\chi-\chi^{-1}|\notin\mathcal{D}(p-1)$. However,

$$\begin{aligned}\left(\chi-\frac{1}{\chi}\right)^{2p}&=\left(\chi^2+\frac{1}{\chi^2}-2\right)^p\\&=\left(\frac{1}{\chi^2}+\chi^2-2\right),\end{aligned}$$

so $(\chi-\chi^{-1})^2$ has multiplicative order dividing $p-1$. Finally, let $|r|\in\mathcal{D}(2(p+1))$. Then,

$$\begin{aligned}\left(r+\frac{1}{r}\right)^{2p}&=\left(r^2+\frac{1}{r^2}+2\right)^p\\&=\left(\frac{1}{r^2}+r^2+2\right),\end{aligned}$$

therefore $(r+r^{-1})^2$ has multiplicative order dividing $p-1$. □

Corallary 3.6. *Let $\chi, r \in \mathbb{F}_{p^2}^\times$ with $|\chi| \in \mathcal{D}(p+1)\backslash\{1,2\}$ and $|r| \in \mathcal{D}(2(p+1))\backslash\mathcal{D}(p+1)$. Then, $\chi - \chi^{-1} = b_1\sqrt{k}$ and $r + r^{-1} = b_2\sqrt{k}$ for $b_1, b_2, k \in \mathbb{F}_p$ and k a quadratic nonresidue. With these choices of χ and r, the orbit* (3.1) *contains triples with all coordinates in $\mathbb{F}_p$.*

Our algorithm will make heavy use of the classification of triples as lying in the endgame, middle game, or opening in the terminology of BGS. The bounds on the endgame were made explicit in [15], where the following inequality was proved.

Lemma 3.7 ([15, Proposition 6.1])**.** *A Markoff triple of order $d \in \mathcal{D}(p \pm 1)$ is connected to $\mathcal{C}_p$ provided*

$$d > \frac{8\sqrt{p}(p \pm 1)\tau(p \pm 1)}{\phi(p \pm 1)}. \tag{3.2}$$

We will call the right hand side of (3.2) for each choice of sign the *endgame breakpoints* and denote them $B_{(p,\pm)}$.

We now wish to bound from below the orders of triples which are part of the middle game. We will consider an orbit of size t, and give an upper bound on the number of triples in that orbit with order less than t. If the number of such triples is less than t, then the orbit being considered must contain a triple of order larger than t. A suitable bound is given in [15] and relies on an approximation of Corvaja and Zannier [11].

Lemma 3.8 ([15, Lemma 2.1])**.** *If $\chi \in \mathbb{F}_{p^2}^\times$ has order $t > 2$, then the number of values n between 0 and $t-1$ for which*

$$\operatorname{ord}_p\left(\frac{\chi + \chi^{-1}}{\chi - \chi^{-1}}\right)\left(sr^n + (sr^n)^{-1}\right)$$

divides d is at most

$$\frac{3}{2}\max\left(\sqrt[3]{6td}, \frac{4td}{p}\right). \tag{3.3}$$

We now introduce the notion of maximal divisors, coined in [15].

Definition 3.9 ([15, Definition 1.2])**.** Let n be a positive integer, and let $x \in \mathbb{R}$. A positive divisor d of n is *maximal with to respect to x* if $d \leq x$ and there is no other positive divisor d' of n such that $d' \leq x$ and $d \mid d'$. The set of maximal divisors of n with respect to x is denoted $\mathfrak{M}_x(n)$.

Fix a divisor t of $p \pm 1$. If we sum (3.3) over all maximal divisors of $p-1$ and $p+1$ with respect to t, then we have an upper bound on the number of triples with order less than t in each orbit of order t. Explicitly,

if

$$t > \sum_{d \in \mathfrak{M}_t(p \pm 1)} \frac{3}{2} \max\left(\sqrt[3]{6td}, \frac{4td}{p}\right), \tag{3.4}$$

then every orbit of order t is guaranteed to have a triple of order larger than t. (In fact, this is still double counting elements of order dividing more than one maximal divisor; sharpening this bound is a direction for future work.)

Let L_p be the smallest $d \in \mathcal{D}(p \pm 1)$ such that every $d' \geq d$, with $d' \in \mathcal{D}(p \pm 1)$, satisfies (3.4). If it exists, then we call L_p the *middle game breakpoint*.

Lemma 3.10. *Every triple of order at least L_p is connected to $\mathcal{C}_p$.*

Proof. Let $v_1 \in \mathcal{M}(\mathbb{F}_p)^\times$ with order $d > L_p$. Then, v_1 is connected to some triple v_2 of order $d' \gneq d$. This argument may be repeated until a triple of order $p \pm 1$ is found, at which point we have reached $\mathcal{C}_p$. □

Algorithm 3.11 shows a procedure for calculating L_p.

Algorithm 3.11: Finding the middle game breakpoint L_p.

```
Input: prime p.
Result: L_p, either a real number or None.
 L_p ← None;
for t in D(p ± 1) \ {2} in ascending order do
    sum ← 0;
    for d in D, d < t do
        sum ← sum + max(∛(6dt), 4dt/p);
    end
    if t ≥ sum and L_p is None then
        L_p ← t;
    end
    else if t < sum then
        L_p ← None;
    end
end
return L_p;
```

In [10], Chen showed that the size of any connected component disconnected from $\mathcal{C}_p$ is divisible by p. This result was used in [15] to show the following.

Lemma 3.12 ([15, Lemma 2.2])**.** *If $p > 3$, then $|\mathcal{G}_p \setminus \mathcal{C}_p|$ is divisible by $4p$.*

Lemma 3.7 and Corollary 3.10 give two bounds on the orders of triples in $\mathcal{C}_p$. Our goal is now to count the number of triples which we have not yet shown are connected to $\mathcal{C}_p$. If the number of triples thus counted is less than $4p$, then $\mathcal{G}_p$ is connected by Lemmas 2.16 and 3.12.

Definition 3.13. Let $\phi : \mathcal{M}(\mathbb{F}_p) \to \{0,1\}$ be the boolean function

$$\phi(a) = \begin{cases} 1 & a \text{ is hyperbolic and } \operatorname{ord}_p(a) < \min(L_p, B_{(p,-)}), \\ 1 & a \text{ is elliptic and } \operatorname{ord}_p(a) < \min(L_p, B_{(p,+)}), \\ 0 & \text{otherwise,} \end{cases}$$

where L_p and $B_{(p,\pm)}$ are the middle game and endgame breakpoints, respectively. A coordinate a has *small order* if $\phi(a) = 1$. The set of coordinates with small order is denoted $\mathcal{S}$.

Definition 3.14. Let φ be a boolean function $\mathcal{M}(\mathbb{F}_p) \to \{0,1\}$. A triple $v \in \mathcal{M}(\mathbb{F}_p)$ is a *bad triple* if $\varphi(v) = 0$. The set of bad triples is denoted $\mathcal{B}$.

We now wish to construct a boolean function φ with two competing mandates. First, φ must be as sensitive as possible; that is, we wish to minimize the number of triples v connected to $\mathcal{C}_p$ but for which $\varphi(v) = 0$. Second, we would like φ to be easy to compute, in the sense that a computer can quickly identify bad triples with respect to φ. To this end, we have two strategies:

Strategy 3.15. For each $(a,b) \in \mathcal{S} \times \mathcal{S}$, determine the zero, one, or two values of c for which $(a,b,c) \in \mathcal{M}(\mathbb{F}_p)$ using

$$c = \frac{1}{2}\left(ab \pm \sqrt{a^2b^2 - 4(a^2+b^2)}\right).$$

Any triple found this way for which a, b, and c all have small order is a bad triple.

Strategy 3.16. For each $a \in \mathcal{S}$ and $a = \chi + \chi^{-1}$, and for each coset of $\langle \chi \rangle$, pick a representative $r \in \mathbb{F}_{p^2}^\times / \langle \chi \rangle$ and calculate the entries in the orbit of rot'_a given by (3.1). Note that $r \in \mathbb{F}_p$ if a is hyperbolic, and $r \in \mathbb{F}_{p^2}^\times$ with order dividing $2(p+1)$ if a is elliptic; see Corollary 3.6. Check a set number of triples in the orbit[1]; if they are all of small order, then every triple in the orbit is bad. Otherwise, none of the triples in the orbit are bad.

For every $a \in \mathbb{F}_p$, we run one of the two above strategies to identify all bad triples with first coordinate a. We choose the strategy requiring fewer checks, i.e., we choose Strategy 3.15 when $|\mathcal{S} \times \mathcal{S}| < (p \pm 1)/|\chi|$, and Strategy 3.16 otherwise.

Remark 3.17. Strategy 3.16 is more sensitive, since a triple may have all coordinates with small order and yet still be "good". It is possible that, for some prime p, choosing to only run Strategy 3.16 could affirmatively show that $\mathcal{G}_p$ is connected while sometimes running Strategy 3.15 is inconclusive.

[1]See Figure 3.3 for the maximum number of triples checked per orbit. We cap the number of triples checked to prevent our program hanging on a prime for which an exceptional number of checks must be performed. In practice, we found no such exceptional prime out of all primes less than one million, or among those randomly sampled less than 110,000,000.

Strategy 3.16 has a second advantage; it will only ever test Markoff triples. Strategy 3.15, on the other hand, must filter through (a, b) pairs for which no value of c gives a Markoff triple. The runtime cost of performing this filter is similar to the weakening of the estimations in [15] caused by assuming each pair (a, b) has two values of c making $(a, b, c) \in \mathcal{M}(\mathbb{F}_p)$. Nevertheless, when the middle game breakpoint or $|\chi|$ is small, Strategy 3.15 may still be faster.

3.2 Computational Machinery

In this section, we describe the data structures and procedures needed to implement the strategies described in Section 2. We begin with the *factor trie*[2], encoding a canonical poset on $\mathcal{D}(p \pm 1)$. This poset will allow us to recursively generate only those elements $a \in \mathbb{F}_p$ satisfying $\mathrm{ord}_p(a)$ less than some upper bound (or a being parabolic), without having to filter the entirety of $\mathbb{F}_p$ by order.

Associate a word $w(n)$ to each $n = p_1^{t_1} \cdots p_n^{t_n}$ with primes in increasing order given by

$$w(n) = \underbrace{p_1 \ldots p_1}_{t_1} \cdots \underbrace{p_n \ldots p_n}_{t_n}. \tag{3.5}$$

Let $\mathcal{D}(n)$ be the set of positive divisors of n. Define a partial ordering $\prec$ on $\mathcal{D}(p \pm 1)$ as $n \prec m$ if the word $w(n)$ is a prefix for the word $w(m)$. If $n \prec m$ and there is no ℓ such that $n \prec \ell \prec m$, then we write $n \prec\!\cdot\, m$.

Define the factor trie $T_n = (V, E)$ as the graph with vertex set $V = \{w(d) : d \in \mathcal{D}(n)\}$ and directed edge set $E = \{(w(\ell), w(m)) : \ell \prec\!\cdot\, m\}$. When the context is clear, we will identify $\mathcal{D}(n)$ and $\prec\!\cdot$ with the vertex and edge sets directly. An example factor trie for $n = 60$ is shown in Figure 3.1.

Associate to each vertex of T_n a finite cyclic group via the covariant functor $\mathcal{F}_n : T_n \to \mathbf{Ab}$ defined by $\mathcal{F}_n : w(d) \mapsto \mathbb{Z}/d\mathbb{Z}$ and taking the directed edge $\ell \prec\!\cdot\, m$ to the inclusion map $\mathbb{Z}/\ell\mathbb{Z} \hookrightarrow \mathbb{Z}/m\mathbb{Z}$. Note that there are no intermediate subgroups between these groups.

The tries $T_{p\pm1}$ will form the call graphs for our recursive procedure, with vertex $d \notin \{1, 2\}$ generating the elements $\{x \in \mathbb{Z}/p\mathbb{Z} : \mathrm{ord}_p(x) = d\}$; these are exactly the generators of $\mathcal{F}_{p\pm1}(d)$. (If $d \in \{1, 2\}$, then $x = \pm 2$ and is parabolic, see Remark 2.18.) If $d_1 \prec d_2$, then $|\mathcal{F}_{p\pm1}(d_1)| < |\mathcal{F}_{p\pm1}(d_2)|$, so our x values are generated with increasing $\mathrm{ord}_p(x)$ moving down the call graph induced by $T_{p\pm1}$, and we can stop the procedure when we have reached the desire bound.

We could represent elements of the group $\mathcal{F}_{p\pm1}(d)$ as integers modulo d. However, it will be more convenient to represent them instead as integer arrays based on the unique decomposition of these groups into

[2] A *trie* is a tree where the vertex set is the set of unique paths from the root. Some authors use this as the definition for a tree; others define a tree as a connected acyclic graph. We use the term *trie* to emphasize the word structure of the graph.

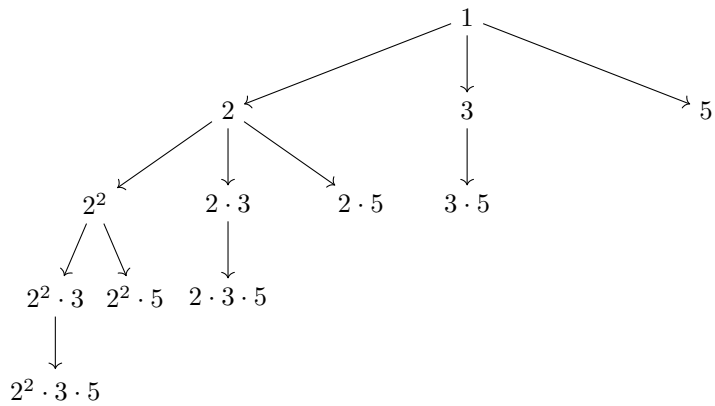


Figure 3.1: The factor trie for $n = 60$.

a direct sum of prime power cyclic groups. The multiplicative group operations can then be performed by the CPU using only componentwise addition and subtraction, which are much faster than direct multiplication and reciprocal taking. One drawback is that addition and subtraction are not easily represented, so a conversion back to $\mathbb{F}_p$ is necessary before any ring operations. We now describe this representation.

Let $p - 1 = p_1^{t_1} \cdots p_n^{t_n}$ and $p + 1 = q_1^{s_1} \cdots q_m^{s_m}$, and assume $p_1 = q_1 = 2$.

Fix, once and for all, a $\mathbb{Z}$-basis of $\mathbb{F}_p^\times$ of the form $\{g_i\}_{i=1}^n$, where $|g_i| = p_i^{t_i}$. Making this choice is a special case of [27, Algorithm 6.1], which is our preferred method. This choice induces a group isomorphism

$$\iota_{p-1} : \mathbb{F}_p^\times \xrightarrow{\sim} \bigoplus_{i=1}^n \mathbb{Z}/p_i^{t_i}\mathbb{Z}$$

given by

$$\prod_{i=1}^n g_i^{r_i} \mapsto (r_1, \ldots, r_n). \tag{3.6}$$

An identical construction exists for the elliptic coordinates by replacing $\mathbb{F}_p^\times$ with the subgroup $E \subset \mathbb{F}_{p^2}^\times$ with order $p + 1$. Taken together, we have the commutative diagram

$$\begin{tikzcd}
 & \mathbb{Z}/p\mathbb{Z} & \\
\mathbb{F}_p^\times \arrow[ur, "\psi_{p-1}"] \arrow[r, hook, "\iota_{p-1}"'] & \left(\bigoplus_{i=1}^n \mathbb{Z}/p_i^{t_i}\mathbb{Z}\right) \amalg \left(\bigoplus_{i=1}^m \mathbb{Z}/q_i^{s_i}\mathbb{Z}\right) \arrow[u, dashed] & E \arrow[ul, "\psi_{p+1}"'] \arrow[l, hook', "\iota_{p+1}"]
\end{tikzcd}$$

where $\psi_{p\pm1}(\chi) = \chi + \chi^{-1}$ (see Lemma 3.4). To the vertex $d = p_1^{d_1} \cdots p_n^{d_n} \in \mathcal{D}(p-1)$ we have now associated both the subgroup of $\mathbb{F}_p^\times$ of order d and the group

$$\mathcal{F}_{p-1}(d) = \mathbb{Z}/d\mathbb{Z} \cong \bigoplus_{i=1}^n \mathbb{Z}/p_i^{d_i}\mathbb{Z}$$

containing integer arrays given by (3.6), and similarly for the divisors of $p+1$.

Remark 3.18. Since $\psi_{p\pm1}(\chi) = \psi_{p\pm1}(\chi^{-1})$, the maps ψ_{p-1} and ψ_{p+1} are 2-to-1 when restricted to $\mathbb{F}_p^\times \setminus \{\pm1\}$ and $E \setminus \{\pm1\}$, respectively. Since $\psi_{p-1}(\pm1) = \psi_{p+1}(\pm1)$, the induced (dotted) map in the commutative diagram is 2-to-1.

We now describe the recursive procedure for generating a value $a \in \mathcal{S}$ (see Definition 3.13) with order $\mathrm{ord}_p(a) = d$. We begin with the integer array $(0, 0, \ldots, 0)$ associated to $w(1)$, and propagate this array down the factor trie recursively, with each node $w(d)$ yielding[3] all the coordinates $a \in \mathbb{F}_p$ with $\mathrm{ord}_p(a) = d$, mapping the integer array to a via the inverse map of (3.6).

Remark 3.19. Parabolic triples are connected to $\mathcal{C}_p$ by definition, so we do not want to generate χ values for which $|\chi| = d \in \{1, 2\}$ (see Remark 2.18).

In light of Remark 3.18, we would like to yield only one representative from each $\{\chi, \chi^{-1}\}$ pair. The following definition and lemma provides a characterization for making this choice canonical.

Definition 3.20. Let $p \pm 1 = p_1^{t_1} \cdots p_n^{t_n}$ with $p_1 = 2$, and let $(r_1, \ldots, r_n) \in \oplus_{i=1}^n \mathbb{Z}/p_i^{t_i}\mathbb{Z}$ be an integer array according to (3.6) not representing a $\chi \in \mathbb{F}_{p^2}^\times$ with order 1 or 2. Let

$$k = \min\{j : r_j \neq 0 \text{ and } (p_j, r_j) \neq (2, 2^{t_1-1})\}.$$

The integer array is *in the lower half* if $r_k \leq p_k^{t_k}/2$.

Lemma 3.21. *Let $\chi \in \mathbb{F}_{p^2}^\times$ such that $|\chi| \notin \{1, 2\}$ and $|\chi| \in \mathcal{D}(p \pm 1)$. Then, exactly one of $\iota_{p\pm1}(\chi)$ or $\iota_{p\pm1}(\chi^{-1})$ is in the lower half.*

Proof. Consider a hyperbolic $a \in \mathbb{F}_p$ with $a = \chi + \chi^{-1}$ and $\chi \in \mathbb{F}_p^\times$. (The elliptic case is handled identically.) Let $\chi = g_1^{r_1} \cdots g_n^{r_n}$. Since $\iota_{p-1}(\chi^{-1}) = (a_1, \ldots, a_n)$ where

$$a_i = (p_i^{t_i} - r_i) \bmod p_i^{t_i},$$

if $|\chi| \notin \{1, 2\}$, then at least one of r_k and $p_k^{t_k} - r_k$ is less than or equal to $p_k^{t_k}/2$. Therefore, at least one of $\iota_{p-1}(\chi)$ and $\iota_{p-1}(\chi^{-1})$ is in the lower half.

Now assume that both χ and χ^{-1} are in the lower half. For each j such that $p_j \neq 2$, it must be that $r_j = 0$. But then our array is either $(0, \ldots, 0)$ or $(2^{t_1-1}, \ldots, 0)$, rearranging if necessary so $p_1 = 2$, and these arrays correspond to $|\chi| = 1$ and $|\chi| = 2$, respectively. □

[3] A `yield` statement is similar to a `return` statement in that it passes data back to the caller of the function; it differs in that execution continues in the callee, allowing for more than one value to be yielded.

We will only yield integer arrays in the lower half, thus generating each desired element of $\mathbb{F}_p$ exactly once. We do this by setting upper bounds on the entries in the arrays yielded by node $w(d)$ for $d = p_1^{d_1} \cdots p_m^{d_m}$ according to

$$r_j \leq \ell_{d,j} = \begin{cases} p_j^{t_j}/2 & j = \min\{j : p_j^{d_j} \neq 2\}, \\ p_j^{t_j} & \text{otherwise.} \end{cases} \tag{3.7}$$

We now present our algorithm for yielding Markoff coordinates $a \in \mathcal{S}$ in Algorithm 3.22. Each value of χ yielded corresponds to a Markoff coordinate $\chi + \chi^{-1}$.

Algorithm 3.22: Generating χ corresponding to desired Markoff coordinates $\chi + \chi^{-1} \in \mathcal{S}$.

```
Input: Factor trie T_{p±1} with p ± 1 = p_1^{t_1} ··· p_n^{t_n},
Limits ℓ_1, ..., ℓ_n as in (3.7),
Node w(d) with d = p_1^{d_1} ··· p_m^{d_m},
Array of integers r_1, ..., r_m.
Base case: propagate(w(1), 0, ..., 0)
Result: Stream of χ values.
fn propagate(w(d), r_1, ..., r_m)
    for i in 0 to p_m do
        r'_m ← r_m + p_m^{t_m − d_m};
        if r'_m > ℓ_{d,m} then
            break;
        end
        if d_m < t_m then
            propagate(w(p_1^{d_1} ··· p_m^{d_m+1}), r_1, ..., r'_m);
        end
        if i = 0 then
            /* Yielding when i = 0 would not respect the trie structure, i.e.,
               there would be multiple call paths to yield the same value. */
            continue;
        end
        if 2 < d < min(L_p, B_{(p,±)}) then
            yield g_1^{r_1} ··· g_{m−1}^{r_{m−1}} · g_m^{r'_m};
        end
        for j in m + 1 to n do
            if d_j < t_j and w(d · p_j) or there is a d' ≻ d such that d' < min(L_p, B_{(p,±)}) then
                propagate(w(d · p_j), r_1, ..., r'_m);
            end
        end
    end
```

Besides generating all $a \in \mathcal{S}$, we can use the factor trie to generate cosets of $\langle \chi \rangle$ for $\chi \in \mathbb{F}_{p^2}^\times$, as in Strategy 3.16. For this we need one representative for each coset of $\langle \chi \rangle$. That is, for $|\chi| \in \mathcal{D}(p \pm 1)$, we need one value of order d for each divisor d of $(p \pm 1)/|\chi|$. Let $|\chi| = p_1^{s_1} \cdots p_n^{s_n}$. We can repurpose Algorithm 3.22

by instead interpreting the yielded values as r in (3.1), and replacing the upper bounds in (3.7) with

$$\ell_{d,j} = \begin{cases} 0 & d_j \leq s_j, \\ p_j^{t_j}/2 & d_j > s_j \text{ and } j = \min\{j : p_j^{d_j} \neq 2\}, \\ p_j^{t_j} & \text{otherwise.} \end{cases} \tag{3.8}$$

3.3 Our Algorithm & Results

In this section, we combine the process described in Section 2 with the machinery constructed in Section 3 to produce a comprehensive algorithm for proving that $\mathcal{G}_p^\times$ is connected. The code described in this section can be found at `https://github.com/colbyaustinbrown/libbgs`, with executable code found in the `examples` directory. We used the Rust programming language, version 1.78, for our `libbgs` library defining the data structures described in Section 3. Rust is a strongly, statically typed imperative language prioritizing speed, memory safety, and concurrency. We now present our algorithm for testing whether $\mathcal{G}_p^\times$ is connected.

Algorithm 3.23 (Testing $\mathcal{G}_p$ for connectivity)**.**

1. Calculate the prime factorizations for $p \pm 1$, and construct the factor tries $T_{p\pm 1}$. Also, fix $\mathbb{Z}$-bases of $\mathbb{F}_p^\times$ and E for (3.6), and also a value m with order $2^t || p^2 - 1$ (see Corollary 3.6) using [27, Algorithm 6.1]. Note that the choice of basis requires $O(\log p)$ choices via randomly sampling elements of $\mathbb{F}_{p^2}$, with each choice being made within an expected 2 samples.

2. Use (3.2) and Algorithm 3.11 to determine the endgame and middle game breakpoints.

3. Generate all coordinates with small order $a \in \mathbb{F}_p$ according to Definition 3.13 via Algorithm 3.22, and proceed to step 4 for each a.

4. If $(p \pm 1)/\operatorname{ord}_p(a) < |\mathcal{S} \times \mathcal{S}|$, go to step 4(b); otherwise, go to step 4(a).

   (a) Perform Strategy 3.15: Use Algorithm 3.22 to generate all $b \in \mathbb{F}_p$ with small orders. For each b value, calculate the 0, 1, or 2 values of c for which $(a, b, c) \in \mathcal{G}_p$. Any c value with small order gives a bad triple (a, b, c).

   (b) Perform Strategy 3.16: Generate r values given by Algorithm 3.22 modified with (3.8), and use them to calculate up to (fixed) n values b from (3.1). (If a is elliptic, the values r must be multiplied by the value m chosen in Step 1 before being plugged in to (3.1), see Corollary 3.6.) If every b has order smaller than L, then every triple in the orbit is bad.

5. Count the number of bad triples found. If the result is less than $4p$, then $\mathcal{G}_p$ is connected. Otherwise, the algorithm is inconclusive.

Using Algorithm 3.23, we confirmed that $\mathcal{G}_p$ is connected for all $p < 1,000,000$, confirming Theorem 3.3. We also ran our algorithm for a random sample of $1,000$ primes $p < 110,000,000$, and $\mathcal{G}_p^\times$ was connected for each of these primes, too. We did not find any $\mathcal{G}_p$ with at least p bad triples, with two exceptions: $p = 7,558,541$ and $p = 96,840,901$, for which the number of bad triples found was $9,716,411$ and $103,370,751$, respectively. The criteria for connectivity of [10] was nearly always sufficient in our tests, although we did rely on the improved bounds of [15] that any $|\mathcal{G}_p^\times \setminus \mathcal{C}_p| \geq 4p$. The number of bad triples per prime is shown in Figure 3.2. Table 3.1 shows the prime factorizations of $p \pm 1$ and bad triple counts for a sample of primes less than one million. Generally, the number of bad triples is inversely correlated with the number of prime divisors (with multiplicity) of $p \pm 1$.

Table 3.1: For $p < 1,000,000$, the 10 primes with the largest ratio $|\mathcal{B}|/p$, and the 5 largest primes with $\mathcal{B} = \emptyset$.

	Hyperbolic		Elliptic	
p	$p-1$	Bad Triples	$p+1$	Bad Triples
825,287	$2 \cdot 7 \cdot 11 \cdot 23 \cdot 233$	277,287	$2^3 \cdot 3 \cdot 137 \cdot 251$	320,209
916,879	$2 \cdot 3 \cdot 17 \cdot 89 \cdot 101$	251,391	$2^4 \cdot 5 \cdot 73 \cdot 157$	425,410
804,203	$2 \cdot 7 \cdot 17 \cdot 31 \cdot 109$	295,979	$2^2 \cdot 3^2 \cdot 89 \cdot 251$	286,714
936,259	$2 \cdot 3 \cdot 17 \cdot 67 \cdot 137$	307,155	$2^2 \cdot 5 \cdot 13^2 \cdot 277$	362,722
734,803	$2 \cdot 3 \cdot 29 \cdot 41 \cdot 103$	171,268	$2^2 \cdot 7^2 \cdot 23 \cdot 163$	351,854
550,811	$2 \cdot 5 \cdot 13 \cdot 19 \cdot 223$	211,593	$2^2 \cdot 3 \cdot 197 \cdot 233$	168,181
858,701	$2^2 \cdot 5^2 \cdot 31 \cdot 277$	279,547	$2 \cdot 3 \cdot 13 \cdot 101 \cdot 109$	304,704
843,229	$2^2 \cdot 3^2 \cdot 59 \cdot 397$	320,154	$2 \cdot 5 \cdot 37 \cdot 43 \cdot 53$	250,655
995,677	$2^2 \cdot 3 \cdot 11 \cdot 19 \cdot 397$	0	$2 \cdot 497,839$	0
995,987	$2 \cdot 497,993$	0	$2^2 \cdot 3 \cdot 7 \cdot 71 \cdot 167$	0
996,783	$2^2 \cdot 3 \cdot 5 \cdot 37 \cdot 449$	0	$2 \cdot 498,391$	0
997,583	$2 \cdot 498,791$	0	$2^4 \cdot 3 \cdot 7 \cdot 7,969$	0
999,958	$2 \cdot 499,979$	0	$2^3 \cdot 3 \cdot 5 \cdot 13 \cdot 641$	0

In the worst case, steps 1 and 2 require on the order of $|\mathcal{D}(p \pm 1)| + |\mathcal{D}(p+1)|$ computations. Step 4 executes at most p times, and each iteration is bounded above by the runtime of step 4(b), which checks at most $\mathcal{D}(p \pm 1)$ orbits. The check on each orbit terminates after finding any triple not in $\mathcal{B}$; the largest number of such checks per prime is shown in Figure 3.3.

For the cap given in step 4(b), we used $n = 60$. In our execution of Algorithm 3.23, step 4(b) was terminated after 60 checks for only one prime, $p = 119,659$, and the algorithm still affirmatively determined $\mathcal{G}_p^\times$ is connected. We note that this cap is necessary to guarentee an almost linear asymptotic runtime at the expense of a possibly inconclusive response by the algorithm; however, the lack of growth in the number of required checks over the large range $p < 110,000,000$ (as shown in Figure 3.3) provides evidence that this

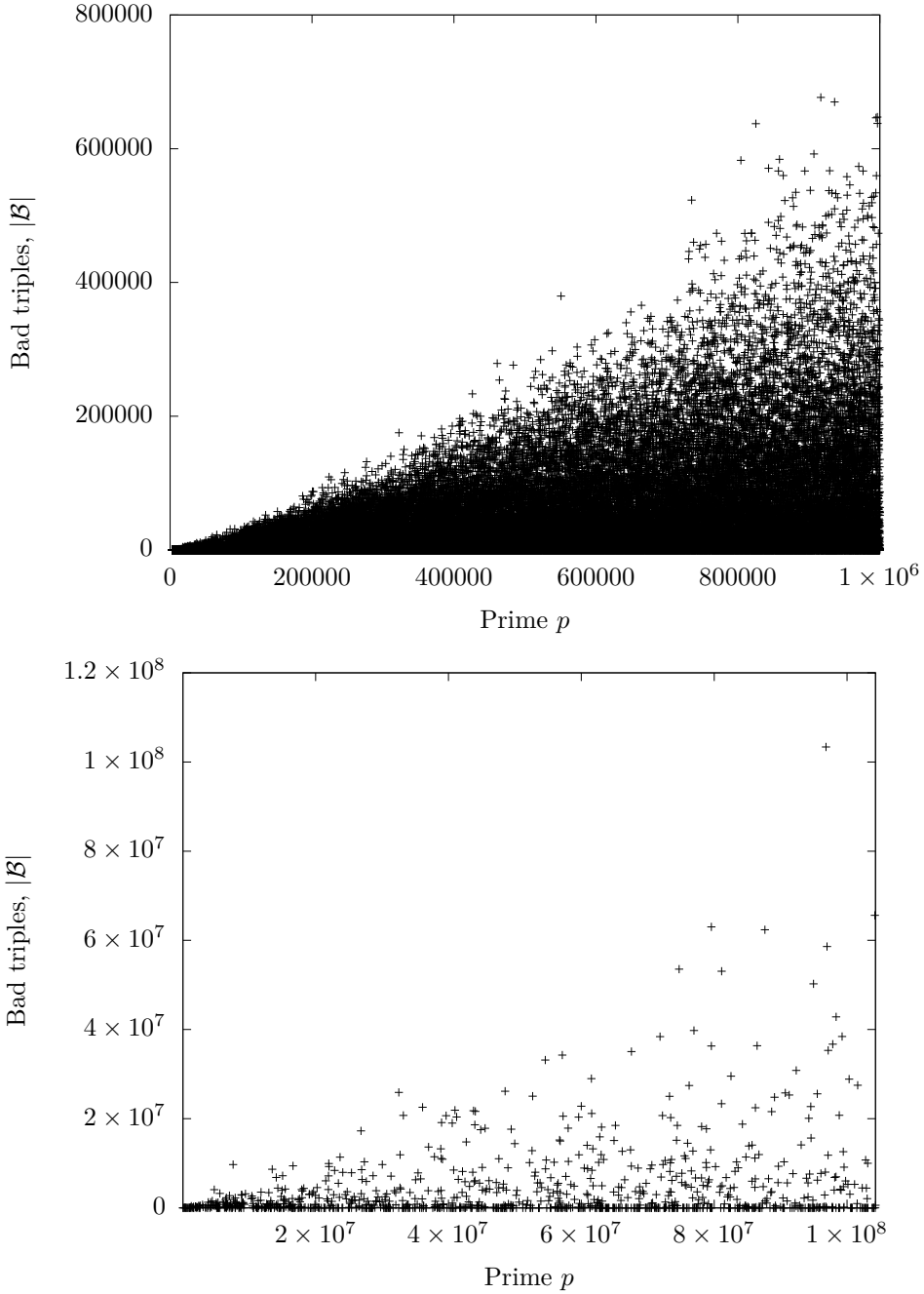


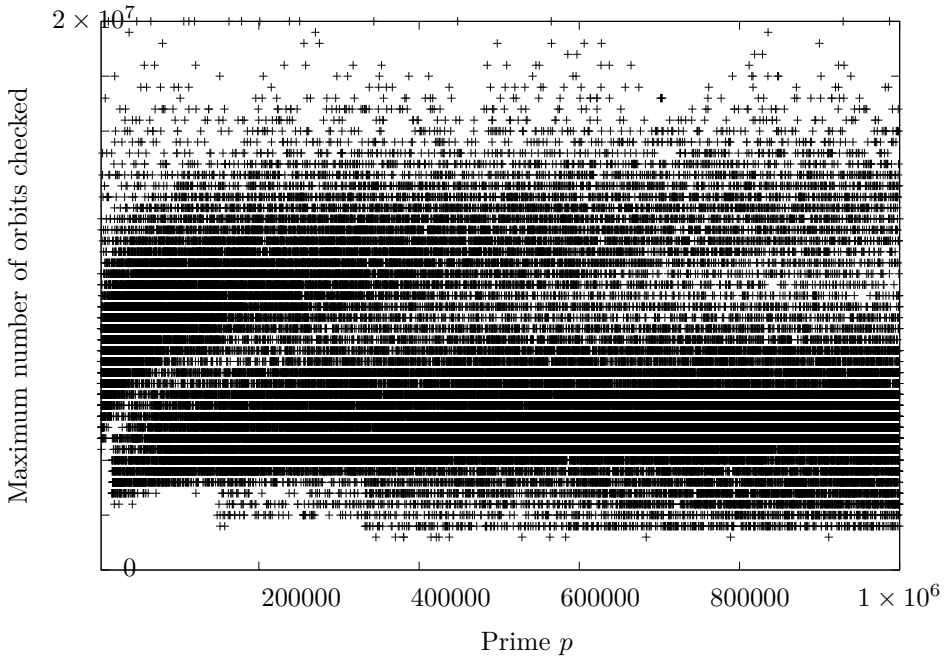
Figure 3.2: Total number of bad triples for all primes less than $1,000,000$ (top) and random primes less than $110,000,000$ (bottom).

check does not reduce the algorithm's power. We use the following result.

Theorem 3.24 ([22, theorem 315])**.** *For every $\epsilon > 0$, the divisor function $\mathcal{D}(n) \in o(n^{\epsilon})$.*

Therefore, our algorithm runs in $o(p^{1+\epsilon})$ time, in the worst case. The runtimes for all the primes we tested are shown in Figure 3.4. As emphasized in the figures, the almost linear bound on the runtime is a worst-case upper bound, and our algorithm performs well below this upper limit for a large number of primes.

The actual time to run Algorithm 3.23 depends heavily on both the endgame and middle game breakpoints. A plot of a the breakpoints for our random primes beneath 110 million, calculated using equation (3.2) and Algorithm 3.11, is shown in Figure 3.5. The first prime for which Algorithm 3.11 returns a middle game breakpoint is $p = 1,328,247$.



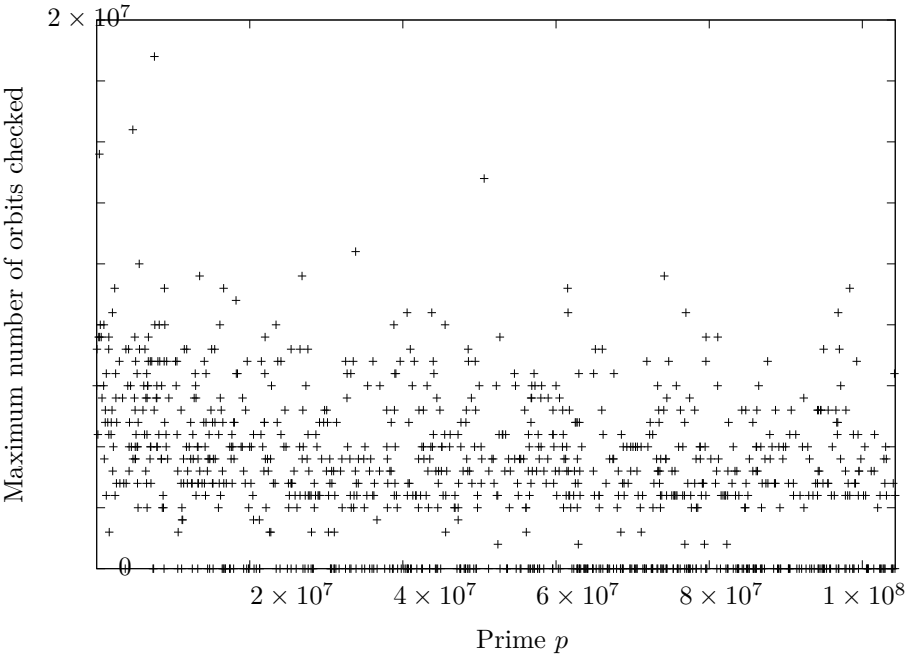


Figure 3.3: Maximum number of orbits checked during any iteration of Step 4(b) for all primes less than $1,000,000$ (top) and random primes less than $110,000,000$ (bottom).

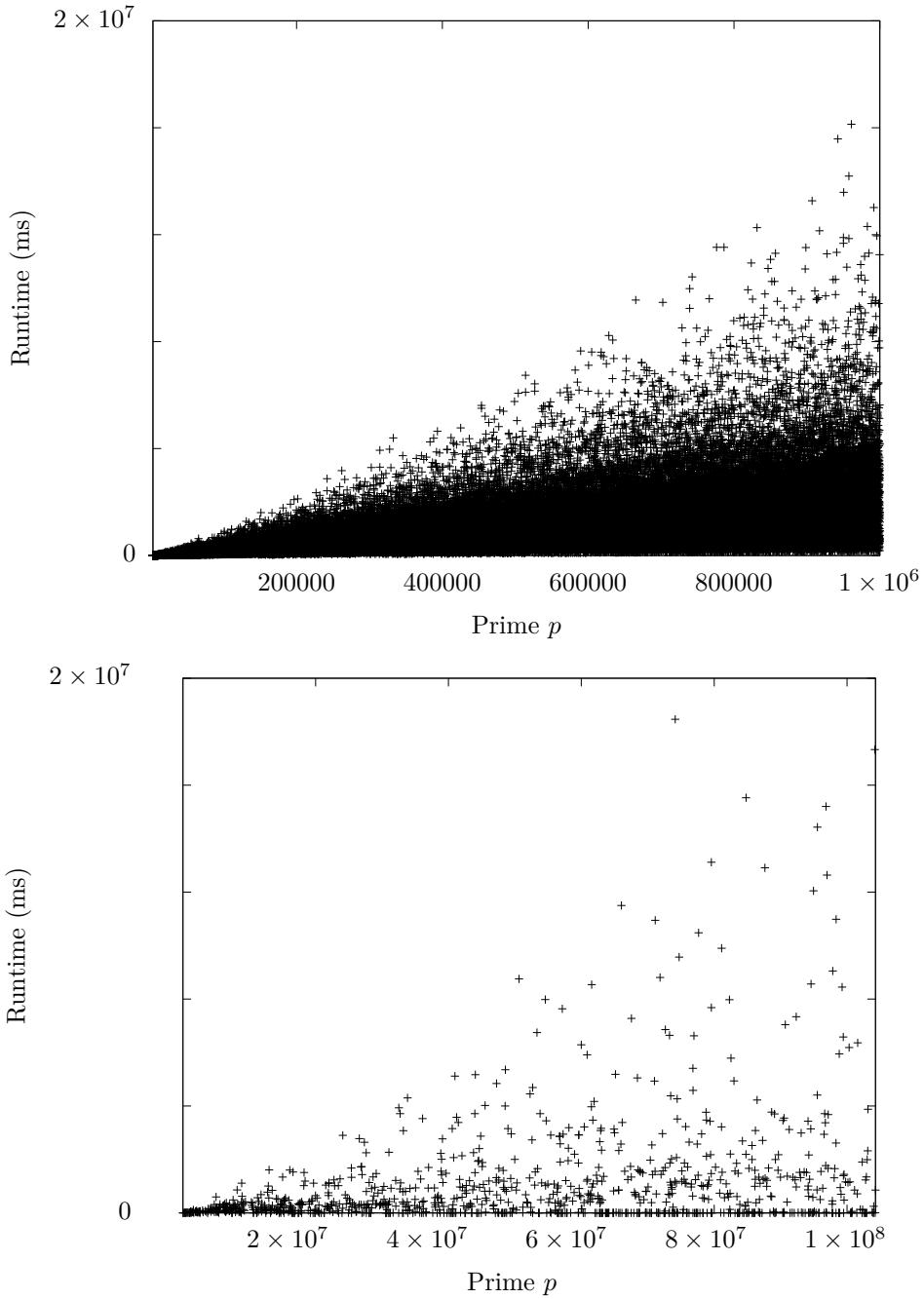


Figure 3.4: Runtimes for Algorithm 3.23 on all primes less than $1,000,000$ (top) and our random sample of primes less than $110,000,000$ (bottom). Measured on an 11th Gen Intel Core i5-11320H processor.

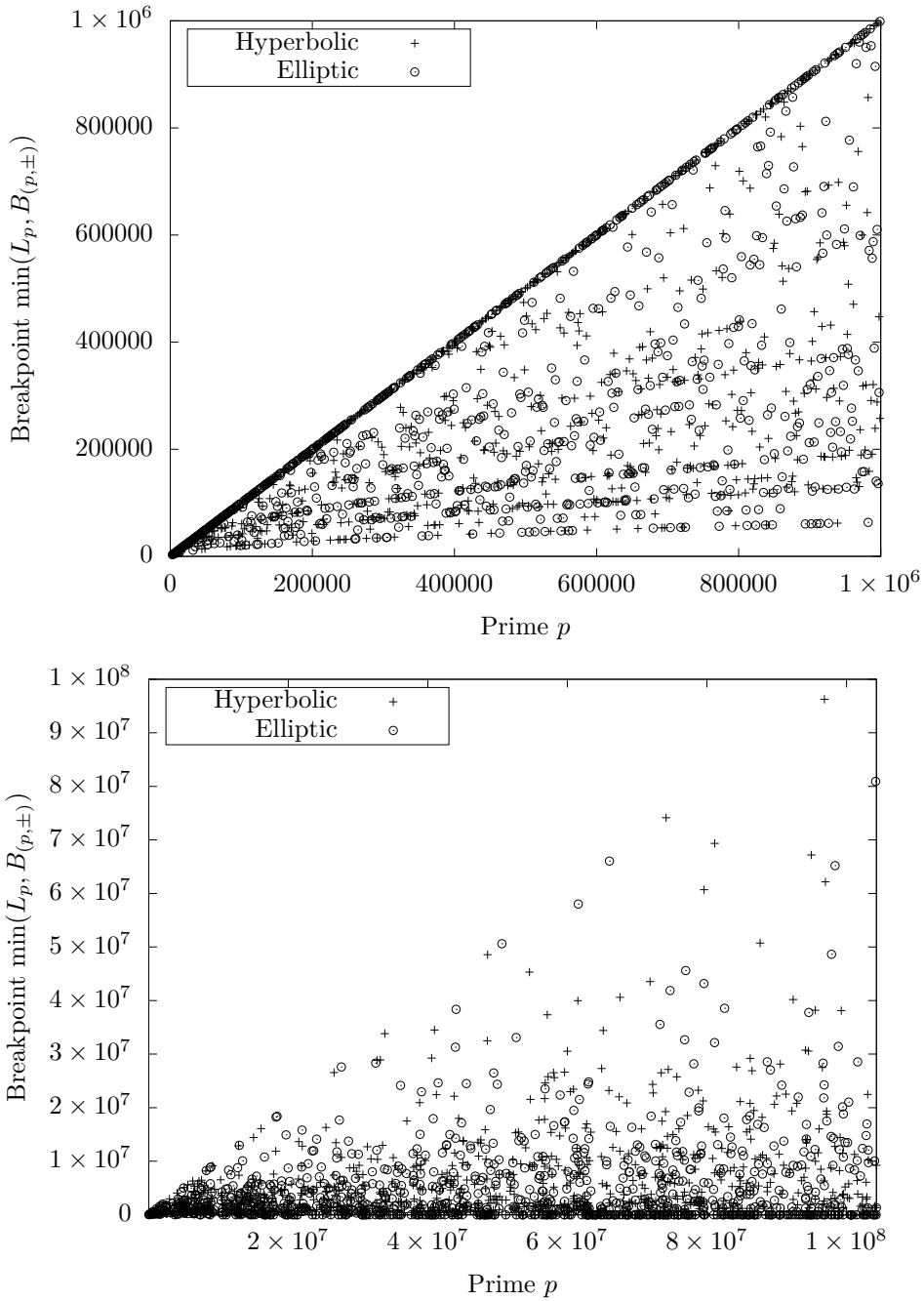


Figure 3.5: Breakpoints for all primes less than $1,000,000$ (top) and random primes less than $110,000,000$ (bottom).

Chapter 4

Markoff Triples in Lean 4

Computations and experimental mathematics have yielded several results on Markoff graphs. De Courcy-Ireland and Lee explicitly computed the adjacency matrices of Markoff graphs and their eigenvalue distributions as a demonstration of the spectra of the Markoff graph following the Kesten-MacKay law in [13]. Eddy et. al. used computers in [15] to test whether Markoff graphs were connected for extremely large p using approximations based on the prime factorizations of $p \pm 1$. Most recently, Martin [32] used computations in Sage as the final step in his proof of the connectivity of Markoff $\bmod p$ graphs of certain sizes. In Chapter 3, we demonstrated an algorithm for testing whether Markoff graphs are connected for small primes, as well as a realization of this algorithm.

In recent years, there has been much interest in assisted proof writing technologies and computer verifiable proofs. Since its introduction in 2013, Lean has been adopted across many mathematical fields. Great progress has been made towards formalizing important results like the prime number theorem [24] and Fermat's last theorem [7]. The broader Lean community has assembled Mathlib [39], a formidable library of constructions and theorems. Lean, as well as many other proof assistant languages, are also compilable to runnable programs, allowing computational mathematicians to prove the validity of their code.

In this chapter, we will use the Markoff graphs as an extended example of a mathematical construction formalized in Lean. We will also consider how our design presented here can provide a flexible tool for future research in Markoff triples.

One of our goals in this chapter will be to formalize the following theorem, originally presented in Chapter 1.

Theorem 4.1. *For every Markoff triple $(a, b, c) \in \mathcal{M}$, the coordinates a, b, and c are pairwise coprime.*

We will prove this theorem by induction on the Markoff tree, a technique also used commonly in many

other proofs on the properties of Markoff triples. Induction on the Markoff triples relies on the fact that, to any Markoff triple, there is a unique path of Vieta involutions from $(1, 1, 1)$. In the Markoff graphs modulo p, these paths are no longer unique, but the strong approximation conjecture suggests that there is still at least one such walk.

One example computation using this inductive technique is lifting a triple $\vec{x}$ from $\mathcal{M}_p$ back to $\mathcal{M}$ by performing the same walk in $\mathcal{G}$ as in $\mathcal{G}_p$ to go from $(1, 1, 1)$ to $\vec{x}$. For example, looking back to Figure 2.1 (and remembering to scale all the values by $3^{-1} \equiv 2 \pmod 5$), we see that a lift of $(0, 1, 2)$ is found by computing $V_1(V_3(1, 1, 1)) = (5, 1, 2)$.

In Section 1, we will review the basics of the Lean 4 type system necessary for our recursive principle. In Section 2, we will present Lean code samples constructing the Markoff triples, the Markoff graph, and proving their basic properties. Next, in Section 3, we will prove that, over the positive integers, the Markoff graph always contains a path to the root node $(1, 1, 1)$. Finally, in Section 4, we will define a recursor-like function for Markoff graphs based on the induction over walks from an arbitrary node to our root and use it to prove Theorem 4.1.

4.1 The Lean 4 Type System

4.1.1 Introduction to Lean

Lean is a language with a type theory capable of encoding mathematical constructions and proofs as well as compiling into executable code. Lean is primarily a metaprogramming language with extensive flexibility for introducing new notation as well as *tactics* which are sequentially run to construct proofs. Examples of tactics include `simp`, which applies series of "simp lemmas" to simplify expressions towards a so-called "simp normal form"; `omega`, which attempts to solve systems of inequalities or prove that no such solution exists with the current hypotheses; and `aesop`, the Automated Extensible Search for Obvious Proofs, which applies sequences of user-defined rules to attempt to automatically prove "easy" goals from the current set of hypotheses. Tactics proofs are intended to be written and inspected in an interactive environment, and proofs written using tactics will therefore sometimes be elided from this section when they contain no content relevant to the current discussion. All of the code will appear unabridged in the Appendix.

The authoritative source for the Lean 4 language is the official Lean reference manual [14].

The portion of the Lean language responsible for verifying the mathematical correctness of constructions and proofs is the *kernel*. To maintain confidence that the kernel will only certify correct proofs, the kernel itself is a relatively small part of Lean; it encodes a minimal set of features. Lean programs entered by

the user are *elaborated* into this kernel language, and the results of this elaboration may be checked by third-party implementations of the kernel as well.

Mario Carneiro in [8] showed that Lean is sound relative to Zermelo-Fraenkel set theory with the axiom of choice and countably many inaccessible cardinals. Basic constructions necessary for any Lean program are included with Lean itself. A huge corpus of mathematical results are implemented in Mathlib, a library which may be imported into Lean.

4.1.2 Function and Inductive Types

Constructions and proofs in Lean are built out of *expressions* which each have a *type*. The sentence "five is a natural number" may be restated as "'five' is an expression with type 'natural number'" and in Lean is written as `5: Nat`. Often, Lean is able to determine the type of an expression without having it written explicitly; when this is the case, we may elide the type annotation and write only `5`.

Function applications follow the type inference rule that if `f : A -> B` is a function which takes expressions of type `A` to expressions of type `B`, and `x : A` is an expression of type `A`, then `f x : B` is an expression of type `B`. The kernel verifies that the type of an argument in a function application matches the type of the domain of the function up to *definitional equality*, checked syntatically. Definitional equality is reflexive and symmetric, but not transitive [8]; Lean therefore has an internal notion of *propositional equality* which is what is used in Lean proofs.

It is tempting to think of types as being analogous to sets, but the comparison should not be taken too far. There are no subtype relations in Lean, and there are no intersection or union operations on types. In Lean, the type of an expression is unique up to definitional equality of types [8].

We now briefly describe some of the basic principles of how the kernel verifies the soundness of expressions. For a more complete treatment of this topic, see [8]. Expressions must follow the inference rules of Lean's type theory; this is enforced by the Lean kernel during *typechecking*. Typechecking is context dependent, since expressions and types may contain previously defined expressions and types as subterms. A *context* is a list of expressions and their types which have already been defined. If a context Γ yields the judgement that e has type α, then we write $\Gamma \vdash e : \alpha$. When one judgement is written above another, we mean that the judgement on bottom follows from the one on top. If e, x_1, x_2 are expressions, and e depends on x_1, then $e[x_2/x_1]$ is the expression obtained by replacing all occurrences of x_1 with x_2 in e.

Functions and function application are defined by the following judgements:

$$\frac{\Gamma, x : \alpha \vdash e : \beta}{\Gamma \vdash \lambda x : \alpha . y : \Pi x : \alpha . \beta}$$

$$\frac{\Gamma \vdash f : \Pi x : \alpha, \beta \; \Gamma \vdash x_0 : \alpha}{\Gamma \vdash f x_0 : \beta[x_0/x]}$$

The first of these rules states that, if a function accepting a parameter $x : \alpha$ yields a result $e : \beta$, then the function $\lambda x : \alpha.e$ has type $\Pi x : \alpha, \beta$, where this resulting type is called a *function type* or a *pi type* and is one of the primitive types implemented in the Lean type theory. The second of these judgements is called *β-reduction* and states that a function $f : \Pi x : \alpha, \beta$ may only accept parameters of type α, and when an expression $x_0 : \alpha$ is applied, the resulting type is β with all occurrences of x replaced by occurrences of x_0. Note that the resulting type of a function may depend on the parameter; such functions are said to be *polymorphic*.

These typing rules make Lean into a *dependent* type theory, where types may depend both on other types as well as terms. We refer to types of the form $\Pi x : \alpha, \beta$ as a *pi type* or a *(dependent) function type*, and these are the types of functions from α to β, where β may depend on x.

Since types are themselves expressions, they also have typing judgements. The "types of types" are called *sorts* and are indexed by natural numbers called the *universe* of the sort. It is only terms with types which are a sort that may appear on the right side of a typing judgement `x: T`, where `T: Sort u` for a natural number u. Sorts form a non-cumulative hierarchy; each type belongs to exactly one sort, and each sort has as its type the next sort, `Sort u : Sort (u + 1)`. We abbreviate `Sort 1` as `Type`, and more generally we abbreviate `Sort u` as `Type (u - 1)` whenever $u > 0$. The bottommost sort, `Sort 0` and abbreviated as `Prop`, has several unique attributes which will be the focus of Section 4.1.4. Types and functions may also be polymorphic in their sorts; this polymorphism is implicit in lean whenever a *universe variable* appears in an expression. For example, the identity function `id.{u} {T: Sort u} -> T -> T` can be used in any sort. Other primitive operations on sorts exist in Lean, but we will elide almost all of our sorts.

The keyword `axiom` is used in Lean to declare a type to be axiomatically inhabited. For example, the axiom of choice is defined in Lean as

```
axiom Classical.choice: {T: Sort u} -> Nonempty T -> T
```

which says that a representative may be chosen out of any nonempty type. (This axiom being universe polymorphic makes it technically equivalent to the axiom of global choice.) Since the expression `Classical.choice` does not reduce to any other expression, the axiom of choice carries no computational data or value. Functions and data which otherwise carries computational information but uses an invocation of an axiom must be marked with the `noncomputable` keyword. Noncomputability is a transitive property, and any function which depends on a noncomputable function will itself be noncomputable. The Lean compiler will be unable to execute any runable code for items marked as `noncomputable`.

Sorts and function types are the only primitive types in Lean. All other types are brought into the type system as an *inductive types*. An inductive type is given by a list of parameters followed by a series of *constructors*, functions which do not β reduce but, when fully applied, type check into an instance of the inductive type. An example of an inductive type is shown below for the natural numbers:

```
inductive Nat where
   | zero: Nat
   | succ: Nat -> Nat
```

This type has two constructors: `zero`, which accepts no parameters and returns a natural number, and `succ`, which accepts a natural number which we understand to the predecessor, and returns a natural number. A second example which demonstrates the dependent nature of Lean types is the type of lists of a specific length:

```
inductive ListOfLength (T: Type) : (l: Nat) -> Type where
   | nil: ListOfLength T 0
   | cons: T -> (ListOfLength T k) -> ListOfLength T (k.succ)
```

This inductive type is parameterized by two variables, `T` representing the type of the elements contained in the list, and `k`, the length of the list. It is made up of two constructors; the first representing a list of length 0, and the second representing a value along with a list of length k, resulting in a list of length $k+1$.

In the previous example, the type `ListOfLength` depended on both `T: Type`, appearing on the left side of the colon, and `l: Nat`, appearing on the right. Type variables on the left are called *type parameters* and must remain fixed in every recursive reference of that type. The variables on the right are called *indices* and may differ between recursive calls. There are other syntactic restrictions on recursive definitions of inductive types, see [14].

Inductive types may be *pattern matched* on, creating different branches in the code for the different invocations of its constructors. For example, here is a function of type `Nat -> Prop` which returns `True` for the natural numbers 0 and 1, and `False` otherwise.

```
def isZeroOrOne: Nat -> Prop
   | .zero => True
   | .succ .zero => True
   | _ => False
```

In the above example, the cases of the parameter being 0 or 1 are considered in order, and then all other natural numbers will match the wildcard (underscore). Pattern matching is implemented at the elaboration

step, during which matches are translated into the more primitive feature of *recursors*, discussed in the next section.

Inductive types with exactly one constructor and no indices may also be created with the `structure` keyword. These inductive types enjoy automatically generated helper methods and lemmas by Lean but are otherwise identical to types declared with the `inductive` keyword.

4.1.3 Recursors

When an inductive data type is defined, Lean adds to its core logic a *primitive recursor* for that inductive type. The recursor encodes an inductive principle for that type. Many primitive operations on data types, such as pattern matching or unpacking, are implemented in terms of the recursor at the elaboration stage. A primitive recursor for a type is created based on its syntactic representation and cannot be override by users. A recursor is made up of three main kinds of parameters: one *motive*, the target type of the recursor invocation; a *minor premise* for each constructor, which must satisfy the motive for every application of that constructor and also carries the data of the recursion itself; and one *major premise*, the specific instance of the inductive type on which the recursor is to be applied.

As an example, the recursor for the natural numbers has the following signature:

```
Nat.rec
   {motive: Nat -> Sort u}
   (zero: motive Nat.zero)
   (succ : (n: Nat) -> motive n -> motive n.succ)
   (t: Nat):
      motive t
```

Using this recursor, a `ListOfLength` (defined previously) of the integers from 1 to n may be created like this:

```
def oneToN (n: Nat): ListOfLength Nat n :=
   Nat.rec
      (motive := fun k => ListOfLength Nat k)
      (zero := ListOfLength.nil)
      (succ := fun k m => ListOfLength.cons k m)
      (t := n)
```

Primitive recursors are created automatically and our inductive type of Markoff numbers will not be recursive. However, we can still define a similar method representing a recursive principle on paths from $(1, 1, 1)$

to an arbitrary Markoff triple in $\mathcal{M}$. The data structure we would like to use for this purpose is a `SimpleGraph.Walk` from the Mathlib library, which has the following definition:

```
inductive SimpleGraph.Walk {V: Type u} (G: SimpleGraph V):
      V -> V -> Type u where
   | .nil {V: Type u} {G: SimpleGraph V} {u: V}: G.Walk u u
   | .cons {V: Type u} {G: SimpleGraph V} {u v w: V} (h: G.Adj u v)
      (p: G.Walk v w): G.Walk u w
```

where `G.Adj u v` is a proof that vertices `u` and `v` are adjacent. Even though this type contains all the information we need and constructing an instance will be the primary goal of Section 3, its recursor is not quite the one we desire.

```
SimpleGraph.Walk.rec {V: Type u} {G: SimpleGraph V}
   {motive: (x y: V) -> G.Walk x y -> Sort u}
   (nil: {x: V} -> motive x x SimpleGraph.Walk.nil)
   (cons: {x y z: V} -> (h: G.Adj x y) -> (p: G.Walk y z) ->
      motive y z p -> motive u w -> motive x z (SimpleGraph.Walk.cons h p))
   {x y: V} {t: G.Walk x y}: motive x y t
```

This recursor has a motive parametrized over all walks between any two (connected) nodes. It is possible to avoid this problem by using a motive which itself has the extra hypothesis that the end of the walk in the root, but we will prefer to instead write our own recursor for this purpose in Section 4.

4.1.4 Propositions and Proofs

The bottommost sort in the Lean type theory is `Sort 0`, also called `Prop`. Its types correspond to what the working mathematician regards as propositions which are to be proven. Each proposition is its own type, and the inhabitants of these types are proofs. As an example, consider the following Lean code, which declares a new proposition and also provides its proof.

```
theorem exists_lt_of_ne_zero {n: Nat} (h: n ≠ 0):
    ∃m: Nat, m < n :=
  ⟨ n.pred, Nat.pred_lt h ⟩
```

The above code segment may be read like this: There is a theorem called `exists_lt_of_ne_zero` parameterized by a natural number `n`; furthermore, we assume that `n` is not equal to 0. The theorem states that there exists a natural number `m` such that `m < n`. The proof of this theorem is the natural number

which satisfies the predicate – namely, the predecessor of `n` – and the proof that this value really is less than `n`. The proof that `n.pred < n` is given by a theorem already in Lean called `Nat.pred_lt` which requires that `n` be not equal to 0; that fact is assumed in our theorem statement and is called `h`.

Remark 4.2. Let us pause for a moment and consider a design choice in the Lean definition of natural numbers which will illustrate the types of decisions to be made in defining Markoff triples. Normally, we define the predecessor function so its domain is all $n \geq 1$. If we wished to mirror this definition exactly in Lean, where all functions are total, we would be forced to define the predecessor function on a different type, the type of natural numbers along with proofs that these natural numbers are at least 1. This is valid but completely impractical. We would be forced to carry terms of type $n > 0$ throughout all of our code, even when it is irrelevant to the proof at hand. Even worse, it may sometimes be desirable to have subtraction as a total function on $\mathbb{N} \times \mathbb{N}$ saturating at 0, and supporting such behavior would require clunky logic. Instead, we simply define the predecessor of 0 to be 0, and then make $n > 0$ a necessary condition for theorems such as `Nat.pred_lt` of the above example. There is no fear of introducing inconsistencies – all we have done is chosen a more convenient definition.

To enable the intuitive understanding of `Prop` as being the "type of propositions" with inhabitants being proofs, `Prop` has a few special characteristics unique among all sorts.

First, `Prop` has *propositional extentionality*, which makes `P` and `Q` propositionally equal when `P ↔ Q`. This allows the typechecker to correctly match types with logically equivalent propositions in their terms. It is implemented as one of the three core axioms in the Lean standard library.[1] Since propositional extentionality is implemented as an axiom and not as a part of the kernel, it may be omitted from the type theory at the cost of most of the standard library, including the law of the excluded middle.

Second, `Prop` has *impredicative quantification*. For sorts higher than `Prop`, function types belong to the least upper bound of the universes for its parameters' types and return type. That is, if `A: Sort u` and `B: Sort v`, then the type `A -> B` lies in `Sort (min u v)`. This restriction on sorts higher than `Prop`, called *predicative quantification*, is necessary to avoid the type theory analogue of Russel's paradox. However, propositions are allowed to quantify over themselves, or over types in higher sorts. For example, we are allowed to write

```
theorem And.imp_and {P: Prop}: P → P ∧ P :=
  fun p => ⟨ p, p ⟩
```

[1]The other two are the axiom of choice, introduced in Section 1.2, and the axiom of quotient soundness, which states that equivalence classes of equivalent representatives are propositionally equal.

specifically as a proposition; without impredicative quantification for `Prop`, the above definition would lie in `Sort 1` and would not enjoy the unique characteristics belonging to `Prop`.

Third, terms of types in `Prop` have *definitional proof irrelevance*. If `P: Prop` is a proposition, and `a: P` and `b: P` are two proofs of `P`, then `a = b` holds definitionally, regardless of their expansion or syntactic representation. Terms of types in `Prop` themselves carry no data or computational value; if a theorem is proven, then it is irrelevant which proof was used. In Lean, any piece of data carrying computational value must be placed into `Sort 1` or higher. Definitional equality is a built-in feature of the Lean type system, is enforced by the kernel, and is not implemented in terms of more primitive features.

Since terms of the same type in `Prop` are all indistinguishable, they may not be matched on to compute a value in any higher sort. A benefit of this is that axioms (declarations in Lean which are given no expression to unfold to), including the axiom of choice, may be used freely to produce terms in `Prop` without affecting the computability of the remaining Lean code. On the other hand, functions which produce values with types lying in `Sort 1` or higher and transitively depend on the use of axioms are marked by the kernel as being *noncomputable*.

Definitional proof irrelevance has an important consequence for recursors of types in `Prop`. Since proofs must be interchangeable, we cannot generally allow recursors for types in `Prop` to have motives lying in higher universes without definitional irrelevance. Otherwise, we could write the following proof that `True = False`:

```
def left_or_right (h: True ∨ True): Prop :=
  match h with
    | .inl _ => True
    | .inr _ => False
example : True = False := by
  show left_or_right (.inl trivial) = left_or_right (.inr trivial)
  rfl
```

Here, we pattern match on the two constructors for the logical disjunction, in one case return the `True` proposition and in the other the `False` proposition. But due to proof irrelevance, the left and right constructors are definitionally equal, so the values of the calls to `left_or_right` must be equal, too. Hence, `True` and `False` are equal, clearly an inconsistency.

To prevent this inconsistency, Lean only allows recursors for inductive types in `Prop` to have motives which eliminate to other types in `Prop`. If the previous example is entered into a Lean environment, an error results identifying the line `match h with` and the fact that the resulting type is `Prop`, which is itself

a type in `Sort 1`. However, we may still pattern match on inductive types with multiple constructors as long as the resulting type is still a `Prop`, reflecting the ability to do "proof by cases" in Lean.

There is one exception to this rule. Consider an inductive proposition which has only one constructor, and that constructor only contains parameters which are all of type `Prop`, or are parameters to the inductive type. Such an inductive proposition carries no distinguishing data between two instances, and are therefore called *subsingleton propositions*. Subsingleton propositions may be eliminated into any universe, reflecting the fact that any elimination of a subsingleton proposition only depends on whether the proposition is true or false. Note that this exception is purely syntactic; the following inductive type can only be inhabited one way, yet is not subject to subsingleton elimination, since it contains two constructors.

```
inductive NotASubsingleton: Prop where
   | possible: NotASubsingleton
   | impossible: False -> NotASubsingleton
```

4.2 The Markoff Triples

The goal of this section is define the Markoff triples in Lean 4. We will also prove the properties which we will need in Section 4.3 where we will not only prove that the Markoff graph is connected but also define an inductive principle on the Markoff triples.

Even though we will ultimately prove our theorem over the natural numbers, there is no harm in setting up our initial definitions with a small amount of generality. We therefore fix a commutative ring `R` and define a `MarkoffTriple k` (over `R`) as a map from `Fin 3` (a type with 3 elements conveniently called 0, 1, and 2) to `R`, along with a proof that the three coordinates solve the Markoff equation.

Remark 4.3. We have have instead defined a Markoff triple as three values in `R`, called `a`, `b`, and `c`, as well as a proof of `a^2 + b^2 + c^2 = a * b * c`. This would be a equally valid way to define Markoff triples. We choose this definition, however, because of the syntactic ease with which we will be able to apply permutations (modeled as elements of `Fin 3 -> Fin 3`, this is simply function composition). These Markoff triples will also interface nicely in the future with multivariable polynomials, which are roughly speaking of type `(Fin 3 -> R) -> R`.

Here is what our definition looks like in Lean.

```
import Mathlib


@[ext]
```

```
structure MarkoffTriple {R: Type u} [CommRing R] (m k: R) where
   -- coordinates of a Markoff triple indexed by 0, 1, and 2.
   coords : Fin 3 -> R
   -- A Markoff triple solves the Markoff equation.
   solves':
      (coords 0)^2 + (coords 1)^2 + (coords 2)^2
         = m * (coords 0) * (coords 1) * (coords 2) + κ
deriving DecidableEq

namespace MarkoffTriple

variable {R: Type u} [CommRing R] {m κ : R}
```

Next, we define some helper abbreviations and establish the extentionality principle that two Markoff triples are equal if they are equal in all of their coordinates.

```
abbrev a (t : MarkoffTriple m κ) := t.coords 0
abbrev b (t : MarkoffTriple m κ) := t.coords 1
abbrev c (t : MarkoffTriple m κ) := t.coords 2

lemma eq_iff {₁t ₂t : MarkoffTriple m κ} :
    ₁t = ₂t ↔ ₁t.a = ₂t.a ∧ ₁t.b = ₂t.b ∧ ₁t.c = ₂t.c := {
  mp h := by aesop
  mpr h := by
    ext i
    fin_cases i <;> aesop
}

lemma solves (t: MarkoffTriple m k) (i: Fin 3):
    t[i] ^2 + t[i + 1] ^2 + t[i + 2] ^2 = k * t[i] * t[i + 1] * t[i + 2] := by
  /- proof in Appendix -/

lemma solves_twist (t: MarkoffTriple m k) (i: Fin 3):
    t[i] ^2 + t[i + 2] ^2 + t[i + 1] ^2 = k * t[i] * t[i + 2] * t[i + 1] := by
  /- proof found in Appendix -/
```

Next, we equip our Markoff triples with an S_3 action.

```
def perm (t: MarkoffTriple m k) ( σ: Perm (Fin 3)): MarkoffTriple m k := {
  coords i := t.coords σ⁻¹( i)
  solves' := by
   /- proof found in Appendix -/
}

@[simp]
lemma perm_get {t: MarkoffTriple m k} { σ: Perm (Fin 3)} {i: Fin 3}:
    (t.perm σ)[i] = t[ σ⁻¹ i] := by
  /- proof found in Appendix -/

instance : MulAction (Perm (Fin 3)) (MarkoffTriple m k) where
  smul g t := t.perm g
  /- proofs found in Appendix -/

@[simp]
lemma mul_get {t: MarkoffTriple m k} { σ: Perm (Fin 3)} {i: Fin 3}:
    ( σ • t)[i] = t[ σ⁻¹ i] := by
  /- proof found in Appendix -/
```

Completing our setup, we define the Markoff-Vieta involution, as well as provide helper lemmas for accessing coordinates after transformations.

```
def vieta (t : MarkoffTriple m k) (i: Fin 3): MarkoffTriple m k := {
   coords j :=
      if i = j then
         m * t[i + 1] * t[i + 2] - t[i]
      else
         t[j]
   solves' := by
      /- proof  found in Appendix -/
}

lemma vieta_eq (t: MarkoffTriple m k) {i: Fin 3}:
    (t.vieta i)[i] = k * t[i + 1] * t[i + 2] - t[i] := by
```

```
  /- proof found in Appendix -/

lemma vieta_eq_of_ne (t: MarkoffTriple m k) {i j: Fin 3} {i_ne_j: i ≠ j}:
    (t.vieta i)[j] = t[j] := by
  /- proof found in Appendix -/

theorem vieta_is_involution {t: MarkoffTriple m k} {i: Fin 3}:
    (t.vieta i).vieta i = t := by
  /- proof found in Appendix -/
```

It is worth considering a particular design choice made in the preceding code blocks. Our construction at this point is for Markoff triples over any commutative ring, yet we are really interested in studying them over the natural numbers, which is not even an additive group. We must therefore define a new type representing Markoff triples over $\mathbb{N}$. For all Markoff triples (a, b, c), we have $3ab > c$, so that $c' = 3ab - c$ is well-defined when restricted to $\mathbb{N}$, and respectively for the other coordinates. However, obtaining this result using only the properties of the natural numbers would be burdensome; for example, we would need to carry around terms of type `3ab > c` through nearly all of our lemmas. We also wish for our new type to interface nicely with other lemmas we may prove for `MarkoffTriple` which specialize to $\mathbb{N}$. For all these reasons, we choose to define the Markoff triples restricted to the first octant as a structure extension of `MarkoffTriple`. A structure which *extends* another structure inherits all of the fields of its parent and receives several helper methods from the elaborator, including coercions. Lean does not have subtyping, however, and values of a type declared as an extension are not inhabitants of the parent structure.

```
structure MarkoffTriplePos (m: ℤ) extends MarkoffTriple m 0 where
   pos {i : Fin 3} : coords i > 0
deriving DecidableEq

def MarkoffTriple.pos {m: ℤ} (t: MarkoffTriple m 0) (h: ∀i: Fin 3, t[i] > 0):
    MarkoffTriplePos m :=
  { t with pos := by aesop }

namespace MarkoffTriplePos
open MarkoffTriple

variable {m: ℤ}
```

```
lemma eq_of_cast_eq {a b: MarkoffTriplePos m}
    (h: a.toMarkoffTriple = b.toMarkoffTriple): a = b :=
  by aesop

instance : GetElem (MarkoffTriplePos m) (Fin 3) ℤ (fun _ _ => True) where
  getElem t i _ := t.toMarkoffTriple[i]

@[simp]
lemma get_eq {t: MarkoffTriplePos k} {i: Fin 3}:
    t.toMarkoffTriple[i] = t[i] := rfl

lemma solves (t: MarkoffTriplePos m) (i: Fin 3):
    t[i] ^2 + t[i + 1] ^2 + t[i + 2] ^2 = k * t[i] * t[i + 1] * t[i + 2] :=
  by simpa using t.toMarkoffTriple.solves i
```

With this setup, we gain access to the plethora of notational conveniences and arithmetic lemmas that comes with the `Int` type in Mathlib. Now, instead of needing to prove that $mbc > a$ for all Markoff triples, we can use the fact that $mbc - a = (b^2 + c^2)/a$ to show that the Markoff-Vieta involution fixes the first octant.

```
lemma vieta_alt_dvd (t: MarkoffTriplePos m) {i: Fin 3}:
    t[i] ∣ (t[i + 1])^2 + (t[i + 2])^2 := by
  /- proof found in Appendix -/

lemma vieta_alt (t: MarkoffTriplePos m) {i: Fin 3}:
    (t.vieta i)[i] = Int.divExact _ _ (t.vieta_alt_dvd (i := i))
   := by
  /- proof found in Appendix -/

def vieta (t: MarkoffTriplePos m) (i: Fin 3):
   MarkoffTriplePos m := {
  t.toMarkoffTriple.vieta i with
  pos := by /- proof found in appendix -/
```

Helper methods for `MarkoffTriplePos` parallel to those defined for `MarkoffTriple` may also be added; their definitions are omitted but may be found in the Appendix.

The Markoff graph $\mathcal{M}$ being connected follows from the observation that the Markoff-Vieta involution applied to the largest coordinate always decreases that coordinate, except for the root case $(1, 1, 1)$; conversely, applying the Markoff-Vieta involution to any other coordinate decreases that coordinate. In order to prove that the Markoff graph is connected, we first establish a theorem allowing us to recognize when we have arrived at our base case. Namely, if two coordinates are equal, then they are both equal to 1.

```
theorem a_eq_one_of_a_eq_b {t: MarkoffTriplePos (3: ℤ)} {i j: Fin 3} (h1: i ≠ j) (h2:
    t[i] = t[j]):
    t[i] = 1 := by
  -- begin by reducing to the case that j = i + 1
  wlog h: j = i + 1
  · have h': j = i + 2 := (by omega;) subst h'
    simpa [Equiv.swap_apply_of_ne_of_ne] using this
      (t := t.perm <| swap (i + 1) (i + 2))
      (j := i + 1) (by simp)
(by simp [Equiv.swap_apply_of_ne_of_ne, h2]) rfl
  subst h; clear h1
  have := t.solves i
  erw [←h2] at this
  conv at this => simp; ring_nf
  -- at this point, we have hypothesis
  --  this: t[i] ^2 * 2 + t[i + 2] ^2 = t[i] ^2 * t[i + 2] * 3
  -- and goal ⊢ t[i] = 1
  -- next, unpack the divisibility statement into m: ℤ and
  -- local hypothesis
  --  hm: t[i + 2] = t[i] * m
  obtain ⟨ m, hm ⟩: t[i] ∣ t[i + 2] := by
    apply (Int.pow_dvd_pow_iff two_ne_zero).mp
    apply (Int.dvd_iff_dvd_of_dvd_add (b := (t[i])^2 * 2) ?_).mp (by simp)
    rw [this, mul_assoc]; apply dvd_mul_right
  rw [
      hm
    , show (t[i])^2 * (t[i] * m) * 3 = (t[i])^2 * ((t[i]) * m * 3) by ring
    , show (t[i])^2 * 2 + (t[i]*m)^2 = (t[i])^2 * (2 + m^2) by ring
  ] at this
  apply Int.eq_of_mul_eq_mul_left
```

```
    (Int.pow_ne_zero <| t.is_pos.ne.symm) at this
  have m_le_two: m ≤ 2 := by
    apply Int.le_of_dvd zero_lt_two
    apply (Int.dvd_iff_dvd_of_dvd_add (c := m^2) ?_).mpr (by simp)
    rw [this]
    apply dvd_mul_of_dvd_left
    apply dvd_mul_of_dvd_right dvd_rfl
  have m_pos: m > 0 := by
    apply Int.pos_of_mul_pos_right (a := t[i]) ?_ t.is_pos
    apply Int.pos_of_mul_pos_left (b := 3) ?_ zero_lt_three
    rw [←this]
    apply Int.add_pos_of_pos_of_nonneg zero_lt_two
    exact Int.pow_nonneg_of_even rfl
  interval_cases m
  · /* m = 1 */ simpa using this
  · /* m = 2 */ simpa [show t[i] * 2 * 3 = 6 * t[i] by ring] using this
```

Next, we establish that a Markoff-Vieta involution applied to a non-strictly maximum coordinate strictly increases that coordinate.

```
theorem z_vieta_gt_self_of_nonmax_of_fundamental_domain
  {t: MarkoffTriplePos 3}
  {i: Fin 3} {p: t[i] ≤ t[i + 1] ∨ t[i] ≤ t[i + 2]}:
    (t.vieta i)[i] > t[i] := by
  wlog h': t[i] ≤ t[i + 1]
  · replace p := p.resolve_left h'
    simpa [Equiv.swap_apply_of_ne_of_ne, vieta_eq] using this
      (t := t.perm <| swap (i + 1) (i + 2)) (i := i)
      (p := by simp [Equiv.swap_apply_of_ne_of_ne, p])
      (by simp [Equiv.swap_apply_of_ne_of_ne, p])
  suffices t[i] * t[i] < t[i] * (3 * t[i + 1] * t[i + 2] - t[i]) by
    rw [vieta_eq]
    exact lt_of_mul_lt_mul_left this (t.is_pos.le)
  ac_change t[i] * t[i] < 3 * t[i] * t[i + 1] * t[i + 2] - t[i] * t[i]
  · ring
  rw [←t.solves]; ring_nf
```

```
  apply lt_add_of_le_of_pos
  · apply pow_le_pow_left₀ t.is_pos.le h'
  · apply Int.pow_pos t.is_pos
```

We next wish to prove that a Markoff-Vieta involution applied to the maximum coordinate will always increase that coordinate, as long as there is a unique maximum coordinate. However, we will find it useful to instead prove a slightly stronger statement first:

Lemma 4.4. *Let $(a, b, c) \in \mathcal{M}$ be a Markoff triple, and without loss of generality let $a > b > c$. Then, $3bc - a < b$.*

Once again, we will define this theorem with arbitrary indices.

```
theorem z_vieta_le_others_of_max_of_fundamental_domain
  {t: MarkoffTriplePos 3} {i: Fin 3}
  {p: t[i] > t[i + 1] ∧ t[i] > t[i + 2]}:
    (t.vieta i)[i] ≤ t[i + 1] ∨ (t.vieta i)[i] ≤ t[i + 2] := by
  wlog h: t[i + 1] >= t[i + 2]
  · simpa [Equiv.swap_apply_of_ne_of_ne, vieta_eq] using or_comm.mp <| this
      (t := t.perm <| swap (i + 1) (i + 2)) (i := i)
      (p := by simp [Equiv.swap_apply_of_ne_of_ne, p])
      (by push_neg at h; simpa using h.le)
  let f (x: ℤ) := (x - t[i]) * (x - (t.vieta i)[i])
  have : f t[i + 1] <= 0 := by
   /- proof found in Appendix -/
  if h': t[i] = 2 then
    rw [vieta_eq, h']
    rw [show t[i + 1] = 1 by have : t[i + 1] > 0 := t.is_pos; omega]
    rw [show t[i + 2] = 1 by have : t[i + 2] > 0 := t.is_pos; omega]
    omega
  else
    suffices f_b_ne_zero: f t[i + 1] ≠ 0 by
      replace this := lt_of_le_of_ne this f_b_ne_zero
      have this := Int.pos_of_mul_neg_right this (by simpa using p.left)
      apply Or.inl
      simpa using this.le
    apply Int.mul_ne_zero
    · exact sub_ne_zero.mpr p.left.ne
    · /- proof found in Appendix -/
```

```
theorem z_vieta_lt_self_of_max_of_fundamental_domain
  {t: MarkoffTriplePos 3} {i: Fin 3}
  (h: t[i] > t[i + 1] ∧ t[i] > t[i + 2]):
    (t.vieta i)[i] < t[i] :=
  match z_vieta_le_others_of_max_of_fundamental_domain (p := h) with
    | .inl h' => h'.trans_lt h.left
    | .inr h' => h'.trans_lt h.right
```

4.3 A recursive principle for walks in the Markoff graph

In this section, we define our recursor-like function for Markoff triples which will encode the process of "walking" from a Markoff triple back to the root. As an application of this function, we will give a proof of Theorem 4.1.

In Mathlib, simple graphs are represented as a symmetric, antireflexive relation on a type of vertices.

```
def graph: SimpleGraph (MarkoffTriplePos 3) := {
  Adj x y := ∃i: Fin 3, x.vieta i = y
  symm x y := by
    intro ⟨i, ⟩h; use i
    apply congrArg (flip vieta i) at h
    erw [flip, vieta_is_involution] at h
    exact h.symm
  loopless := ⟨fun x => by
    push_neg; intro i
    suffices (x.vieta i)[i] ≠ x[i] by aesop
    if h: x[i] > x[i + 1] ∧ x[i] > x[i + 2] then
      apply ne_of_lt
      exact z_vieta_lt_self_of_max_of_fundamental_domain h
    else
      set_option push_neg.use_distrib true in push_neg at h
      apply ne_of_gt
      exact z_vieta_gt_self_of_nonmax_of_fundamental_domain (p := h)
  ⟩
}
```

We are now ready to construct our walk from any node back to the root. We do so recursively, and provide a proof for the termination side goal.

```
def walkToRoot (t: MarkoffTriplePos 3):
    SimpleGraph.Walk graph t MarkoffGraphZk3.root :=
  if h: t = MarkoffGraphZk3.root then
   -- t and root are propositionally, but not definitionally, equal and therefore
   -- .nil by itself will not typecheck as ‘Walk root root‘.
   -- We therefore use the substitution operator ▸ to convert.
    h ▸ .nil
  else
    let i := max_of_ne_root h
    .cons (v := t.vieta i) (by use i) (MarkoffGraphZk3.walkToRoot <| t.vieta i)
termination_by maxNat t
decreasing_by
   -- goal is:
   --    maxNat (t.vieta (max_of_ne_root h)) < maxNat t
   /- proof found in Appendix -/
```

With this construction, we can prove that the Markoff graph over the positive integers is connected by using the convenient Mathlib lemma that a graph is connected if and only if there is a vertex reachable by all vertices in the graph.

```
theorem is_connected: graph.Connected :=
  graph.connected_iff_exists_forall_reachable.mpr <| by
    use MarkoffGraphZk3.root
    intro w
    use (walkToRoot w).reverse
```

We can now define our recursor for the Markoff graph over positive integers. Our recursor will accept any node as the “start node”, but unlike for the primitive recursor for `SimpleGraph.Walk`, this start node will remain fixed across the motive and all minor premises.

```
def markoff_graph_rec {t s: MarkoffTriplePos 3} (walk: SimpleGraph.Walk graph t s)
  {motive: MarkoffTriplePos 3 → Sort u} (start: motive s)
  (step: {x y: MarkoffTriplePos 3} → graph.Adj y x → motive x → motive y):
    motive t :=
  match _: walk with
```

```
    | @SimpleGraph.Walk.nil _ _ u =>
      (by aesop: u = s) ▸ start
    | @SimpleGraph.Walk.cons _ _ _ _ u h p =>
      have : u = s := by aesop
      step h (markoff_graph_rec (this ▸ p) start step)
termination_by walk.length
```

Finally, we demonstrate our recursor by proving Theorem 4.1, that the coordinates of a Markoff triple are pairwise coprime.

```
theorem coords_coprime (t: MarkoffTriplePos 3) {i j: Fin 3} (i_ne_j: i ≠ j):
    Int.gcd t[i] t[j] = 1 :=
  markoff_graph_rec (motive := fun x => Int.gcd x[i] x[j] = 1) (walkToRoot t)
    (start := by simp)
    (step := by
      intro x y e h
      wlog hj: j = i + 1
      · have h': j = i + 2 := by omega
        have := this t (i := i + 2) (j := i)
          (by fin_cases i <;> decide) e
          (by simpa [←h', Int.gcd_comm] using h)
          (by simp [add_assoc])
        simpa [←h', Int.gcd_comm] using this
      subst hj
      obtain ⟨k, ⟩hk := e
      apply (Int.gcd_eq_iff (g := 1)).mpr
      refine ⟨by simp, by simp, ?⟩_
      intro c ₁ch ₂ch
      apply (Int.gcd_eq_iff (g := 1)).mp at h
      apply And.right ∘ And.right at h
      specialize h c
      if k = i ∨ k = i + 1 then
        rename_i this; cases this
        all_goals
          rename_i this; subst this
          rw [←hk] at h
```

```
        rw [vieta_eq, vieta_eq_of_ne _ (by simp)] at h
      · apply h ?_ ₂ch
        apply Int.dvd_sub ?_ ₁ch
        apply Int.dvd_mul_of_dvd_left
        exact Int.dvd_mul_of_dvd_right ₂ch
      · apply h ₁ch ?_
        apply Int.dvd_sub ?_ ₂ch
        apply Int.dvd_mul_of_dvd_right
        simpa [add_assoc] using ₁ch
    else
      have : k = i + 2 := (by omega); subst this
      rw [←hk] at h
      repeat rw [vieta_eq_of_ne _ (by simp)] at h
      exact h ₁ch ₂ch
  )
```

4.4 Decidability and Computability

The strong approximation conjecture is equivalent to showing `SimpleGraph.Connected` for all primes and can be expressed as

```
def strongApproximation: Prop := (graph p).Connected
```

where the universal quantifier is implicit. In [15], approximation methods are used to show that the conjecture is true for a large class of primes, namely those larger than 10^{400}. This method cannot, however, be used by itself to construct a walk from any vertex back to $(1, 1, 1)$.

In Lean's type theory, the law of the excluded middle is provable from the axiom of choice, definitional proof irrelevance, and propositional extensionality. As a result, every proposition is definitionaly equal to either `True` or `False`. However, since propositions cannot generally be eliminated into sorts with computational value, propositions cannot be used by themselves as the predicate for `if-else` blocks. Instead, we must lift the data of the truthiness of a proposition to `Sort 1` or higher, using the `Decidable` class.

```
class inductive Decidable (p: Prop) where
  | isTrue (h: p): Decidable p
  | isFalse (h: Not p): Decidable p
```

An instance of `Decidable p` carries the data of whether `p` is `True` or `False`, as well as a proof of that fact. (Actually, using the axiom of choice, all propositions are decidable; however, these `Decidable` instances are marked as `noncomputable` by the compiler, as described in Section 1.2. For the remainder of this chapter, we will not consider `noncomputable` instances of `Decidable`.)

The proposition `SimpleGraph.Connected` already contains a (computable) `Decidable` instance whenever the vertex set is finite. We may therefore give `(graph p).Connected` an instance of `Decidable` with very little additional work. Additionally, this instance will compile into a runnable program and can be used to test whether Markoff graphs are connected. However, it is a very naive algorithm, suitable for any finite graph at all. Instances on Markoff graphs with specific values of `p` may be added one at a time using suitable methods for specific primes.

Chapter 5

Tree decompositions of Markoff graphs

5.1 Expander Families and the Cheeger Constant

We have seen that the Markoff graphs $\mathcal{M}_p$ form a family of graphs with $\mathcal{O}(p^2)$ vertices as $p \to \infty$. In [5], the authors conjecture that this family is in fact an *expander* family. Expander families of graphs are those which maintain good connective properties while remaining sparse. We quantify this notion for a graph G by comparing the size of each subset of G to the size of its boundary.

Definition 5.1. Let G be a graph and $S \subseteq V(G)$ a subset of vertices. The *boundary of* S is defined as the set of edges which connect a vertex in S to a vertex outside of S; in particular,

$$\partial S := \{\{e, v\} \in E(G) | e \in S, v \notin S\}.$$

Definition 5.2. Let $\alpha \in \mathbb{R}$ with $0 < \alpha \leq 1/2$. The *Cheeger constant* $h_\alpha(G)$ of a graph G with n vertices is

$$h_\alpha(G) := \min_{\substack{S \subseteq V(G) \\ 0 < |S| \leq \alpha |V(G)|}} \frac{|\partial S|}{|S|}.$$

If $h_\alpha(G) > 0$, then G is called an $(\alpha, h(G))$-*expander*. A family of graphs $\{G_i\}_{i \in \mathbb{N}}$ is an α-*expander family* if

$$\inf_{i \in \mathbb{N}} h_\alpha(G_i) > 0.$$

A family of graphs is called an *expander family* when it is an α-expander family for some $\alpha > 0$.

Disconnected graphs have Cheeger constant 0 since $\partial S = \emptyset$ whenever S is a connected component. For the rest of this section, we will consider only connected components of the Markoff graph; if the Strong Approximation Conjecture is true, this will be all of $\mathcal{G}_p^\times$ for all primes p.

Random walks on the Markoff graph have appeared in the literature as a way to search for paths between vertices of $\mathcal{G}_p^\times$ (as in [38]) and as a way to search for lifts of Markoff triples (as in [4]). In order to find desired paths and vertices in Markoff graphs via random walks, we hope that the probability of ending at any vertex rapidly approaches the uniform distribution as the length of the walk $\to \infty$. The rate of convergence for a k-regular graph G is controlled essentially by the largest eigenvalue magnitude $|\lambda| \neq \pm k$ of the adjacency matrix of G. We have the following theorem of Alon and Boppana.

Theorem 5.3 ([28, Corollary 4.5.5])**.** *Fix $k \in \mathbb{N}$ and an index set $I \subseteq \mathbb{N}$. Let $\{X_{k,n}\}_{n \in I}$ be a family of k-regular connected graphs where $X_{k,n}$ has n vertices. Let $\delta_{X_{k,n}}$ denote the adjacency matrix of $X_{k,n}$. Then,*

$$\liminf_{n\to\infty} \max\{|\lambda| \mid \lambda \text{ an eigenvalue of } \delta_{k,n}, \lambda \neq \pm k\} \geq 2\sqrt{k-1}.$$

Families of graphs which achieve the best possible bound on its second-largest eigenvalue of $2\sqrt{k-1}$ are called *Ramanujan*.

In [13], de Courcey-Ireland and Lee proved that the Markoff graphs adhere to the Kesten-McKay law for eigenvalue distributions of random graphs. They also conjectured that the spectral gap remains bounded away from zero for all Markoff graphs, and furthermore approach $2\sqrt{k-1}$ along $p \to \infty$ when $p \equiv 3 \pmod 4$. Implicit in this conjecture is the idea that the family of Markoff graphs mod p for $p \equiv 3$ mod 4 are Ramanujan.

Conjecture 5.4 ([13, Conjecture 1.1])**.** *The family of Markoff graphs $\{\mathcal{M}_p^\times\}_{p \text{ prime}}$ is an expander family. Furthermore, $\{\mathcal{M}_p^\times\}_{\substack{p \text{ prime} \\ p \equiv 3 \pmod 4}}$ is Ramanujan.*

5.2 Tree decompositions and treewidth

We now present the notion of treewidth and pathwidth which are dual (in a sense we will make precise) to expander families and Cheeger constants.

Many problems in graph theory are solved using a dynamic programming technique, where solutions to subproblems are quickly found and then synthesized into solutions to larger subproblems. The following illustrative example is adapted from [16, Section 11.4]. In the k-Colorability problem (for fixed k), we

are given a graph G and asked whether G is vertex colorable using k colors; that is, whether there is an assignment $V(G) \to C$ for C a set of k colors such that no two adjacent vertices receive the same color. Naively, we could iterate through all $k^{|V(G)|}$ colorings and check whether any of them are valid.

Imagine for a moment that we are given two sets of vertices $A, B \subset V(G)$ with the property that there are no edges between $A \setminus B$ and $B \setminus A$. We can check the $k^{|A|}$, $k^{|B|}$, and $k^{|A \cap B|}$ colorings for $G[A]$, $G[B]$, and $G[A \cap B]$, respectively. Now, a coloring for $A \cup B$ can be found by filtering the colorings for $G[A \cap B]$ and retaining only those $f : A \cap B \to C$ for which there are $g_A : A \to C$ and $g_B : B \to C$ such that f coincides with both g_A and g_B.

This process can be repeated recursively as long as we are given a family of subsets $\beta : T \to \mathcal{P}(V(G))$ indexed by the nodes $t \in T$ for some tree T satisfying a small number of properties. Beginning with the leaves of T, call one of them ℓ, calculate the set S_ℓ of all colorings of $G[\beta(\ell)]$. If $t \in T$ and we have already calculated the sets $S_{t'}$ for all children t' of t, we can run the operation described in the previous paragraph to determine S_t as long as $\beta(t_1) \cap \beta(t_2) \subseteq \beta(t)$ for all pairs of children t_1, t_2 of t. As long as every edge (a, b) appears inside some set as $(a, b) \subset \beta(s)$, this process will result in a coloring on the whole graph (if there is an isolated vertex v, just give it any color at all).

If we add one more restriction to our family β, then a trick emerges. Root our tree T at some node r and let $s \prec t$ whenever s is a descendent of t from r. Assume that every vertex v satisfies $v \in \beta(t_2)$ whenever $v \in \beta(t_1) \cap \beta(t_3)$ and $t_1 \prec t_2 \prec t_3$. Now, the trick is that we are allowed to "forget" the color of a vertex $v \in \beta(t) \setminus \beta(t')$ in our calculation of S_t. This is because if $v \in \beta(t) \setminus \beta(t')$, then v can not appear in any $\beta(s)$ for $t \prec s$. Loosely speaking, the colorings of vertices appearing in the sets A which are "above" $\beta(t)$ in the tree can never interact with the colorings. All of the colorings involving only v and its neighbors have already been considered, and no edge adjacent to v will appear in any of the sets $\beta(s)$ for which we must still compute colorings.

If every set $\beta(t)$ has size bounded by w, then our algorithm will run in time exponential in w and linear in $|T|$. Unfortunately, determining a suitable tree T is NP-hard and is generally extremely difficult to feasibly compute. The data of the tree T is called a *tree decomposition*. Roughly, a tree decomposition of a graph G encodes order in which subsets of vertices must be considered in a dynamic programming algorithm like the one just described.

Definition 5.5. Let G be a graph. A *tree decomposition* $\mathcal{T} = (T, \beta)$ of G is a tuple of a tree T (called the "shape" of the decomposition) and a family of subsets $\beta : V(T) \to 2^{V(G)}$ (called the "bags") which satisfies the following.

- *Vertex coverage*: For every $v \in V(G)$, there is a $t \in T$ such that $v \in \beta(t)$.

- *Edge coverage*: For every $\{u, v\} \in E(G)$, there is a $t \in T$ such that $\{u, v\} \subseteq \beta(t)$.
- *Coherence*: For every $v \in V(G)$, the set of bags in which v appears is connected in T. That is, $\{t \in T \mid v \in \beta(t)\}$ is a subtree of T.

To help avoid confusion, we will always refer to the elements of $V(G)$ as *vertices* of the graph G, and to the elements of $V(T)$ as the *nodes* of the tree T.

As an example of the helpfulness of a tree decomposition, consider any edge $e = \{t_1, t_2\} \in E(T)$. Call the set of vertices in $\beta(t_1) \cap \beta(t_2)$ the *adhesion* of the edge e. Now, the edge e gives a tripartition of vertices $(A \setminus B, A \cap B, B \setminus A)$, where A is the set of all vertices which appear in the bags on side of e but not in the adhesion of e, and B defined similarly for the other side of e. Note that this is exactly the data we used in our outline of the solution the Maximum Independent Set problem; the tree decomposition of G then describes exactly the order of subsets for which we should solve the subproblems.

The most important measure of a tree decomposition's size is the sizes of its bags, which also bounds the sizes of its adhesions.

Definition 5.6. Let $\mathcal{T} = (T, \beta)$ be a tree decomposition of G. The *width* of $\mathcal{T}$ is

$$\text{width}(\mathcal{T}) = \max_{t \in T} |\beta(t)| - 1.$$

The *treewidth* of G is then the smallest possible width across all tree decompositions.

Remark 5.7. Every graph G has a treewdith since every graph has at least one decomposition, in particular the decomposition with one node and a bag holding the entire vertex set: $(\{*\}, * \mapsto V(G))$. Accordingly, every graph G trivially has treewidth bounded by $|V(G)| - 1$.

If G is a graph with two connected components C_1 and C_2 which have tree decompositions $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively, then a tree decomposition of G can be formed by taking a tree with one node whose bag is empty and appending C_1 and C_2 as children. This tree decomposition has width $\max(\text{width}(C_1), \text{width}(C_2))$. We will therefore generally consider only the connected components of G.

Lemma 5.8. *A connected graph G has treewidth 1 if and only if it is a tree.*

Proof. Define a tree decomposition $\mathcal{T}$ for G by

$$T = (E(G), \{\{e_1, e_2\} \subseteq E(G) \mid e_1 \text{ and } e_2 \text{ are adjacent}\}$$

and bags $\beta(e) = e$. By inspection, all of the properties of Definition 5.5 are satisfied and each bag has size 2. Thus, $\mathcal{T}$ is a tree decomposition with width 1. $\square$

In Definition 5.5, we required the shape of the decomposition to be a tree. We may drop this requirement to obtain decompositions of other shapes. For example, if we insist that T is instead a path (T is connected and every vertex has order 1 or 2), then $\mathcal{T}$ is called a *path decomposition*. Since every path decomposition is also a tree decomposition, we trivially have $\text{pathwidth}(G) \geq \text{treewidth}(G)$ for every graph G.

We now turn to an instructive example of a tree decomposition. Let G be a graph made up of a single cycle, i.e. G is isomorphic to $G = (V, E)$ where $V = \mathbb{Z}/n\mathbb{Z}$ and $E = \{(i, i+1)\}_{i=1}^{n}$. Then, G has a path decomposition with nodes indexed by $(\mathbb{Z}/n\mathbb{Z}) \setminus \{0\}$ and bags given by

$$\beta(i) = \{n\} \cup \{i, i+1\}.$$

More plainly, we form bags of adjacent edges $\{1, 2\}, \{2, 3\}, \ldots, \{n-2, n-1\}$, and then to each bag we add vertex n. This path decomposition has width 2 and also serves as a tree decomposition. By Lemma 5.8, there is no tree decomposition of width 1, hence there is no path decomposition of width 1. We conclude that the pathwidth (hence treewidth) of every cycle graph is 2.

A *minor* of a graph G is a graph obtained from G by a sequence of vertex deletions and edge contractions. For a family of graphs $\mathcal{F}$, if H is not a minor of any graph in $\mathcal{F}$, then we say H is a *forbidden minor* of $\mathcal{F}$.

Trees may be characterized as graphs containing no K_3 minor where K_3 is the complete graph on three vertices. We may therefore reframe the previous result as the following lemma.

Lemma 5.9. *A graph has treewidth 1 if and only if it contains no K_3 minor.*

More generally, treewidth may be characterized in terms of forbidden minors. Consider a graph G with tree decomposition $\mathcal{T}$. Let H be a minor of G obtained by some sequence of vertex deletions and edge contractions. A tree decomposition $\mathcal{T}'$ may be obtained for H by removing from each bag all deleted vertices and identifying all vertices identified in G via edge contractions. Since these operations only ever decrease the size of the bags in $\mathcal{T}$, we conclude that $\text{treewidth}(H) \leq \text{treewidth}(G)$. We therefore conclude that, for every $k > 0$, the family $\mathcal{F}_k$ of graphs of treewidth bounded above by k is minor-closed, and by the Robertson and Seymour graph minors theorem [35], $\mathcal{F}_k$ is completely determined by a finite set of forbidden minors. Unfortunately, determining the set of forbidden minors for particular k is extremely challenging and has only been done for small values of k, but the number of obstructions must grow as an exponential function of $\sqrt{k}$, see [34].

A graph is called *chordal* if it contains no chordless cycles of length 3, and a chordal graph $\widehat{G}$ is called a *chordal completion* of G if G is a subgraph of $\widehat{G}$ with the same vertex set. The previous example of the cycle graph illustrates the general notion that chordless cycles are not themselves obstructions to tree

decompositions with small width, made precise by the following folklore theorem which appears in [9].

Theorem 5.10 ([9, Theorem 3])**.** *G has treewidth at most k if and only if it is the subgraph of a chordal graph with maximum clique size $k+1$.*

5.3 Decompositions of Markoff Graphs

In this section, we give upper bounds on the pathwidth and treewidth of the Markoff graphs $\mathcal{M}_p^\times$. Before we do so, we note that we expect tree decompositions for Markoff graphs to be particularly nasty due to the following theorem of Grohe and Marx and in light of Section 5.1.

Theorem 5.11. *[21, Theorem 3] For every class $\mathcal{C}$ of graphs and every α with $0 < \alpha \leq 1/2$, the following statements are equivalent:*

1. *There is a constant $\beta > 0$ such that treewidth$(G) \geq \beta|V(G)|$ for every $G \in \mathcal{C}$.*
2. *There are constants $\gamma_1, \gamma_2 > 0$ such that every graph $G \in \mathcal{C}$ has a subgraph H such that $|V(H)| \geq \gamma_1|V(G)|$ and $h_\alpha(H) \geq \gamma_2$.*

In particular, expander families are not bounded in their treewidth, and in fact the treewidths grow linearly with respect to the size of the vertex set. These graphs have treewidths which are "as bad as possible" since treewidths can only grow linearly according to Remark 5.7. Accordingly we have the following conjecture which is a necessary condition for Conjecture 5.4.

Conjecture 5.12. *The treewidths of Markoff graphs $\mathcal{M}_p^\times$ grow as $\mathcal{O}(p^2)$.*

With the bad news dispensed with, let us now turn towards bounding the treewidth of $\mathcal{M}_p^\times$ from above. Our constructions will rely on the assignments of the three Vieta involutions to each edge of the Markoff graph. We therefore begin with the following definition.

Definition 5.13. A graph G is *k-edge colorable* if there is a finite set C of size $|C| = k$ (the "colors") and an assignment $E(G) \to C$ of colors to each edge such that adjacent edges are assigned different colors.

Lemma 5.14. *The Markoff graphs $\mathcal{M}_p^\times$ (and also $\mathcal{M}^\times$) are 3-edge colorable exactly corresponding to the three Vieta involutions.*

Proof. Since V_i for $i \in [3]$ are all involutions, every vertex $v \in \mathcal{M}_p^\times$ (or $v \in \mathcal{M}^\times$) has no more than 1 adjacent edge arising from V_i. □

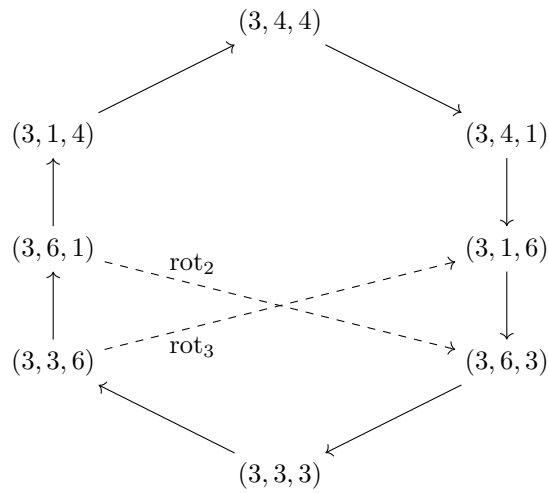


Figure 5.1: A subgraph of $\mathcal{B}_7^\times$ containing the rot_1 cycle at $(3,3,3)$. The solid edges are rot_1 rotations. Note that there is actually a second pair of crossing chords above the two depicted.

If $c_1 \neq c_2$ are distinct colors, then a (c_1, c_2)-bichromatic walk is a walk where the edges alternate between the two colors. If $C = (v_1, \ldots, v_n, v_1)$ is a cycle of a graph, then a *chord* of C is an edge $v_i, v_j \in C$ such that $|i - j| \neq 1$.

In Section 2.3, we introduced the rotation maps as a permutaton composed with a Vieta involution. Repeated applications of a rotation maps are essentially alternating applications of Vieta involutions plus some coordinate permutations. For example,

$$\mathrm{rot}_1^n(a,b,c) = \begin{cases} (V_3 \circ V_2)^{n/2} & n \text{ is even,} \\ \tau_{1,2} \circ V_2 \circ (V_3 \circ V_2)^{(n-1)/2} & n \text{ is odd.} \end{cases}$$

In the BGS graph $\mathcal{B}_p^\times$ (Definition 2.12), the cycles formed by repeated applications of a rotation map played an important role in finding paths between Markoff triples. Generally, these cycles are not chordless.

Example 5.15. The BGS graph modulo 7 contains a cycle with four chords, two of which are depicted in Figure 5.1.

More generally, let $(a,a,c) \in \mathcal{M}_p$ with $a, c \neq 0$. Then,

$$\begin{aligned} \mathrm{rot}_1(a,a,c) &= (a, c, ac - a), \\ \mathrm{rot}_3(a,a,c) &= (a, ac - a, c). \end{aligned} \tag{5.1}$$

If the rot_1 cycle containing (a,a,c) also contains (a,b,b) for any $b \in \mathbb{F}_p$ or (a,b,d) where $a^2 + d^2 = b^2$ (see

Lemma 2.8), then both of the triples of (5.1) will lie in the same cycle. Accordingly, there will be a chord in that cycle. Furthermore,

$$\mathrm{rot}_2(\mathrm{rot}_1(a,a,c)) = (a,c,a),$$
$$\mathrm{rot}_1(\mathrm{rot}_3(a,a,c)) = (a,c,a),$$

hence there is a second, crossing chord in this cycle.

However, the corresponding cycles in the Markoff graph $\mathcal{G}_p^\times$ formed by alternating Vieta involutions (i.e., bichromatic cycles) do not contain these chords. Now, for $i \in [3]$, there are either $3(p-3)/2)$ or $3(p-5)/2$ maximal (V_{i+1}, V_{i+2})-bichromatic paths, corresponding exactly to the paths which start and end at a vertex fixed by V_i (see Lemma 2.11 for the count of the number of these fixed points). We therefore put the following lemma in terms of these maximal paths instead of in terms of the cycles only.

Lemma 5.16. *Let $P = (v_1, \ldots, v_n)$ be a bichromatic path maximal under set inclusion in $\mathcal{G}_p^\times$. Then v_i is not adjacent to v_j for any $|i-j| > 1$.*

Proof. Without loss of generality let the path P be made by alternating V_2, V_3. If there were an edge between nonconsecutive vertices in this path, it must be a V_1 edge. Then we have two vertices (a,b,c) and $(bc-a,b,c)$ in P with $bc-a$. But this is absurd since V_2 and V_3 leave the first coordinate fixed. □

Theorem 5.17. *Let G be a graph with n vertices, 3-edge colorable such that no bichromatic cycle contains a chord, and let M be the length of the longest bichromatic walk. Then, G has a path decomposition of width no more than*

$$\frac{n}{4} + M + \mathcal{O}(\log_3 n).$$

We will prove this by describing an algorithm constructing the desired path decomposition in two phases. In the first phase, we will construct a path decomposition of width up to $n/4$ by extracting chordless cycles and placing their boundary into every subsequent bag. In the second phase, we appeal to the following result by Fomin and Høie describing path decompositions of 3-regular graphs.

Lemma 5.18 ([17, Lemma 4])**.** *Let G be a graph on n vertices and with maximum vertex degree at most 3. Then for any vertex subset $X \subseteq V(G)$ there is a path decomposition $\mathcal{P} = (X_1, X_2, \ldots, X_r)$ of G of width at most*

$$\max(|X|, \lfloor n/3 \rfloor + 1) + (2/3)\log_3 n + 1$$

and such that $X = X_r$.

Proof of Lemma 5.17. Initialize $S_0, X_0 := \emptyset$ and $\mathcal{P}$ to be the empty path decomposition of the empty graph. (We will think of the sets S_t as being the vertices for which we are "done" and have no more neighbors to add to the decomposition, while X_t is the set of vertices which have at least one neighbor, but possibly not all of them, in our decomposition.) At each time step t, we will maintain the following invariants.

1. $N(S_t) \subseteq X_t \subset V(G)$.

2. $|X_t| \leq |S_t|$

3. $\mathcal{P}_t$ is a path decomposition of $G[S_t \cup X_t]$.

4. $\mathcal{P}_t$ begins with bag X_t.

5. $\mathcal{P}_t$ has width no more than $|X_t| + 2$.

At step $t \in \mathbb{N}$, consider $G_t = G \setminus (S_t \cup W_t)$. Let W be a bichromatic walk in G_t, maximal under set inclusion; such a walk must exist since X_t is a proper subset of G. Let $S_{t+1} := S_t \cup W$ and $X_{t+1} := X_t \cup N(W)$. Invariant 1 is immediately satisfied by this construction. Since W is maximal under set inclusion, each vertex $w \in W$ has at most one neighbor outside of W; namely, the neighbor whose adjacency color is excluded from the walk. Therefore, $|W| \leq |N(W)|$, and invariant 2 is preserved. Now, $G[W]$ is a chordless cycle and thus has a path decomposition $(\beta_1, \ldots, \beta_k)$ of width no more than 2. The desired path decomposition $\mathcal{P}_{t+1}$ is given by

$$(X_{t+1}, (\beta_1 \cup X_{t+1}), \ldots, (\beta_k \cup X_{t+1})) :: \mathcal{P}_t$$

where $L_1 :: L_2$ denotes the concatenation of L_1 and L_2. By inspection, this path decomposition satisfies invariants 3, 4, and 5.

Terminate this process at the first time t_{stop} such that

$$X_{t_{\text{stop}}} + M > \frac{|V(G)| - |S_{t_{\text{stop}}}|}{3}.$$

At this point, we may use the algorithm of [17, Lemma 4] to obtain a path decomposition $\mathcal{P}'$ of $G \setminus S_{t_{\text{stop}}}$ with ending bag $X_{t_{\text{stop}}}$. This path decomposition has width no more than

$$\begin{aligned}\text{width}(\mathcal{P}') &\leq \max\left(|X_{t_{\text{stop}}}| + M, \left\lfloor \frac{n - |S_{t_{\text{stop}}}|}{3} \right\rfloor + 1\right) + (2/3)\log_3(n - |S_{t_{\text{stop}}}|) + 1 \\ &\leq |X_{t_{\text{stop}}}| + M + \mathcal{O}(\log_3(n)).\end{aligned}$$

We finally note that $|X_{t+1}| - |X_t| \leq M$. Combining this observation with our description of t_{stop} yields

$$|X_{t_{\text{stop}}}| \leq \frac{n - |S_{t_{\text{stop}}}|}{3} \leq \frac{n - |X_{t_{\text{stop}}}|}{3}$$

and therefore

$$|X_{t_{\text{stop}}}| \leq \frac{n}{4}.$$

Our final path decomposition of G is then $\mathcal{P}' :: \mathcal{P}_{t_{\text{stop}}}$. □

Corallary 5.19. *The Markoff graphs $\mathcal{M}_p$ have pathwidth bounded above by $p^2/4 + \mathcal{O}(p^{1+\epsilon})$.*

The algorithm of Lemma 5.17 applies not only to Markoff graphs but also to their subgraphs. The separators that we find by identifying bichromatic walks are, unfortunately, neither balanced nor "small" – in fact, we expect them to be almost the same size as the paths they are separating from the rest of the graph.

Unfortunately, finding small balanced seperators is a difficult problem with few results on general classes of graphs. In [33], Monien and Preis give an asymptotic upper bound of $(n/6 + \epsilon)|V|$ for bisection widths of 3-regular graphs. We can exploit this result by changing the stopping time of the algorithm in the proof of Lemma 5.17 to stop when our path decomposition reaches width $n/6$. This gives us a marginal improvement in the asympototic pathwidth of Markoff graphs.

Theorem 5.20. *For all $\epsilon > 0$, there is an integer n_ϵ such that, for all $p > n_\epsilon$, the pathwidth of $\mathcal{G}_p^\times$ is at most $(1/7 + \epsilon)p^2 + \mathcal{O}(p^{1+\epsilon})$.*

Proof. Replicate the proof of Lemma 5.17 but with stopping time defined defined to be the last step such that

$$X_{t_{\text{stop}}} + M > \frac{n - |S_{t_{\text{stop}}}|}{6}$$

Now, the width of the path decomposition $\mathcal{P}_{t_{\text{stop}}}$ is bounded by $|X_{t_{\text{stop}}}| \leq n/7$ and

$$|V(G) \setminus S_{t_{\text{stop}}}| = (6n/7) + \mathcal{O}(M).$$

For sufficiently large n, the algorithm of [33, Theorem 2] then gives a path decomposition of $G_{t_{\text{stop}}} = G \setminus |S_{t_{\text{stop}}}|$ of width bounded by

$$\left((1/6) + \frac{7\epsilon}{6}\right)|V(G_{t_{\text{stop}}})| = (1/7 + \epsilon)n.$$

Combining the path decompositions yields a path decomposition of width $p^2/7 + \mathcal{O}(p^{1+\epsilon})$.

□

5.4 Future work

In this section, we outline potential research towards Conjecture 5.12. Our plan is to construct tree decompositions of the Markoff graph in two phases. First, we will find a tree decomposition on a contraction of the Markoff graph with roughly $\mathcal{O}(p^{1+\epsilon})$ vertices. Then, we will attempt to "lift" this decomposition to a tree decomposition on the whole Markoff graph. If both steps produce tree decompositions of sufficiently large width, then the conjecture will be proven. Otherwise, if either step produces a tree decomposition of sublinear treewidth, then the conjecture will be refuted.

Let $i \in [3]$ correspond to one of the three Vieta involutions and define G_p^i as the graph with vertex set $V(G_p^i) = V(\mathcal{M}_p^\times)/\sim_i$ where $t_1 \sim_i t_2$ iff there is a (possibly empty so $\sim_i$ will be reflexive) (V_{i+1}, V_{i+2})-bichromatic path containing both t_1 and t_2. In short, we are contracting all bichromatic paths into a single vertex. The calculation for the number of bichromatic paths is essentially the same as in Section 3.3. In short, each $\chi \in \mathbb{F}_{p^2}^\times \setminus \{\pm 1\}$ with $|\chi| \in \mathcal{D}(p \pm 1)$ gives $(p \pm 1)/|\chi|$ bichromatic paths, up to a factor of either 1/2 or 2. As in Chapter 3, the total number of these bichromatic orbits will be $\mathcal{O}(p^{1+\epsilon})$.

In $\mathcal{G}_p^i$, a vertex corresponding to a bichromatic path of length k will have exactly k neighbors, corresponding to the k V_i-neighbors of the vertices of the path in $\mathcal{G}_p$. At first it may seem that some of these neighbors will be identified, but that is not the case: let (x, y, z) be adjacent to two vertices (a, b_1, c_1) and (a, b_2, c_2) lying on the same (V_2, V_3)-bichromatic path. Since (x, y, z) must be adjacent to both other vertices via a V_1 Vieta involution, $y = b_1 = b_2$ and $z = b_1 = b_2$. But (a, b_1, c_1) and (a, b_2, c_2) are actually the same vertex.

Now, imagine that a helpful friend has given us a tree decomposition $\mathcal{T}$ of $\mathcal{G}_p^i$. We may naively expand this decomposition into a decomposition $\mathcal{T}'$ of $\mathcal{G}_p^\times$ by replacing in every bag of $\mathcal{T}$ each vertex $v \in V(\mathcal{G}_p^\times)$ with the entire bichromatic path from which it came. Since these paths are of length no more than $2p$, we have $\text{treewidth}(\mathcal{T}') \leq 2p \cdot \text{treewidth}(\mathcal{T})$. In fact, this is the worst possible way to pull back the tree decomposition $\mathcal{T}'$. Therefore, if we were to discover that the treewidth of $\mathcal{G}_p^i$ grows sublinearly with respect to p, the treewidths of $\mathcal{G}_p^\times$ would have to be less than $\mathcal{O}(p^2)$, and Conjecture 5.12 would be refuted.

On the other hand, if we can prove that $\text{treewidth}(\mathcal{G}_p^i)$ is $\mathcal{O}(f)$ and pulling back these tree decompositions always results in the bags growing by a factor of $\mathcal{O}(g)$ where $\mathcal{O}(fg) = \mathcal{O}(p^2)$, then Conjecture 5.12 will be proven. However, it is not yet clear to us where the extra factor of p^ϵ should go. Since the treewidths of $\mathcal{G}_p^\times$ can grow no faster than $\mathcal{O}(p^2)$, either f is only $\mathcal{O}(p)$ instead of $\mathcal{O}(p^{1+\epsilon})$, or g grows as $\mathcal{O}(p^{1-\epsilon})$, or something similarily slightly sublinear.

We will conclude this section with a brief, heuristic description of one possible algorithm to perform the pullback from $\mathcal{T}$ to $\mathcal{T}'$. However, we are doubtful it will be of much practical use; the graphs on which we wish to run this algorithm are simply too large to compute the necessary tree decomposition $\mathcal{T}$ in the first

place. The description here will hopefully be of use in illustrating the potential challenges which will have to be overcome in future research.

For any vertex $v \in \mathcal{G}_p^i$, let $\mathcal{T}[v] = (T[v], \beta|_{T[v]})$ denote the tree decomposition with shape

$$T[v] = \{t \in T \mid v \in \beta(t)$$

and bag function β restricted to these nodes. Root $T[v]$ at some node r and consider the partial order $\prec$ where $n_1 \prec n_2$ if n_1 is an ancestor of n_2, i.e. if n_1 is nearer to r than n_2 and there is a path in $T[v]$ of the form $r, \ldots, n_1, \ldots, n_2$.

Let $v_1, \ldots, v_k$ be the vertices in the bichromatic walk corresponding to v. Each v_i has a unique neighbor in $\mathcal{G}_p^i \setminus \{v_i\}_{i=1}^n$, call them $u_1, \ldots, u_n$. If we are so lucky that there are nodes $t_1, \ldots, t_j$ such that $v_i \in t_i$ for all i and also $t_i \prec t_j$ whenever $i < j$, then we can construct the tree decomposition "blowing up" the bichromatic cycle v back into $v_1, \ldots, v_n$ easily: Beginning from the leaves ℓ of $T[v]$, choose a single vertex $u_{i'} \in \ell$ and replace v with $v_{i'}$ and add $v_{i'}$ also to the parent. When there are multiple $T[v]$ contains multiple $u_{i'}$'s in a single node n, we may replace n with two copies of n_1, n_2 where n_2 is the only child of n_1 and the children of n are now the children of n_2. This process may then be repeated for every bichromatic path.

The procedure just described will replace every vertex in every bag with at most 2 vertices. Since this would increase the treewidth by only a constant factor and we do not expect this to be the case, the precondition $i < j \implies t_i \prec t_j$ is highly probably not always satisfiable. The trouble is this: to each leaf ℓ we may choose a neighbor $u_i \in \beta(\ell)$ from which we get two orderings on the neighbors which we may think of as coming from the "clockwise" and "counterclockwise" traversal of the bichromatic cycle (resp. "left" and "right" in case v corresponds to a path instead of a cycle). When neither of these orders aligns with $\prec$, we are forced to add more vertices u to each bag.

This algorithm closely models the sort of dynamic programming procedure which a tree decomposition models. We should therefore be able to determine the addition of vertices to bags of $\mathcal{T}$ which minimally increases the width for each vertex v. However, there are two points which are not yet clear. First, if we run this algorithm for each vertex v and greedily take the "blown up" tree decomposition of minimal width, will this lead to the global minimal choice for blowing up all v? Second, this algorithm depends essentially on the chosen tree decomposition $\mathcal{T}$. However, tree decompositions, even those of minimum width, are not unique, and it may be the case that some are more optimal than others following the "blowing up" procedure.

Appendix: Lean code for Chapter 4

File: `Chap4/MarkoffTriple.lean`

```
module

public import Mathlib

@[expose] public section

@[ext]
structure MarkoffTriple {R : Type u} [CommRing R] (m κ : R) where
  coords : Fin 3 → R
  solves':
    (coords 0)^2 + (coords 1)^2 + (coords 2)^2
      = m * (coords 0) * (coords 1) * (coords 2) + κ
deriving DecidableEq

namespace MarkoffTriple

variable {R : Type u} [CommRing R]

open Equiv

variable {m κ : R}

@[reducible]
def mk' (a b c : R) (solves : a ^ 2 + b ^ 2 + c ^ 2 = m * a * b * c + κ) :
    MarkoffTriple m κ := {
```

```
  coords | 0 => a | 1 => b | 2 => c
  solves' := by simp_all
}

abbrev a (t : MarkoffTriple m κ) := t.coords 0
abbrev b (t : MarkoffTriple m κ) := t.coords 1
abbrev c (t : MarkoffTriple m κ) := t.coords 2

lemma eq_iff {₁t ₂t : MarkoffTriple m κ} :
    ₁t = ₂t ↔ ₁t.a = ₂t.a ∧ ₁t.b = ₂t.b ∧ ₁t.c = ₂t.c := {
  mp h := by aesop
  mpr h := by
    ext i
    fin_cases i <;> aesop
}

@[reducible]
instance : GetElem (MarkoffTriple m κ) (Fin 3) R (fun _ _ => True) where
  getElem t i _ := t.coords i

lemma ext' {₁t ₂t : MarkoffTriple m κ}
  (h : ∀ i : Fin 3, ₁t[i] = ₂t[i]):
    ₁t = ₂t := eq_iff.mpr ⟨h 0, h 1, h ⟩2

lemma solves (t : MarkoffTriple m κ) (i : Fin 3) :
    t[i] ^2 + t[i + 1] ^2 + t[i + 2] ^2 = m * t[i] * t[i + 1] * t[i + 2] + κ := by
  fin_cases i
  all_goals
    dsimp
    rw [
        show t[(0: Fin 3)] = t.a by rfl
      , show t[(1: Fin 3)] = t.b by rfl
      , show t[(2: Fin 3)] = t.c by rfl
    ]
    ac_change t.a ^2 + t.b ^2 + t.c^2 = m * t.a * t.b * t.c + κ
```

```
    exact t.solves'

lemma solves_twist (t : MarkoffTriple m κ) (i : Fin 3) :
    t[i] ^2 + t[i + 2] ^2 + t[i + 1] ^2 = m * t[i] * t[i + 2] * t[i + 1] + κ := by
  ac_change t[i] ^2 + t[i + 1] ^2 + t[i + 2] ^2 = m * t[i] * t[i + 1] * t[i + 2] + κ
  exact t.solves i

def perm (t : MarkoffTriple m κ) σ( : Perm (Fin 3)) : MarkoffTriple m κ := {
  coords i := t.coords σ⁻¹( i)
  solves' := by
    have := t.solves σ⁻¹( 0)
    have := t.solves_twist σ⁻¹( 0)
    fin_cases σ
    all_goals
      simp_all [swap_apply_of_ne_of_ne]
    all_goals aesop
}

@[simp, grind =]
lemma perm_get {t : MarkoffTriple m κ} σ{ : Perm (Fin 3)} {i : Fin 3} :
    (t.perm σ)[i] = tσ⁻¹[ i] := by
  fin_cases σ
  <;> fin_cases i
  all_goals
    simp_all [getElem, perm]

@[simp, grind =]
lemma coords_eq_getElem {t : MarkoffTriple m κ} {i : Fin 3} :
    t.coords i = t[i] := by rfl


@[simp, grind =]
lemma perm_a {t : MarkoffTriple m κ} σ{ : Perm (Fin 3)} :
    (t.perm σ).a = tσ⁻¹[ 0] := by rfl

```

```
instance : MulAction (Perm (Fin 3)) (MarkoffTriple m κ) where
  smul g t := t.perm g
  mul_smul ₁g ₂g t := by
    apply ext'; intro i
    convert_to (t.perm (₁g * ₂g))[i] = ((t.perm ₂g).perm ₁g)[i]
    simp
  one_smul t := by
    apply ext'; intro i
    convert_to (t.perm 1)[i] = t[i]
    simp

@[simp, grind =]
lemma mul_get {t : MarkoffTriple m κ} σ{ : Perm (Fin 3)} {i : Fin 3} :
    σ( • t)[i] = tσ⁻¹[ i] := by
  convert_to (t.perm σ)[i] = tσ⁻¹[ i]
  simp

def vieta (t : MarkoffTriple m κ) (i : Fin 3) : MarkoffTriple m κ := {
  coords j :=
    if i = j then
      m * t[i + 1] * t[i + 2] - t[i]
    else
      t[j]
  solves' := by
    have := t.solves (i := i)
    fin_cases i <;> grind
}

lemma vieta_eq (t : MarkoffTriple m κ) (i : Fin 3) :
    (t.vieta i)[i] = m * t[i + 1] * t[i + 2] - t[i] := by
  fin_cases i
  all_goals
    simp only [vieta, getElem]
    simp

```

```
lemma vieta_eq_of_ne {t : MarkoffTriple m κ} {i j : Fin 3} (i_ne_j : i ≠ j) :
    (t.vieta i)[j] = t[j] := by
  fin_cases i
  <;> fin_cases j
  <;> first | rfl | simp at i_ne_j

@[simp]
theorem vieta_is_involution {t : MarkoffTriple m κ} {i : Fin 3} :
    (t.vieta i).vieta i = t := by
  fin_cases i
  <;> ext i
  <;> fin_cases i
  all_goals simp [vieta_eq, vieta_eq_of_ne]

lemma vieta_eq_iff_eq_vieta {x y : MarkoffTriple m κ} {i : Fin 3} :
    x.vieta i = y ↔ x = y.vieta i := {
  mp h := by simp [←h]
  mpr h := by simp [h]
}

@[simp]
lemma perm_vieta {t : MarkoffTriple m κ} σ{ : Perm (Fin 3)} {i : Fin 3} :
    σ( • t).vieta i = σ • (t.vieta σ⁻¹( i)) := by
  apply ext'
  intro j
  if h: i = j then
    subst h
    rw σ[( • t).vieta_eq]
    repeat rw [mul_get]
    rw [t.vieta_eq]
    suffices h: tσ⁻¹[ (i + 1)] * tσ⁻¹[ (i + 2)] = tσ⁻¹[ i + 1] * tσ⁻¹[ i + 2] by
      rw [mul_assoc, h]; ring
    by_cases h: σ⁻¹ (i + 1) = σ⁻¹ i + 1
    case' pos =>
      rw [h, (?_: σ⁻¹ (i + 2) = σ⁻¹ i + 2)]
```

```
    case' neg =>
      rw [(?_: σ⁻¹ (i + 1) = σ⁻¹ i + 2), (?_: σ⁻¹ (i + 2) = σ⁻¹ i + 1)]
      · ring
    all_goals
      fin_cases i <;> fin_cases σ
      <;> simp_all [swap_apply_of_ne_of_ne]
  else
    simp only [mul_get]
    repeat rw [vieta_eq_of_ne]
    · simp only [mul_get]
    · by_contra con; absurd h
      simpa using congrArg σ⁻¹( ·) con
    · exact h

def zero : MarkoffTriple m 0 :=
  ⟨fun _ => 0, by ⟩simp

def doubleSignChange (t : MarkoffTriple m κ) (j : Fin 3) : MarkoffTriple m κ := {
  coords i :=
    if i = j then
      t[i]
    else
      -t[i]
  solves' := by
    have := t.solves'
    grind
}

@[simp]
lemma doubleSignChange.involution {t : MarkoffTriple m κ} {i : Fin 3} :
    (t.doubleSignChange i).doubleSignChange i = t := by
  fin_cases i
  <;> ext j
  <;> fin_cases j
  all_goals
```

```
    have := t.solves
    simp only [doubleSignChange, getElem]
    simp

instance : Nonempty (MarkoffTriple m 0) := ⟨⟩zero

end MarkoffTriple
```

File: `Chap4/MarkoffNConnected.lean`

```

import Mathlib

import Chap4.MarkoffTriple

@[ext]
structure MarkoffTriplePos (m : ℤ) extends MarkoffTriple m 0 where
  pos {i : Fin 3} : coords i > 0
deriving DecidableEq

def MarkoffTriple.pos {m : ℤ} (t : MarkoffTriple m 0) (h : ∀ i : Fin 3, t[i] > 0) :
    MarkoffTriplePos m :=
  { t with pos := by aesop }

namespace MarkoffTriplePos

open MarkoffTriple

variable {m : ℤ}

lemma eq_of_cast_eq {a b : MarkoffTriplePos m}
    (h : a.toMarkoffTriple = b.toMarkoffTriple) : a = b := by
  aesop

@[reducible]
instance : GetElem (MarkoffTriplePos m) (Fin 3) ℤ (fun _ _ => True) where
  getElem t i _ := t.toMarkoffTriple[i]
```

```
@[simp, grind =]
lemma get_eq {t : MarkoffTriplePos m} {i : Fin 3} :
    t.toMarkoffTriple[i] = t[i] := by rfl

lemma solves (t : MarkoffTriplePos m) (i : Fin 3) :
    t[i] ^2 + t[i + 1] ^2 + t[i + 2] ^2 = m * t[i] * t[i + 1] * t[i + 2] :=
  by simpa using t.toMarkoffTriple.solves i

lemma vieta_alt_dvd (t : MarkoffTriplePos m) (i : Fin 3) :
    t[i] ∣ (t[i + 1])^2 + (t[i + 2])^2 := by
  apply (Int.dvd_add_right (b := t[i] ^2) (by aesop)).mp
  erw [←add_assoc, t.solves i]
  apply dvd_mul_of_dvd_left
  apply dvd_mul_of_dvd_left
  apply dvd_mul_of_dvd_right
  apply Int.dvd_refl

lemma vieta_alt (t : MarkoffTriplePos m) {i : Fin 3} :
    (t.vieta i)[i] = Int.divExact _ _ (t.vieta_alt_dvd i)
   := by
  rw [Int.divExact_eq_ediv _]
  apply Int.eq_ediv_of_mul_eq_right t.pos.ne.symm
  rw [t.toMarkoffTriple.vieta_eq]; ring_nf
  apply Int.sub_eq_iff_eq_add'.mpr
  ac_change m * t[i] * t[i + 1] * t[i + 2] = (t[i])^2 + (t[i + 1])^2 + (t[i + 2])^2
  · rw [show t.toMarkoffTriple[i] = t[i] by rfl]; grind
  simpa using (t.solves i).symm

def vieta (t : MarkoffTriplePos m) (i : Fin 3) : MarkoffTriplePos m :={
  t.toMarkoffTriple.vieta i with
  pos :=by
    intro j
    if h: i ≠ j then
      simpa [vieta_eq_of_ne h] using t.pos
```

```
    else
      push Not at h; subst h
      simp only [coords_eq_getElem, t.vieta_alt]
      apply Int.ediv_pos_of_pos_of_dvd ?_ ?_ (t.vieta_alt_dvd _)
      · apply add_pos_of_nonneg_of_pos
        focus apply le_of_lt
        all_goals exact Int.pow_pos t.pos
      · exact t.pos.le
}

@[simp]
lemma cast_vieta {t : MarkoffTriplePos m} {i : Fin 3} :
    t.toMarkoffTriple.vieta i = (t.vieta i).toMarkoffTriple := by rfl

@[simp]
lemma vieta_is_involution {t : MarkoffTriplePos m} {i : Fin 3} :
    (t.vieta i).vieta i = t := eq_of_cast_eq <| by
  repeat rw [←cast_vieta]
  exact MarkoffTriple.vieta_is_involution

lemma vieta_eq_iff_eq_vieta {x y : MarkoffTriplePos m} {i : Fin 3} :
    x.vieta i = y ↔ x = y.vieta i := {
  mp h := by aesop
  mpr := by aesop
}

lemma vieta_eq (t : MarkoffTriplePos m) (i : Fin 3) :
    (t.vieta i)[i] = m * t[i + 1] * t[i + 2] - t[i] := t.toMarkoffTriple.vieta_eq i

lemma vieta_eq_of_ne (t : MarkoffTriplePos m) {i j : Fin 3} (h : i ≠ j) :
    (t.vieta i)[j] = t[j] := t.toMarkoffTriple.vieta_eq_of_ne h

open Equiv (Perm swap)

def perm (t : MarkoffTriplePos m) σ( : Perm (Fin 3)) : MarkoffTriplePos m :=
```

```
  (t.toMarkoffTriple.perm σ).pos <| by
    intro i
    have := t.pos (i := σ⁻¹ i)
    simpa using t.pos

@[simp]
lemma perm_get {t : MarkoffTriplePos m} σ{ : Perm (Fin 3)} {i : Fin 3} :
    (t.perm σ)[i] = tσ⁻¹[ i] := MarkoffTriple.mul_get

@[simp]
lemma perm_vieta {t : MarkoffTriplePos m} σ{ : Perm (Fin 3)} {i : Fin 3} :
    (t.perm σ).vieta i = (t.vieta σ⁻¹( i)).perm σ :=
  eq_of_cast_eq t.toMarkoffTriple.perm_vieta

theorem a_eq_one_of_a_eq_b {t : MarkoffTriplePos 3} {i j : Fin 3}
  (₁h : i ≠ j) (₂h : t[i] = t[j]) :
    t[i] = 1 := by
  wlog h: j = i + 1
  · have h': j = i + 2 := by omega
    subst h'
    simpa [Equiv.swap_apply_of_ne_of_ne] using this
      (t := t.perm <| swap (i + 1) (i + 2))
      (j := i + 1) (by simp)
      (by simp [Equiv.swap_apply_of_ne_of_ne, ₂h]) rfl
  subst h; clear ₁h
  have := t.solves i
  erw [←₂h] at this
  conv at this => simp; ring_nf
  obtain ⟨m, ⟩hm: t[i] ∣ t[i + 2] := by
    apply (Int.pow_dvd_pow_iff two_ne_zero).mp
    apply (Int.dvd_iff_dvd_of_dvd_add (b := (t[i])^2 * 2) ?_).mp (by simp)
    rw [this, mul_assoc]; apply dvd_mul_right
  rw [
      hm
    , show (t[i])^2 * (t[i] * m) * 3 = (t[i])^2 * ((t[i]) * m * 3) by ring
```

```
    , show (t[i])^2 * 2 + (t[i]*m)^2 = (t[i])^2 * (2 + m^2) by ring
  ] at this
  apply Int.eq_of_mul_eq_mul_left
    (Int.pow_ne_zero <| t.pos.ne.symm) at this
  have m_le_two: m ≤ 2 := by
    apply Int.le_of_dvd zero_lt_two
    apply (Int.dvd_iff_dvd_of_dvd_add (c := m^2) ?_).mpr (by simp)
    rw [this]
    apply dvd_mul_of_dvd_left
    apply dvd_mul_of_dvd_right dvd_rfl
  have m_pos: m > 0 := by
    apply Int.pos_of_mul_pos_right (a := t[i]) ?_ t.pos
    apply Int.pos_of_mul_pos_left (b := 3) ?_ zero_lt_three
    rw [←this]
    apply Int.add_pos_of_pos_of_nonneg zero_lt_two
    exact Int.pow_nonneg_of_even rfl
  interval_cases m
  · simpa using this
  · simpa [show t[i] * 2 * 3 = 6 * t[i] by ring] using this

theorem z_vieta_gt_self_of_nonmax_of_fundamental_domain
  {t : MarkoffTriplePos m}
  {i : Fin 3} {p : t[i] ≤ t[i + 1] ∨ t[i] ≤ t[i + 2]} :
    (t.vieta i)[i] > t[i] := by
  wlog h': t[i] ≤ t[i + 1]
  · replace p := p.resolve_left h'
    simpa [Equiv.swap_apply_of_ne_of_ne, vieta_eq] using this
      (t := t.perm <| swap (i + 1) (i + 2)) (i := i)
      (p := by simp [Equiv.swap_apply_of_ne_of_ne, p])
      (by simp [Equiv.swap_apply_of_ne_of_ne, p])
  suffices t[i] * t[i] < t[i] * (m * t[i + 1] * t[i + 2] - t[i]) by
    rw [vieta_eq]
    exact lt_of_mul_lt_mul_left this (t.pos.le)
  ac_change t[i] * t[i] < m * t[i] * t[i + 1] * t[i + 2] - t[i] * t[i]
  · ring
```

```
  rw [←t.solves]; ring_nf
  apply lt_add_of_le_of_pos
  · apply ₀pow_le_pow_left t.pos.le h'
  · apply Int.pow_pos t.pos

theorem z_vieta_le_others_of_max_of_fundamental_domain
  {t : MarkoffTriplePos 3} {i : Fin 3} {p : t[i] > t[i + 1] ∧ t[i] > t[i + 2]} :
    (t.vieta i)[i] ≤ t[i + 1] ∨ (t.vieta i)[i] ≤ t[i + 2] := by
  wlog h: t[i + 1] >= t[i + 2]
  · simpa [Equiv.swap_apply_of_ne_of_ne, vieta_eq] using or_comm.mp <| this
      (t := t.perm <| swap (i + 1) (i + 2)) (i := i)
      (p := by simp [Equiv.swap_apply_of_ne_of_ne, p])
      (by push Not at h; simpa using h.le)
  let f (x: ℤ) := (x - t[i]) * (x - (t.vieta i)[i])
  have : f t[i + 1] <= 0 := by
    dsimp [f]; rw [t.vieta_eq i]
    calc (t[i + 1] - t[i]) * (t[i + 1] - (3 * t[i + 1] * t[i + 2] - t[i]))
      _ = (t[i + 1])^2 + 3 * t[i] * t[i + 1] * t[i + 2] - (t[i])^2 - 3 * (t[i + 1])^2
    * t[i + 2] :=
        by ring
      _ = (t[i + 1])^2 + ((t[i])^2 + (t[i + 1])^2 + (t[i + 2])^2) -
          (t[i])^2 - 3 * (t[i + 1])^2 * t[i + 2] :=
        by rw [t.solves]
      _ = 2 * (t[i + 1])^2 + (t[i + 2])^2 - 3 * (t[i + 1])^2 * t[i + 2] :=
        by ring
    simp only [sub_nonpos]
    calc 2 * (t[i + 1])^2 + (t[i + 2])^2
      _ = 2 * (t[i + 1])^2 + (t[i + 2] * t[i + 2]) := by ring
      _ <= 2 * (t[i + 1])^2 + (t[i + 1] * t[i + 1]) :=
        add_le_add_right (Int.mul_le_mul h h t.pos.le t.pos.le) _
      _ = (3 * (t[i + 1])^2) * 1 := by
        ring
      _ <= (3 * (t[i + 1])^2) * t[i + 2] :=
        Int.mul_le_mul
          le_rfl t.pos zero_le_one
```

```
          (by simpa using Int.pow_nonneg t.pos.le)
  if h': t[i] = 2 then
    rw [vieta_eq, h']
    rw [show t[i + 1] = 1 by have : t[i + 1] > 0 := t.pos; omega]
    rw [show t[i + 2] = 1 by have : t[i + 2] > 0 := t.pos; omega]
    omega
  else
    suffices f_b_ne_zero: f t[i + 1] ≠ 0 by
      replace this := lt_of_le_of_ne this f_b_ne_zero
      have this := Int.pos_of_mul_neg_right this (by simpa using p.left)
      apply Or.inl
      simpa using this.le
    apply Int.mul_ne_zero
    · exact sub_ne_zero.mpr p.left.ne
    · apply sub_ne_zero.mpr
      rw [←show (t.vieta i)[i + 1] = t[i + 1] by apply t.vieta_eq_of_ne (by simp)]
      by_contra con
      have a_vieta_one := a_eq_one_of_a_eq_b (t := t.vieta i) (by simp) con.symm
      have b_one := con.symm.subst
        (motive := fun x => x = 1) a_vieta_one
      beta_reduce at b_one
      erw [t.vieta_eq_of_ne (by omega)] at b_one
      have eq := t.solves i
      rw [show t[i + 1] = 1 by assumption] at h eq
      have c_one: t[i + 2] = 1 := by
        have : t[i + 2] > 0 := t.pos
        interval_cases t[i + 2]; rfl
      rw [c_one] at eq; ring_nf at eq
      have a_dvd_2: t[i] ∣ 2 := by
        have := Int.dvd_mul_right t[i] 3
        rw [←eq] at this
        apply Int.dvd_iff_dvd_of_dvd_add at this
        exact this.mpr (by simp)
      apply Int.le_of_dvd zero_lt_two at a_dvd_2
      omega
```

```
theorem z_vieta_lt_self_of_max_of_fundamental_domain
  {t : MarkoffTriplePos 3} {i : Fin 3} (h : t[i] > t[i + 1] ∧ t[i] > t[i + 2]) :
    (t.vieta i)[i] < t[i] :=
  match z_vieta_le_others_of_max_of_fundamental_domain (p := h) with
    | .inl h' => h'.trans_lt h.left
    | .inr h' => h'.trans_lt h.right

lemma vieta_ne_self {t : MarkoffTriplePos 3} {i : Fin 3} :
    t.vieta i ≠ t := by
  suffices (t.vieta i)[i] ≠ t[i] by aesop
  if h: t[i] > t[i + 1] ∧ t[i] > t[i + 2] then
    apply ne_of_lt
    exact z_vieta_lt_self_of_max_of_fundamental_domain h
  else
    set_option push_neg.use_distrib true in push Not at h
    apply ne_of_gt
    exact z_vieta_gt_self_of_nonmax_of_fundamental_domain (p := h)

def graph : SimpleGraph (MarkoffTriplePos 3) := {
  Adj x y := ∃i: Fin 3, x.vieta i = y
  symm x y := by
    intro ⟨i, ⟩h; use i
    apply congrArg (flip vieta i) at h
    erw [flip, vieta_is_involution] at h
    exact h.symm
  loopless := ⟨fun x => by
    push Not; intro i
    exact vieta_ne_self
  ⟩
}

def root : MarkoffTriplePos 3 :=
  ⟨⟨fun _ => 1, by ⟩simp, by ⟩simp
```

```
@[simp]
lemma root_get {i : Fin 3} : root[i] = 1 := by
  fin_cases i <;> rfl

def maxIdxOfNeRoot {t : MarkoffTriplePos 3} (h : t ≠ root) :
    {i: Fin 3 // t[i] > t[i + 1] ∧ t[i] > t[i + 2]} :=
  if ₁h: t[(0: Fin 3)] > t[(1: Fin 3)] then
    if ₂h: t[(0: Fin 3)] > t[(2: Fin 3)] then
      ⟨0, ₁h, ₂⟩h
    else if ₂h': t[(0: Fin 3)] = t[(2: Fin 3)] then
      by
        have : t[(0: Fin 3)] = 1 := a_eq_one_of_a_eq_b (by trivial) ₂h'
        have : t[(1: Fin 3)] > 0 := t.pos
        omega
    else
      have : t[(0: Fin 3)] < t[(2: Fin 3)] := by omega
      ⟨2, this, ₁h.trans ⟩this
  else if ₁h': t[(0: Fin 3)] = t[(1: Fin 3)] then
    ⟨2, by
      rw [show 2 + 1 = (0: Fin 3) by rfl, show 2 + 2 = (1: Fin 3) by rfl]
      have t0_eq_1: t[(0: Fin 3)] = 1 := a_eq_one_of_a_eq_b (by trivial) ₁h'
      have t2_eq_1: t[(1: Fin 3)] = 1 := a_eq_one_of_a_eq_b (by trivial) ₁h'.symm
      rw [t0_eq_1, t2_eq_1]
      simp only [and_self]
      have : t[(2: Fin 3)] > 0 := t.pos
      have : t[(2: Fin 3)] ≠ 1 := by
        by_contra; absurd h
        ext i; fin_cases i
        all_goals assumption
      omega
    ⟩
  else if ₂h: t[(1: Fin 3)] > t[(2: Fin 3)] then
    ⟨1, ₂h, by rw [show 1 + 2 = (0: Fin 3) by rfl]; ⟩omega
  else if ₂h': t[(1: Fin 3)] = t[(2: Fin 3)] then
    by
```

```
      have : t[(1: Fin 3)] = 1 := a_eq_one_of_a_eq_b (by trivial) ₂h'
      have : t[(0: Fin 3)] > 0 := t.pos
      omega
  else
    ⟨2, by simp; omega, by simp; ⟩omega

@[reducible]
def max (t : MarkoffTriplePos 3) : ℤ :=
  if h: t = root then 1
  else let i := maxIdxOfNeRoot h; t[i.val]

@[reducible]
def maxNat (t : MarkoffTriplePos 3) : ℕ :=
  t.max.toNat

@[simp]
lemma root_max_eq_one :
    root.max = 1 := by rfl

@[simp]
lemma root_maxNat_eq_one :
    root.maxNat = 1 := by rfl

lemma le_max (t : MarkoffTriplePos 3) (i : Fin 3) :
    t[i] ≤ t.max := by
  if h : t = root then
    fin_cases i <;> all_goals aesop
  else match _ : i - (maxIdxOfNeRoot h).val with
    | 0 | 1 | 2 => grind

lemma le_maxNat (t : MarkoffTriplePos 3) (i : Fin 3) :
    t[i] ≤ t.maxNat := by
  if h: t = root then
    fin_cases i <;> all_goals aesop
  else
```

```
    simp only [Int.ofNat_toNat, le_sup_iff]
    apply Or.inl
    match h': i - (maxIdxOfNeRoot h).val with
      | 0 | 1 | 2 => grind

lemma max_eq_one_iff_eq_root (t : MarkoffTriplePos 3) :
    t.max = 1 ↔ t = root := {
  mp h := by
    suffices t.toMarkoffTriple = root.toMarkoffTriple by aesop
    apply MarkoffTriple.ext'
    intro i
    have : t[i] > 0 := t.pos
    have : t[i] <= 1 := by simpa [←h] using t.le_max i
    simp
    omega
  mpr h := by subst h; decide
}

lemma maxNat_lt_of_max_lt (₁t ₂t : MarkoffTriplePos 3)
  (h : max ₁t < max ₂t) :
    (max ₁t).toNat < (max ₂t).toNat := by
  apply Int.ofNat_lt.mp
  suffices h': (max ₁t) < (max ₂t) ∧ 0 < (max ₂t) by
    simp [h']
  refine ⟨h, ?⟩_
  by_cases h': ₂t = root
  · simp [max, h']
  · simpa only [max, h'] using ₂t.pos

lemma vieta_max_le {t : MarkoffTriplePos 3} {i j : Fin 3} (h : t[i] > t[i + 1] ∧ t[i]
    > t[i + 2]) :
    (t.vieta i)[j] ≤ t[j] :=
  if h': i = j then
    h' ▸ (z_vieta_lt_self_of_max_of_fundamental_domain h).le
  else by rw [t.vieta_eq_of_ne h']
```

```
lemma existsUnique_vieta_dec_of_ne_root {t : MarkoffTriplePos 3} (h : t ≠ root) :
    ∃! i : Fin 3, (t.vieta i).max < t.max := by
  let i := t.maxIdxOfNeRoot h
  use i.val
  split_ands
  · beta_reduce
    if h' : (t.vieta i) = root then
      simp only [max, h', h]
      change 1 < t[i.val]
      apply lt_of_le_of_lt (b := t[i.val + 1]) ?_ i.property.left
      have := t.pos (i := i.val + 1)
      have : 0 < t[i.val + 1] := by simp_all
      omega
    else
      rw [show t.max = t[i.val] by simp [max, h, maxIdxOfNeRoot]; aesop]
      unfold max
      simp only [h', ↓reduceDIte]
      if h'' : (t.vieta i).maxIdxOfNeRoot h' = i.val then
        rw [h'']
        apply z_vieta_lt_self_of_max_of_fundamental_domain
        exact i.property
      else
        push Not at h''
        rw [t.vieta_eq_of_ne h''.symm]
        match (t.maxIdxOfNeRoot h) - i.val with
          | 0 | 1 | 2 => grind
  · intro j h'
    rw [show t.max = t[i.val] by grind] at h'
    if h'' : t.vieta j = root then
      by_contra con
      apply congrArg ·([i.val]) at h''
      simp [t.vieta_eq_of_ne con] at h''
      grind
    else
```

```
      suffices (t.vieta j)[i.val] < t[i.val] by
        by_contra con
        simp [t.vieta_eq_of_ne con] at this
      apply lt_of_le_of_lt ?_ h'
      apply (t.vieta j).le_max

def walkToRoot (t : MarkoffTriplePos 3) :
    SimpleGraph.Walk graph t root :=
  if h: t = root then
    h ▸ .nil
  else
    let i := maxIdxOfNeRoot h
    .cons (v := t.vieta i) (by use i) (walkToRoot <| t.vieta i)
termination_by maxNat t
decreasing_by
  if h': t.vieta i = root then
    rw [h']
    rw [show maxNat root = 1 by decide]
    calc 1
      _ ≤ t[i.val + 1].toNat := (Int.le_toNat <| t.pos.le).mpr <| t.pos
      _ < t[i.val].toNat := by
        suffices t[i.val + 1] < t[i.val] ∧ 0 < t[i.val] by simp [this]
        refine ⟨i.property.left, t.⟩pos
      _ ≤ maxNat t := by simp only [Int.toNat_le, le_maxNat]
  else
    let j := maxIdxOfNeRoot (t := t.vieta i.val) h'
    rw [show maxNat (t.vieta i.val) = (t.vieta i.val)[j.val].toNat by
      simp [maxNat, max, h']; rfl
    ]
    calc (t.vieta i.val)[j.val].toNat
      _ ≤ t[j.val].toNat := by simp [vieta_max_le i.property]
      _ < maxNat t := ?_
    apply lt_of_le_of_ne (by simp [le_max t j.val])
    rw [show maxNat t = t[i.val].toNat by
      simp [maxNat, max, h]; rfl
```

```
    ]
    symm; by_contra con
    apply congrArg (Nat.cast (R := ℤ) ·) at con
    repeat rw [Int.toNat_eq_max] at con
    rw [
      show Max.max t[i.val] 0 = t[i.val] by simpa using t.pos.le,
      show Max.max t[j.val] 0 = t[j.val] by simpa using t.pos.le
    ] at con
    apply a_eq_one_of_a_eq_b at con
    · have := i.property.left
      have : t[i.val + 1] > 0 := t.pos
      omega
    · by_contra con'
      have := z_vieta_le_others_of_max_of_fundamental_domain
        (t := t)
        (i := i.val) (p := i.property)
      cases this
      case' ₁h.inl => have := j.property.left
      case' ₁h.inr => have := j.property.right
      all_goals
        rw [←con'] at this
        rw [vieta_eq_of_ne (t := t) (j := i.val + _) (by simp)] at this
        omega

theorem is_connected : graph.Connected :=
  graph.connected_iff_exists_forall_reachable.mpr <| by
    use root
    intro w
    use (walkToRoot w).reverse

def markoff_graph_rec {t s : MarkoffTriplePos 3} (walk : SimpleGraph.Walk graph t s)
  {motive : MarkoffTriplePos 3 → Sort u} (start : motive s)
  (step : {x y : MarkoffTriplePos 3} → graph.Adj y x → motive x → motive y) :
    motive t :=
  match _: walk with
```

```
    | @SimpleGraph.Walk.nil _ _ u =>
      (by aesop: u = s) ▸ start
    | @SimpleGraph.Walk.cons _ _ _ _ u h p =>
      have : u = s := by aesop
      step h (markoff_graph_rec (this ▸ p) start step)
termination_by walk.length

theorem coords_coprime (t : MarkoffTriplePos 3) {i j : Fin 3} (i_ne_j : i ≠ j) :
    Int.gcd t[i] t[j] = 1 :=
  markoff_graph_rec (motive := fun x => Int.gcd x[i] x[j] = 1) (walkToRoot t)
    (by simp)
    (by
      intro x y e h
      wlog hj: j = i + 1
      · have h': j = i + 2 := by omega
        have := this t (i := i + 2) (j := i)
          (by fin_cases i <;> decide) e
          (by simpa [←h', Int.gcd_comm] using h)
          (by simp [add_assoc])
        simpa [←h', Int.gcd_comm] using this
      subst hj
      obtain ⟨k, ⟩hk := e
      apply (Int.gcd_eq_iff (g := 1)).mpr
      refine ⟨by simp, by simp, ?⟩_
      intro c ₁ch ₂ch
      apply (Int.gcd_eq_iff (g := 1)).mp at h
      apply And.right ∘ And.right at h
      specialize h c
      if k = i ∨ k = i + 1 then
        rename_i this; cases this
        all_goals
          rename_i this; subst this
          rw [←hk] at h
          rw [vieta_eq, vieta_eq_of_ne _ (by simp)] at h
        · apply h ?_ ₂ch
```

```
          apply Int.dvd_sub ?_ ₁ch
          apply Int.dvd_mul_of_dvd_left
          exact Int.dvd_mul_of_dvd_right ₂ch
        · apply h ₁ch ?_
          apply Int.dvd_sub ?_ ₂ch
          apply Int.dvd_mul_of_dvd_right
          simpa [add_assoc] using ₁ch
      else
        have : k = i + 2 := (by omega); subst this
        rw [←hk] at h
        repeat rw [vieta_eq_of_ne _ (by simp)] at h
        exact h ₁ch ₂ch
    )

end MarkoffTriplePos
```

Bibliography

[1] M. Aigner. *Markov's Theorem and 100 Years of the Uniqueness Conjecture*. Springer, 2013.

[2] A. Baragar. "On the Unicity Conjecture for Markoff Numbers". *Canad. Math. Bull.* 39.1 (1996), 3–9.

[3] A. Baragar. "The Markoff equation and equations of Hurwitz". Dissertation. Brown University, 1991.

[4] E. Bellah, S. Chen, E. Fuchs, and L. Ye. *Bounding Lifts of Markoff Triples mod p*. 2023. arXiv: 2311.11468 [math.NT]. URL: https://arxiv.org/abs/2311.11468.

[5] J. Bourgain, A. Gamburd, and P. Sarnak. "Markoff triples and strong approximation". *Comptes rendus. Mathèmatique* 354.2 (2016), 131–135.

[6] C. A. Brown. "An almost linear time algorithm testing whether the Markoff mod p graphs are connected". *Res. Number Theory* 11.3 (2024).

[7] K. Buzzard and R. Taylor. *FLT*. 2025. URL: github.com/ImperialCollegeLondon/FLT.

[8] M. Carneiro. "The Type Theory of Lean" (2019). URL: https://github.%20com/digama0/lean-type-theory/releases.

[9] J. Castellví, M. Drmota, M. Noy, and C. Requilé. "Chordal graphs with bounded tree-width". *Adv. Appl. Math.* 157 (2024), 102700. ISSN: 0196-8858. DOI: https://doi.org/10.1016/j.aam.2024.102700. URL: https://www.sciencedirect.com/science/article/pii/S0196885824000319.

[10] W. Y. Chen. "Nonabelian level structures, Nielsen equivalence, and Markoff triples". *Annals of Mathematics* 199.1 (2024). ISSN: 0003-486X.

[11] P. Corvaja and U. Zannier. "Greatest common divisors of $u-1$, $v-1$ in positive characteristic and rational points on curves over finite fields". *J. Eur. Math. Soc. (JEMS)* 15.5 (2013), 1927–1942. DOI: 10.4171/JEMS/409.

[12] T. W. Cusick and M. E. Flahive. *The Markoff and Lagrange Spectra*. American Mathematical Society, 1989.

[13] M. De Courcy-Ireland and S. Lee. "Experiments with the Markoff Surface". *Exp. Math.* 31.3 (2022), 814–829.

[14] The Lean Developers. *The Lean Language Reference*. 4.30.0-rc2. URL: `lean-lang.org/reference`.

[15] J. Eddy, E. Fuchs, M. Litman, D. Martin, N. Tripeny, and D. Vanyo. *Connectivity of Markoff Mod-P Graphs and Maximal Divisors*. 2023. arXiv: `2308.07579 [math.NT]`.

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. eng. 1st ed. 2006. Texts in Theoretical Computer Science. An EATCS Series. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. ISBN: 1-280-62574-0.

[17] F. V. Fomin and K. Høie. "Pathwidth of cubic graphs and exact algorithms". *Inform. Procc. Lett.* 97.5 (2005), 191–196. DOI: `https://doi.org/10.1016/j.ipl.2005.10.012`.

[18] Von G. Frobenius. "Über die Markoffschen Zahlen". Vol. Jan-Juni 1913. https://www.biodiversitylibrary.org/bibliography/42231. Berlin, Deutsche Akademie der Wissenschaften zu Berlin, 1882-1918, 1913, 458–487. URL: `https://www.biodiversitylibrary.org/page/40702191`.

[19] E. Fuchs, K. Lauter, M. Litman, and A. Tran. *A Cryptographic Hash Function from Markoff Triples*. Cryptology ePrint Archive, Paper 2021/983. 2021. URL: `https://eprint.iacr.org/2021/983`.

[20] A. Ghosh and P. Sarnak. "Integral points on Markoff type cubic surfaces". *Invent. Math.* 229.2 (2022), 689–749. ISSN: 0020-9910.

[21] M. Grohe and D. Marx. "On tree width, bramble size, and expansion". eng. *J. Combin. Theory Ser. B* 99.1 (2009), 218–228. ISSN: 0095-8956.

[22] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 1975.

[23] A Hurwitz. "Über die angenäherte Darstellung der Irrationalzahlen durch rationale Brüche". *Math. Ann.* 39 (1891), 279–378.

[24] A. Kontorovich and T. Tao. *Prime Number Theorem and More*. Version 0.1.0. Jan. 2024. URL: `https://github.com/AlexKontorovich/PrimeNumberTheoremAnd`.

[25] S. V. Konyagin, S. V. Makarychev, I. E. Shparlinski, and I. V. Vyugin. "On the structure of graphs of Markoff triples". *Q. J. Math.* 71 (2020).

[26] J. Liouville. "Nouvelle démonstration d'un théorème sur les irrationnelles algb́riques, inséré dans le Compte Rendu de la dernière séance". *C. R. Math. Acad. Sci. Paris.* A 18 (1844), 910–911.

[27] F. Lübeck. “Standard generators of finite fields and their cyclic subgroups”. *J. Symbolic Comput.* 117.51-67 (2023).

[28] A. Lubotzky. *Discrete groups, expanding graphs, and invariant measures.* Progress in mathematics ; v. 125. Birkhäuser, 1994. ISBN: 081765075X.

[29] A. Markoff. “Sur les formes quadratiques binaires indéfinies”. *Math. Ann.* 15.3-4 (1879).

[30] A. Markoff. “Sur les formes quadratiques binaires indéfinies II”. *Math. Ann.* 17 (1880).

[31] D. E Martin. “A new proof of Chen’s theorem for Markoff graphs”. *Invent. Math.* 241 (2025), 623–626.

[32] D. E. Martin. *Markoff Triples and Nielsen Equivalence in* $SL_2(\mathbb{F}_p)$. 2025. arXiv: 2510.0775772 [math.NT].

[33] B. Monien and R. Preis. “Upper bounds on the bisection width of 3- and 4-regular graphs”. *Journal of discrete algorithms* 4.3 (2005), 475–498.

[34] S. Ramachandramurthi. “The Structure and Number of Obstructions to Treewidth”. *SIAM J. Discrete Math.* 10.1 (1997), 146–157. DOI: 10.1137/S0895480195280010. eprint: https://doi.org/10.1137/S0895480195280010.

[35] N. Robertson and P. D. Seymour. “Graph Minors. XX. Wagner’s conjecture”. *J. Combin. Theory Ser. B* 92.2 (2004), 325–357. ISSN: 0095-8956. DOI: https://doi.org/10.1016/j.jctb.2004.08.001.

[36] K. F. Roth. “Rational approximations to algebraic numbers”. eng. *Mathematika* 2.1 (1955), 1–20. ISSN: 0025-5793.

[37] H. Schecker. “Über die Menge der Zahlen, die als Minima quadratischer Formen auftreten”. *J. Number Theory* 9 (1977), 121–141.

[38] J. H. Silverman. “A heuristic subexponential algorithm to find paths in Markoff graphs over finite fields”. *Res. Number Theory* 11.3 (2025).

[39] The mathlib Community. “The Lean Mathematical Library”. *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs.* CPP 2020. New Orleans, LA, USA: ACM, Jan. 2020. DOI: 10.1145/3372885.3373824. URL: https://doi.org/10.1145/3372885.3373824.

ProQuest Number: 32739863

INFORMATION TO ALL USERS
The quality and completeness of this reproduction is dependent on the quality
and completeness of the copy made available to ProQuest.

Distributed by
ProQuest LLC a part of Clarivate (2026).
Copyright of the Dissertation is held by the Author unless otherwise noted.

This work is protected against unauthorized copying under Title 17,
United States Code and other applicable copyright laws.

This work may be used in accordance with the terms of the Creative Commons license
or other rights statement, as indicated in the copyright statement or in the metadata
associated with this work. Unless otherwise specified in the copyright statement
or the metadata, all rights are reserved by the copyright holder.

ProQuest LLC
789 East Eisenhower Parkway
Ann Arbor, MI 48108 USA