EFFICIENT ENUMERATION OF QUADRATIC LATTICES

ERAN ASSAF, VICTOR CHEN, ROHAN GARG, AND BENNY WANG

Abstract. We present an algorithm to enumerate isometry classes of integral quadratic lattices of a given rank and determinant, and we analyze its running time by establishing bounds on the number of genus symbols for a fixed rank and determinant. Our work uses the enumeration of genus symbols following Conway and Sloane, together with algorithms to produce representatives from a given genus symbol and an algorithm, guided by the Smith–Minkowski–Siegel mass formula, to generate the remaining equivalence classes. We build on previous work of Brandhorst, Hanke, and Dubey–Holenstein. We analyze the running times of their respective algorithms and compare the practical performance of their implementations with our own. Our implementations are publicly available.

1. Introduction

1.1. **Motivation.** The study of quadratic lattices has been central in mathematics for centuries, with their classification playing a major role. This traces back to Gauss [Gau01], who classified binary quadratic lattices up to isometry and introduced the concept of the genus, with important applications to number theory. Gauss's methods were used as recently as [CS99] to produce tables of binary quadratic forms.

Later developments by Minkowski, Hasse, and Witt led to Eichler's complete classification of indefinite forms [Eic52], and these advances went hand in hand with tabulation efforts such as [Jon35] and [BI58] for lattices of rank 3. This continued in the work of Nipp [Nip91], who used computers to produce tables for ranks 4 and 5, now available in Nebe and Sloane's catalog [NS]. These efforts continue today, with works such as [KL13] and [EH19] classifying all isometry classes with class number one in ranks 3 and 4, respectively.

Our work continues these efforts to tabulate isometry classes of lattices. We present an algorithm that constructs lattices representing each isometry class of a given rank and determinant. As an application, we obtain a comprehensive database of isometry classes, which will be added to the L-functions and Modular Forms Database [LMFDB].

1.2. **Results.** We follow the ideas of [CS99, Chapter 15] to generate all valid genus symbols of a given determinant and rank, and use these to produce lattices representing every isometry class in the corresponding genera. This is done in two steps. We first construct a single lattice using the algorithm of [DH14a], with improvements appearing in [Han13] and unpublished work of Simon Brandhorst, implemented in [SageMath].

Date: June 1, 2026.

The complexity analysis below is in terms of arithmetic operations in the ring of integers. Note that the size of integers throughout the algorithms is bounded by the determinant D.

Theorem 1.1. *There exists an algorithm that, given positive integers n and D such that $\log D \ll n$, outputs a Gram matrix of an integral lattice in every genus of rank n and determinant D, with running time complexity*

$$n^4 D^{\alpha + \frac{\pi(1+o(1))}{\log\log D}},$$

where $\alpha = \log_2(3+\sqrt{17}) - 1 \approx 1.833$.

Moreover, by refining the analysis, one obtains some saving in the running-time complexity when running the algorithm for all determinants up to D.

Theorem 1.2. *There exists an algorithm that, given positive integers n and D such that $\log D \ll n$, outputs a Gram matrix of an integral lattice in every genus of rank n and determinant at most D, with running time complexity*

$$O(n^4 D^{\alpha+1} (\log D)^{e^\pi - 1} \log\log D).$$

Applying Eichler's theorem of strong approximation as in [CS99, Corollary 22], it has the following immediate corollary.

Corollary 1.3. *There exists an algorithm that, given positive integers n and D such that $\log D \ll n$, outputs a Gram matrix of an integral lattice in every indefinite isometry class of rank n and determinant at most D, with running time complexity*

$$O(n^4 D^{\alpha+1} (\log D)^{e^\pi - 1} \log\log D).$$

Remark 1.4. For definite lattices, the running time complexity is dominated by exhausting the isometry classes within a genus, for which we do not have an improvement over the known methods of enumerating p-neighbors, which are exponential in the rank n. The basic idea was first described in [SP91], and by using the isometry testing of [PS97] instead of Minkowski reduction, it can be adapted to work for any rank, see [Han13, Algorithm 5.8].

An implementation of the algorithm is publicly available at `https://github.com/assaferan/k3s-lmfdb/tree/main/lattices`. An overview of the method is given in Section 3, after the necessary background in Section 2.

A key step in the algorithm is computing a representative in a given genus. This step is the bottleneck for the running time of the overall procedure. In Section 4 we describe such an algorithm, originally implemented in [SageMath]. We compare its running time complexity to a previously described algorithm [DH14a].

The algorithm, implemented in [SageMath] by Brandhorst, following [MM09], uses two main components: the construction of a maximal integral overlattice, and local modifications to an integral lattice to match a specified p-adic genus symbol at a single prime p. These steps are described, respectively, in Sections 5 and 6.

An algorithm for finding a maximal overlattice was proposed in [Han13]. In Section 5, we analyze its running time and compare it with Brandhorst's. Our final algorithm incorporates these improvements into Brandhorst's method.

Section 7 analyzes the running time of our algorithm and completes the proofs of Theorem 1.1 and Theorem 1.2.

2. Background and Notation

In this section, we recall some needed definitions and fix notations. For reference, the interested reader may consult any standard text on quadratic forms, such as [O'M63], [Cas78], [Bue89], [CS99], or [Cox22].

We work in a fixed dimension n. Let R be a PID with field of fractions F, and let V be an F-vector space of dimension n, so that $V \simeq F^n$. Our main object of study is lattices.

Definition 2.1. An **R-lattice** in V is a finitely generated R-submodule $L \subseteq V$ such that $FL = V$. We say that L has **rank** n. When $R = \mathbb{Z}$, we simply say that L is a **lattice**.

Remark 2.2. Some authors omit the second condition in the definition of an R-lattice and say that L is **full** if $FL = V$. We will not encounter R-lattices that are not full, so we avoid this terminology.

In this paper, all lattices will be equipped with a symmetric bilinear form.

Definition 2.3. Let M, N be R-modules. A **symmetric bilinear form on M with values in N** is a function $B : M \times M \to N$ satisfying

- $B(u, v + w) = B(u, v) + B(u, w)$ for all $u, v, w \in M$.
- $B(u, v) = B(v, u)$ for all $u, v \in M$.
- $B(\lambda u, v) = \lambda B(u, v)$ for all $u, v \in M$ and $\lambda \in R$.

When $N = R$, we simply say that B is a **symmetric bilinear form** on M.

The form B induces a homomorphism $\phi_B : M \to \mathrm{Hom}_R(M, N)$ given by $u \mapsto (v \mapsto B(u, v))$. We say that B is **regular** when ϕ_B is an isomorphism. When $R = F$ is a field, we also say that B is **non-degenerate**.

Given a symmetric bilinear form B on V, any choice of a basis E for V gives rise to an $n \times n$ symmetric matrix $M_{B,E} \in M_n(F)$ with

$$B(u, v) = u^{\intercal} M_{B,E} v \qquad \forall\, u, v \in V.$$

This is the **Gram matrix** of B with respect to E. If E is a basis for an R-lattice L, the same matrix is a Gram matrix for L.

We will be interested in lattices in quadratic spaces, and there will be a slight difference in integrality notions between the quadratic form and the symmetric bilinear form, due to issues at $p = 2$. We therefore proceed to define quadratic forms.

Definition 2.4. A **quadratic form** on V is a function $Q : V \to F$ such that:

$$Q(\lambda v) = \lambda^2 Q(v) \qquad \forall\, v \in V,\ \lambda \in F,$$

and such that

$$B(u, v) = Q(u + v) - Q(u) - Q(v) \tag{2.5}$$

defines a bilinear form. The pair (V, Q) is called a **quadratic space**.

Every quadratic form gives rise to a bilinear form. If $\mathrm{char}(F) \neq 2$, the converse holds with $Q(v) = \frac{1}{2} B(v, v)$.

Definition 2.6. Two quadratic spaces (V_1, Q_1) and (V_2, Q_2) are **isometric** if there exists an F-linear isomorphism $S : V_1 \to V_2$ such that

$$Q_2(S(v)) = Q_1(v) \qquad \forall v \in V_1.$$

We then write $V_1 \simeq V_2$ and the map S is called an **isometry** between V_1 and V_2.

If E is a basis for V, isometries correspond to matrices $S \in \mathrm{GL}_n(F)$ satisfying $S^\intercal M_2 S = M_1$.

Definition 2.7. Lattices L_1, L_2 in quadratic spaces V_1, V_2 are **isometric** if there exists an isometry $S : V_1 \to V_2$ with $S(L_1) = L_2$. We write $L_1 \simeq L_2$.

Scaling the form, we may assume $B(L, L) \subseteq R$. This leads to the following.

Definition 2.8. An **integral R-lattice** in (V, Q) is a lattice L with $B(L, L) \subseteq R$. It is **even** if $Q(L) \subseteq R$.

An R-lattice L is integral if and only if it has a Gram matrix $G \in M_n(R)$. Changing basis by $A \in \mathrm{GL}_n(R)$ replaces G by $A^\intercal G A$, so determinants are well defined modulo squares.

Definition 2.9. If L is an integral R-lattice with Gram matrix G, its **determinant** is

$$\det(L) = \det(G) R^{\times 2} \in R/R^{\times 2}.$$

When $R = \mathbb{Z}$, this is simply an integer.

Our goal is to classify integral lattices up to isometry for fixed rank n and determinant D. However, the determinant D is a coarse invariant, and one obtains a finer one by working locally. For $\mathbb{Z}$-lattices, we can define the following invariant, measuring the local-to-global obstruction.

Definition 2.10. Let L_1, L_2 be quadratic lattices. If

$$L_1 \otimes \mathbb{Z}_p \simeq L_2 \otimes \mathbb{Z}_p \quad \text{for all primes } p,$$

and

$$L_1 \otimes \mathbb{R} \simeq L_2 \otimes \mathbb{R},$$

we say they lie in the same **genus** and write $L_1 \sim L_2$.

We describe algorithms to produce Gram matrices in each such genus. These use some standard constructions, which we proceed to describe below.

Definition 2.11. The **dual lattice** of an R-lattice L in a non-degenerate quadratic space is

$$L^\sharp = \{u \in V : B(u, v) \in R \text{ for all } v \in L\}.$$

L is integral if and only if $L \subseteq L^\sharp$.

Definition 2.12. The **discriminant module** of an integral R-lattice L is the R-module

$$D(L) = L^\sharp / L,$$

equipped with the induced bilinear form

$$B_D : D(L) \times D(L) \to F/R.$$

Definition 2.13. If L is an integral $\mathbb{Z}$-lattice, its **p-discriminant group** is

$$D_p(L) = D(L_p) \qquad \text{where } L_p = L \otimes \mathbb{Z}_p.$$

We often need to construct a maximal integral lattice containing a given lattice L, for which a first approximation can be given by saturating the lattice. Here we give precise definitions for what this means.

Definition 2.14. An integral $\mathbb{Z}$-lattice L is **p-saturated** if $pD_p(L) = 0$.

Definition 2.15. An **overlattice** of L is any lattice L' with $L \subseteq L'$.

Definition 2.16. An integral R-lattice L is **R-maximal** if $L \subseteq L'$ and L' integral imply $L = L'$.

Finally, for describing the genus, one can use a finite list of symbols, encoding the local isometry class at each place. At ∞, this is simply the signature. We proceed below to recall the definitions of the signature and, more generally, the local genus symbols.

Definition 2.17. When $F = \mathbb{R}$, the **signature** of (V, Q) is the triple (n_0, n_+, n_-) counting the zero, positive, and negative eigenvalues of a Gram matrix. If $n_0 = 0$, we write (n_+, n_-).

Following [CS99], it is convenient to treat real equivalence as equivalence over $\mathbb{Z}_p$ for $p = -1$.

A **genus symbol** is a finite collection of local invariants, one for each place p (including $p = -1$), that uniquely determines the genus. We use the explicit system of symbols defined in [CS99]. In particular, a genus symbol is simply the tuple of all its local symbols, and two symbols are equal exactly when the associated genera are equal. Our algorithms always treat genera as given in this symbolic form, so no preprocessing such as reordering constituents is required.

In our complexity analysis, some arithmetic functions occur naturally. We denote by $\omega(D)$ the number of distinct prime factors of D, by $\nu_p(D)$ the valuation of D at p, and by $\Omega(D) = \sum_{p|D} \nu_p(D)$ the total number of prime factors of D. We also denote by $\psi \approx 2.373$ the exponent for the complexity of matrix multiplication, so that multiplication of $n \times n$ matrices costs n^ψ multiplications in the base ring.

3. Overview of the algorithm

Our objective is to find a Gram matrix of a lattice in every genus of integral lattices with a fixed rank n and determinant D. To do so, we first enumerate a list of the genera with rank n and determinant D.

Let $\mathcal{G}(n, D)$ denote the (finite) set of genera of integral lattices of rank n and determinant D. Then, for each genus $\gamma \in \mathcal{G}(n, D)$, we compute a representative lattice in γ. For a formal description of this algorithm, see Algorithm 1.

Example 3.1. For $n = 1$ and $D \neq 0$, there is a single genus of one-dimensional lattices of determinant D, and its isometry classes correspond to the one-dimensional bilinear form $B(x, y) = Dxy$, with Gram matrix (D). In this trivial case, the algorithm reduces to choosing the representative $L = (D)$.

Lemma 3.2. *Algorithm 1 returns a set of representatives for all genera of integral lattices with rank n and determinant D.*

Proof. Let L be an integral lattice with rank n and determinant D. Let γ be its genus symbol. By Lemma 3.3, `GenusSymbols`(n, D) returns all valid genus symbols of rank n and determinant D, hence $\gamma \in S$. By Theorem 4.29, $g =$ `Representative`(γ) is a Gram matrix for a lattice L' with genus symbol γ, so $L' \sim L$. □

Algorithm 1: Construct Gram matrices for every genus of integral lattices of given rank n and determinant D.

Data: A positive integer n and an integer D.
$S \leftarrow \texttt{GenusSymbols}(n, D)$;
$G \leftarrow \emptyset$;
for $\gamma \in S$ **do**
  $G \leftarrow G \cup \texttt{Representative}(\gamma)$;
end
return G;

Algorithm 2: $\texttt{GenusSymbols}(n, D)$. Construct a valid genus symbol for each genus with given rank n and determinant D.

Data: A positive integer n and an integer D.
$S \leftarrow \emptyset$;
for $p \in \texttt{PrimeDivisors}(2D)$ **do**
  $S_p \leftarrow \texttt{LocalGenusSymbols}(n, D, p)$;
end
for $\gamma = (\gamma_p)_p \in \prod_{p|2D} S_p$ **do**
  if $\texttt{IsValid}(\gamma)$ **then**
    $S \leftarrow S \cup \{\gamma\}$;
  end
end
return S;

In Section 7 we analyze the running time complexity of Algorithm 1 to complete the proof of Theorem 1.1 and Theorem 1.2.

We proceed to describe each of the different steps.

3.1. **Enumeration of genus symbols.** The first subalgorithm of Algorithm 1 is $\texttt{GenusSymbols}(n, D)$. The objective of this subalgorithm is to generate a list of the genus symbols of all genera with fixed determinant D and rank n. We accomplish this task as follows.

(1) For each prime p dividing $2D$, we compute all possible local genus symbols with rank n and determinant D.
(2) We generate all possible combinations of local genus symbols and filter out combinations that do not yield any lattices. The remaining combinations correspond to genera that each contain at least one lattice, and these are the genera that we return.

For a formal description of this algorithm, see Algorithm 2. Here, the function $\texttt{LocalGenusSymbols}(n, D, p)$ returns all (finitely many) options for a p-adic genus symbol of rank n and determinant D, and $\texttt{IsValid}(\gamma)$ is a function that returns true if γ satisfies the determinant, oddity and p-adic existence conditions [CS99, (29)-(35)], and false otherwise.

Lemma 3.3. *Algorithm 2 returns all valid genus symbols having rank n and determinant D.*

Proof. Let $\gamma = (\gamma_p)_p$ be a valid genus symbol of rank n and determinant D. Then $\gamma_p \in S_p$ for all $p \mid 2D$, and by [CS99, Theorem 11], `IsValid`(γ) returns true. □

3.2. **Finding a representative.** In order to find a representative, we use Algorithm 3. In [DH14a], a different algorithm was described. The following Lemma compares their respective running time, and shows that Algorithm 3 has better running time complexity.

Lemma 3.4. *For $\log D \ll n$, the running time complexity of Algorithm 3, using* [Han13, Algorithm 4.6] *for the maximal overlattice algorithm, has better running time complexity than that of* [DH14a].

Proof. According to Theorem 4.26 the time complexity of [DH14a] is

$$O(n^8 + n^{2+\psi} \sum_{p|D} \log \nu_p(D) + n^3 \omega(D) \sum_{p|D} (\log p)^2,$$

where ψ is such that multiplication of $n \times n$ matrices runs in $O(n^\psi)$, $\omega(D) = \sum_{p \in P} 1$ is the number of prime divisors of D, and $\nu_p(D)$ is the valuation of D at p.

According to Corollary 4.33, the time complexity of Algorithm 3, when using [Han13, Algorithm 4.6] for the maximal overlattice algorithm, is

$$O(n^3 \omega(D) + n^\psi \sum_{p|D} \log(\nu_p(D)) + n\omega(D)\Omega(D) \log D),$$

where $\Omega(D) = \sum_{p \in P} \nu_p(D)$.

Since $\omega(D) \ll \log D \ll n$, we have $n^3 \omega(D) \ll n^8$. Clearly,

$$n^\psi \sum_{p|D} \log \nu_p(D) \ll n^{2+\psi} \sum_{p|D} \log \nu_p(D),$$

and as $\Omega(D) \ll \log D$, we also have

$$n\omega(D)\Omega(D) \log D \ll n\omega(D)(\log D)^2 \ll n^3 \omega(D) \sum_{p|D} (\log p)^2.$$

Adding these three inequalities, we obtain the result. □

4. Representative of a genus

In this section we describe two algorithms that, given a genus symbol, return a representative lattice in that genus. We begin by quickly reviewing the algorithm described in [DH14a] and carefully analyzing its running time complexity. We then describe Algorithm 3, and analyze its running time complexity, which is being used in Lemma 3.4.

4.1. **Dubey–Holenstein Algorithm.** In [DH14a], an algorithm to find a representative in polynomial time is established, but no exact upper bound was determined. We will characterize the complexity of the Dubey–Holenstein Algorithm in terms of

- the rank n of the input genus;
- the determinant D of the input genus;
- the set of distinct prime divisors of $2D$ (not including -1);
- a maximum probability of failure ε.

In [DH14a] the term quadratic form is used interchangeably with either the lattice or its Gram matrix by fixing a basis for the lattice, so we will adopt this convention throughout this section.

First, we recall some definitions used in [DH14a].

As a shorthand, we define $k_2 = 3$ and $k_p = 1$ for odd primes p. Moreover, we define the notation $\overline{q}$ as follows:

$$\overline{q} = q \prod_{p \mid 2q} p^{k_p}. \tag{4.1}$$

Moreover, we use the notation $\nu_p(n)$ to denote the largest integral power of p dividing n, where p is a prime integer and n is a positive integer.

Definition 4.2. A vector $v \in (\mathbb{Z}/p^k\mathbb{Z})^n$ is **primitive** if there exists an entry v_i such that $p \nmid v_i$.

We can generalize this notion to vectors $v \in (\mathbb{Z}/q\mathbb{Z})^n$, for composite q: such a vector is **primitive** if it is primitive modulo p^k for all $p^k \mid q$. We say that an integer t **has a primitive** q-representation by a quadratic form Q if there exists a primitive $\mathbf{x} \in (\mathbb{Z}/q\mathbb{Z})^n$ for which $\mathbf{x}^\intercal Q\mathbf{x} \equiv t \pmod{q}$. Evidently, if an integer t has a primitive q-representation by one quadratic form in a genus γ, it follows that t has a primitive q-representation by every quadratic form in γ. This allows us to define the following:

Definition 4.3. An integer t is said to **have a primitive representation in a genus γ** if, for each prime power p^k, the integer t has a primitive p^k-representation by every quadratic form in γ.

By [DH14a, Theorem 22], the above is equivalent to t having just a primitive $\overline{\mathrm{lcm}(t, \det(\gamma))}$-representation (notation as in (4.1)) by every quadratic form in γ.

Definition 4.4. Two rank n integral quadratic forms Q_1 and Q_2 are ***q*-equivalent** if there exists an $n \times n$ matrix $U \in \mathrm{GL}(\mathbb{Z}/q\mathbb{Z})$ for which $U^\intercal Q_1 U \equiv Q_2 \pmod{q}$.

Definition 4.5. A quadratic form Q is ***q*-equivalent to a genus** γ if, for every form $Q_1 \in \gamma$, Q and Q_1 are q-equivalent.

Definition 4.6. Let γ be a genus and let n be a positive integer. We define **scalar multiplication of γ by n** as the genus $n\gamma$ obtained from scaling each of the local quadratic forms of γ by n.

It is clear that if Q is a quadratic form in the genus γ, then for all positive integers n, nQ is a quadratic form in the genus $n\gamma$; conversely, all quadratic forms in $n\gamma$ can be divided by n, and doing so gives a quadratic form in γ.

Definition 4.7. Let γ be a genus. The **GCD of γ** is the largest integer n for which there exists a genus γ' such that $n\gamma' = \gamma$.

We now provide a high level summary of the algorithm.

(1) We identify an integer t that has a primitive representation in γ.
(2) We compute a quadratic form Q_0 that is $\overline{(Dt^{n-1})}$-equivalent to γ, with its top-left entry equal to t.
(3) Based off of Q_0, we generate a genus γ_{n-1} of rank $n-1$. We recursively call the algorithm on γ_{n-1}, obtaining a representative form Q_{n-1} and construct our representative based off of Q_0 and Q_{n-1}.

The details of the algorithm can be found in [DH14a, Theorem 33], and a proof of correctness can be found in [DH14a, Theorem 19]. The overview above is provided to help give context for our complexity analysis of the algorithm.

In our analysis, we will ignore bit complexity. In particular, we will assume that the following operations take $O(1)$ time:

- Addition, subtraction and multiplication in $\mathbb{Z}/p^k\mathbb{Z}$,
- addition, subtraction, multiplication and division in $\mathbb{Z}$,
- choosing a random integer in a given range.

First, we state some well-established results useful for our running time complexity analysis.

Lemma 4.8. *Let p be a fixed prime, and suppose an integer k satisfying $(\frac{k}{p}) = -1$ is known. Given a fixed odd prime power p^k and a quadratic residue r with $0 \leq r < p^k$, a square root of r in $\mathbb{Z}/p^k\mathbb{Z}$ can be computed in $O(\log p + \log k)$ time. (The value of r does not affect the time complexity.)*

Proof. If $r = 0$ we are instantly done. If $r > 0$, write $r = p^e \cdot r'$ where e is a nonnegative integer and $p \nmid r'$. One must have e even and r' a quadratic residue modulo p. Since $e \leq k$, computing e by binary search takes $O(\log k)$ time. Then, given a square root x of r' modulo p^{k-e}, $p^{e/2}x$ is a square root of r modulo p^k. With this reduction, we may assume that $p \nmid r$.

By the Tonelli–Shanks algorithm [DS73], we can compute a square root of r mod p in $O(\log p)$ time, making use of our integer k with $(\frac{k}{p}) = -1$. Now, to lift this square root to a square root modulo p^k, use Hensel lifting: if $x^2 \equiv r \pmod{p^t}$ and $2x$ is invertible mod p, then one may correct x to a root modulo p^{2t}, and iterating yields a root modulo p^k. Setting the number of lifting steps to $O(\log k)$ produces the claimed bound. □

Lemma 4.9. *Given a fixed power of two 2^k and a quadratic residue r with $0 \leq r \leq 2^k$, a square root of r in $\mathbb{Z}/2^k\mathbb{Z}$ can be computed in $O(k)$ time.*

Proof. We must have $r \equiv 1 \pmod 8$ when r is an odd residue, so 1 is a square root of r mod 8 in that case. If $x^2 \equiv r \pmod{2^t}$ then $x^2 \equiv r$ or $x^2 \equiv r + 2^t \pmod{2^{t+1}}$; if the latter holds then $(x + 2^{t-1})^2 \equiv r \pmod{2^{t+1}}$. Thus at each step one can lift a solution from modulus 2^t to modulus 2^{t+1} by checking at most one correction; repeating k times gives $O(k)$ time. □

Lemma 4.10. *Given an integer k and a prime p, the Legendre symbol $(\frac{k}{p})$ can be computed in $O(\log p)$ time.*

Proof. It is standard that

$$\left(\frac{k}{p}\right) \equiv k^{(p-1)/2} \pmod p,$$

and the right-hand exponentiation can be computed in $O(\log p)$ time by fast modular exponentiation. □

We now establish the time complexity of certain precomputations we make prior to the main recursion of [DH14a].

Lemma 4.11. *Let p be an odd prime. There exists an algorithm which takes p and a target failure probability $\delta \in (0,1)$ as input and returns an integer k with $(\frac{k}{p}) = -1$ in*

$$O\left(\log\left(\tfrac{1}{\delta}\right)\log p\right)$$

time, with chance of failure at most δ.

Proof. Exactly half of the elements of $\mathbb{F}_p^\times$ are quadratic residues. Sampling uniformly from $\{1,\dots,p-1\}$ gives a non-residue with probability $1/2$; each Legendre-symbol test costs $O(\log p)$ by Lemma 4.10. Repeating $O(\log(1/\delta))$ independent samples yields the claimed time bound and failure probability. □

Lemma 4.12. *Let $p \geq 5$ be an odd prime, let t be an integer and let $(\varepsilon_1,\varepsilon_2,\varepsilon_3) \in \{-1,1\}^3$. There exists a randomized algorithm which takes $p, \varepsilon_1, \varepsilon_2, \varepsilon_3$ and a target failure probability $\delta \in (0,1)$ as input and returns integers a_1, a_2, a_3 such that*

$$a_1 + a_2 \equiv a_3 \pmod{p}, \qquad \left(\frac{a_i}{p}\right) = \varepsilon_i \text{ for } i \in \{1,2,3\},$$

in $O(\log(\frac{1}{\delta})\log p)$ time, with chance of failure at most δ.

Proof. Sample a_1, a_2 uniformly at random from $\mathbb{F}_p^\times$ and set $a_3 = a_1 + a_2$. For a fixed triple of desired Legendre symbols $(\varepsilon_1,\varepsilon_2,\varepsilon_3)$, the proportion of pairs (a_1,a_2) giving those signs is a positive constant (bounded below uniformly in p). Therefore $O(\log(1/\delta))$ repetitions suffice, and each repetition costs $O(\log p)$ to check the three Legendre symbols, yielding the stated bound. □

From here on, we will assume that the algorithms in Lemma 4.11 and Lemma 4.12 have been precomputed for every single possible set of inputs for $p \leq B$ for an appropriate bound B, so that the result of each algorithm is instantly accessible. In Theorem 4.26, we will determine a finite set of precomputations that are sufficient for the rest of the algorithm. In [DH14a], the algorithms run with a constant probability of failure. However, given the precomputations in Lemma 4.11 and Lemma 4.12, the failing situations are eliminated.

We now establish the time complexity of some smaller tasks in the main recursion of [DH14a].

Lemma 4.13. *Let $m \geq 2$ be a positive integer, let k be an integer with an inverse modulo m. Assuming GRH, there exists a randomized algorithm which takes k, m, and a target failure probability $\delta \in (0,1)$ as input and returns a prime $p > N$ with*

$$p \notin P, \qquad p \equiv k \pmod{m}$$

in $\log(\frac{1}{\delta})\log(m)$ time, with chance of failure at most δ.

Proof. By GRH, there is a constant C, independent of k, such that there are $C\frac{m^3}{\log(m)}$ valid prime p in the set $S = \{k, k+m, \dots, k+m^3-m\}$. We repeatedly sample uniformly from S, with a $\frac{C}{\log(m)}$ probability of success each time. Thus, if we take $\lceil \log(\delta)/\log(1-\frac{C}{\log(m)})\rceil$ samples, we have at most a δ chance of failure. This comes out to

$$O(\log(\tfrac{1}{\delta})\log(m)),$$

as desired. □

Lemma 4.14. *Let γ be a genus. There exists an algorithm which takes γ as input and returns* $\gcd(\gamma)$ *and* $\frac{\gamma}{\gcd(\gamma)}$*, running in*

$$O\left(\sum_{p\in P}\log p\right)$$

time (recall definition 4.7). Here, P is the set of distinct prime divisors of $2\det(\gamma)$.

Proof. For $p \in P$, suppose p^{e_p} is the largest power of p dividing every entry in γ_p. Then, we have

$$\gcd(\gamma) = \prod_{p\in P} p^{e_p}.$$

To compute $\gamma' = \frac{\gamma}{\gcd(\gamma)}$, it suffices to compute γ'_p for each prime $p \in P$. This is

$$r\left(\frac{\gamma}{p^{e_p}}\right),$$

where r is an integer such that $(\frac{r}{p}) = (\frac{\gcd(\gamma)/p^{e_p}}{p})$. Computing r takes $\log(p)$ time, giving us the desired complexity. □

Lemma 4.15. *Let γ be a genus with rank at least 4. There exists an algorithm which takes γ as input and returns an integer t dividing* $\det(\gamma)$ *with a primitive representation in γ, running in*

$$O\left(\sum_{p\in P}\left(\log p + \log\left(\nu_p(D)\right)\right)\right)$$

time (recall Definition 4.3). Here, D is the determinant of γ and P is the set of distinct prime divisors of $2D$.

Proof. In [DH14a, Lemma 26], the existence of such a t is asserted, and a construction is given. We will not reproduce the full construction, but analyze the complexity of the steps used in constructing t.

(1) For each prime p, we perform a variety of constant time computations. One such computation is to look up an integer k with $(\frac{k}{p}) = -1$, which has been precomputed as per Lemma 4.11). This step takes $O(|P|)$ time.
(2) For each prime $p \in P$, we compute a Legendre symbol mod p. This takes $O\left(\sum_{p\in P}\log p\right)$ time.
(3) From step 2, we obtain $\nu_p(t)$ for each $p \in P$. To compute t from these values takes up to
$$O\left(\sum_{p\in P}\log \nu_p(D)\right)$$
time, since the algorithm guarantees that $\nu_p(t) \le \nu_p(D)$.

Summing these times gives the desired complexity. □

Lemma 4.16. *Let γ be a genus with rank 2 or 3. There exists an algorithm which takes γ and a target failure probability $\delta \in (0, 1)$ as input and returns an integer t*

dividing $\det(\gamma)$ *with a primitive representation in* γ*, running in*

$$O\left(\sum_{p\in P}\left(\log(\tfrac{1}{\delta})\log p + \log\left(\nu_p(D)\right)\right)\right)$$

time (recall Definition 4.3), with chance of failure at most δ*. Again,* D *is the determinant of* γ *and* P *is the set of distinct prime divisors of* $2D$*.*

Proof. The algorithm is very similar to that of Lemma 4.15. The only difference in time complexity comes from the need to compute a prime $q \notin P$ satisfying a particular congruence modulo

$$2\prod_{p\in P} p.$$

By Lemma 4.13, computing q takes

$$O\left(\log(\tfrac{1}{\delta})\sum_{p\in P}\log p\right)$$

time. □

Lemma 4.17. *Let* p *be an odd prime, let* n *be an integer and let* $(\varepsilon_1,\varepsilon_2)\in\{-1,1\}^2$*. There exists an algorithm which takes* p *and* n *as input and returns integers* a_1 *and* a_2 *such that*

$$a_1+a_2\equiv n \pmod p, \qquad \left(\frac{a_i}{p}\right)=\varepsilon_i \text{ for } i\in\{1,2\}$$

in $O(1)$ *time, or notes if no such integers exist.*

Proof. If $p=3$, we can check all cases. In particular, if $\varepsilon_1\neq\varepsilon_2$ or $(\frac{n}{p})\in\{\varepsilon_1,\varepsilon_2\}$, there are no solutions. By Lemma 4.12, we have precomputed a triple (b_1,b_2,b_3) such that

$$b_1+b_2\equiv b_3 \pmod p$$

with $(\frac{b_1}{p})=\varepsilon_1$, $(\frac{b_2}{p})=\varepsilon_2$ and $(\frac{b_3}{p})=(\frac{n}{p})$. Now the pair $(a_1,a_2)=(\frac{nb_1}{n_3},\frac{nb_2}{b_3})$ can be instantly computed and works. □

Lemma 4.18. *Let* $Q=\operatorname{diag}(a,b)$ *be a quadratic form where* a *and* b *are integers with* $\nu_p(a)=0$ *and* $\nu_p(b)$ *even. Let* t *and* k *be positive integers with* $p\nmid t$ *and* $(\frac{t}{p})\neq(\frac{a}{p})$*. There exists an algorithm that takes* Q*,* t *and* k *as input and returns a primitive* p^k*-representation of* t *by* Q*, running in* $O(\log p+\log k)$ *time.*

Proof. In [DH14a, Lemma 25], the existence of a solution is asserted, and a construction is given. We will not reproduce the full construction, but analyze the complexity of the steps used there.

(1) First, one finds two quadratic residues whose sum is $\frac{t}{a}$ modulo p. This can be done in $O(1)$ time, by 4.17.
(2) Next, to lift a primitive representation modulo p to one modulo p^k, one computes modular square roots and applies Hensel lifting; by Lemma 4.8 this takes $O(\log p+\log k)$ time.

Summing these costs gives the claimed bound. □

Lemma 4.19. *Let*

$$Q = \begin{pmatrix} 2a & b \\ b & 2c \end{pmatrix}$$

be a quadratic form where a and c are integers and b is odd. Let t and k be positive integers with $t \equiv 2 \pmod 4$. There exists an algorithm that takes Q, t and k as input and returns a primitive 2^k-representation of t by Q, running in $O(k)$ time.

Proof. This algorithm is given in [DH14b, Lemma 17]. By simple trial and error we can compute a 2^2-representation. Now we show how to lift a 2^i-representation (y_1, y_2) to a 2^{i+1}-representation. We split into cases; note that y_1 and y_2 cannot both be even.

- Suppose y_1 is odd. If $ay_1^2+by_1y_2+cy_2^2 \not\equiv t \pmod{2^{i+1}}$, then, by assumption, we must have $ay_1^2 + by_1y_2 + cy_2^2 \equiv t + 2^i \pmod{2^{i+1}}$. Now, note that $(y_1, y_2 + 2^i)$ is a valid 2^{i+1}-representation since

$$ay_1^2 + by_1(y_2 + 2^i) + c(y_2 + 2^i)^2 \equiv ay_1^2 + by_1y_2 + 2^i + cy_2^2 \equiv t \pmod{2^{i+1}}.$$

- Suppose y_1 is even. Then, y_2 is odd and we proceed similarly to the previous case.

By repeatedly lifting, we obtain a 2^k-representation in $O(k)$ time, as desired. □

Lemma 4.20. *Let $Q = \operatorname{diag}(a, b, c, d)$ be a quadratic form with $\nu_2(a) \le \nu_2(b) \le \nu_2(c) \le \nu_2(d)$. Let t and k be positive integers with t odd. There exists an algorithm that takes Q, t and k as input and returns a primitive 2^k-representation of t by Q, running in $O(1)$ time.*

Proof. This algorithm is given in [DH14a, Lemma 24]. First, we write a, b, c and d in the form $a = \tau_1 \cdot 2^{i_1}, \ldots, d = \tau_4 \cdot 2^{i_4}$ where $\tau_1, \ldots, \tau_4$ are odd. Let (x_1, x_2, x_3, x_4) be the representation we are searching for. We substitute $x_j = 2^{\lceil (i_4 - i_j)/2 \rceil} y_j$ for $1 \le j \le 3$, where $y_j \in \mathbb{Z}$, while committing to forcing x_4 to be odd. Dividing out 2^{i_4} on both sides reduces the problem to the case of $k = 4$.

This is now a finite problem, and can thus be solved in constant time. □

We now cite the key result of [DH14c], providing an algorithm to canonicalize a p-adic quadratic form.

Lemma 4.21 ([DH14c, Theorems 12, 14])**.** *Let p^k be a prime power, let Q be an integral quadratic form of rank n and let k be an integer at least $\nu_p\left(\overline{\det Q}\right)$. Let $\delta \in (0, 1)$. Assuming GRH, there exists an algorithm which takes p, Q and k as input and outputs a matrix $U \in \mathrm{GL}_n(\mathbb{Z}/p^k\mathbb{Z})$ such that*

$$U^\intercal Q U \pmod{p^k}$$

is a canonical p-adic form in the sense of [DH14c]*, running in*

- $O\left(n^{1+\psi} \log k + n \log p\right)$ *time if p is odd and*
- $O\left(n^{1+\psi} \log k + nk^3\right)$ *time if $p = 2$.*

(Here, ψ denotes the exponent of the time complexity for matrix inversion.)

In [DH14c], the algorithms run with a constant probability of failure. However, given the precomputations in Lemma 4.11 and Lemma 4.12, the failing situations are eliminated.

Lemma 4.22. *Let p^k be a prime power and let A and B be p^k-equivalent rank n quadratic forms. Assuming GRH, there exists an algorithm which takes p, k, A and B as input and outputs a matrix $U \in \mathrm{GL}_n(\mathbb{Z}/p^k\mathbb{Z})$ such that*

$$U^\intercal AU \equiv B \pmod{p^k},$$

running in

- $O\left(n^{1+\psi}\log k + n\log p\right)$ *time if p is odd and*
- $O\left(n^{1+\psi}\log k + nk^3\right)$ *time if $p = 2$.*

(These are the same time complexities as those in Lemma 4.21.)

Proof. Using Lemma 4.21, we may compute U_A and U_B such that $U_A^\intercal AU_A \equiv U_B^\intercal BU_B \pmod{p^k}$. Then, since

$$(U_AU_B^{-1})^\intercal A(U_AU_B^{-1}) \equiv B \pmod{p^k},$$

we output $U = U_AU_B^{-1}$. (The n^ψ time it takes to compute U_B^{-1} is negligible compared to the complexity of Lemma 4.21.) □

We are now ready to compute the overall complexity of the algorithm. Let $\mathcal{G}(n, D)$ be the set of genera of rank n and determinant D. For notational convenience, let e_p denote $\nu_p(D)$. Let $f(n, D)$ denote the time complexity to compute a representative of $\gamma \in \mathcal{G}(n, D)$.

Lemma 4.23. *Suppose $n \geq 4$. We have*

$$f(n, D) = O\left(ne_2^3 + \sum_{p\mid 2D}\left(n^{1+\psi}\log e_p + n^2\omega(D)(\log p)^2\right)\right) + f(n-1, 2^nD)$$

Proof. We break the time complexities down step by step:

(1) Our first step is to find an integer t with a primitive representation in γ, such that $t \mid D$. As per Lemma 4.15, this step takes

$$O\left(\sum_{p\mid 2D}(\log p + \log e_p)\right)$$

time. Define $q = \overline{Dt^{n-1}}$, and let $K_p = \nu_p(q)$ for each prime p. Note that by our construction of t with $t \mid D$, the set of primes dividing q is the same as the set of primes dividing D. Note that

$$K_p = (n-1)\nu_p(t) + e_p \leq ne_p,$$

which will be used in step (3).

(2) Next, for each prime $p \mid 2D$, we construct an $n \times n$ Gram matrix S_p that is p^{K_p}-equivalent to γ. By construction [DH14a, Lemma 17], this S_p is block diagonal, with blocks of size at most 2×2. Overall, the complexity of this step is

$$O(n\omega(D)).$$

(3) Next, for each prime $p \mid 2D$, we construct a $1 \times n$ column vector $\mathbf{x}_p$ of integers representing t over $\mathbb{Z}/p^{K_p}\mathbb{Z}$, as well as an $(n-1) \times n$ matrix A_p such that $[\mathbf{x}_p, A_p] \in \mathrm{GL}_n(\mathbb{Z}/p^{K_p}\mathbb{Z})$.

We follow the construction outlined in [DH14a, Theorem 33]. By this construction, constructing A_p takes $O(1)$ time, so it really suffices to compute $\mathbf{x}_p$. This can be done by Lemma 4.18, Lemma 4.19 and Lemma 4.20. We must apply the algorithm of Lemma 4.18 once for each odd prime dividing D, for a total of $\omega(2D) - 1$ times. This step thus runs in

$$O\left(K_2 + \sum_{p|D}(\log p + \log(K_p))\right)$$
$$= O\left(ne_2 + \sum_{p|D}(\log p + \log(ne_p))\right)$$

time.

Note that, by construction, $[\mathbf{x}_p, A_p]$ is block diagonal with blocks of size at most 4×4.

(4) Next, for each prime $p \mid 2D$, we compute the products

$$\mathbf{d}_p = \mathbf{x}_p^\intercal S_p A_p \pmod{p^{K_p}}, \qquad H_p = \left(tA_p^\intercal S_p A_p - \mathbf{d}_p^\intercal \mathbf{d}_p\right).$$

The multiplication $\mathbf{d}_p^\intercal \mathbf{d}_p$ takes $O(n^2)$ time. Besides that, since A_p and S_p are block diagonal with blocks of size at most 4, many of the required multiplications take $O(n)$ time; aggregating the cost over $p \mid 2D$ gives

$$O(n^2\omega(D))$$

time.

(5) Next, we define γ_{n-1} as the genus symbol with determinant Dt^{n-2} and local symbol given by H_p for each prime p (note that H_p is an $(n-1) \times (n-1)$ matrix). Now, define

$$\gamma'_{n-1} = \frac{\gamma_{n-1}}{\gcd(\gamma_{n-1})}.$$

(This computation is demonstrated in Lemma 4.14.) We compute a representative of γ_{n-1} by recursively applying our algorithm on γ'_{n-1} and scaling the result by $\gcd(\gamma_{n-1})$. It is shown in [DH14a, Theorem 33] that $\det(\gamma'_{n-1}) \mid \det(\gamma) \cdot 2^n$. Therefore, computing a representative of γ_{n-1} takes at most

$$f(n-1, 2^n D) + O\left(\sum_{p|2D} \log p\right)$$

time. In particular, recall that we constructed t in step (1) to be a divisor of d, so the set of primes dividing Dt^{n-2} is precisely P.

Denote the representative of γ_{n-1} as $\widetilde{H}$.

(6) Next, from our $\omega(2D)$ vectors of the form $\mathbf{x}_p$, we compute a vector $\mathbf{x}$ of integers such that $\mathbf{x} \equiv \mathbf{x}_p \pmod{p^{K_p}}$ for each $p \mid 2D$. We analogously define A, S and H. Computing these vectors/matrices is done by the Chinese Remainder Theorem on individual entries. Since H is not diagonal, we have $O(n)$ nonzero entries; for each entry, there is a system of $\omega(2D)$ modular congruences we need to solve. Each entry takes $O(\omega(D)\sum_{p|2D}(\log p)^2)$

time to solve. So, the overall complexity of this step is

$$f_6(n, D) = O\left(n^2\omega(D)\sum_{p|2D}(\log p)^2\right).$$

(7) Next, for each $p \mid 2D$, we compute a matrix U_p such that

$$U_p^\intercal H U_p \equiv \widetilde{H} \pmod{p^{L_p}},$$

where $L_p = \nu_p(\overline{Dt})$. Note that, since $t \mid D$, we always have $L_p \leq 2e_p + 1$ for odd p and $L_p \leq 2e_p + 3$ for $p = 2$.

The complexity for computing U_p is computed in lemma 4.22. Using this, we complete this step in

$$O\left(n^{1+\psi}\log L_2 + nL_2^3\right) + \sum_{p|D} O\left(n^{1+\psi}\log L_p + n\log p\right)$$

$$= O\left(nL_2^3 + \sum_{p|2D}\left(n^{1+\psi}\log L_p + n\log p\right)\right)$$

$$= O\left(ne_2^3 + \sum_{p|2D}\left(n^{1+\psi}\log\left(e_p\right) + n\log p\right)\right)$$

time.

(8) Similarly to step (6), we compute a matrix U of integers such that

$$U \equiv U_p \pmod{p^{K_p}}$$

for each $p \mid 2D$. This takes

$$f_8(n, D) = O\left(n^2\omega(D)\sum_{p|2D}(\log p)^2\right)$$

time to solve.

(9) Now, we compute the matrix multiplications in [DH14a, (61)]. These multiplications take

$$O(n^2)$$

time to compute.

Summing these steps, we see that the dominant terms are coming from steps (5), (6), (7) and (8). Therefore

$$f(n, D) = (1 + o(1))\sum_{i=6}^{8} f_i(n, D) + f(n-1, 2^n D),$$

yielding the result. □

Lemma 4.24. *For $n \in \{2, 3\}$, assuming GRH, we have*

$$f(n, D) = O\left(e_2^3 + \sum_{p|2D}\left(\log e_p + \omega(D)\log(p)^2 + \log(p)\log(\tfrac{1}{\delta})\right)\right) + f(n-1, 2^n D).$$

Proof. The algorithm is essentially the same as that of Lemma 4.23, but at step 1, we need to use Lemma 4.16 instead of Lemma 4.15. □

Lemma 4.25. *For $n = 1$, we can compute a representative of a genus in $O(\omega(D))$ time.*

Proof. We simply compute $\det(\gamma)$ by multiplying together the determinants of all local symbols. Multiplying these $\omega(D)$ determinants takes $O(\omega(D))$ time. □

Theorem 4.26. *Assuming GRH, the Dubey–Holenstein algorithm runs in*

$$O\left(n^2(e_2+n^2)^3 + n^{2+\psi}\sum_{p|2D}\log e_p + n^3\omega(D)\sum_{p|2D}(\log p)^2 + \log\left(\tfrac{\omega(D)}{\delta}\right)\sum_{p|2D}\log p\right).$$

time, with at most a δ chance of failure.

Proof. In Lemma 4.23, we make use of the fact that we have done the precomputations in Lemma 4.11 and Lemma 4.12. As such, before the recursion, our first step is to apply these two algorithms on every prime $p \mid 2D$. To ensure that our algorithm fails with probability at most δ, it suffices to allow a $\frac{\delta}{2\omega(2D)}$ chance of failure for each call of either algorithm. Thus, our precomputations take

$$O\left(\log\left(\tfrac{\omega(D)}{\delta}\right)\sum_{p|2D}\log p\right)$$

time total. Notice that this value already exceeds the value of the additional work required in Lemma 4.24 (compared to the time complexity in Lemma 4.23). Henceforth, we do not worry about this special case.

We now compute the total runtime of the recursive part of the algorithm, using the recursion we computed in Lemma 4.23. Recall that we have

$$f(n,D) = O\left(ne_2^3 + \sum_{p|2D}\left(n^{1+\psi}\log e_p + n^2\omega(D)(\log p)^2\right)\right) + f(n-1,2^nD).$$

Now, let

$$\begin{aligned}
A(n,D) &= O\left(n^2\omega(D)\sum_{p|2D}(\log p)^2\right),\\
B(n,D) &= O\left(ne_2^3\right),\\
C(n,D) &= O\left(n^{1+\psi}\sum_{p|2D}\log e_p\right).
\end{aligned}$$

So, $f(n,D) = A(n,D) + B(n,D) + C(n,D) + f(n-1,2^nD)$. We now compute the overall complexity of each individual function:

- The overall contribution of $A(n,D)$ in the algorithm is

$$\sum_{i=1}^{n} A(i,D) = O\left(n^3\omega(D)\sum_{p|2D}(\log p)^2\right).$$

- Note that $\sum_{j=n-i+1}^{n} j = \binom{n+1}{2} - \binom{i+1}{2} \leq \binom{n+1}{2} < n^2$, hence
$$\sum_{i=1}^{n} B\left(i, D \cdot 2^{\sum_{j=n-i+1}^{n} j}\right) = \sum_{i=1}^{n} i\left(e_2 + \binom{n+1}{2} - \binom{i+1}{2}\right)^3 = O\left(n^2(e_2+n^2)^3\right).$$
- The overall contribution of $C(n, D)$ in the algorithm is
$$\sum_{i=1}^{n} C(i, D \cdot 2^{n^2}) = O\left(n^{2+\psi}\left(\log(e_2+n^2) + \sum_{p \mid 2D} \log e_p\right)\right).$$

Summing everything together gives us the final claimed complexity. The bottleneck with respect to n lies in the n^8 term, which can be traced back to computing the canonical 2-adic form of H in step 7 of Lemma 4.23. □

Remark 4.27. We note that the above bound is sharp, as for genera whose local 2-adic symbol consists of n blocks which are 1-dimensional of type II, the recursive step increases the 2-adic valuation of D by n.

We may also express the complexity of Theorem 4.26 in terms of just the determinant and rank.

Corollary 4.28. *Let γ be a genus. Let n and D be the rank and determinant of γ. As just a function of n and $\ell = \log D$, the algorithm laid out by Dubey Holenstein runs in*
$$O\left(n^2(n^2+\ell)^3 + n^{2+\psi}\ell\log(\ell) + n^3\ell^2\log(\ell)^2 + \ell^2\log\left(\frac{\ell}{\delta}\right)\right)$$
time, with at most a δ chance of failure.

4.2. **Brandhorst algorithm.** In [SageMath], an algorithm by Simon Brandhorst is given to compute a representative lattice of a given genus. In this subsection we give an overview of this algorithm, prove its correctness, and analyze its running time complexity.

Given a genus symbol γ, we define `Gram-Rational-Representative(`γ`)` to be the Gram matrix of a rational representative of γ. Similarly, for a $\mathbb{Z}_p$ genus symbol γ_p, we define `Gram-`$\mathbb{Z}_p$`-Representative(`γ_p`)` to be the Gram matrix of a representative of γ_p.

For a lattice L, the function `Maximal-Overlattice(`L`)` returns a maximal overlattice of L, by using either Algorithm 8 or [Han13, Algorithm 4.6].

For a lattice L, a prime p, and a Gram matrix G_p over $\mathbb{Z}_p$, the function `Local-Modification` (L, G_p, p) returns a lattice L' such that for all $q \neq p$, $L'_q = L_q$, and L_p has Gram matrix G_p. This is done by using Algorithm 9.

Using these building blocks, we present Algorithm 3 to compute a representative lattice of a given genus.

Algorithm 3: Implements the function `Representative(`γ`)` : Constructs a representative of a genus symbol γ.

Data: Genus symbol γ
$D \leftarrow$ determinant of γ;
$q \leftarrow$ `Gram-Rational-Representative(`γ`)`;
$L \leftarrow$ the integral lattice with Gram matrix $4q$;
$L \leftarrow$ `Maximal-Overlattice(`L`)`;
$\gamma_2 \leftarrow$ the $\mathbb{Z}_2$ genus symbol of γ;
$G_2 \leftarrow$ `Gram-`$\mathbb{Z}_2$`-Representative(`γ_2`)`;
if γ *is not even* **then**
  $L \leftarrow$ `Local-Modification(`L, $4G_2$, 2`)`;
  $L \leftarrow L/2$;
else
  $L \leftarrow$ `Local-Modification(`L, G_2, 2`)`;
end
for *odd prime* p *dividing* d **do**
  $\gamma_p \leftarrow$ the $\mathbb{Z}_p$ genus symbol of γ;
  $G_p \leftarrow$ `Gram-`$\mathbb{Z}_p$`-Representative(`γ_p`)`;
  $L \leftarrow$ `Local-Modification(`L, G_p, p`)`;
end
return L

Theorem 4.29. *Algorithm 3 returns a lattice L with $L \in \gamma$.*

Proof. We show that, at the end of the algorithm, L has the correct genus symbol at each prime p. First consider the prime $p = 2$.

If γ is not even, then we locally modify L such that L_2 has Gram matrix $4G_2$. Observe that L is maximal, $4G_2$ is even, and L is isometric to the lattice with Gram matrix $4G_2$. Thus, by Theorem 6.1, the output of `Local-Modification`, call it L', satisfies that L'_2 has a basis for which the Gram matrix is $4G_2$. Hence, $L'_2/2$ has Gram matrix G_2 with respect to the scaled basis.

Now consider an odd prime p such that $p \mid D$. After the local modification at p, it is clear that L_p has Gram matrix G_p, so L indeed has the correct genus symbol at p.

Finally, for odd primes p that do not divide D, the initial choice of L already had the correct genus symbols at such p, and by Theorem 6.1 we have not modified these genus symbols. Thus, at the end of the algorithm, L has the correct genus symbol at every prime p. □

We begin by giving a complexity analysis which depends on the costs of `Maximal-Overlattice` and `Local-Modification`. We then specialize, depending on the algorithms used. Write $\omega(D)$ for the number of prime divisors of D.

Theorem 4.30. *Suppose the time complexity of doing a local modification at a prime p of a lattice L with rank n and determinant D is $\Gamma(n, D, p)$. Likewise, suppose the time complexity of finding the maximal overlattice of a lattice L with rank n and determinant D is $\rho(n, D)$. Further assume that $n^2 \ll \rho(n, D)$ and*

$n\omega(D) \ll \Gamma(n, D, p)$. *Then Algorithm 3 has time complexity*

$$O\left(\rho(n, D) + \sum_{p|D} \Gamma(n, D, p)\right).$$

Proof. We first estimate the time complexity of the subroutine `Gram-Rational-Representative(`γ`)`. Computing a rational representative given the genus symbol takes $O(n\omega(D) + n^2)$ time: determining the diagonal blocks of the rational Gram matrix is $O(n\omega(D))$ (bounded by the number of blocks) and assembling the full Gram matrix is $O(n^2)$.

Similarly, each `Gram-`$\mathbb{Z}_p$`-Representative(`γ_p`)` can be computed in $O(n)$ time, so the total time to compute the rational representative and the local Gram matrices is

$$O\left(n^2 + \sum_{p|D} n\right) = O\big(n^2 + n\omega(D)\big),$$

which, by our assumptions, we may treat as a lower-order term relative to the costs of `Maximal-Overlattice` and `Local-Modification`.

By definition, computing the maximal overlattice of L costs $\rho(n, D)$. We then perform a local modification for each prime $p \mid D$, and each such local modification costs $\Gamma(n, D, p)$. Summing those costs yields

$$O\left(\rho(n, D) + \sum_{p|D} \Gamma(n, D, p)\right),$$

which dominates the lower-order $O(n^2 + n\omega(D))$ term (reading and assembling matrices is $O(n^2)$, hence dominated by the listed costs in typical parameter regimes). This establishes the stated complexity. □

As a corollary, we obtain the running time complexity when Algorithm 8 is used for `Maximal-Overlattice`, as follows.

Corollary 4.31. *Let $\delta \in (0, 1)$ be an arbitrary probability and let $\omega(D)$ denote the number of distinct primes dividing D. For a prime p, let*

$$c_p = \begin{cases} \lambda_p(\log p) \log\left(\frac{\omega(D)}{\delta}\right) & p \neq 2, \\ 2^{\lambda_2} & p = 2, \end{cases} \tag{4.32}$$

where

$$\lambda_p = \begin{cases} \min(\nu_2(D), 4n) & \text{if } p = 2, \\ \min(\nu_p(D), n) & \text{otherwise.} \end{cases}$$

Assuming arithmetic operations take $O(1)$ time, when using Algorithm 8 for `Maximal-Overlattice`*, Algorithm 3 has time complexity*

$$O\left(n \sum_{p|D} \log(\nu_p(D)) + \lambda_p n^\psi + c_p\right)$$

with probability of failure at most δ.

Proof. The bound follows from Theorem 5.10 and Corollary 6.3, which give the stated costs for computing the maximal overlattice and for the local modifications respectively; combining those bounds with the preceding paragraph yields the claimed overall complexity and failure probability. □

Corollary 4.33. *Let $\omega(D)$ denote the number of distinct primes dividing D and $\Omega(D)$ denote the sum of the valuations of D, i.e.,*

$$\Omega(D) = \sum_{p|D} \nu_p(D).$$

Then, when using [Han13, Algorithm 4.6] *for* `Maximal-Overlattice`*, Algorithm 3 has time complexity*

$$O\left(n^3\omega(D) + n^\psi \sum_{p|D} \log(\nu_p(D)) + n\omega(D)\Omega(D)\log D\right).$$

In terms of n and $\log D$, this gives the following complexity.

$$O\left(n^3 \frac{\log D}{\log\log D} + n^\psi \log D + n\frac{(\log D)^3}{\log\log D}\right).$$

Proof. This follows from Theorem 5.18 and Corollary 6.4. The latter time complexity is obtained by noting that $\omega(D) = O\left(\frac{\log D}{\log\log D}\right)$, $\Omega(D) = O(\log D)$, and $\sum_{p|D} \log \nu_p(D) = O(\log D)$. □

5. Maximal overlattices

In this section we describe and analyze the running-time complexity of two algorithms that, given a lattice L, return a maximal overlattice of L. The first was implemented by Simon Brandhorst in [SageMath] and the second is described in [Han13]. Although [Han13] describes and proves correctness of their algorithms, we require an analysis of the time complexities of these algorithms for our purposes; this is the content of this section.

5.1. Brandhorst Maximal Overlattice algorithm.

In this section we describe, prove correctness of, and analyze the running-time complexity of the Brandhorst algorithm, implemented in [SageMath], for finding a maximal overlattice of a given integral lattice L. In both this section and section 6, normal form refers to the definition of normal form in [MM09, Def. 4.6]. Unless stated otherwise, n and D denote the rank and determinant, respectively, of L. Further, let $M(k)$ denote the time cost of multiplying two k-bit integers. Using the fastest known multiplication algorithm (Harvey–van der Hoeven), one may take $M(k) = O(k \log k)$ [HvdH21].

There are two parts to this algorithm: finding a maximal overlattice over $\mathbb{Z}_p$ for odd primes p, and finding a maximal overlattice over $\mathbb{Z}_2$. The overall algorithm simply takes the maximal overlattice over $\mathbb{Z}_p$ for all relevant primes p.

For an odd prime p, in order to find a maximal overlattice over $\mathbb{Z}_p$, one begins by p-saturating the lattice. Recall that an integral $\mathbb{Z}$-lattice L is **p-saturated** if $pD_p(L) = 0$, where $D_p(L) = D(L_p) = L_p^\sharp/L_p$ is its **p-discriminant** group. Algorithm 4 describes a simple algorithm to p-saturate a lattice.

Algorithm 4: `OP-Saturate(`L`,`p`)`. Constructs a p-saturated overlattice of a $\mathbb{Z}$-lattice L for an odd prime p.

Data: A lattice L and an odd prime p
$\mathcal{D} \leftarrow$ the p-discriminant group of L;
$Q \leftarrow$ the quadratic form on $\mathcal{D}$;
$B \leftarrow$ a basis for $\mathcal{D}$ such that the Gram matrix is in normal form;
$B' \leftarrow$ empty list;
for *generator* g *in* B **do**
  $g' \leftarrow$ a preimage of g in $L^\vee$;
  Add $p^{\left\lfloor \frac{-\nu_p(Q(g))+1}{2} \right\rfloor} \cdot g'$ to B';
end
$L' \leftarrow$ the lattice with basis B';
return L';

Theorem 5.1. *Algorithm 4 returns a p-saturated lattice L' containing L.*

Proof. For a generator g of $\mathcal{D}$ we have

$$\nu_p\left(Q\left(p^{\left\lfloor \frac{-\nu_p(Q(g))+1}{2} \right\rfloor} \cdot g\right)\right) = 2\left\lfloor \frac{-\nu_p(Q(g))+1}{2} \right\rfloor + \nu_p(Q(g)).$$

Since

$$-\nu_p(Q(g)) \le 2\left\lfloor \frac{-\nu_p(Q(g))+1}{2} \right\rfloor \le -\nu_p(Q(g)) + 1,$$

we obtain

$$0 \le \nu_p\left(Q\left(p^{\left\lfloor \frac{-\nu_p(Q(g))+1}{2} \right\rfloor} \cdot g\right)\right) \le 1.$$

Thus every value of the form on the new basis has nonnegative valuation at p and valuation at most 1, so L' is integral at p and p-saturated. □

Theorem 5.2. *Algorithm 4 runs in*

$$O\big((n^\psi + n\log(\nu_p(D)))\ M(\nu_p(D)\log p)\big)$$

time.

Proof. Computing the p-discriminant group $\mathcal{D}$ has the same asymptotic cost as computing a Smith normal form; that can be done with $O(n^\psi)$ ring operations (where $\psi \approx 2.373$ denotes the exponent for matrix multiplication/inversion) over the relevant modulus, see [Sto96]. Here we work over $\mathbb{Z}/p^{1+\nu_p(D)}\mathbb{Z}$, so the cost of a single multiplication in that ring is $M(\nu_p(D)\log p)$.

For the remaining steps we loop over the generators of $\mathcal{D}$. For a single generator g, the exponentiation by a power of p costs $O(\log(\nu_p(Q(g))))$ ring multiplications. Since

$$\sum_g \nu_p(Q(g)) = \nu_p(D),$$

and the number of generators m is at most n, we have

$$\sum_g \log(\nu_p(Q(g))) \le m\log\left(\frac{\nu_p(D)}{m}\right) \le n\log(\nu_p(D)).$$

Hence the total number of ring multiplications performed in exponentiation is $O(n\log(\nu_p(D)))$.

Algorithm 5: `OP-Overlattice(`L`, `p`)`.

Data: A lattice L and an odd prime p
$\mathcal{D} \leftarrow$ the p-discriminant group of L;
$Q \leftarrow$ the quadratic form on $\mathcal{D}$;
$B \leftarrow$ a basis of $\mathcal{D}$ with Gram matrix in normal form;
$D \leftarrow \det(Q)$;
if $-\nu_p(D) \leq 1$ **then**
  return L
else if $-\nu_p(D) = 2$ **then**
  Let $\mathbf{a}, \mathbf{b}$ be the two elements of B;
  $a \leftarrow p \cdot Q(\mathbf{a}), \quad b \leftarrow p \cdot Q(\mathbf{b})$;
  if *$-b/a$ is not a quadratic residue modulo p* **then**
    return L
  else
    Let x be such that $x^2 \equiv -\frac{b}{a} \pmod{p}$;
    $\mathbf{t} \leftarrow x\mathbf{a} + \mathbf{b}$
  end
else if $-\nu_p(D) \geq 3$ **then**
  Choose $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ to be three elements of B;
  $a \leftarrow p \cdot Q(\mathbf{a}), \quad b \leftarrow p \cdot Q(\mathbf{b}), \quad c \leftarrow p \cdot Q(\mathbf{c})$;
  if *$-b/a$ is a quadratic residue modulo p* **then**
    Let x be such that $x^2 \equiv -\frac{b}{a} \pmod{p}$;
    $\mathbf{t} \leftarrow x\mathbf{a} + \mathbf{b}$
  else
    Let x and y be such that $ax^2 + by^2 + c \equiv 0 \pmod{p}$;
    $\mathbf{t} = x\mathbf{a} + y\mathbf{b} + \mathbf{c}$
  end
$\mathbf{t}' \leftarrow$ a preimage of $\mathbf{t}$ in $L^\vee$;
return $L + \langle \mathbf{t}' \rangle$;

Combining the $O(n^\psi)$ cost for the Smith-type computations, the cost for exponentiations, which amount to $O(n \log(\nu_p(D)))$, and multiplying by the bit complexity $M(\nu_p(D) \log p)$ per ring multiplication, yields the stated bound. □

Algorithm 5 implements `OP-Overlattice(`L`, `p`)`, where L is a p-saturated $\mathbb{Z}$-lattice and p is an odd prime. It constructs a p-saturated $\mathbb{Z}$-lattice L' satisfying

(1) $L_p \subseteq L'_p$, with strict inclusion when L_p is not $\mathbb{Z}_p$-maximal;
(2) $L_q = L'_q$ for all primes $q \neq p$.

If L_p is already maximal, the algorithm returns L.

Theorem 5.3. *When L is a p-saturated $\mathbb{Z}$-lattice and p is an odd prime, Algorithm 5 returns the $\mathbb{Z}$-lattice L' satisfying the above properties.*

Proof. Let $\mathcal{D}$ be the p-discriminant group of L written in normal form, and let $D = \det(D)$. Because L is p-saturated, $\mathcal{D}$ is an $\mathbb{F}_p$-vector space of dimension $-\nu_p(D)$. The lattice L is maximal at p exactly when $\mathcal{D}$ contains no nonzero isotropic vectors.

Algorithm 6: `OP-maximal(`L`, `p`)`.

```
Data: A lattice L and an odd prime p
L' ← OP-Saturate(L, p);
while L' is not maximal do
    L'' ← OP-Overlattice(L', p);
    if L' = L'' then
        return L'
    else
        L' ← L''
    end
end
```

If $-\nu_p(D) \le 1$ then $\mathcal{D}$ has dimension at most 1 and therefore contains no isotropic vectors, so L is already maximal.

If $-\nu_p(D) = 2$, let $\mathbf{a}, \mathbf{b}$ be the basis elements. Writing $Q(\mathbf{a}) = a/p$ and $Q(\mathbf{b}) = b/p$ with $p \nmid a, b$, one checks that $\mathcal{D}$ contains a nontrivial isotropic vector if and only if $-b/a$ is a quadratic residue modulo p. If it is not, L is maximal; otherwise choosing x with $x^2 \equiv -b/a \pmod p$ and setting $\mathbf{t} = x\mathbf{a} + \mathbf{b}$ yields an isotropic vector whose preimage $\mathbf{t}'$ produces a nontrivial p-overlattice $L + \langle \mathbf{t}' \rangle$.

The case $-\nu_p(D) \ge 3$ is analogous: either $-b/a$ is a square and we proceed as in the previous case, or one finds x, y with $ax^2 + by^2 + c \equiv 0 \pmod p$ and sets $\mathbf{t} = x\mathbf{a} + y\mathbf{b} + \mathbf{c}$. In every case $Q(\mathbf{t})$ is $\mathbb{Z}_p$-integral, so $L + \langle \mathbf{t}' \rangle$ is an integral overlattice at p and strictly larger than L when an isotropic vector exists. Thus the algorithm produces the desired overlattice. □

Theorem 5.4. *Let $\delta \in (0,1)$ be an arbitrary probability. Using Tonelli–Shanks* [DS73] *and Cipolla's algorithm to compute square roots modulo p, Algorithm 5 has time complexity*

$$O\big(n^{\psi} M(\nu_p(D)\log p) + (\log p)\log(1/\delta)\; M(\log p)\big)$$

with probability of failure at most δ.

Proof. Computing $\mathcal{D}$ costs $O(n^{\psi} M(\nu_p(D)\log p))$ by the same reasoning as before.

Computing square roots modulo p costs $O(\log p)$ field operations in many cases (for example when $p \equiv 3 \pmod 4$), while Tonelli–Shanks and Cipolla's runs in $O(\log p)$ worst-case field operations and can be made Las Vegas with expected running time of that order; each field operation has bit-cost $M(\log p)$. Repeating the randomized square-root attempt $O(\log(1/\delta))$ times reduces the failure probability to at most δ, so the cost of root-finding contributes $O((\log p)\log(1/\delta) M(\log p))$. Combining terms yields the stated bound. □

Theorem 5.5. *Algorithm 6 terminates and returns a $\mathbb{Z}_p$-maximal overlattice of L, where p is an odd prime.*

Proof. Algorithm 5 returns the same lattice when it is already maximal and otherwise returns a strictly larger p-overlattice. Hence, if $L' = L''$ we are done.

If not, we produce a strictly increasing chain of lattices

$$\Lambda_0 \subset \Lambda_1 \subset \Lambda_2 \subset \cdots,$$

and the determinants satisfy

$$\det(\Lambda_0) > \det(\Lambda_1) > \det(\Lambda_2) > \cdots .$$

Since determinants are positive integers, this strictly decreasing sequence cannot continue indefinitely. Therefore the loop terminates at a $\mathbb{Z}_p$-maximal overlattice of L. □

Theorem 5.6. *Let $\delta \in (0,1)$ be arbitrary and set $\lambda_p = \min(\nu_p(D), n)$. Then Algorithm 6 has time complexity*

$$O\left(\left(n^\psi + n\log(\nu_p(D))\right) M(\nu_p(D)\log p) + \lambda_p n^\psi M(\lambda_p \log p) + c_p M(\log p)\right)$$

with failure probability at most δ, where $c_p = \lambda_p(\log p)\log(1/\delta)$.

Proof. Saturation costs $O((n^\psi + n\log(\nu_p(D)))M(\nu_p(D)\log p))$ by Theorem 5.2. After saturation the p-adic determinant exponent is at most λ_p, and each application of `OP-Overlattice` reduces that exponent by at least 1, so the while-loop iterates at most λ_p times.

By Theorem 5.4, one iteration costs

$$O\big(n^\psi M(\lambda_p \log p) + (\log p)\log(1/\delta) M(\log p)\big),$$

and multiplying this by the number of iterations λ_p and adding the saturation cost yields the stated bound. □

Algorithm 7 implements `2-maximal(`L`)`, where L is a $\mathbb{Z}$-lattice. It constructs a $\mathbb{Z}$-lattice L' for which L'_2 is $\mathbb{Z}_2$-maximal and $L'_p = L_p$ for all odd primes p.

Theorem 5.7. *Algorithm 7 returns a $\mathbb{Z}_2$-maximal overlattice of L.*

Proof. The lattice with basis B' is an overlattice of L and ensures that the exponent of the discriminant is at most 4. Adding an isotropic subspace I to L' also yields an integral overlattice, and is used to accelerate the computation.

The final while-loop searches for nonzero isotropic vectors in the current discriminant group $\mathcal{D}'$; when such a vector is found its preimage is adjoined to the lattice, producing a strictly larger overlattice. As in the odd-prime case, this process strictly decreases the relevant 2-adic exponent and therefore must terminate at a $\mathbb{Z}_2$-maximal lattice. □

Theorem 5.8. *Let $\lambda_2 = \min(\nu_2(D), 4n)$. Then Algorithm 7 has time complexity*

$$O\Big((n^\psi + n\log(\nu_2(D)))M(\nu_2(D)) + \lambda_2 n^\psi M(\lambda_2) + 2^{\lambda_2}\Big).$$

Proof. As in Theorem 5.2, computing the initial discriminant group costs

$$O((n^\psi + n\log(\nu_2(D)))M(\nu_2(D))).$$

The cardinality of $\mathcal{D}'$ is bounded by both $2^{\nu_2(D)}$ and 2^{4n}, so scanning all vectors of $\mathcal{D}'$ to find an isotropic one takes $O(2^{\lambda_2})$ time. Each iteration of the outer while-loop recalculates the 2-discriminant group (costing $O(n^\psi M(\lambda_2))$) and there are at most $O(\lambda_2)$ iterations. Combining these costs yields the stated bound. □

Theorem 5.9. *Algorithm 8 produces a maximal overlattice of L.*

Algorithm 7: `2-maximal(`L`)`.

Data: A lattice L
$\mathcal{D} \leftarrow$ the 2-discriminant group of L;
$Q \leftarrow$ quadratic form on $\mathcal{D}$;
$B \leftarrow$ a basis of $\mathcal{D}$;
$B' \leftarrow$ empty list;
for *generator* g *in* B **do**
  $e \leftarrow$ the additive order of g;
  $v \leftarrow \nu_2(e)$;
  $q \leftarrow Q(g)$;
  $\delta \leftarrow$ the remainder when qe is divided by 2;
  Add $2^{\left\lfloor \frac{v+1}{2} \right\rfloor + \delta} g$ to B';
end
$L' \leftarrow$ the lattice with basis B';
$I \leftarrow$ isotropic subspace of L';
$L' \leftarrow L' + I$;
$\mathcal{D}' \leftarrow$ the 2-discriminant group of L';
while L' *is not maximal* **do**
  $v \leftarrow$ empty vector;
  for *vector* v' *in* $\mathcal{D}'$ **do**
    $v \leftarrow v'$;
    if $v' \neq 0$ *and* $Q(v') = 0$ **then**
      Break out of the for loop;
    end
  end
  if $Q(v) \neq 0$ **then**
    return L'
  else
    $v' \leftarrow$ a preimage of v in $L'^{\vee}$;
    $L' \leftarrow L' + \langle v' \rangle$;
    $\mathcal{D}' \leftarrow$ the 2-discriminant group of L'
  end
end
return L'

Proof. By Theorems 5.5 and 5.7 we know that the subroutines produce the correct $\mathbb{Z}_p$-maximal overlattices at every prime dividing $\det(G)$. Suppose for contradiction that the returned L' is not integrally maximal; then there exists an overlattice $L'' \supset L'$ with $\det(L'') \mid \det(L')$ and some prime p such that $L''_p \neq L'_p$. That prime must divide $\det(G)$, contradicting the maximality at p. Hence L' is maximal. □

Theorem 5.10. *Let $\delta \in (0,1)$ and let $\omega(D)$ denote the number of distinct prime factors of D. Set $\lambda_2 = \min(\nu_2(D), 4n)$ and $\lambda_p = \min(\nu_p(D), n)$ for odd primes p.*

Algorithm 8: `Maximal-Overlattice(`L`)`. Constructs an integral maximal overlattice of a lattice L.

Data: A lattice L
$G \leftarrow$ Gram matrix of L;
$D \leftarrow \det(G)$;
$L' \leftarrow L$;
if $2 \mid D$ **then**
  $L' \leftarrow$ `2-maximal(`L'`)`;
end
for *odd prime* p *dividing* d **do**
  $L' \leftarrow$ `OP-maximal(`L'`,` p`)`;
end
return L'

Then Algorithm 8 has time complexity

$$O\left(\sum_{p|D}\left(n^{\psi}+n\log(\nu_p(D))\right)M(\nu_p(D)\log p)+\lambda_p n^{\psi}M(\lambda_p\log p)+c_p\right)$$

with probability of failure at most δ, where

$$c_p=\begin{cases}\lambda_p(\log p)\log\left(\frac{\omega(D)}{\delta}\right)M(\log p) & p\neq 2,\\ 2^{\lambda_2} & p=2.\end{cases}$$

Proof. Apply Theorems 5.6 and 5.8 to each prime dividing D. Assigning a failure probability of at most $\delta/\omega(D)$ to each odd prime and summing the costs yields the stated global bound. □

Corollary 5.11. *Let $\delta \in (0,1)$. Assuming that arithmetic operations take constant time, i.e., $M(x)=1$ for all x, then Algorithm 8 has time complexity*

$$O\left(2^{4n}+n^{\psi+1}\log(D)+n\log(D)^2\log(1/\delta)\right)$$

with probability of failure δ.

Proof. We build off of the previous theorem. Since $M(x)=1$, $\lambda_2 \leq 4n$, and $\nu_2(D)=O(\log D)$, we get that

$$O(n^{\psi}+n\log(\nu_2(D)))M(\nu_2(D))+\lambda_2 n^{\psi}M(\lambda_2)+2^{\lambda_2})$$

is of the same magnitude as

$$O(n^{\psi}+n\log(\log(D))+n^{\psi+1}+2^{4n})=O(n\log(\log(D))+2^{4n}).$$

We now consider the portion of the time complexity within the summation. Since $\lambda_p \leq n$, $\nu_p(D)=O(\log D)$, and $\omega(D)=O(\log D)$, we get that

$$O\left(\left(n^{\psi}+n\log(\nu_p(D))\right)M(\nu_p(D)\log p)+\lambda_p n^{\psi}M(\lambda_p\log p)+c_p\right)$$

is of the same order of magnitude as

$$O\Big(n^{\psi}+n\log(\log(D))+n(n^{\psi}+(\log p)\log(\log(D)/\delta))\Big)$$

which implies a running time of

$$O\Big(n^{\psi+1}+n(\log p)\log(D)\log(1/\delta)\Big).$$

Note that we are summing this over all $p \mid D$. Clearly the number of primes dividing D is bounded by $\log D$. Likewise, the sum of $\log p$ for $p \mid D$ is bounded by $\log D$. Thus, the summation portion has time complexity

$$O(n^{\psi+1}\log(D) + n(\log D)^2 \log(1/\delta)).$$

Combining this with the portion of the time complexity outside of the summation yields the desired

$$O\left(2^{4n} + n^{\psi+1}\log(D) + n\log(D)^2\log(1/\delta)\right). \qquad \square$$

5.2. **Hanke algorithm.** In this section we describe the running-time analysis of Hanke's algorithm.

Throughout this section let Q be a rational quadratic form on an n-dimensional vector space V with (absolute) determinant D. Fix an $\mathcal{O}_F$-basis of V and express all Gram matrices with respect to that basis; thus a lattice $L \subset V$ is represented by an $n \times n$ matrix of coordinates. Let ψ denote the exponent for matrix multiplication. We assume that arithmetic on individual entries (including lcm of small denominators and modular reductions) is $O(1)$.

For a Gram matrix $G \in \mathbb{Q}^{n\times n}$ clear denominators by letting $d =$ lcm of the denominators of entries of G and set $A = dG \in \mathbb{Z}^{n\times n}$.

Lemma 5.12 (Cost of [Han13, Algorithm 3.8])**.** *The steps performed in* [Han13, Algorithm 3.8] *have costs*

- *lattice choice and denominator lcm:* $O(n^2)$*,*
- *single dual computation:* $O(n^\psi)$*,*
- *refinement loop:* $O\left(n^\psi \sum_{p\mid D}\log(\nu_p(D))\right)$*.*

Proof. Let $G \in \mathbb{Q}^{n\times n}$ be the Gram matrix for the chosen basis and let d be the least common multiple of the denominators of entries of G, so $A = dG \in \mathbb{Z}^{n\times n}$. Reading G (or A) requires n^2 entries, hence $O(n^2)$; computing the lcm of n^2 small denominators is $O(n^2)$ under our unit-cost assumption for small-integer arithmetic.

Computing the dual lattice $L^\sharp$ reduces to inverting A over $\mathbb{Q}$, followed by clearing denominators which can be done via Gaussian Elimination and thus takes $O(n^\psi)$ time assuming ring operations are constant time.

The refinement loop repeatedly modifies the lattice to reduce local invariants $m_{\mathfrak{p}}$. Each iteration requires one dual-lattice computation. An initial local invariant $m_{\mathfrak{p}}$ can be reduced to 1 in $O(\log m_{\mathfrak{p}})$ steps, and summing these valuations over all primes dividing the determinant yields $O\left(\sum_{p\mid D}\nu_p(D)\right)$ iterations overall. The cost per iteration is $O(n^\psi)$ and we multiply this by the number of iterations to get $O\left(n^\psi \sum_{p\mid D}\log(\nu_p(D))\right)$. $\square$

Lemma 5.13 (Cost of [Han13, Algorithm 3.11])**.** *Finding an isotropic vector in an isotropic n-dimensional quadratic space over $\mathbb{F}_q$ by the randomized single-variable reduction in Algorithm 3.11 requires expected time $O(n^2\log q)$ per prime q. Summing over primes $q \mid D$ gives expected time $O(n^2\log D)$.*

Proof. The randomized reduction picks $\vec{a}, \vec{m} \in \mathbb{F}_q^n$ uniformly at random and reduces the isotropy test to solving a quadratic equation in one variable

$$Q_B(\vec{a} + t\vec{m}) \equiv 0 \pmod{q}.$$

Evaluating the quadratic polynomial and computing its discriminant takes $O(n^2)$ field operations. Testing whether the discriminant is a square and extracting a square root (when it is) is done by modular exponentiation, which costs $O(\log q)$ field operations. Each random trial succeeds with constant probability (bounded below by some absolute constant), so the expected number of trials is $O(1)$. Therefore the expected cost per prime is $O(n^2 \log q)$; summing $\log q$ over primes dividing D yields the stated bound. □

Lemma 5.14 (Cost of [Han13, Algorithm 3.12])**.** *Extending a nonzero isotropic vector $\vec{v}$ to a basis as in Algorithm 3.12 costs $O(n^2)$ arithmetic operations per local factor, and $O(n^2\omega(D))$ when performed over all primes dividing D.*

Proof. With respect to the chosen basis compute the row $r = \vec{v}^T G$, where G is the Gram matrix; this multiplication costs $O(n)$. Find an index i with $r_i \neq 0$ (if one exists) and perform elementary column and row operations (shears and scalings) to make $\vec{v}$ the first basis vector and update the remaining basis vectors accordingly. Each elementary operation updates $O(n)$ entries and at most $O(n)$ such operations are required in the worst case, yielding $O(n^2)$ arithmetic operations per local factor. Repeating the same construction at each prime dividing D accumulates the factor $O(\omega(D))$ and gives $O(n^2\omega(D))$. □

Lemma 5.15 (Cost of [Han13, Algorithm 3.13, 3.14])**.** *Computing the radical (right kernel of the Gram matrix) costs $O(n^\psi)$. Splitting off up to $\lfloor n/2 \rfloor$ hyperbolic planes and performing the required local normalizations contributes at most $O(n^2)$ arithmetic operations per plane; hence, over all planes the cost is $O(n^3)$, and performing these operations for all local factors incurs an additional $O(\omega(D))$ multiplicative factor, giving $O(n^3\omega(D))$.*

Proof. Let $A = dG \in \mathbb{Z}^{n\times n}$ be the integral Gram matrix after clearing denominators. Computing the radical is computing the nullspace of A, which can be achieved by Gaussian elimination in $O(n^\psi)$ time. Once a basis for the radical is obtained, extracting one hyperbolic pair requires a sequence of elementary row/column operations (basis changes and shears) that update $O(n)$ entries per elementary operation; extracting a single hyperbolic plane therefore costs $O(n^2)$. Extracting up to $\lfloor n/2 \rfloor$ planes yields $O(n^3)$ work in the worst case. Repeating this process for each local factor (primes dividing D) introduces a multiplicative factor bounded by $O(\omega(D))$, hence the claimed bound $O(n^3\omega(D))$. □

Lemma 5.16 (Cost of [Han13, Algorithm 3.16])**.** *Let $A = dG \in \mathbb{Z}^{n\times n}$ be the integral Gram matrix obtained by clearing denominators of G. Computing the Smith normal form of A and performing all lifts of p-primary invariant directions to elements of $L^\sharp$ has total cost*

$$O(n^\psi + n\omega(D)\Omega(D)\log D).$$

Proof. For a fixed invariant factor $s_i = p^{k_1}k_2$ with $\gcd(k_2, p) = 1$, lifting a p-primary element of the discriminant to an element of $L^\sharp$ with vanishing non-p components amounts to solving systems of modular congruences. Each such system typically has $O(\omega(D))$ modulus sizes to consider (in the sense of valuations across primes), and solving a single such modular system can be implemented in $O(\omega(D)\log D)$ time. The total number of independent lifts is at most $O(n\Omega(D))$ (each prime contributes at most n directions and the sum of valuations is $O(\Omega(D))$). Multiplying these

estimates gives $O(n\omega(D)\Omega(D)\log D)$ for the lifting phase. Combining this with the $O(n^\psi)$ cost for SNF, see [Sto96], yields the stated bound. □

Lemma 5.17 (Cost of [Han13, Algorithm 4.6])**.** *Finding an even 2-neighbor, or finding that none exist, can be done in $O(n^\psi)$ time.*

Proof. Let L_{even} denote the sublattice of L consisting of all vectors $v \in L$ with $B(v,v) \equiv 0 \pmod 2$. Following the description in [Han11], every even 2-neighbor L' of L can be represented in the form

$$\frac{1}{2}\vec{w} + \{\vec{v} \in L : H(\vec{v},\vec{w}) \equiv 0 \pmod 2\},$$

for some $\vec{w} \in L_{\text{even}}^\perp$ satisfying the additional congruence condition $B(\vec{w},\vec{w}) \equiv 0 \pmod 8$.

However, finding $\vec{w}$ is not computationally difficult, as if a solution to $B(\vec{w},\vec{w}) \equiv 0 \pmod 8$ exists, it exists with only up to 5 components of $\vec{w}$ being nonzero. Therefore, we can enumerate 8^5 possible $\vec{w}$ and check those.

Computing the neighbor after finding $\vec{w}$ can be done in n^ψ. □

Theorem 5.18 (Overall time complexity of the Hanke algorithm)**.** *Under the hypotheses above, Hanke's algorithm runs in time*

$$O\big(n^3\omega(D) + n^\psi \sum_{p|D} \log(\nu_p(D)) + n\omega(D)\Omega(D)\log D\big).$$

Proof. Summing the complexities of the algorithms above, namely Lemma 5.12, Lemma 5.15, and Lemma 5.16, we get the result. □

6. Local modifications

In this section, we describe an algorithm implemented by Simon Brandhorst in [SageMath] to locally modify a lattice. That is, given a prime p, a $\mathbb{Z}$-lattice M such that M_p is maximal, and a Gram matrix G of a lattice isomorphic to M over $\mathbb{Q}_p$, we return the lattice L such that L_p has Gram matrix G and $L_q = M_q$ over all primes $q \neq p$.

Algorithm 9: Locally modifying a lattice

Data: A prime p, a $\mathbb{Z}_p$ maximal lattice M, and a Gram matrix G of a lattice isomorphic to M over $\mathbb{Q}_p$

$D \leftarrow$ the least positive integer such that $D \cdot G^{-1}$ is an integral matrix;
$d \leftarrow p^{\nu_p(D)}$;
$L \leftarrow$ integral lattice with standard basis and Gram matrix G;
$L_{\max} \leftarrow$ a $\mathbb{Z}_p$-maximal overlattice of L;
$G_{\max} \leftarrow$ the Gram matrix of $L_{\max}$;
$L_B \leftarrow$ the basis of $L_{\max}$;
$T_L \leftarrow$ the transformation matrix over $\mathbb{Z}_p$ such that $T_L G_{\max} T_L^\top$ is in normal form;
$B \leftarrow (T_L L_B)^{-1}$;
$G_M \leftarrow$ the Gram matrix of M;
$M_B \leftarrow$ the basis of M;
$T_M \leftarrow$ the transformation matrix over $\mathbb{Z}_p$ such that $T_M G_M T_M^\top$ is in normal form;
$B' \leftarrow B \cdot T_M \cdot M_B$;
$N \leftarrow$ the lattice spanned by B';
return $(N \cap M) + d \cdot M$

Theorem 6.1. *Algorithm 6 produces the correct output.*

Proof. We first note that $N \cap M$ and $d \cdot M$ are clearly both sublattices of M, so their sum is also a sublattice of M.

Now consider some prime $q \neq p$. Now in $\mathbb{Z}_q$, $d \cdot M$ is simply the same lattice as M, as d is a power of p and is therefore invertible in $\mathbb{Z}_q$. Thus, in $\mathbb{Z}_q$,

$$(N_q \cap M_q) + d \cdot M_q = M_q,$$

so we indeed find that local modification returns a lattice that is locally equal to M for all primes $q \neq p$.

Now we wish to show that in $\mathbb{Z}_p$, $(N_p \cap M_p) + d \cdot M_p$ is equivalent to the lattice with Gram matrix G. First, let

$$U_L = T_L L_B \quad \text{and} \quad U_M = T_M M_B.$$

Note that U_L and U_M are the bases for $L_{\max}$ and M respectively when the normal form is achieved. Further, it is clear that

$$B' = U_L^{-1} U_M.$$

Now observe that the normal form at p is uniquely determined by the rational invariants [MM09, Prop. 4.8]. Thus, since $L_{\max}$ and M are both $\mathbb{Z}_p$-maximal lattices, and $L_{\max}$ is rationally equivalent to L, which is itself rationally equivalent to M, we find that $L_{\max}$ and M have the same normal form J. Thus, if M_I is the inner product matrix of M,

$$\begin{aligned} U_L G U_L^\top = J = U_M M_I U_M^\top &\implies (U_L^{-1} U_M) M_I (U_L^{-1} U_M)^\top = G \\ &\implies B' M_I B'^\top = G \end{aligned}$$

In particular, B' is the basis of the lattice in the space of M that has Gram matrix G. Thus, N is the integral lattice with Gram matrix G.

To finish, it suffices to show that in $\mathbb{Z}_p$, $(N \cap M) + d \cdot M$ is the same lattice as N. First, observe that $L_{\max}$ contains L, and therefore each vector in the basis of L can be expressed as a linear combination in $\mathbb{Z}_p$ of the vectors in the basis of $L_{\max}$. But the basis of L is the standard basis, i.e., the identity matrix I. Thus, there exists a matrix Q with all entries in $\mathbb{Z}_p$ such that

$$U_L Q = I \implies U_L^{-1} = Q.$$

Therefore, U_L^{-1} has all entries in $\mathbb{Z}_p$. Now since $B' = U_L^{-1} U_M$, each vector in the basis B' can be expressed as a linear combination in $\mathbb{Z}_p$ of vectors in the basis U_M. Thus, N must be contained in M, so in fact, $N \cap M = N$ in $\mathbb{Z}_p$.

To finish, we note that all entries in dU_L must be in $\mathbb{Z}_p$. Notably, if we let $dU_L = Q$, then

$$d \cdot U_M = Q U_L^{-1} U_M \implies d \cdot U_M = QB.$$

Since Q has all entries in $\mathbb{Z}_p$, each vector in the basis $d \cdot U_M$ can be expressed as a linear combination in $\mathbb{Z}_p$ of vectors in the basis B. Thus, $d \cdot M$ must be contained in N.

But this just means that in $\mathbb{Z}_p$,

$$(N_p \cap M_p) + d \cdot M_p = N_p + d \cdot M_p = N_p,$$

so we indeed find that in $\mathbb{Z}_p$, $(N_p \cap M_p) + d \cdot M_p$ is the lattice with Gram matrix G. This completes our proof. □

Theorem 6.2. *Suppose the time complexity of finding the $\mathbb{Z}_p$-maximal overlattice of a lattice L with rank n and determinant D is $\rho(n, D, p)$. Then, Algorithm 9 has time complexity*

$$O(n^\psi + \rho(n, D, p))$$

Proof. By definition, taking the $\mathbb{Z}_p$-maximal overlattice of L has time complexity $\rho(n, D, p)$. The remainder of the algorithm simply has time complexity $O(n^\psi)$, which yields our desired time complexity. □

Corollary 6.3. *Let $\delta \in (0, 1)$ be some arbitrary probability. Assuming that arithmetic takes $O(1)$ time, i.e., $M(x) = 1$ for all x, then with the Algorithms 6 and 7 for the maximal overlattice, Algorithm 9 has time complexity*

(1) $O\left(n \log(\nu_p(d)) + \lambda_p \left(n^\psi + (\log p) \log(1/\delta)\right)\right)$ *for odd primes p*
(2) $O\left(n \log(\nu_2(d)) + \lambda_2 n^\psi + 2^{\lambda_2}\right)$ *for $p = 2$*

where

$$\lambda_p = \begin{cases} \min(\nu_2(d), 4n) & \text{if } p = 2 \\ \min(\nu_p(d), n) & \text{otherwise} \end{cases}$$

Proof. With Algorithms 6 and 7 for the maximal overlattice, we know from Theorems 5.6 and 5.8 that $\rho(n, d, p)$ has time complexity

(1) $O\left(n \log(\nu_p(d)) + \lambda_p \left(n^\psi + (\log p) \log(1/\delta)\right)\right)$ for odd primes p, with probability of failure at most δ
(2) $O\left(n \log(\nu_2(d)) + \lambda_2 n^\psi + 2^{\lambda_2}\right)$ for $p = 2$

where we assumed $M(x) = 1$ for all x. Note that the remainder of the algorithm has time complexity $O(n^\psi)$, which is subsumed by the time complexity for finding the $\mathbb{Z}_p$-maximal overlattice of L, so we are done. □

Corollary 6.4. *With Hanke's Algorithm, Algorithm 9 has time complexity*

$$O(n^3 + n^\psi \log(\nu_p(D))).$$

Proof. This follows immediately from a simple modification of Theorem 5.18 for maximal overlattices over $\mathbb{Z}_p$. □

7. Counting Genus Symbols

We aim to asymptotically compute the number of genus symbols with a fixed rank and determinant. In this section, we describe how to compute the number of genus symbols for an odd prime p when $n > \nu_p(D)$, and we give upper and lower bounds in the case $p = 2$.

Theorem 7.1. *Let $n, d \in \mathbb{Z}$ with $n > 0$. Let $p \neq 2$ be an odd prime dividing D with $\nu_p(D) < n$. Let $L_p(n, D)$ denote the number of valid p-adic genus symbols of rank n and determinant D. Then*

$$L_p(n, D) \sim \frac{1}{8\,\nu_p(D)} \exp\left(\pi\sqrt{\nu_p(D)}\right).$$

Proof. Fix an odd prime $p \mid D$ and write $k = \nu_p(D)$. A p-adic genus symbol may be described by a choice, for each $i \geq 1$, of a nonnegative integer d_i equal to the multiplicity of the p^i-constituent and, for each nonzero constituent, of a sign $\varepsilon_{p^i} \in \{\pm 1\}$ (the value of the Legendre symbol of the local determinant). The constraints on dimensions are

$$\sum_{i \geq 1} i\, d_i = k,$$

and, if t denotes the number of indices i with $d_i > 0$, then the vector of signs contributes a factor of $2^{\,t-1}$ (one sign is determined by the determinant condition). Thus the combinatorial count we need is, up to an innocuous factor of $1/2$, the coefficient c_k in the power series expansion of the generating function

$$G(x) = \prod_{i \geq 1} \big(1 + \sum_{t \geq 1} 2x^{ti}\big).$$

The inner sum is a geometric series, so

$$G(x) = \prod_{i \geq 1} \frac{1 + x^i}{1 - x^i}.$$

Define integers a_i by $a_i = 2$ when i is odd and $a_i = 1$ when i is even; then

$$G(x) = \prod_{i \geq 1} (1 - x^i)^{-a_i}.$$

To obtain the asymptotic behaviour of c_k we apply Meinardus's theorem (see, e.g., [Mei54]; for hypotheses and detailed statements see [GSE08]). The Dirichlet series associated to the sequence (a_i) is

$$L(s) = \sum_{i \geq 1} a_i i^{-s} = \sum_{\substack{i \geq 1 \\ i \text{ odd}}} \frac{2}{i^s} + \sum_{\substack{i \geq 2 \\ i \text{ even}}} \frac{1}{i^s} = (2 - 2^{-s})\zeta(s),$$

which is meromorphic for $\Re(s) > 0$ with the sole pole inherited from $\zeta(s)$ at $s = 1$. The residue of $L(s)$ at $s = 1$ is

$$A = \operatorname{Res}_{s=1} L(s) = 2 - 2^{-1} = \tfrac{3}{2}.$$

One checks the remaining analytic hypotheses of Meinardus (growth in vertical strips and suitable bounds for the generating function off the positive real axis) hold because they reduce to classical estimates for the Riemann zeta function; we omit the routine verification.

Evaluate L and its derivative at $s = 0$. Since $\zeta(0) = -\frac{1}{2}$ and $\zeta'(0) = -\frac{1}{2}\ln(2\pi)$, and noting $(2 - 2^{-s})'|_{s=0} = \ln 2$, we obtain

$$L(0) = -\tfrac{1}{2}, \qquad L'(0) = \ln 2 \cdot (-\tfrac{1}{2}) + 1 \cdot \left(-\tfrac{1}{2}\ln(2\pi)\right) = -\tfrac{1}{2}\ln(4\pi).$$

Apply Meinardus with parameter $r = 1$ (the pole order). The theorem gives, for $k \to \infty$,

$$c_k \sim C\, k^{-(D(0)+1)/2} \exp\left((r+1)\left(A\Gamma(r+1)\zeta(r+1)\right)^{1/(r+1)} k^{r/(r+1)}\right),$$

where the constant C can be written in terms of A, $L(0)$ and $L'(0)$. Substituting $r = 1$, $A = \frac{3}{2}$, $\Gamma(2) = 1$ and $\zeta(2) = \pi^2/6$ yields the exponential factor

$$\exp\left(2\sqrt{A\Gamma(2)\zeta(2)}\,\sqrt{k}\right) = \exp\left(\pi\sqrt{k}\right).$$

A direct evaluation of the prefactor (standard in the Meinardus set-up) gives

$$c_k \sim \frac{1}{8}\, k^{-1} \exp\left(\pi\sqrt{k}\right).$$

Recalling the factor $1/2$ introduced earlier and the identification $k = \nu_p(D)$, we obtain

$$L_p(n, D) \sim \frac{1}{8\,\nu_p(D)} \exp\left(\pi\sqrt{\nu_p(D)}\right),$$

which is the claimed asymptotic. □

We may now return to the original problem. Ignoring $p = 2$ for the moment, the number of odd prime local genus symbols is asymptotically the product over odd primes $p \mid d$ of the factors above. Writing $\Omega_0(D) = \#\{p \neq 2 : p \mid D\}$ and

$$\Omega_{1/2}(D) = \sum_{2 \neq p \mid D} \sqrt{\nu_p(D)},$$

and noting that the local determinant conditions introduce a mild divisor-type factor in the denominator which we denote by $\tau(D) = \prod_{p \neq 2} \nu_p(D)$, one obtains the estimate

$$\prod_{2 \neq p \mid D} \frac{1}{8\nu_p(D)} \exp\left(\pi\sqrt{\nu_p(D)}\right) = \frac{1}{8^{\Omega_0(D)}\tau(D)} \exp\left(\pi\Omega_{1/2}(D)\right).$$

Finally, allowing choice of real signature (there are $n + 1$ possible signatures for a rank n form) multiplies the count by $n + 1$. Thus, denoting by $L^{(2)}(n, D)$ the total number of genus symbols ignoring $p = 2$, we have

$$L^{(2)}(n, D) \sim \frac{(n+1)\exp\left(\pi\Omega_{1/2}(D)\right)}{8^{\Omega_0(D)}\tau(D)}.$$

7.1. $p = 2$ **case.** We give a crude but explicit bound for the number of 2-adic genus symbols. The counting is less regular at $p = 2$ because of the richer train /compartment structure described in Conway and Sloane; our estimate deliberately overcounts several choices to obtain a simple closed form.

Theorem 7.2. *Let $n, D \in \mathbb{Z}$ with $n > 0$ and assume $\nu_2(D) < n$. Let $L_2(n, D)$ be the number of valid 2-adic genus symbols of rank n and determinant D. Put $k = \nu_2(D)$. Then*

$$\frac{1}{4\sqrt{3}\,k}\exp\left(\pi\sqrt{\tfrac{2k}{3}}\right) \le L_2(n, D) \le \frac{3}{2\sqrt{51}\,k}\left(\frac{3+\sqrt{17}}{2}\right)^k \exp\left(\pi\sqrt{\tfrac{2k}{3}}\right). \tag{7.3}$$

Proof. Write $k = \nu_2(D)$ and retain the notation of Conway–Sloane (trains and compartments). Any genus symbol determines a partition of k via the multiplicities $(a_1, a_2, \ldots, a_k)$ satisfying

$$a_1 + 2a_2 + \cdots + ka_k = k.$$

Let $p(k)$ denote the number of partitions of k into positive integers. By the Hardy–Ramanujan asymptotic (with a classical error term; see e.g. [HR17]),

$$p(k) \asymp \frac{1}{4k\sqrt{3}}\exp\left(\pi\sqrt{\tfrac{2k}{3}}\right)\left(1 + O(k^{-1/2})\right).$$

Hence the number of possible rank assignments (the (a_i)-vectors) is $p(k)$, and this gives the lower bound in (7.3) since at least one choice of types/oddities/signs is compatible with any fixed partition.

For the upper bound we overcount: for each part we choose a type (Type I or II), for each compartment we choose an oddity (at most 4 choices), and for each train we choose a sign (we allow a factor of 2 even when the train might have zero total rank; this overcounts but simplifies the bound). Encode the type sequence of length $k+2$ as a binary string with 1 indicating Type I and 0 Type II, for valuations between 0 and $k+1$, so that the last digit is always 0. A compartment corresponds to an occurrence of the substring 10; a pair of adjacent Type II blocks (substring 00) contributes an extra factor coming from train structure. Denote by $N_b(k, x, y)$ the number of binary strings of length $k+1$ ending with b with x occurrences of 10 and y occurrences of 00, and let $S_b(k) = \sum_{x,y} N_b(k, x, y)4^x 2^y$. Then

$$\begin{aligned} N_0(k, x, y) &= N_0(k-1, x, y-1) + N_1(k-1, x-1, y), \\ N_1(k, x, y) &= N_0(k-1, x, y) + N_1(k-1, x, y), \end{aligned}$$

showing that $S_0(k) = 2S_0(k-1) + 4S_1(k-2)$ and $S_1(k) = S_0(k-1) + S_1(k-1)$, hence $S_0(k) = 3S_0(k-1) + 2S_0(k-2)$. Since $S_0(0) = 0$ and $S_0(1) = 6$, it follows that $S_0(k) = \frac{6}{\sqrt{17}}(\lambda_1^k - \lambda_2^k)$, where $\lambda_{1,2} = \frac{3\pm\sqrt{17}}{2}$. Multiplying this combinatorial bound by the partition count $p(k)$ (to account for choices of ranks) produces the stated upper bound. Combining the two estimates gives (7.3). □

Before we conclude the proof of our main theorem, for the running time complexity estimates, the following Lemma will be useful.

Lemma 7.4. *Let us denote*

$$f(D) = \prod_{p|D} \frac{\exp\left(\pi\sqrt{\nu_p(D)}\right)}{\nu_p(D)}. \tag{7.5}$$

Then for all D, we have

$$f(D) \leq D^{\frac{\pi(1+o(1))}{\log\log D}}.$$

Proof. Applying Cauchy-Schwarz, we get

$$\log f(D) \leq \pi \sum_{p|D} \sqrt{\nu_p(D)} \leq \pi\sqrt{\omega(D)\Omega(D)}.$$

Since $\omega(D)\Omega(D)$ is maximized at primorials, we have

$$\omega(D)\Omega(D) \leq (1+o(1)) \left(\frac{(\log D)^2}{(\log\log D)^2}\right),$$

and it follows that

$$\log f(D) \leq \pi(1+o(1))\frac{\log D}{\log\log D},$$

yielding the result. □

7.2. **Proof of Theorem 1.1.** We are now ready to prove Theorem 1.1.

Proof. We combine the results from Corollary 4.33, Theorem 7.1 and Theorem 7.2. Here, we use e_p as a shorthand for $\nu_p(D)$, and let $\lambda = \frac{3+\sqrt{17}}{2}$. Since the signature has $O(n)$ possible values, the counts of local symbols in Theorem 7.1 (for odd primes) and in Theorem 7.2 (for $p = 2$) show that there are at most $O\left(n\lambda^{e_2} f(D)\right)$ genera with rank n and determinant D, where $f(D)$ is as in (7.5). By Corollary 4.33, computing a genus in each representative takes

$$O\left(n^3\frac{\log D}{\log\log D} + n^\psi \log D + n\frac{(\log D)^3}{\log\log D}\right) = O\left(n^3\frac{\log D}{\log\log D}\right)$$

time, where we use the assumption that $\log D \ll n$ to see the first term is dominating. Multiplying everything together, the running time complexity is

$$O\left(n^4\lambda^{e_2} f(D)\frac{\log D}{\log\log D}\right).$$

Using Lemma 7.4 and $e_2 \leq \log_2 D$, this is the same as $n^4\lambda^{\log_2 D} D^{\frac{\pi(1+o(1))}{\log\log D}}$, which can be further simplified to $n^4 D^{\log_2\lambda + \frac{\pi(1+o(1))}{\log\log D}}$, proving the claim. □

7.3. **Proof of Theorem 1.2.** While a naive application of Theorem 1.1 would increase the exponent of D by 1, refining the above estimates, we give a proof of Theorem 1.2.

Proof. Fix $d \leq D$. By Corollary 4.33, Algorithm 3 has time complexity

$$O\left(n^3\omega(d) + n^\psi \sum_{p|d} \log(\nu_p(d)) + n\omega(d)\Omega(d)\log d\right) = O(n^3\omega(D)),$$

where we use the hypothesis $\log D \ll n$ to see that $n^3\omega(d)$ is the dominant term, as $\omega(d)\Omega(d)\log d \leq n^2\omega(d)$ and $\sum_{p|d}\log(\nu_p(d)) \leq \omega(d)\log\log d$, which is also less than $\omega(d)n^{3-\psi}$.

Therefore, applying Theorem 7.1 and Theorem 7.2 again, the running time complexity for finding representatives for each genus with a given d is

$$O\left(n^4\lambda^{e_2} f(d)\omega(d)\right) = n^4 d^{\log_2\lambda} f(d)\omega(d).$$

Let $F(D) = \sum_{d=1}^{D} f(d)\omega(d)$. Then Abel summation yields

$$\sum_{d=1}^{D} d^{\log_2 \lambda} f(d)\omega(d) = D^{\log_2 \lambda} F(D) - \log_2 \lambda \int_1^D t^{\log_2 \lambda - 1} F(t)\, dt. \tag{7.6}$$

On the other hand, setting $L(z,s) = \sum_{n=1}^{\infty} \frac{f(n) z^{\omega(n)}}{n^s}$, multiplicativity of $f(n)$ and $z^{\omega(n)}$ yields the Euler product

$$L(z,s) = \prod_p \sum_{k=0}^{\infty} \frac{f(p^k) z}{p^{ks}} = \prod_p \sum_{k=0}^{\infty} \frac{z e^{\pi\sqrt{k}}}{p^{ks}}.$$

Writing $L(z,s) = \zeta(s)^{z e^{\pi}} G(z,s)$, we have

$$\sum_{d=1}^{\infty} \frac{f(d)\omega(d)}{d^s} = \frac{\partial}{\partial z} L(z,s)|_{z=1} = \zeta(s)^{e^{\pi}} \frac{\partial}{\partial z} G(z,s)|_{z=1} + e^{\pi} \zeta(s)^{e^{\pi}} \log \zeta(s) G(1,s).$$

Since $G(z,s)$ converges uniformly for all $\Re s > \frac{1}{2}$, we get that

$$\sum_{d=1}^{\infty} \frac{f(d)\omega(d)}{d^s} \sim e^{\pi} G(1,1) \zeta(s)^{e^{\pi}} \log \zeta(s) \sim -e^{\pi} G(1,1) (s-1)^{-e^{\pi}} \log(s-1)$$

as $s \to 1^+$. By [Del54, Théoréme IV], it follows that

$$F(D) \sim e^{\pi} \frac{G(1,1)}{\Gamma(e^{\pi})} D (\log D)^{e^{\pi}-1} \log\log D.$$

Plugging that back into (7.6) we see that

$$\sum_{d=1}^{D} d^{\log_2 \lambda} f(d)\omega(d) = O(D^{\log_2 \lambda + 1} (\log D)^{e^{\pi}-1} \log\log D),$$

hence the result. □

8. Acknowledgments

The authors would like to thank the MIT PRIMES program for facilitating this research opportunity, and in particular Dr. Thomas Rüd and Prof. Tanya Khovanova for their helpful comments. Assaf was supported by a grant from the Simons Foundation (SFI-MPS-Infrastructure-00008651, AS).

References

[BI58] Heinrich Brandt and Oskar Intrau, *Tabellen reduzierter positiver ternärer quadratischer Formen*, Abh. Sächs. Akad. Wiss. Math.-Nat. Kl. **45** (1958), no. 4, 261 (German).

[Bue89] Duncan A. Buell, *Binary Quadratic Forms: Classical Theory and Modern Computations*, Springer-Verlag, 1989.

[Cas78] J. W. S. Cassels, *Rational quadratic forms*, London Mathematical Society Monographs, vol. 13, Academic Press, Inc. [Harcourt Brace Jovanovich, Publishers], London-New York, 1978.

[CS99] J. H. Conway and N. J. A. Sloane, *Sphere packings, lattices and groups*, 3rd ed., Grundlehren der mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences], vol. 290, Springer-Verlag, New York, 1999. With additional contributions by E. Bannai, R. E. Borcherds, J. Leech, S. P. Norton, A. M. Odlyzko, R. A. Parker, L. Queen and B. B. Venkov.

[Cox22] David A. Cox, *Primes of the Form $x^2 + ny^2$: Fermat, Class Field Theory, and Complex Multiplication*, 3rd ed., American Mathematical Society, 2022.

[Del54] Hubert Delange, *Généralisation du théorème de Ikehara*, Ann. Sci. École Norm. Sup. (3) **71** (1954), 213–242 (French).

[DH14a] Chandan K. Dubey and Thomas Holenstein, *Generating a Quadratic Forms from a Given Genus*, CoRR **abs/1409.6913** (2014), available at `1409.6913`.

[DH14b] ———, *Sampling a Uniform Random Solution of a Quadratic Equation Modulo p^k*, CoRR **abs/1404.0281** (2014), available at `1404.0281`.

[DH14c] ———, *Computing the p-adic Canonical Quadratic Form in Polynomial Time*, CoRR **abs/1409.6199** (2014), available at `1409.6199`.

[EH19] Andrew G. Earnest and Anna Haensch, *Classification of one-class spinor genera for quaternary quadratic forms*, Acta Arith. **191** (2019), no. 3, 259–287, DOI 10.4064/aa180309-19-2.

[Eic52] Martin Eichler, *Die Ähnlichkeitsklassen indefiniter Gitter*, Math. Z. **55** (1952), 216–252, DOI 10.1007/BF01268656 (German).

[Gau01] Carl Friedrich Gauss, *Disquisitiones Arithmeticae*, Gerh. Fleischer, jun., 1801 (Latin).

[GSE08] Boris L. Granovsky, Dudley Stark, and Michael Erlihson, *Meinardus' theorem on weighted partitions: extensions and a probabilistic proof*, Adv. in Appl. Math. **41** (2008), no. 3, 307–328, DOI 10.1016/j.aam.2007.11.001.

[Han11] Jonathan Hanke, *Quadratic Forms and Automorphic Forms – Expanded Notes from the 2009 Arizona Winter School*, 2009 Arizona Winter School, 2011. `https://swc-math.github.io/aws/2009/09HankeNotes.pdf`.

[Han13] ———, *Algorithms for computing maximal lattices in bilinear (and quadratic) spaces over number fields*, Diophantine methods, lattices, and arithmetic theory of quadratic forms, Contemp. Math., vol. 587, Amer. Math. Soc., Providence, RI, 2013, pp. 111–130, DOI 10.1090/conm/587/11697.

[HR17] G. H. Hardy and S. Ramanujan, *Asymptotic formulæ in combinatory analysis [Proc. London Math. Soc. (2) **16** (1917), Records for 1 March 1917]*, Collected papers of Srinivasa Ramanujan, AMS Chelsea Publ., Providence, RI, 1917, pp. 244.

[HvdH21] David Harvey and Joris van der Hoeven, *Integer multiplication in time $O(n \log n)$*, Annals of Mathematics **193** (2021), no. 2, 563 – 617, DOI 10.4007/annals.2021.193.2.4.

[Jon35] Burton W. Jones, *A Table of Eisenstein-reduced Positive Ternary Quadratic Forms of Determinant* **97-98** (1935).

[KL13] Markus Kirschmer and David Lorch, *Single-class genera of positive integral lattices*, LMS J. Comput. Math. **16** (2013), 172–186, DOI 10.1112/S1461157013000107.

[LMFDB] The LMFDB Collaboration, *The L-functions and Modular Forms Database*, 2025. `https://www.lmfdb.org/`.

[Mei54] Günter Meinardus, *Asymptotische Aussagen über Partitionen*, Math. Z. **59** (1954), 388–398, DOI 10.1007/BF01180268 (German).

[MM09] Rick Miranda and David R. Morrison, *Embeddings of Integral Quadratic Forms*, 2009. Available at `https://web.math.ucsb.edu/~drm/manuscripts/eiqf.pdf`.

[NS] Gabriele Nebe and Neil J.A. Sloane, *Catalogue of lattices*. Available at `http://www.research.att.com/~njas/lattices/abbrev.html`, last accessed June 2025.

[Nip91] Gordon L. Nipp, *Quaternary quadratic forms*, Springer-Verlag, New York, 1991. Computer generated tables; With a 3.5″ IBM PC floppy disk.

[O'M63] O. Timothy O'Meara, *Introduction to Quadratic Forms*, Die Grundlehren der mathematischen Wissenschaften, vol. 117, Springer-Verlag, 1963.

[PS97] W. Plesken and B. Souvignier, *Computing isometries of lattices*, J. Symbolic Comput. **24** (1997), no. 3-4, 327–334, DOI 10.1006/jsco.1996.0130. Computational algebra and number theory (London, 1993).

[SageMath] The Sage Developers, *Sagemath, the Sage Mathematics Software System (Version 10.5)*, 2024. `https://www.sagemath.org`.

[SP91] Rainer Schulze-Pillot, *An algorithm for computing genera of ternary and quaternary quadratic forms*, Proceedings of the 1991 international symposium on Symbolic and algebraic computation, 1991, pp. 134–143.

[DS73] Daniel Shanks, *Five Number-theoretic Algorithms*, Proceedings of the Second Manitoba Conference on Numerical Mathematics (1973), 51–70.

[Sto96] Arne Storjohann, *Near optimal algorithms for computing Smith normal forms of integer matrices*, Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation, 1996, pp. 267–274, DOI 10.1145/236869.237084.