SIAM J. COMPUT.
Vol. 39, No. 3, pp. 874–903

AN LLL ALGORITHM WITH QUADRATIC COMPLEXITY*

PHONG Q. NGUYEN† AND DAMIEN STEHLÉ‡

Abstract. The Lenstra–Lenstra–Lovász lattice basis reduction algorithm (called LLL or L^3) is a fundamental tool in computational number theory and theoretical computer science, which can be viewed as an efficient algorithmic version of Hermite's inequality on Hermite's constant. Given an integer d-dimensional lattice basis with vectors of Euclidean norm less than B in an n-dimensional space, the L^3 algorithm outputs a reduced basis in $O(d^3 n \log B \cdot \mathcal{M}(d \log B))$ bit operations, where $\mathcal{M}(k)$ denotes the time required to multiply k-bit integers. This worst-case complexity is problematic for applications where d or/and $\log B$ are often large. As a result, the original L^3 algorithm is almost never used in practice, except in tiny dimension. Instead, one applies floating-point variants where the long-integer arithmetic required by Gram–Schmidt orthogonalization is replaced by floating-point arithmetic. Unfortunately, this is known to be unstable in the worst case: the usual floating-point L^3 algorithm is not even guaranteed to terminate, and the output basis may not be L^3-reduced at all. In this article, we introduce the L^2 algorithm, a new and natural floating-point variant of the L^3 algorithm which provably outputs L^3-reduced bases in polynomial time $O(d^2 n(d+\log B) \log B \cdot \mathcal{M}(d))$. This is the first L^3 algorithm whose running time (without fast integer arithmetic) provably grows only quadratically with respect to $\log B$, like Euclid's gcd algorithm and Lagrange's two-dimensional algorithm.

Key words. lattice reduction, L^3, floating-point arithmetic

AMS subject classifications. 11Y16, 11H06, 11H55

DOI. 10.1137/070705702

1. Introduction.

For $1 \leq d \leq n$, let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be linearly independent vectors in $\mathbb{R}^n$: often, $n = d$ or $n = O(d)$. We denote by $L(\mathbf{b}_1, \ldots, \mathbf{b}_d) = \{\sum_{i=1}^d x_i \mathbf{b}_i \mid x_i \in \mathbb{Z}\}$ the set of all integer linear combinations of the $\mathbf{b}_i$'s. This set is called a *lattice* of $\mathbb{R}^n$ and $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ a *basis* of that lattice. A lattice basis is usually not unique, but all the bases have the same number d of elements, called the *dimension* of the lattice. If $d \geq 2$, there are infinitely many bases, but some are more interesting than others: they are called *reduced*. Roughly speaking, a reduced basis is a basis made of reasonably short vectors which are almost orthogonal. Finding good reduced bases has proved invaluable in many fields of computer science and mathematics (see the books [12, 6]), particularly in cryptology (see [36, 29]). This problem is known as *lattice reduction* and can intuitively be viewed as a vectorial generalization of gcd computations.

Hermite [13][1] published in 1850 the first lattice reduction algorithm in arbitrary dimension, by trying to generalize Lagrange's two-dimensional algorithm [21][2] (often

*Received by the editors October 18, 2007; accepted for publication (in revised form) January 21, 2009; published electronically August 26, 2009. A preliminary version of this work appeared in conference proceedings as [34].
http://www.siam.org/journals/sicomp/39-3/70570.html

†INRIA & Ecole normale supérieure, DI, 45 rue d'Ulm, 75005 Paris, France (pnguyen@di.ens.fr, http://www.di.ens.fr/~pnguyen/). Part of this author's work was supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT.

‡CNRS & Ecole normale supérieure de Lyon/LIP/INRIA Arenaire/Université de Lyon, 46 allée d'Italie, F-69364 Lyon Cedex 07, France, and ACAC/Department of Computing, Macquarie University, Sydney NSW 2109, Australia, and Department of Mathematics and Statistics, University of Sydney, Sydney NSW 2006, Australia (damien.stehle@ens-lyon.fr, http://perso.ens-lyon.fr/damien.stehle/).

[1]This text is also available in the first volume (pages 100–121) of Hermite's collected works, published by Gauthier-Villars. The exact date of the letters is unknown, but the first letter seems to have been written in 1847.

[2]This text can also be found on pages 695–795 of the third volume of Lagrange's collected works.

wrongly attributed to Gauss). In his famous letters [13] to Jacobi, Hermite actually described two reduction algorithms: the first letter presented an algorithm to show the existence of Hermite's constant (which guarantees the existence of short lattice vectors), while the second letter presented a slightly different algorithm to further prove the existence of lattice bases with bounded orthogonality defect. The subject had a revival around 1981 with Lenstra's celebrated work on integer programming [24, 25], which was, among others, based on a novel lattice reduction technique. This reduction technique, which can be found in the preliminary version [24] of [25], turns out to be a relaxed variant of Hermite's second algorithm, perhaps because the reduction goal of [24] was to bound the orthogonality defect. Lenstra's algorithm is polynomial-time only for fixed dimension, which was, however, sufficient in [24]. This inspired Lovász to develop a polynomial-time variant of the algorithm, which reached a final form in the seminal paper [23], where Lenstra, Lenstra, and Lovász applied it to factor rational polynomials in polynomial time, from which the name LLL or L^3 comes: the L^3 algorithm was the first polynomial-time reduction algorithm, and it provides bases almost as reduced as Hermite's second algorithm. Further refinements of the L^3 algorithm were later proposed, notably by Kaltofen [17], Schnorr [39, 40], Gama et al. [8], and Gama and Nguyen [9]: [17, 40] improve the running time of L^3, and [39, 8, 9] improve the output quality of L^3 while keeping polynomial-time complexity. Reduction algorithms (in particular, L^3) have arguably become the most popular tool in public-key cryptanalysis (see the survey [36]). In the past 25 years, they have been used to break many public-key cryptosystems, including knapsack cryptosystems [37], RSA in particular settings [7, 5, 4], DSA and similar signature schemes in particular settings [15, 32], etc.

Given as input an integer d-dimensional lattice basis whose n-dimensional vectors have norm less than B, the L^3 algorithm outputs a so-called L^3-reduced basis in time $O(d^3 n \log B \cdot \mathcal{M}(d \log B))$, using arithmetic operations on integers of bit-length $O(d \log B)$. Kaltofen [17] improved the analysis and obtained the complexity bound $O(\frac{d^4 n \log^2 B}{d + \log B} \cdot \mathcal{M}(d + \log B))$, proving that the worst-case multiplications involve operands of bit-sizes $O(d \log B)$ and $O(d + \log B)$. This worst-case complexity turns out to be problematic in practice, especially for lattices arising in cryptanalysis where d or/and $\log B$ are often large. For instance, in a typical RSA application of Coppersmith's lattice-based theorem [7], we may need to reduce a 64-dimensional lattice with vectors having coefficients whose size is a multiple of an RSA modulus (at least 2048 bits), in which case the complexity of [23] becomes "$d^5 n \log^3 B = 2^{69}$," with naive integer multiplication. As a result, the original L^3 algorithm is seldom used in practice. Instead, one applies floating-point variants, where the long-integer arithmetic required by Gram–Schmidt orthogonalization (which plays a central role in L^3) is replaced by floating-point arithmetic on much smaller numbers. The use of floating-point arithmetic in the L^3 algorithm dates back to the early eighties when the L^3 algorithm was used to solve low-density knapsacks [20]. Unfortunately, floating-point arithmetic may lead to stability problems, both in theory and practice, especially when the dimension increases: the running time of floating-point variants of the L^3 algorithm such as that in Schnorr and Euchner [42] is not guaranteed to be polynomial or even finite, and the output basis may not be L^3-reduced at all. This phenomenon is well known to L^3 practitioners and is usually solved by sufficiently increasing the precision. For instance, experimental problems arose during the cryptanalyses [31, 30] of lattice-based cryptosystems, which led to improvements in Shoup's NTL library [45].

There is, however, one provable floating-point variant of L^3, due to Schnorr [40], which significantly improves the worst-case complexity. Schnorr's variant outputs an approximate L^3-reduced basis in time $O(d^3 n \log B \cdot \mathcal{M}(d+\log B))$, using $O(d+\log B)$ precision floating-point numbers. However, this algorithm is mostly of theoretical interest and is not implemented in any of the main computational libraries [45, 28, 3, 27]. This is perhaps explained by the following reasons: it is not clear which floating-point arithmetic model is used, the algorithm is not easy to describe, and the hidden complexity constants are rather large. More precisely, the required precision of floating-point numbers in [40] seems to be higher than $12d + 7\log_2 B$.

Our results. We present the L^2 algorithm, a new and simple floating-point variant of the L^3 algorithm, in a standard floating-point arithmetic model, which provably outputs approximate L^3-reduced bases in polynomial time. More precisely, its complexity is $O(d^2 n(d+\log B)\log B \cdot \mathcal{M}(d))$ using a precision of only $(\log_2 3)\cdot d$ bits, which is independent of $\log B$. This is the first L^3 algorithm whose running time grows only quadratically with respect to $\log B$ (hence the name L^2), whereas the growth is cubic—without fast integer arithmetic—for all other known provable L^3 algorithms. This improvement is significant for lattices where $\log B$ is larger than d, for example those arising from minimal polynomials [6] and Coppersmith's technique [7]. Interestingly, the L^3 algorithm can be viewed as a generalization of the famous Euclid gcd algorithm and Lagrange's two-dimensional algorithm [21] whose complexities are quadratic without fast integer arithmetic, not cubic like the original L^3. This arguably makes L^2 closer to Euclid's algorithm. The table of Figure 1 draws a comparison between L^3 [23], Schnorr's floating-point LLL algorithm from [40], and L^2.

At first glance, it may look surprising that there is an $\mathcal{M}(d)$ term in the complexity of L^2 but no $\mathcal{M}(\log B)$ term. This can be explained as follows. Most of the integer multiplications in L^2 are of two types: either two arguments of bit-length $O(d)$, or one argument of bit-length $O(d)$ and the other of bit-length $O(\log B)$. Multiplications of the second type can be performed within $\mathcal{M}(d)(1+\frac{\log B}{d})$ bit operations, by subdividing the $O(\log B)$-bit number in $O(1+\frac{\log B}{d})$ numbers of bit-length $O(d)$. Of course, L^2 also performs a few multiplications between $O(\log B)$-bit numbers (in the initial computation of the Gram matrix), but these are much less frequent: their overall cost vanishes with the $\log^2 B$ term of the complexity bound above.

	L^3 [23]	Schnorr [40]	L^2
Required precision	$O(d\log B)$	$> 12d + 7\log_2 B$	$d\log_2 3 \approx 1.58d$
$O()$-complexity with naive multiplication	$d^5 n \log^3 B$	$d^3 n(d+\log B)^2 \log B$	$d^4 n(d+\log B)\log B$
$O()$-complexity with fast multiplication	$d^{4+\varepsilon} n \log^{2+\varepsilon} B$	$d^3 n(d+\log B)^{1+\varepsilon}\log B$	$d^{3+\varepsilon} n(d+\log B)\log B$

FIG. 1. *Comparison of different L^3 algorithms.*

The L^2 algorithm is based on several improvements, both in the L^3 algorithm itself and more importantly in its analysis. From an algorithmic point of view, we improve the accuracy of the usual Gram–Schmidt computations by a systematic use of the Gram matrix, and we adapt Babai's nearest plane algorithm [2] to floating-point arithmetic in order to stabilize the so-called size-reduction process extensively used in L^3. We give tight bounds on the accuracy of Gram–Schmidt computations to prove the correctness of L^2. The analysis led to the discovery of surprisingly bad lattices: for instance, we found a 55-dimensional lattice with 100-bit vectors which makes NTL's LLL_FP [45] (an improved version of [42]) loop forever, which contradicts [19], where it is claimed that double precision is sufficient in [42] to L^3-reduce lattices up to dimension 250 with classical Gram–Schmidt. However, for random looking lattice bases, stability problems seem to arise only in dimensions much higher than 55, due perhaps to the well-known experimental fact that for such input bases, the L^3 algorithm outputs better bases than for the worst case (see [35]). Finally, to establish a quadratic running time, we generalize a well-known cascade phenomenon in the complexity analysis of the Gaussian and Euclidean algorithms. This was inspired by the so-called greedy lattice reduction algorithm of [33],[3] which is quadratic in low dimension thanks to another cascade. The cascade analysis is made possible by the efficiency of our floating-point variant of Babai's algorithm and cannot be adapted to the standard L^3 algorithm: it is unlikely that the complexity of the original L^3 algorithm could be proved quadratic in $\log B$.

Related work. Much work [43, 40, 48, 18, 19, 41] has been devoted to improve L^3, specifically the exponent of d in the complexity, but none has improved the $\log^3 B$ factor (except [51, 44] for dimension two). We hope that some of these improvements might be adaptable to L^2.

Floating-point stability has long been a mysterious issue in L^3. When it was realized during experiments that classical Gram–Schmidt orthogonalization could be very unstable, it was suggested in the late nineties to use well-known alternative techniques (see [22, 11]) like Givens rotations (implemented in NTL) or Householder reflections, which are more expensive but seem to be more stable in practice. However, from a theoretical point of view, the best results known on the worst-case accuracy of such techniques are not significantly better than the so-called modified Gram–Schmidt algorithm. Besides, most numerical analysis results refer to backward stability and not accuracy: such a mistake is made in [19], where a theorem from [22] is incorrectly applied. At the moment, it is therefore not clear how to provably exploit known results on Givens rotations and Householder reflections to improve the L^3 algorithm theoretically, though Schnorr [41] provides heuristic arguments suggesting it might be possible. This is why L^2 uses a process only close to classical Gram–Schmidt.

Road map. In section 2 we provide necessary background on lattices and L^3. We describe the L^2 algorithm in section 3. Section 4 proves the correctness of L^2, while section 5 analyzes its complexity. In section 6, we generalize the L^2 algorithm to generating sets.

2. Preliminaries.

Notation. All logarithms are in base 2. Let $\|\cdot\|$ and $\langle\cdot,\cdot\rangle$ be the Euclidean norm and inner product of $\mathbb{R}^n$. The notation $\lceil x \rfloor$ denotes a closest integer to x. Bold variables are vectors. All the lattices we consider are integer lattices, as usual. The complexity model we use is the RAM model, and the computational cost is measured in elementary operations on bits. Our floating-point arithmetic model is a smooth extension of the IEEE-754 standard [16], as provided by NTL [45] (RR class) and MPFR [49]. With an ℓ-bit working precision, an *fp*-number is of the form $x = \pm m_x \cdot 2^{e_x}$, where the mantissa $m_x \in [1/2, 1)$ is $(\ell+1)$-bit long and the exponent e_x is an integer. If a is a real number, we denote by $\diamond(a)$ a closest *fp*-number to it. This directly implies that $|\diamond(a) - a| \leq 2^{-\ell-1}|a|$. We expect all four basic floating-point operations to be correctly rounded: the returned value for $(a \text{ op } b)$ with $\text{op} \in \{+,-,/,*\}$ is $\diamond(a \text{ op } b)$. In our complexity analysis, we do not consider the cost of the arithmetic on the exponents: it can be checked easily that the exponents are integers of length $O(\log(d + \log B))$, so that the cost is indeed negligible.

[3]The full version of this extended abstract will appear in *ACM Transactions on Algorithms*.

We now recall a few basic notions from algorithmic geometry of numbers (see [29]) before describing the classical LLL algorithm.

2.1. Lattice background. The *Gram matrix* $G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ of $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ is the $d \times d$ symmetric positive definite matrix $(\langle \mathbf{b}_i, \mathbf{b}_j \rangle)_{1 \leq i,j \leq d}$ formed by all the inner products.

Lattice volume. A lattice L has infinitely many lattice bases as soon as $\dim L \geq 2$. Any two bases are related to each other by an integral matrix of determinant ± 1, and therefore the determinant of the Gram matrix of a basis depends only on the lattice. The square root of this determinant is called the *volume* $\operatorname{vol} L$ (or *determinant*) of the lattice. This volume is sometimes called *covolume* because it is the volume of the torus $\operatorname{span}(L)/L$.

Hermite's constant. Any lattice L has a nonzero vector of minimal Euclidean norm: this minimal norm is called the *first minimum* and is denoted by $\lambda_1(L)$. It is a classical fact that for any d-dimensional lattice L, $\lambda_1(L)/\operatorname{vol}(L)^{1/d}$ can be upper bounded independently of L: Hermite's constant γ_d is defined as the supremum of $(\lambda_1(L)/\operatorname{vol}(L)^{1/d})^2$ over all d-dimensional lattices L.

Gram–Schmidt orthogonalization. Let $\mathbf{b}_1, \ldots, \mathbf{b}_d$ be linearly independent vectors. Their *Gram–Schmidt orthogonalization* (GSO) $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^*$ is the orthogonal family defined recursively as follows: the vector $\mathbf{b}_i^*$ is the component of the vector $\mathbf{b}_i$ which is orthogonal to the linear span of the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. We have $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \mathbf{b}_j^*$, where $\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}$. For $i \leq d$ we let $\mu_{i,i} = 1$. The reason why the GSO is widely used in lattice reduction is because the matrix representing the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_d$ with respect to the orthonormal basis $(\mathbf{b}_1^*/\|\mathbf{b}_1^*\|, \ldots, \mathbf{b}_d^*/\|\mathbf{b}_d^*\|)$ is lower triangular, with diagonal coefficients $\|\mathbf{b}_1^*\|, \ldots, \|\mathbf{b}_d^*\|$. It follows that the lattice L spanned by the $\mathbf{b}_i$'s satisfies $\operatorname{vol} L = \prod_{i=1}^d \|\mathbf{b}_i^*\|$.

Notice that the GSO family depends on the order of the vectors. If the $\mathbf{b}_i$'s are integer vectors, the $\mathbf{b}_i^*$'s and the $\mu_{i,j}$'s are rational. We also define the variables $r_{i,j}$ for $i \geq j$ as follows: for any $i \in [\![1, d]\!]$, we let $r_{i,i} = \|\mathbf{b}_i^*\|^2$, and for any $i \geq j$ we let $r_{i,j} = \mu_{i,j} \|\mathbf{b}_j^*\|^2$. In what follows, the *GSO family* denotes the $r_{i,j}$'s and $\mu_{i,j}$'s. Some information is redundant in rational arithmetic, but in the context of our floating-point calculations, it is useful to have all these variables.

Size-reduction. A basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is *size-reduced* with factor $\eta \geq 1/2$ if its GSO family satisfies $|\mu_{i,j}| \leq \eta$ for all $1 \leq j < i \leq d$. The ith vector $\mathbf{b}_i$ is *size-reduced* if $|\mu_{i,j}| \leq \eta$ for all $j \in [\![1, i-1]\!]$. Size-reduction usually refers to $\eta = 1/2$, but it is essential for the L^2 algorithm to allow at least slightly larger factors η.

2.2. From Hermite's reductions to the LLL reduction. A pair $(\mathbf{b}_1, \mathbf{b}_2)$ of linearly independent vectors is *Lagrange-reduced* [21] if $\|\mathbf{b}_1\| \leq \|\mathbf{b}_2\|$ and $|\langle \mathbf{b}_1, \mathbf{b}_2 \rangle| \leq \|\mathbf{b}_1\|^2/2$. This definition is often wrongly attributed to Gauss [10]. Lagrange's algorithm [21] shows that any two-dimensional lattice has a Lagrange-reduced basis: the algorithm is very similar to Euclid's gcd algorithm. Furthermore, such a basis $(\mathbf{b}_1, \mathbf{b}_2)$ satisfies the following property:

$$\|\mathbf{b}_1\| \cdot \|\mathbf{b}_2\| \leq \sqrt{4/3} \cdot \operatorname{vol}(L(\mathbf{b}_1, \mathbf{b}_2)),$$

which means that the vectors $\mathbf{b}_1$ and $\mathbf{b}_2$ are almost orthogonal. It also follows that

$$\|\mathbf{b}_1\| \leq (4/3)^{1/4} \cdot \operatorname{vol}(L(\mathbf{b}_1, \mathbf{b}_2))^{1/2}, \tag{1}$$

which gives a tight upper bound on Hermite's constant $\gamma_2 = \sqrt{4/3}$.

Hermite's reductions. In his famous letters to Jacobi (see [13]), Hermite described two reduction notions (along with algorithms) in the language of quadratic forms. We will briefly describe both, since Hermite's algorithms can be viewed as the ancestors of the L^3 algorithm.

The first reduction notion is described at the end of the first letter [13], but we only present a simplified version in the spirit of the second letter, which we call H1. To the best of our knowledge, it is the first reduction notion in arbitrary dimension, which was introduced to prove the existence of Hermite's constant, by establishing an upper bound called Hermite's inequality. The H1 reduction is defined by induction:

- A single vector $\mathbf{b}_1 \neq \mathbf{0}$ is always H1-reduced.
- A d-dimensional basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is H1-reduced if and only if the following hold:
  - The basis is size-reduced with factor 1/2; that is, the GSO satisfies all the inequalities $|\mu_{i,j}| \leq 1/2$ for all $i > j$.
  - The first basis vector $\mathbf{b}_1$ satisfies the following inequality:

    $$\|\mathbf{b}_1\| \leq (4/3)^{(d-1)/4} \cdot \mathrm{vol}(L(\mathbf{b}_1, \ldots, \mathbf{b}_d))^{1/d}, \tag{2}$$

    which generalizes (1) and implies Hermite's inequality:

    $$\gamma_d \leq \gamma_2^{d-1} = (4/3)^{(d-1)/2}. \tag{3}$$

  - By induction, the projected $(d-1)$-tuple $(\mathbf{b}'_2, \ldots, \mathbf{b}'_d)$ itself is H1-reduced, where for any $i \in [\![2, d]\!]$ the vector $\mathbf{b}'_i$ is the component of the vector $\mathbf{b}_i$ which is orthogonal to the vector $\mathbf{b}_1$.

This H1 reduction notion is useful only to prove Hermite's inequality: the first vector of an H1-reduced basis may be arbitrarily far from the first minimum of the lattice, and the orthogonality defect of the basis may be arbitrarily large. To prove the existence of H1-reduced bases, Hermite presented the following recursive algorithm:

- Let $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a basis of a lattice L.
- Apply recursively the algorithm to the projected basis $(\mathbf{b}'_2, \ldots, \mathbf{b}'_d)$, in such a way that all the basis vectors $\mathbf{b}_2, \ldots, \mathbf{b}_d$ are size-reduced with respect to $\mathbf{b}_1$: $|\mu_{i,1}| \leq 1/2$ for all $i \geq 2$.
- If $\mathbf{b}_1$ satisfies Hermite's inequality, the algorithm terminates. Otherwise, it can be shown that $\|\mathbf{b}_2\| < \|\mathbf{b}_1\|$, so exchange $\mathbf{b}_1$ and $\mathbf{b}_2$, and restart from the beginning.

The main differences with this algorithm and L^3 are the following: L^3 starts working with the first two basis vectors, but Hermite will start working with the last two basis vectors; and Hermite's algorithm uses Hermite's inequality instead of the so-called Lovász condition. Hermite proved that his algorithm must terminate. However, because his algorithm did not match Lagrange's algorithm in dimension two, and perhaps also because the orthogonality defect of a H1-reduced basis can be arbitrarily large, Hermite presented a slightly different algorithm in his second letter [13] to Jacobi:

- Let $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a basis of a lattice L.
- Ensure that $\mathbf{b}_1$ has minimal norm among all $\mathbf{b}_i$'s: otherwise, swap $\mathbf{b}_1$ with the shortest basis vector $\mathbf{b}_i$.
- Apply recursively the algorithm to the projected basis $(\mathbf{b}'_2, \ldots, \mathbf{b}'_d)$, in such a way that all the basis vectors $\mathbf{b}_2, \ldots, \mathbf{b}_d$ are size-reduced with respect to $\mathbf{b}_1$: $|\mu_{i,1}| \leq 1/2$ for all $i \geq 2$.

- If $\mathbf{b}_1$ has minimal norm among all $\mathbf{b}_i$'s, the algorithm terminates. Otherwise, swap $\mathbf{b}_1$ with the shortest basis vector $\mathbf{b}_i$, and restart from the beginning.

Hermite also proved that this algorithm must terminate. One can note that this algorithm matches Lagrange's algorithm when $d = 2$. But one can also note that this second algorithm achieves a reduction notion (H2) which is stronger than H1:

- A single vector $\mathbf{b}_1 \neq \mathbf{0}$ is always H2-reduced.
- A d-dimensional basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is H2-reduced if and only if the following hold:
  - The basis is size-reduced with factor 1/2; that is, the GSO satisfies all the inequalities $|\mu_{i,j}| \leq 1/2$ for all $i > j$.
  - The first basis vector $\mathbf{b}_1$ has minimal norm among all basis vectors: $\|\mathbf{b}_1\| \leq \|\mathbf{b}_i\|$ for all i.
  - By induction, the $(d-1)$-tuple $(\mathbf{b}'_2, \ldots, \mathbf{b}'_d)$ itself is H2-reduced, where for any $i \in [\![2, d]\!]$ the vector $\mathbf{b}'_i$ is the component of the vector $\mathbf{b}_i$ which is orthogonal to the vector $\mathbf{b}_1$.

As opposed to H1, this reduction notion implies a bounded orthogonality defect: more precisely, an H2-reduced basis satisfies $\prod_{i=1}^{d} \|\mathbf{b}_i\| \leq (4/3)^{\frac{d(d-1)}{4}} \cdot \mathrm{vol}(L)$. Surprisingly, Lenstra's reduction notion [24] (resp., his algorithm) turns out to be a relaxed variant of H2 (resp., Hermite's second algorithm): more precisely, one replaces the conditions $\|\mathbf{b}_1\| \leq \|\mathbf{b}_i\|$ by $c\|\mathbf{b}_1\| \leq \|\mathbf{b}_i\|$ for some constant $1/4 < c < 1$. This allowed Lenstra [24] to prove that his algorithm was polynomial-time in fixed dimension d. The H2 reduction was also rediscovered by Schnorr and Euchner [42] in 1994 with their L^3 algorithm with deep insertion: Schnorr–Euchner's algorithm is different from Hermite's second algorithm, but both try to achieve the same reduction notion. It is unknown if Hermite's algorithms are polynomial-time in varying dimension.

The LLL reduction. Roughly speaking, the LLL reduction [23] modifies Hermite's second reduction notion by replacing the conditions $\|\mathbf{b}_1\| \leq \|\mathbf{b}_i\|$ by the single condition $c\|\mathbf{b}_1\| \leq \|\mathbf{b}_2\|$ for some constant $1/4 < c < 1$. More precisely, a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is *L^3-reduced* with factor (δ, η), where $\delta \in (1/4, 1]$ and $\eta \in [1/2, \sqrt{\delta})$ if the basis is size-reduced with factor η and if its GSO satisfies the $(d-1)$ Lovász conditions: for all $2 \leq \kappa \leq d$,

$$\left(\delta - \mu_{\kappa,\kappa-1}^2\right) \cdot r_{\kappa-1,\kappa-1} \leq r_{\kappa,\kappa},$$

or equivalently $\delta \cdot \|\mathbf{b}^*_{\kappa-1}\|^2 \leq \|\mathbf{b}^*_\kappa + \mu_{\kappa,\kappa-1}\mathbf{b}^*_{\kappa-1}\|^2$. This implies that the norms $\|\mathbf{b}^*_1\|, \ldots, \|\mathbf{b}^*_d\|$ of the GSO vectors never drop too much: intuitively, the vectors are not far from being orthogonal. Such bases have very useful properties, like providing approximations to the shortest vector problem and the closest vector problem. In particular, their first vector is relatively short. More precisely, the following result is classical.

THEOREM 1 (see [23]). *Let $\delta \in (1/4, 1]$ and $\eta \in [1/2, \sqrt{\delta})$. Let $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a (δ, η)-L^3-reduced basis of a lattice L. Then*

$$\|\mathbf{b}_1\| \leq \left(1/(\delta - \eta^2)\right)^{\frac{d-1}{4}} \cdot (\mathrm{vol}\ L)^{\frac{1}{d}}, \tag{4}$$

$$\prod_{i=1}^{d} \|\mathbf{b}_i\| \leq \left(1/(\delta - \eta^2)\right)^{\frac{d(d-1)}{4}} \cdot (\mathrm{vol}\ L). \tag{5}$$

Proof. We have $r_{i-1,i-1} \leq (\delta - \eta^2)^{-1} \cdot r_{i,i}$ for any $i \in [\![2, d]\!]$. This implies that for any $j \leq i$

$$r_{j,j} \leq (\delta - \eta^2)^{-(i-j)} \cdot r_{i,i}.$$

Taking $j = 1$ and multiplying these inequalities for all $i \in [\![1, d]\!]$ gives (4). Since the basis is size-reduced, we have, for any i,

$$\begin{aligned}
\|\mathbf{b}_i\|^2 = r_{i,i} + \sum_{j=1}^{i-1} \mu_{i,j}^2 r_{j,j} &\leq \left(1 + \sum_{j=1}^{i-1} \eta^2 \frac{r_{j,j}}{r_{i,i}}\right) \cdot r_{i,i} \\
&\leq \left(1 + \eta^2 \sum_{j=1}^{i-1} \left(\delta - \eta^2\right)^{-(i-j)}\right) \cdot r_{i,i} \\
&= \left(1 + \eta^2 \frac{\left(\delta - \eta^2\right)^{-i} - \left(\delta - \eta^2\right)^{-1}}{\left(\delta - \eta^2\right)^{-1} - 1}\right) \cdot r_{i,i} \\
&= \frac{1 - \delta + \eta^2 \left(\delta - \eta^2\right)^{-i+1}}{1 - \delta + \eta^2} \cdot r_{i,i} \\
&\leq \left(\delta - \eta^2\right)^{-i+1} \cdot r_{i,i}.
\end{aligned}$$

By multiplying these d inequalities, we obtain (5). ☐

The inequality (4) is reminiscent of Hermite's inequality (3). Indeed, if we take $(\delta, \eta) = (1, 1/2)$, then (4) becomes exactly (2), and (5) corresponds to the orthogonality defect of the H2 reduction.

The L^3 reduction usually refers to the factor $(3/4, 1/2)$ initially chosen in [23], in which case the approximation constant $1/(\delta - \eta^2)$ is equal to 2. But the closer δ and η are, respectively, to 1 and 1/2, the shorter the vector $\mathbf{b}_1$ should be. In practice, one usually selects $\delta \approx 1$ and $\eta \approx 1/2$, so that we almost have $\|\mathbf{b}_1\| \leq (4/3)^{\frac{d-1}{4}} \cdot (\mathrm{vol}\ L)^{\frac{1}{d}}$. The L^3 algorithm obtains in polynomial time a basis reduced with factor $(\delta, 1/2)$, where $\delta < 1$ can be chosen arbitrarily close to 1: this algorithm can thus be viewed as an efficient algorithmic version of Hermite's inequality (3). The new L^2 algorithm achieves a factor (δ, η), where $\delta < 1$ can be arbitrarily close to 1 and $\eta > 1/2$ arbitrarily close to 1/2. It is unknown whether or not $\delta = 1$ can be achieved in polynomial time (attempts to prove it can be found in [1] and [26]). However, one can achieve $\eta = 1/2$ in quadratic time by first running the L^2 algorithm on the given input basis with $\delta' = \delta + 2(\eta - 1/2)$ and an $\eta > 1/2$ such that $\delta' < 1$, and then running the L^3 algorithm on the output basis. Because the first reduction outputs an almost-reduced basis, the second reduction will perform only size-reduction operations, in which case L^3 has the same complexity bound as the L^2 algorithm given in Theorem 2.

The $\mathbf{L^3}$ algorithm. The usual L^3 algorithm [23] is described in Figure 2. It computes an L^3-reduced basis in an iterative fashion: the index κ is such that at any stage of the algorithm, the truncated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ is L^3-reduced. At each loop iteration, the index κ is either incremented or decremented: the loop stops when the index κ reaches the value $d + 1$, in which case the entire basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ is L^3-reduced.

The L^3 algorithm performs two kinds of operations: swaps of consecutive vectors and size-reductions (see Figure 3), which consist of at most d translations of the form $\mathbf{b}_\kappa \leftarrow \mathbf{b}_\kappa - m \cdot \mathbf{b}_i$, where m is some integer and $i < \kappa$. Swaps are used to achieve Lovász's conditions, while the size-reduction algorithm is used to size-reduce vectors. Intuitively, size-reductions intend to shorten (or upper bound) the projection of $\mathbf{b}_\kappa$ over the linear span of $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$, while swaps shorten the projection of $\mathbf{b}_\kappa$ over the orthogonal complement of $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$, that is, $\mathbf{b}_\kappa^*$. We explain steps 4–6: if Lovász's condition is satisfied, nothing happens in step 5 and the index κ is

```
Input: A basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ and $\delta \in (1/4, 1)$.
Output: An L$^3$-reduced basis with factor $(\delta, 1/2)$.

1. Compute the rational GSO, i.e., all the $\mu_{i,j}$'s and $r_{i,i}$'s.
2. $\kappa \leftarrow 2$. While $\kappa \leq d$ do
3.    Size-reduce the vector $\mathbf{b}_\kappa$ using the size-reduction algorithm of Figure 3,
   which updates the GSO.
4.    $\kappa' \leftarrow \kappa$. While $\kappa \geq 2$ and $\delta \cdot r_{\kappa-1,\kappa-1} > r_{\kappa',\kappa'} + \sum_{i=\kappa-1}^{\kappa'-1} \mu_{\kappa',i}^2 r_{i,i}$, do $\kappa \leftarrow \kappa - 1$.
5.    Insert the vector $\mathbf{b}_{\kappa'}$ right before the vector $\mathbf{b}_\kappa$ and update the GSO accordingly.
6.    $\kappa \leftarrow \kappa + 1$.
7. Output $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
```

FIG. 2. *The L^3 algorithm.*

```
Input: A basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, its GSO, and an index $\kappa$.
Output: The basis where the vector $\mathbf{b}_\kappa$ is size-reduced and the updated GSO.

1. For $i = \kappa - 1$ down to 1 do
2.    $\mathbf{b}_\kappa \leftarrow \mathbf{b}_\kappa - \lceil \mu_{\kappa,i} \rfloor \cdot \mathbf{b}_i$.
3.    Update the GSO accordingly.
```

FIG. 3. *The size-reduction algorithm.*

incremented like in more classical descriptions of the L^3 algorithm. Otherwise, step 4 finds the right index to insert the vector $\mathbf{b}_\kappa$, by collecting consecutive failures of Lovász's test.

If the L^3 algorithm terminates, it is clear that the output basis is L^3-reduced. What is less clear a priori is why this algorithm has a polynomial-time complexity. A standard argument shows that each swap decreases the quantity $\Delta = \prod_{i=1}^{d} \|\mathbf{b}_i^*\|^{2(d-i+1)}$ by at least a factor $\delta < 1$. On the other hand, we have that $\Delta \geq 1$ because the $\mathbf{b}_i$'s are integer vectors and Δ can be viewed as a product of squared volumes of lattices spanned by some subsets of the $\mathbf{b}_i$'s. This proves that there can be no more than $O(d^2 \log B)$ swaps, and therefore loop iterations, where B is an upper bound on the norms of the input basis vectors. It remains to estimate the cost of each loop iteration. This cost turns out to be dominated by $O(dn)$ arithmetic operations on the basis matrix and GSO coefficients $\mu_{i,j}$ and $r_{i,i}$ which are rational numbers of bit-length $O(d \log B)$. Thus, the overall complexity of the L^3 algorithm described in Figure 2 can be bounded by

$$O\left(\left(d^2 \log B\right) \cdot dn \cdot \mathcal{M}\left(d \log B\right)\right) = O\left(d^3 n \log B \cdot \mathcal{M}\left(d \log B\right)\right).$$

Note, however, that, as mentioned in the introduction, this analysis can be (slightly) improved. First, Kaltofen [17] showed that the most expensive arithmetic operations do not involve two operands of bit-length $O(d \log B)$: at least one of them has bit-length at most $O(d + \log B)$, which leads to the bound

$$O\left(\frac{d^4 n \log^2 B}{d + \log B} \cdot \mathcal{M}(d + \log B)\right).$$

Furthermore, the dependency with respect to n can be improved when $n \gg d$: the arithmetic operations on the basis vectors are actually cheaper than the operations on the GSO coefficients, because they involve only operands of bit-length $O(d + \log B)$.

2.3. The L^3 algorithm with floating-point arithmetic. The cost of the L^3 algorithm is dominated by the arithmetic operations on the GSO coefficients which are rationals with huge numerators and denominators. It is therefore tempting to replace the exact GSO coefficients by floating-point approximations that will be represented with far fewer bits. But doing so in a straightforward manner leads to instability. The algorithm is no longer guaranteed to be polynomial-time: it may not even terminate, because the quantity Δ used to bound the complexity of L^3 algorithm no longer necessarily decreases at each swap. It could be that the new algorithm performs a swap when the initial L^3 algorithm would not have performed such a swap. And if the algorithm ever terminates, the output basis may not be L^3-reduced, due to potential inaccuracy in the GSO coefficients. Prior to this work, the only provable floating-point L^3 algorithm was the one of Schnorr [40], which simulates the behavior of the L^3 algorithm using floating-point approximations of the coefficients of the inverse matrix of the $\mu_{i,j}$'s. The number of loop iterations and the number of arithmetic operations (in each iteration) remain the same as in the L^3 algorithm (up to a constant factor): only the cost of each arithmetic operation related to the GSO decreases. Instead of handling integers of length $O(d \log B)$, Schnorr's algorithm uses floating-point numbers with $O(d + \log B)$-bit long mantissae (with large hidden constants, as mentioned in the introduction). This decreases the worst-case complexity of the L^3 algorithm to $O(d^3 n \log B \cdot \mathcal{M}(d + \log B))$. With naive multiplication, this cost remains cubic in $\log B$. Because this algorithm is mostly of theoretical interest, the main number theory computer packages [3, 28, 45] used to contain heuristic floating-point variants of the L^3 algorithm à la Schnorr and Euchner [42], which suffer from stability problems in high dimension. This is no longer the case in [28], which contains an implementation of the L^2 algorithm.

3. The L^2 algorithm.

In this section, we present the L^2 algorithm. Its complexity analysis is postponed to the next section.

3.1. Overview. The L^2 algorithm is a natural floating-point variant of the L^3 algorithm, which follows the same structure as the L^3 algorithm [23] described in Figure 2, with two important differences:

- Instead of keeping the GSO in exact rational arithmetic, we will keep only a sufficiently good floating-point approximation, and we will try to simulate the execution of the rational L^3 algorithm. If the floating-point precision is too large, it will be too expensive to compute with the GSO. But if the floating-point precision is too small, the approximation might become too inaccurate, to the point of being meaningless: accuracy is crucial for the size-reductions and for checking the Lovász conditions. We will select a floating-point precision linear in d only, whereas Schnorr's floating-point L^3 algorithm described in [40] uses a larger precision (linear in both d and $\log B$). The fact that the floating-point precision is independent of $\log B$ is crucial to the quadratic complexity of L^2.
- We replace the size-reduction algorithm described in Figure 3 by an algorithm better suited to floating-point arithmetic. The new algorithm will perform more operations, but it will be more stable: it will tolerate an approximation of the GSO. We will use the fact that when L^2 calls the size-reduction algorithm, we already know that the first $\kappa - 1$ vectors are almost L^3-reduced.

When computing a floating-point approximation of the GSO, it is very important to use exact scalar products $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$: we will keep only a floating-point approximation of the GSO, but we will also keep the exact Gram matrix formed by $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ and update it during the reduction.

3.2. Gram–Schmidt computations. It is important for the L^2 algorithm to have accurate formulae for the computation of the GSO coefficients. In [42], the following recursive formulae were used:

$$\mu_{i,j} = \frac{\langle \mathbf{b}_i, \mathbf{b}_j \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot \mu_{i,k} \cdot \|\mathbf{b}_k^*\|^2}{\|\mathbf{b}_j^*\|^2} \quad \text{and} \quad \|\mathbf{b}_i^*\|^2 = \|\mathbf{b}_i\|^2 - \sum_{j=1}^{i-1} \mu_{i,j}^2 \cdot \|\mathbf{b}_j^*\|^2.$$

In these formulae, the inner products $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ are computed in floating-point arithmetic, which possibly leads to an absolute error of $2^{-\ell} \cdot \|\mathbf{b}_i\| \cdot \|\mathbf{b}_j\|$, where ℓ is the chosen precision. This happens, for example, when the vectors $\mathbf{b}_i$ and $\mathbf{b}_j$ are almost orthogonal, i.e., when their scalar product is very small compared to the product of their norms. This has the following drawback: to ensure that the basis returned by the L^2 algorithm is size-reduced, absolute error bounds on the $\mu_{i,j}$'s are required; if the absolute error on $\langle \mathbf{b}_i, \mathbf{b}_j \rangle$ can be larger than $2^{-\ell}\|\mathbf{b}_i\|\|\mathbf{b}_j\|$, the precision ℓ must be $\Omega(\log B)$ in the worst case (for example, when the vector $\mathbf{b}_i$ is very long while $\|\mathbf{b}_j\| = O(1)$). The analyses of [40, 41] do not tackle this issue. We solve this problem by computing the exact Gram matrix at the beginning of the execution of the L^2 algorithm and by updating it every time the basis matrix is modified: in fact, the transformations are computed from the Gram matrix and applied to the basis matrix. The additional cost is proportional to the cost of the update of the basis matrix. The advantage is that it allows us to require a floating-point precision of only $O(d)$ bits within the underlying orthogonalization process.

Besides, we use slightly different formulae by introducing the quantities $r_{i,j} = \mu_{i,j} \cdot \|\mathbf{b}_j^*\|^2 = \langle \mathbf{b}_i, \mathbf{b}_j^* \rangle$ for all $i \geq j$:

$$r_{i,j} = \langle \mathbf{b}_i, \mathbf{b}_j \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot r_{i,k} \quad \text{and} \quad \mu_{i,j} = \frac{r_{i,j}}{r_{j,j}}.$$

The accuracy is improved because the inner products are extracted from the exact Gram matrix and because each term of the sum now requires only a single multiplication instead of two. For $i = j$, we have $r_{i,i} = \|\mathbf{b}_i\|^2 - \sum_{k=1}^{i-1} \mu_{i,k} \cdot r_{i,k}$, which suggests defining the quantities $s_j^{(i)} = \|\mathbf{b}_i\|^2 - \sum_{k=1}^{j-1} \mu_{i,k} \cdot r_{i,k}$ for all $j \in [\![1, i]\!]$. In particular, we have $s_1^{(i)} = \|\mathbf{b}_i\|^2$ and $s_i^{(i)} = \|\mathbf{b}_i^*\|^2 = r_{i,i}$. The quantities $s_j^{(i)}$ will be useful to check consecutive Lovász's conditions. Indeed, Lovász's condition $\left(\delta - \mu_{\kappa,\kappa-1}^2\right) \cdot \|\mathbf{b}_{\kappa-1}^*\|^2 \leq \|\mathbf{b}_\kappa^*\|^2$ can be rewritten as $\delta \cdot \|\mathbf{b}_{\kappa-1}^*\|^2 \leq \|\mathbf{b}_\kappa^*\|^2 + \mu_{\kappa,\kappa-1}^2 \|\mathbf{b}_{\kappa-1}^*\|^2$, i.e.,

$$\delta \cdot r_{\kappa-1,\kappa-1} \leq s_{\kappa-1}^{(\kappa)}.$$

Whenever the condition is not satisfied, the L^3 algorithm would swap the vectors $\mathbf{b}_{\kappa-1}$ and $\mathbf{b}_\kappa$ and check the following Lovász's condition:

$$\delta \cdot r_{\kappa-2,\kappa-2} \leq s_{\kappa-2}^{(\kappa)}.$$

Thus, storing the $s_j^{(\kappa)}$'s allows us to check consecutive Lovász's conditions (when consecutive swaps occur) without any additional cost since they appear in the calculation of $r_{\kappa,\kappa}$. The computation of the $r_{i,j}$'s, $\mu_{i,j}$'s, and $s_j^{(d)}$'s is summarized in the so-called Cholesky factorization algorithm (CFA) of Figure 4.

```
Input: The Gram matrix of $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
Output: All the $r_{i,j}$'s, $\mu_{i,j}$'s, and $s_j^{(d)}$'s defined by the GSO.

1. For $i = 1$ to $d$ do
2.   For $j = 1$ to $i - 1$ do
3.     $r_{i,j} \leftarrow \langle \mathbf{b}_i, \mathbf{b}_j \rangle$.
4.     For $k = 1$ to $j - 1$ do $r_{i,j} \leftarrow r_{i,j} - \mu_{j,k} \cdot r_{i,k}$.
5.     $\mu_{i,j} \leftarrow r_{i,j} / r_{j,j}$.
6.     $s_1^{(i)} \leftarrow \|\mathbf{b}_i\|^2$. For $j = 2$ to $i$ do $s_j^{(i)} \leftarrow s_{j-1}^{(i)} - \mu_{i,j-1} \cdot r_{i,j-1}$.
7.     $r_{i,i} \leftarrow s_i^{(i)}$.
```

FIG. 4. *The Cholesky factorization algorithm.*

Of course, because one uses floating-point arithmetic, the exact values are unknown. Instead, one computes floating-point approximations $\bar{r}_{i,j}$, $\bar{\mu}_{i,j}$, and $\bar{s}_j^{(i)}$. Steps 3–6 are performed in the following way:

$$\begin{aligned}
\bar{r}_{i,j} &\leftarrow \diamond\left(\langle \mathbf{b}_i, \mathbf{b}_j \rangle\right), \\
\bar{r}_{i,j} &\leftarrow \diamond\left(\bar{r}_{i,j} - \diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k})\right), \\
\bar{\mu}_{i,j} &\leftarrow \diamond\left(\bar{r}_{i,j} / \bar{r}_{j,j}\right), \\
\bar{s}_j^{(i)} &\leftarrow \diamond\left(\bar{s}_{j-1}^{(i)} - \diamond(\bar{\mu}_{d,j-1} \cdot \bar{r}_{d,j-1})\right).
\end{aligned}$$

We will not use the CFA directly in the L^2 algorithm. Instead, we will use parts of it during the execution of the algorithm: because the orthogonalization is performed vector by vector, there is no need for recomputing everything from scratch if the $r_{i,j}$'s and $\mu_{i,j}$'s are already known for any i and j below some threshold. The CFA will also prove useful in section 4, as a first step in the proof of correctness of the L^2 algorithm.

3.3. A lazy floating-point size-reduction algorithm. The core of the L^2 algorithm is a lazy floating-point version of the size-reduction algorithm, described in Figure 5. Instead of size-reducing the vector $\mathbf{b}_\kappa$ at once like in Figure 3, our floating-point version tries to do the same progressively in several small steps that use the CFA of Figure 4. Size-reducing in a single step requires a very high accuracy for the floating-point calculations: the required mantissa size could possibly be linear in $\log B$ (see the discussion after Theorem 4). Rather than doing this, we perform the size-reduction progressively: we make the $\mu_{i,j}$'s decrease a bit, we recompute them with more accuracy (with the CFA), and we go on until they are small enough and known with good accuracy. If we consider the integer translation coefficients x_i that would have been computed if we had performed the size-reduction at once, it intuitively means that we discover them progressively, from the most significant bits to the least significant ones. Although the lazy size-reduction requires more steps than the initial size-reduction, it will overall be less expensive because replacing the exact rational arithmetic of the GSO by floating-point arithmetic outweighs the cost.

At step 1, we define $\bar{\eta} \leftarrow \frac{\diamond(\diamond(\eta)+1/2)}{2}$. The division by 2 is exact because the base of the floating-point arithmetic is 2. The variable $\bar{\eta}$ is meant to be in $(1/2, \eta)$, which is the case if ℓ is large enough. At step 6, it suffices to update the scalar

Input: A factor $\eta > 1/2$, an fp-precision $\ell + 1$, an index κ, a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ with its Gram matrix, and floating-point numbers $\bar{r}_{i,j}$ and $\bar{\mu}_{i,j}$ for $j \leq i < \kappa$.
Output: Floating-point numbers $\bar{r}_{\kappa,j}$, $\bar{\mu}_{\kappa,j}$, and $\bar{s}_j^{(\kappa)}$ for $j \leq \kappa$,
a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}, \mathbf{b}'_\kappa, \mathbf{b}_{\kappa+1}, \ldots, \mathbf{b}_d)$ with its Gram matrix, where
$\mathbf{b}'_\kappa = \mathbf{b}_\kappa - \sum_{i<\kappa} x_i \cdot \mathbf{b}_i$ for some $x_i \in \mathbb{Z}$ and $|\langle \mathbf{b}'_\kappa, \mathbf{b}^*_i \rangle| \leq \eta \|\mathbf{b}^*_i\|^2$ for any $i < \kappa$.

```
1. $\bar{\eta} \leftarrow \frac{\diamond(\diamond(\eta)+1/2)}{2}$.
2. Compute the $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s, $\bar{s}_j^{(\kappa)}$'s with steps 2–7 of the CFA with "$i = \kappa$."
3. If $\max_{j<\kappa} |\bar{\mu}_{\kappa,j}| \leq \bar{\eta}$, terminate. Else, for $i = \kappa - 1$ down to 1, do
4.   $X_i \leftarrow \lfloor \bar{\mu}_{\kappa,i} \rceil$.
5.   For $j = 1$ to $i - 1$, $\bar{\mu}_{\kappa,j} \leftarrow \diamond(\bar{\mu}_{\kappa,j} - \diamond(X_i \cdot \bar{\mu}_{i,j}))$.
6. $\mathbf{b}_\kappa \leftarrow \mathbf{b}_\kappa - \sum_{i=1}^{\kappa-1} X_i \cdot \mathbf{b}_i$, update $G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ accordingly.
7. Goto step 2.
```

FIG. 5. *The lazy size-reduction algorithm.*

Input: A valid pair (δ, η), like in Theorem 2, a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, and an fp-precision $\ell + 1$.
Output: An L^3-reduced basis with factor pair (δ, η).
Variables: An integral matrix G, floating-point numbers $\bar{r}_{i,j}$, $\bar{\mu}_{i,j}$, and $\bar{s}_i$.

```
1. Compute exactly $G = G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
2. $\bar{\delta} \leftarrow \frac{\diamond(\diamond(\delta)+1)}{2}$, $\bar{r}_{1,1} \leftarrow \diamond(\langle \mathbf{b}_1, \mathbf{b}_1 \rangle)$, $\kappa \leftarrow 2$. While $\kappa \leq d$, do
3.   Size-reduce the vector $\mathbf{b}_\kappa$ using the algorithm of Figure 5, updating the fp-GSO.
4.   $\kappa' \leftarrow \kappa$. While $\kappa \geq 2$ and $\diamond(\bar{\delta} \cdot \bar{r}_{\kappa-1,\kappa-1}) > \bar{s}_{\kappa-1}^{(\kappa')}$, do $\kappa \leftarrow \kappa - 1$.
5.   For $i = 1$ to $\kappa - 1$ do $\bar{\mu}_{\kappa,i} \leftarrow \bar{\mu}_{\kappa',i}$, $\bar{r}_{\kappa,i} \leftarrow \bar{r}_{\kappa',i}$, $\bar{r}_{\kappa,\kappa} \leftarrow \bar{s}_\kappa^{(\kappa')}$.
6.   Insert the vector $\mathbf{b}_{\kappa'}$ right before the vector $\mathbf{b}_\kappa$ and update $G$ accordingly.
7.   $\kappa \leftarrow \kappa + 1$.
8. Output $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
```

FIG. 6. *The L^2 algorithm.*

products $\langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle$ for $i \leq d$ with the following relations:

$$\|\mathbf{b}'_\kappa\|^2 = \|\mathbf{b}_\kappa\|^2 + \sum_{j \neq \kappa} X_j^2 \cdot \|\mathbf{b}_j\|^2 - 2 \sum_{j \neq \kappa} X_j \cdot \langle \mathbf{b}_j, \mathbf{b}_\kappa \rangle + 2 \sum_{j \neq \kappa, i \neq \kappa} X_i \cdot X_j \cdot \langle \mathbf{b}_i, \mathbf{b}_j \rangle,$$

$$\langle \mathbf{b}_i, \mathbf{b}'_\kappa \rangle = \langle \mathbf{b}_i, \mathbf{b}_\kappa \rangle - \sum_{j \neq \kappa} X_j \cdot \langle \mathbf{b}_i, \mathbf{b}_j \rangle \quad \text{for} \quad i \neq \kappa.$$

3.4. Main results. A description of the L^2 algorithm is given in Figure 6.

There is no need for keeping approximations of all the GSO coefficients: because the algorithm is iterative, it suffices to have approximations up to the threshold κ. Notice that the cost of the first step is bounded by $O(d^2 n \mathcal{M}(\log B))$ and is thus negligible compared to the rest of the execution. At step 4, instead of δ we use $\bar{\delta} = \frac{\diamond(\diamond(\delta)+1)}{2}$, which is hopefully in $(\delta, 1)$, to take into account the fact that the quantities $\bar{r}_{\kappa-1,\kappa-1}$ and $\bar{s}_{\kappa-1}^{(\kappa')}$ are known only approximately. The main result of this paper is the following.

THEOREM 2. *Let (δ, η) be such that $1/4 < \delta < 1$ and $1/2 < \eta < \sqrt{\delta}$. Let $c > \log \frac{(1+\eta)^2}{\delta - \eta^2}$ be a constant. Given as input a d-dimensional lattice basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ in $\mathbb{Z}^n$ with $\max_i \|\mathbf{b}_i\| \leq B$, the L^2 algorithm of Figure* 6 *with $\ell = cd + o(d)$ outputs* *a (δ,η)-L^3-reduced basis in time $O\left(d^2 n \log B(d+\log B)\cdot\mathcal{M}(d)\right)$. Furthermore, the running time is $O\left(n(\tau + d\log dB)(d+\log B)\cdot\mathcal{M}(d)\right)$ if τ denotes the number of loop iterations.*

Let us make a few remarks.

1. If one considers naive multiplication, then the L^2 algorithm decreases by a multiplicative factor $\frac{d+\log B}{d}$ the complexity bound $O(d^3 n\log B(d+\log B)^2)$ of [40].
2. We can choose δ arbitrarily close to 1 and η arbitrarily close to $1/2$, so that the coefficient $c > \log\frac{(1+\eta)^2}{\delta-\eta^2}$ can be chosen arbitrarily close to $\log 3 < 1.585$. More precisely, we can show the following complexity bound, up to a multiplicative universal $O(\cdot)$-constant:
$$\frac{dn\log B(d+\log B)\mathcal{M}\left(cd-\log(1-\delta)-\log(\eta-1/2)\right)\cdot\left(d-\log\left(\eta-\frac{1}{2}\right)\right)}{-\log\delta\cdot\left(c-\log\frac{(1+\eta)^2}{\delta-\eta^2}\right)}.$$
We can therefore choose $\eta = 1/2 + 2^{-\Theta(d)}$ and $\delta = 1-\frac{1}{\Theta(\log d)}$ while keeping almost the same complexity bound as in Theorem 2. The right-most terms in the numerator and denominator come from the number of loop iterations of the lazy size-reductions, and the term $-\log\delta$ comes from the loop iterations of the L^2 algorithm.
3. The additional statement of the theorem that is related to the number of loop iterations is useful for certain lattices which arise in practice, like lattices arising in knapsacks and minimal polynomials, where we possibly have $\tau = O(d\log B)$ instead of the worst-case bound $O(d^2\log B)$.
4. Finally, the $o(d)$ term in the condition $\ell = c\cdot d + o(d)$ can be made explicit (see Theorem 6) and may be used backwards: if we perform calculations with a fixed precision (e.g., in the normalized 53-bit mantissae double precision), the correctness will be guaranteed up to some computable dimension. We give in Figure 7 the dimensions up to which common precisions will suffice for two pairs of parameters (η,δ). Note that the values are not very high. We refer the reader to [47] for a description of a practical and provable implementation of L^2.

(δ,η)	Double precision ($\ell = 52$)	Double extended precision ($\ell = 63$)	Quadruple precision ($\ell = 112$)
$(0.75, 0.55)$	11	15	34
$(0.99, 0.51)$	15	21	50

FIG. 7. *Largest valid dimensions for given precisions and parameters.*

The following two sections are devoted to proving Theorem 2. We will obtain the correctness property in section 4, by studying successively the CFA when given as input bases appearing during an L^3-reduction (Theorem 3), and the lazy size-reduction algorithm (Theorem 5), to finally obtain the desired result in Theorem 6. The complexity statement of Theorem 2 will be proved in section 5.

4. Correctness of the L^2 algorithm.

To guarantee the correctness of the L^2 algorithm, we need to estimate the accuracy of the floating-point approximations at various stages.

4.1. Accuracy of the Gram–Schmidt computations. In general, the classical Gram–Schmidt algorithm is known to have very poor numerical stability properties [11, 22, 50]. However, it must be stressed that in the context of the L^3 algorithm, the bases are reduced in an iterative fashion, which implies that we can study the accuracy of Gram–Schmidt computations under the hypothesis that the first $d-1$ vectors of the input basis are L^3-reduced. In this particular setting, because an L^3-reduced basis is almost orthogonal, the following result shows that a working precision of $\approx d\log 3$ bits is sufficient for the CFA if the reduction factor (δ,η) is sufficiently close to the optimal pair $(1, 1/2)$.

THEOREM 3. *Let (δ,η) be a valid factor pair like in Theorem 2, $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ with $\varepsilon \in (0, 1/2]$, and $(\mathbf{b}_1,\ldots,\mathbf{b}_d)$ in $\mathbb{Z}^n$ be a d-dimensional lattice basis whose Gram matrix is given as input to the CFA described in Figure 4. Assume that $(\mathbf{b}_1,\ldots,\mathbf{b}_{d-1})$ is (δ,η)-L^3-reduced. In the case of floating-point arithmetic with ℓ satisfying $\ell \geq 5 + 2\log d - \log\varepsilon + d\log\rho$, the output floating-point numbers satisfy the following equations: for all $j \leq i < d$,*

$$\frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}} \leq d\rho^j 2^{-\ell+2} \quad \textit{and} \quad |\bar{\mu}_{i,j} - \mu_{i,j}| \leq d\rho^j 2^{-\ell+4}.$$

Furthermore, if $M = \max_{j<d} |\mu_{d,j}|$, then we have for any $j < d$

$$\frac{|\bar{r}_{d,j} - r_{d,j}|}{r_{j,j}} \leq d\rho^j 2^{-\ell+2}\cdot M \quad \textit{and} \quad |\bar{\mu}_{d,j} - \mu_{d,j}| \leq d\rho^j 2^{-\ell+4}\cdot M.$$

Finally, if the vector $\mathbf{b}_d$ is η-size-reduced with respect to $(\mathbf{b}_1,\ldots,\mathbf{b}_{d-1})$, then for any $j \leq d$

$$\left|\bar{s}_j^{(d)} - s_j^{(d)}\right| \leq d\rho^j 2^{-\ell+9}\cdot r_{j,j} + d2^{-\ell}\cdot s_j^{(d)}.$$

The second set of inequalities is useful for the analysis of the size-reduction algorithm, while the last set provides guarantees when checking Lovász's conditions.

It is crucial in the theorem to assume that the first vectors are L^3-reduced. More precisely, if the Lovász condition or the size-reduction condition is not satisfied, the result is no longer valid. Heuristically, the Lovász condition ensures that when some $r_{i,j}$ is computed by using the $r_{i,k}$'s with $k < j$, the error on $r_{i,k}$ relative to $r_{k,k}$ cannot be arbitrarily large relative to $r_{j,j}$ (since $r_{j,j}$ cannot be arbitrarily small compared to $r_{k,k}$). The size-reduction condition allows us to bound the error on $r_{i,j}$ relative to $r_{j,j}$, independently of the vector $\mathbf{b}_i$ (this vector could be arbitrarily longer than the vector $\mathbf{b}_j$). At the end, it provides an absolute error on the $\mu_{i,j}$'s.

Before giving the proof of Theorem 3, we first give a sketch of it for the case $\eta \approx 1/2$ and $\delta \approx 1$. Most of the accuracy loss comes from step 4, which amplifies the error. We define $err_j = \max_{j\leq i<d} \frac{|\bar{r}_{i,j}-r_{i,j}|}{r_{j,j}}$, i.e., the error on $r_{i,j}$ relative to $r_{j,j}$, and we bound its growth as j increases. Obviously

$$err_1 \leq \frac{|\diamond\langle\mathbf{b}_i,\mathbf{b}_1\rangle - \langle\mathbf{b}_i,\mathbf{b}_1\rangle|}{\|\mathbf{b}_1\|^2} \leq 2^{-\ell}\cdot\max_{i<d}\frac{|\langle\mathbf{b}_i,\mathbf{b}_1\rangle|}{\|\mathbf{b}_1\|^2} \leq 2^{-\ell},$$

because of the size-reduction condition. We now choose $j \in [\![2, d-1]\!]$. The result for $i = d$ can intuitively be derived from the proof for $i \leq d-1$, by replacing "$\mathbf{b}_d$" by "$\frac{1}{M}\mathbf{b}_d$" in it. Because of step 5, we have, for any $i < d$ and any $k < j$,

$$|\bar{\mu}_{i,k} - \mu_{i,k}| \leq \left|\frac{r_{k,k}}{\bar{r}_{k,k}}\right| err_k + |r_{i,k}|\left|\frac{1}{\bar{r}_{k,k}} - \frac{1}{r_{k,k}}\right| \lesssim \left(\frac{3}{2}\right) err_k,$$

where we neglected low-order terms and used the fact that $|r_{i,k}| \lesssim \frac{1}{2}\|\mathbf{b}_k\|^2$, which comes from the size-reduction condition. This implies that

$$\begin{aligned} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| &\leq |\bar{\mu}_{j,k} - \mu_{j,k}| \cdot |\bar{r}_{i,k}| + |\mu_{j,k}| \cdot |\bar{r}_{i,k} - r_{i,k}| \\ &\lesssim \left(\frac{5}{4}\right) err_k \cdot \|\mathbf{b}_k^*\|^2, \end{aligned}$$

where we also neglected low-order terms and used the size-reduction condition twice. Thus,

$$err_j \lesssim \left(\frac{5}{4}\right) \sum_{k<j} \frac{\|\mathbf{b}_k^*\|^2}{\|\mathbf{b}_j^*\|^2} err_k \lesssim \left(\frac{5}{4}\right) \sum_{k<j} \left(\frac{4}{3}\right)^{j-k} err_k,$$

by using Lovász's conditions. This last inequality finally gives

$$err_j \lesssim 3^j \cdot err_1 \lesssim 3^j 2^{-\ell}.$$

Proof. The result being easy to obtain for $d = 1$, we assume that $d \geq 2$. We start with the first inequalities. The goal is to bound the error that we make while computing the $r_{i,j}$'s. At step 4 of the CFA, we compute the sum $\langle \mathbf{b}_i, \mathbf{b}_j \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot r_{i,k}$ in a naive way. It is classical (see [14], for example) that the error performed during the summation is bounded by

$$\frac{d 2^{-\ell-1}}{1 - d 2^{-\ell-1}} \cdot \left(|\diamond \langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k})| \right).$$

As a consequence, the quantity $|\bar{r}_{i,j} - r_{i,j}|$ is, by the triangular inequality,

$$\begin{aligned} &\leq \frac{d 2^{-\ell-1}}{1 - d 2^{-\ell-1}} \cdot \left(|\diamond \langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k})| \right) \\ &\quad + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| + |\diamond \langle \mathbf{b}_i, \mathbf{b}_j \rangle - \langle \mathbf{b}_i, \mathbf{b}_j \rangle| \\ &\leq \frac{\left(|\diamond \langle \mathbf{b}_i, \mathbf{b}_j \rangle - \langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| + \frac{d}{2^{\ell+1}} (|\langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} |\mu_{j,k} r_{i,k}|) \right)}{1 - d 2^{-\ell-1}} \\ &\leq \frac{\left(|\diamond \langle \mathbf{b}_i, \mathbf{b}_j \rangle - \langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| + \frac{d}{2^{\ell+1}} \cdot (|\langle \mathbf{b}_i, \mathbf{b}_j \rangle| + \sum_{k=1}^{j-1} r_{k,k}) \right)}{1 - d 2^{-\ell-1}}, (*) \end{aligned}$$

where we used the fact that $\eta < 1$, which implies that $|\mu_{j,k} r_{i,k}| \leq \eta \cdot r_{k,k} \leq r_{k,k}$. We show by induction on $j < d$ the following error bound:

$$\forall i \in [\![j, d-1]\!], \ \frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}} \leq d \rho^j 2^{-\ell+2}.$$

To do this, we define $err_j = \max_{i \in [\![j,d-1]\!]} \frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}}$ and $E = d\rho^d 2^{-\ell+2} \leq \varepsilon/16$. We first consider the case $j = 1$. We have

$$|\diamond \langle \mathbf{b}_i, \mathbf{b}_1 \rangle - \langle \mathbf{b}_i, \mathbf{b}_1 \rangle| \leq 2^{-\ell-1} |\mu_{i,1}| r_{1,1} \leq 2^{-\ell-1} \eta r_{1,1} \leq 2^{-\ell-1} r_{1,1}.$$

Assume now that $j \in [\![2, d-1]\!]$ and that we know that the result holds for all $k < j$. If $k < j \leq i$, we have

$$\begin{aligned}
|\bar{\mu}_{i,k} - \mu_{i,k}| &\leq \left|\bar{\mu}_{i,k} - \frac{\bar{r}_{i,k}}{\bar{r}_{k,k}}\right| + \left|\frac{\bar{r}_{i,k}}{\bar{r}_{k,k}} - \frac{r_{i,k}}{r_{k,k}}\right| \\
&\leq \left(1 + 2^{-\ell-1}\right) \cdot \left|\frac{\bar{r}_{i,k}}{\bar{r}_{k,k}} - \frac{r_{i,k}}{r_{k,k}}\right| + 2^{-\ell-1}\eta \\
&\leq \left(1 + 2^{-\ell-1}\right) \cdot \left(\frac{|\bar{r}_{i,k} - r_{i,k}|}{r_{k,k}} \frac{r_{k,k}}{\bar{r}_{k,k}} + |r_{i,k}| \frac{|\bar{r}_{k,k} - r_{k,k}|}{r_{k,k}} \frac{1}{\bar{r}_{k,k}}\right) + 2^{-\ell-1} \\
&\leq \left(1 + 2^{-\ell-1}\right) err_k (1+\eta) \frac{r_{k,k}}{\bar{r}_{k,k}} + 2^{-\ell-1} \\
&\leq err_k (1+\eta) \frac{1 + 2^{-\ell-1}}{1 - E} + 2^{-\ell-1},
\end{aligned}$$

by the induction hypothesis. Hence, if $k < j \leq i$, $|\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| \cdot \frac{1}{r_{k,k}}$ is

$$\begin{aligned}
&\leq (1 + 2^{-\ell-1}) \cdot \frac{|\bar{\mu}_{j,k}\bar{r}_{i,k} - \mu_{j,k} r_{i,k}|}{r_{k,k}} + 2^{-\ell-1} \\
&\leq (1 + 2^{-\ell-1}) \cdot \left(\frac{|\bar{r}_{i,k} - r_{i,k}|}{r_{k,k}} \bar{\mu}_{j,k} + |\bar{\mu}_{j,k} - \mu_{j,k}| \frac{r_{i,k}}{r_{k,k}}\right) + 2^{-\ell-1} \\
&\leq (1 + 2^{-\ell-1}) \cdot (\eta \cdot err_k + (\eta + E)|\bar{\mu}_{j,k} - \mu_{j,k}|) + 2^{-\ell-1} \\
&\leq (1 + 2^{-\ell-1}) \cdot err_k \left(\eta(2+\eta) + \frac{8E}{1-E}\right) + 129 \cdot 2^{-\ell-7} \\
&\leq err_k \left(\eta(2+\eta) + 2^{-\ell+2} + 9E\right) + 129 \cdot 2^{-\ell-7} \\
&\leq err_k \left(\eta(2+\eta) + \varepsilon\right) + 129 \cdot 2^{-\ell-7},
\end{aligned}$$

where we used the induction hypothesis and the inequalities $E \leq \varepsilon/16 \leq 1/32$, $\ell \geq 6$, $\varepsilon \leq 1/2$, and $\eta < 1$. Hence, if $j \leq i$, then $\sum_{k=1}^{j-1} |\diamond(\bar{\mu}_{j,k} \cdot \bar{r}_{i,k}) - \mu_{j,k} r_{i,k}| \times \frac{1}{r_{j,j}}$ is

$$\begin{aligned}
&\leq \sum_{k=1}^{j-1} \left((\eta(2+\eta) + \varepsilon)\, err_k \frac{r_{k,k}}{r_{j,j}} + 129 \cdot 2^{-\ell-7} (\delta - \eta^2)^{k-j} \right) \\
&\leq (\eta(2+\eta) + \varepsilon) \cdot \sum_{k=1}^{j-1} \left(err_k (\delta - \eta^2)^{k-j}\right) + 3 \cdot 129 \cdot 2^{-\ell-7} (\delta - \eta^2)^{-j},
\end{aligned}$$

where we used the fact that $\delta - \eta^2 \leq 3/4$. Similarly, we have

$$\frac{|\langle \mathbf{b}_i, \mathbf{b}_j \rangle|}{r_{j,j}} \leq \frac{|r_{i,j}|}{r_{j,j}} + \sum_{k=1}^{j-1} |\mu_{j,k}||\mu_{i,k}| \frac{r_{k,k}}{r_{j,j}} \leq \sum_{k=1}^{j} (\delta - \eta^2)^{k-j} \leq 3(\delta - \eta^2)^{-j}.$$

Finally, by using the inequalities $d 2^{-\ell-1} \leq 2^{-7}$ and $d \geq 2$, we get, with equation $(*)$,

$$\begin{aligned}
err_j &\leq \frac{1}{1 - d2^{-\ell-1}} \cdot \left(2^{-\ell-1} + (\eta(2+\eta) + \varepsilon) \sum_{k=1}^{j-1} \left(err_k (\delta - \eta^2)^{k-j}\right) + 6d2^{-\ell-1} (\delta - \eta^2)^{-j}\right) \\
&\leq \frac{\eta(2+\eta) + \varepsilon}{1 - d2^{-\ell-1}} \cdot \sum_{k=1}^{j-1} \left(err_k (\delta - \eta^2)^{k-j}\right) + \frac{d 2^{-\ell+2}}{1 - d2^{-\ell-1}} (\delta - \eta^2)^{-j}.
\end{aligned}$$

Let $A = \frac{\eta(2+\eta)+\varepsilon}{1-d2^{-\ell-1}}$ and $B = \frac{d2^{-\ell+2}}{1-d2^{-\ell-1}}$. We define $u_1 = B$ and $u_i = A\sum_{k=1}^{i-1} u_k + B$ for $i > 1$. Then we have $err_i \leq (\delta - \eta^2)^{-i} u_i$ for any $i \leq j$. Besides, we also have $u_i \leq B(1+A)^{i-1}$. Overall, this gives

$$\begin{aligned} err_j &\leq \frac{d2^{-\ell+2}}{1-d2^{-\ell-1}}(\delta-\eta^2)^{-j} \cdot \left(1 + \frac{\eta(2+\eta)+\varepsilon}{1-d2^{-\ell-1}}\right)^{j-1} \\ &\leq \frac{d2^{-\ell+2}}{1-d2^{-\ell-1}} \cdot \left(\frac{1+\eta(2+\eta)+\varepsilon}{\delta-\eta^2}\right)^j \cdot \frac{1}{1+\eta(2+\eta)+\varepsilon} \cdot \left(1+d2^{-\ell+3}\right)^j \\ &\leq d2^{-\ell+2}\rho^j, \end{aligned}$$

where we used the fact that $(1+d2^{-\ell+3})^d \leq 2$, itself implied by $d^2 2^{-\ell+5} \leq 1$.

The result on the μ's is straightforward if we use the facts that $E \leq 1/32$ and $\ell \geq 6$. Furthermore, it is easy to slightly modify the proof to get the desired results on the $r_{d,j}$'s and $\mu_{d,j}$'s for $j < d$.

We now consider the third set of inequalities. We assume that the vector $\mathbf{b}_d$ is size-reduced for the factor η. The analysis is similar to the one just above. If $j \leq d$, we can bound the quantity $|\bar{s}_j^{(d)} - s_j^{(d)}|$ by

$$\begin{aligned} &\frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left(\diamond\|\mathbf{b}_d\|^2 + \sum_{k=1}^{j-1}|\diamond(\bar{\mu}_{d,k}\cdot\bar{r}_{d,k})|\right) + \sum_{k=1}^{j-1}|\diamond(\bar{\mu}_{d,k}\cdot\bar{r}_{d,k}) - \mu_{d,k}r_{d,k}| \\ &\quad + \left|\diamond\|\mathbf{b}_d\|^2 - \|\mathbf{b}_d\|^2\right| \\ &\leq \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left(s_j^{(d)} + 2\sum_{k=1}^{j-1} r_{k,k}\right) \\ &\quad + \frac{1}{1-d2^{-\ell-1}}\left(\sum_{k=1}^{j-1}|\diamond(\bar{\mu}_{d,k}\cdot\bar{r}_{d,k}) - \mu_{d,k}r_{d,k}| + 2^{-\ell-1}\|\mathbf{b}_d\|^2\right) \\ &\leq d2^{-\ell}\cdot s_j^{(d)} + d2^{-\ell+1}\cdot\sum_{k=1}^{j-1} r_{k,k} + d2^{-\ell+6}\cdot\sum_{k=1}^{j-1} r_{k,k}\rho^k, \end{aligned}$$

where we used the second set of inequalities of the theorem. As a consequence, we have

$$\begin{aligned} \left|\bar{s}_j^{(d)} - s_j^{(d)}\right| &\leq d2^{-\ell}\cdot s_j^{(d)} + r_{j,j}d2^{-\ell+4}(\delta-\eta^2)^{-j} + r_{j,j}d2^{-\ell+6}\cdot\sum_{k=1}^{j-1}(\delta-\eta^2)^{k-j}\rho^k \\ &\leq d2^{-\ell}s_j^{(d)} + r_{j,j}d2^{-\ell+9}\rho^j. \end{aligned}$$

This ends the proof of the theorem. ▯

The bound in Theorem 3 seems to be tight in the worst case: the classical Gram–Schmidt algorithm or the CFA become experimentally inaccurate with a precision $\leq d\log 3$ bits for certain lattice bases. Consider indeed the L^3-reduced lattice basis given by the rows of the following $d \times d$ matrix L:

$$L_{i,j} = \begin{cases} \left(\sqrt{4/3}\right)^{d-i} & \text{if} \quad i = j, \\ (-1)^{i-j+1}L_{j,j}\cdot\text{random}[0.49, 0.5] & \text{if} \quad j < i, \\ 0 & \text{if} \quad j > i. \end{cases}$$

To obtain an integral lattice, one can multiply the matrix L by a large scaling factor and round its entries. This matrix is already L^3-reduced. With double precision calculations, i.e., with 53-bit mantissae, the error on the $\mu_{i,j}$'s becomes significant (higher than 0.5) in dimension 35.

These bases show the tightness of the $d \log 3$ bound. By adding a suitable random vector to such bases, it is possible to make the LLL_FP routine of NTL loop forever in dimension 55 (see http://perso.ens-lyon.fr/damien.stehle/FPLLL.html). Nevertheless, these lattice bases seem to be pathological and the floating-point calculations seem to behave much more nicely in practice than in the worst case (see [35] for more details).

4.2. Accuracy of Babai's nearest plane algorithm. To estimate the accuracy of the lazy floating-point size-reduction algorithm given in Figure 5 and used in the L^2 algorithm, we first study a simpler floating-point version that is described in Figure 8.

```
Input:  A factor $\eta > 1/2$, an fp-precision $\ell + 1$, a basis $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, its Gram
matrix $G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$, and floating-point numbers $\bar{r}_{i,j}$ and $\bar{\mu}_{i,j}$ for $j \leq i < d$.
Output:  $x_1, \ldots, x_{d-1} \in \mathbb{Z}$, a vector $\mathbf{b}'_d = \mathbf{b}_d - \sum_{i<d} x_i \mathbf{b}_i$, and $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}'_d)$.

1. Compute the $\bar{\mu}_{d,j}$'s for $j < d$ with steps 2–7 of the CFA with "$i = d$."
2. For $i = d - 1$ down to 1 do
3.    $x_i \leftarrow \lfloor \bar{\mu}_{d,i} \rceil$.
4.    For $j = 1$ to $i - 1$ do $\bar{\mu}_{d,j} \leftarrow \diamond(\bar{\mu}_{d,j} - \diamond(x_i \cdot \bar{\mu}_{i,j}))$.
5. Compute $\mathbf{b}'_d$ and $G(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}, \mathbf{b}'_d)$.
```

FIG. 8. *A floating-point size-reduction algorithm.*

We use Theorem 3 to show stability properties of the algorithm of Figure 8.

THEOREM 4. *Let (δ, η) be a valid reduction factor (like in Theorem 2) and $\rho = \frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ with $\varepsilon \in (0, 1/2]$. Let $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ in $\mathbb{Z}^n$ be a d-dimensional lattice basis given as input to the algorithm of Figure 8 and $B = \max_i \|\mathbf{b}_i\|$. Assume that the truncated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{d-1})$ is (δ, η)-L^3-reduced and that the given Gram–Schmidt coefficients $\bar{r}_{i,j}$ and $\bar{\mu}_{i,j}$ are those that would have been returned by the CFA with working precision $\ell + 1$. Let $M = \max_j |\mu_{d,j}|$. Suppose that ℓ satisfies $\ell \geq 5 + 2\log d - \log \varepsilon + d \log \rho$. Then the algorithm of Figure 8 finds integers $x_1, \ldots, x_{d-1}$ such that for any $i < d$*

$$|x_i| \leq 3(1+\eta+\varepsilon)^{d-1-i}(M+1) \quad \textit{and} \quad \frac{|\langle \mathbf{b}'_d, \mathbf{b}^*_i \rangle|}{\|\mathbf{b}^*_i\|^2} \leq \frac{1}{2} + d\rho^d (M+1) 2^{-\ell+8}.$$

Furthermore, the execution finishes in $O\left(\frac{dn}{\ell}(\ell + d + \log B) \cdot \mathcal{M}(\ell)\right)$ bit operations.

Proof. We define $\mu^{(i)}_{d,j} = \mu_{d,j} - \sum_{k=i}^{d-1} x_k \mu_{k,j}$ for $i > j$. By using the first set of inequalities of Theorem 3, we have

$$\begin{aligned} |\diamond(x_i \cdot \bar{\mu}_{i,j}) - x_i \mu_{i,j}| &\leq \left(1 + 2^{-\ell-1}\right) |x_i| |\bar{\mu}_{i,j} - \mu_{i,j}| + 2^{-\ell-1} |x_i| |\mu_{i,j}| \\ &\leq |x_i| 2^{-\ell-1} \left(1 + 2^5 (1 + 2^{-\ell-1}) d\rho^j\right) \\ &\leq |x_i| d\rho^j 2^{-\ell+5}, \end{aligned}$$

where we used the inequalities $\eta < 1$ and $\ell \geq 6$. At step 4, we update the quantity $\mu_{d,j}$. It can be seen that in fact we compute sequentially the sum $\bar{\mu}^{(i)}_{d,j} = \bar{\mu}_{d,j} - \sum_{k=i}^{d-1} x_k \cdot \bar{\mu}_{k,j}$.

By using the error bound of [14] for this naive summation, we obtain that we can bound the quantity $|\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}|$ by

$$\begin{aligned}
&\frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left(|\bar{\mu}_{d,j}| + \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j})|\right) + \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j}) - x_k\mu_{k,j}| \\
&\quad + |\bar{\mu}_{d,j} - \mu_{d,j}| \\
&\leq \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left(M + \sum_{k=i}^{d-1} |x_k\mu_{k,j}|\right) + \frac{1}{1-d2^{-\ell-1}} \sum_{k=i}^{d-1} |\diamond(x_k \cdot \bar{\mu}_{k,j}) - x_k\mu_{k,j}| \\
&\quad + \frac{|\bar{\mu}_{d,j} - \mu_{d,j}|}{1-d2^{-\ell-1}} \\
&\leq \frac{d2^{-\ell-1}}{1-d2^{-\ell-1}}\left(M + \sum_{k=i}^{d-1} |x_k|\right) + \frac{d2^{-\ell+5}\rho^j}{1-d2^{-\ell-1}} \sum_{k=i}^{d-1} |x_k| + \frac{d2^{-\ell+4}\rho^j M}{1-d2^{-\ell-1}} \\
&\leq d2^{-\ell+6}\rho^j\left(M + \sum_{k=i}^{d-1} |x_k|\right),
\end{aligned}$$

where we used the second set of inequalities of Theorem 3. Assume that $x_j \neq 0$. By taking $i = j+1$ in the previous inequality and using $d\rho^j 2^{-\ell+6} \leq \min(\varepsilon, 1/2)$, we get

$$\begin{aligned}
|x_j| &\leq \frac{1}{2} + \left|\bar{\mu}_{d,j}^{(j+1)}\right| \\
&\leq \frac{1}{2} + \left|\bar{\mu}_{d,j}^{(j+1)} - \mu_{d,j}^{(j+1)}\right| + \left|\mu_{d,j}^{(j+1)}\right| \\
&\leq \frac{1}{2} + \left(1 + d2^{-\ell+6}\rho^j\right) M + (\eta+\varepsilon) \sum_{k=j+1}^{d-1} |x_k| \\
&\leq (1+\eta+\varepsilon)^{d-1-j}\left(\frac{1}{2} + \left(1 + d2^{-\ell+6}\rho^j\right) M\right) \\
&\leq \frac{3}{2}(1+\eta+\varepsilon)^{d-1-j}(1+M).
\end{aligned}$$

So far, we have proved the first statement of the theorem. Besides, for any $i > j$, we have the following inequalities:

$$\begin{aligned}
\left|\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}\right| &\leq \left(d2^{-\ell+6}\rho^j\right)\left(M + \frac{3}{2}(1+M)\sum_{k=i}^{d-1}(1+\eta+\varepsilon)^{d-1-k}\right) \\
&\leq d2^{-\ell+8}\rho^j(1+M)(1+\eta+\varepsilon)^{d-i},
\end{aligned}$$

where we used the fact that $\eta > 1/2$. This allows us to prove the second statement of the theorem and to bound the numbers occurring during the computation:

$$\begin{aligned}
\frac{|\langle \mathbf{b}'_d, \mathbf{b}_i^* \rangle|}{\|\mathbf{b}_i^*\|^2} &\leq \frac{1}{2} + d2^{-\ell+8}\rho^d(1+M), \\
\left|\bar{\mu}_{d,j}^{(i)}\right| &\leq d2^{-\ell+8}\rho^j(1+M)(1+\eta+\varepsilon)^{d-i} + M + \sum_{k=i}^{d-1} |x_k| = 2^{O(d)}(1+M).
\end{aligned}$$

We now consider the costs of arithmetic operations. The most expensive operations involving floating-point numbers are the multiplications. The computations on the exponents that correspond to these operations cost $O\left(\log\log(2^{O(d)}(1+M))\right)$ bit operations. From now on, we consider only the bit costs coming from the operations on the mantissae. Step 1 costs $O(d\log B + d^2 \cdot \mathcal{M}(\ell))$ bit operations. The cost of step 3 is lower than the cost of step 4. There are at most $O(d^2)$ arithmetic operations during the successive steps 4. Among them, the multiplications $\diamond(x_i \cdot \bar{\mu}_{i,j})$ are the most expensive. Since $x_i = \lfloor \bar{\mu}_{d,i}^{(i+1)} \rceil$, the integer x_i can be represented on $O(\ell)$ bits: if $\ell < d$, then x_i can be written $2^{t_i} x_i'$ for some integers x_i' and $t_i = O(\log(d + \log(1+M)))$. As a consequence, the total cost of the successive steps 4 is $O(d^2 \cdot \mathcal{M}(\ell))$. At step 5, we have $O(dn)$ arithmetic operations involving integers only. By using the relation $|\langle \mathbf{b}_i, \mathbf{b}_d \rangle| \leq \|\mathbf{b}_i\| \cdot \|\mathbf{b}_d\|$ and our bound on the x_i's, we see that all the involved integers are of length $2^{O(d)} \cdot B^2$. The most costly operations are of the type $x_i \cdot \langle \mathbf{b}_d, \mathbf{b}_i \rangle$, and since the x_i's are of length $O(\ell)$, the cost of step 5 is $O\left(\frac{dn}{\ell}(d + \log B) \cdot \mathcal{M}(\ell)\right)$. ◻

Notice that by using the relation $\log M = O(d + \log B)$ (coming from the fact that the $d-1$ first vectors are L^3-reduced), the last theorem implies that taking $\ell = O(d + \log B)$ is sufficient to make the $|\mu_{d,i}|$'s smaller than η. The drawback of this approach is that one should have previously computed the $r_{i,j}$'s and $\mu_{i,j}$'s with precision $O(d + \log B)$. This seems an overkill since $O(d + \log M)$ bits suffice and M is usually far smaller than B. Indeed, in the case of Euclid's gcd algorithm, the analogy is that the quotients would be computed by using all the bits of the remainders, instead of only the most significant ones (see below for more details).

The lazy size-reduction algorithm from Figure 5 is a way to work around the difficulty that M cannot be tightly bounded in advance. Using only a $O(d)$-bit precision, it finds the x_j's progressively by performing successive size-reduction steps, each one making $\log M$ decrease by $\Omega(d)$, until we reach $\max_{j<\kappa} |\bar{\mu}_{\kappa,j}| \leq \bar{\eta}$. This strategy is somewhat similar to the size-reduction routine of the floating-point L^3 algorithm of NTL [45], which repeatedly applies the size-reduction algorithm until nothing happens.

The lazy size-reduction algorithm uses $\ell = (\log \frac{(1+\eta)^2}{\delta - \eta^2} + C)d + o(d)$ with an arbitrary $C > 0$. The CFA with ℓ gives the input $\bar{r}_{i,j}$'s and $\bar{\mu}_{i,j}$'s, which by Theorem 3 have their $\approx Cd$ leading bits correct. Therefore, the $r_{i,j}$'s and $\mu_{i,j}$'s may not be known sufficiently well to perform the size-reduction in one single step, but Theorem 4 gives that their approximations suffice to make $M = \max_{i<\kappa} \frac{|\langle \mathbf{b}_\kappa, \mathbf{b}_i^* \rangle|}{\|\mathbf{b}_i^*\|^2}$ decrease by $\approx Cd$ bits. By making $O(1 + \frac{\log M}{d})$ such incomplete size-reductions, size-reduction can be achieved.

Theorem 5. *Let (δ, η) be a valid pair (like in Theorem 2) and $\rho = \frac{(1+\eta)^2 + \varepsilon}{\delta - \eta^2}$ with $\varepsilon \in (0, 1/2]$. Let $C > 0$ be a constant. Let $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ be a d-dimensional lattice basis in $\mathbb{Z}^n$ given as input to the algorithm of Figure 5 and $B = \max_i \|\mathbf{b}_i\|$. Assume that the truncated basis $(\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1})$ is (δ, η)-L^3-reduced and that the given $\bar{r}_{i,j}$'s and $\bar{\mu}_{i,j}$'s are those that would have been returned by the CFA. Let $M = \max_{j<\kappa} |\mu_{\kappa,j}|$. If $\ell \geq 10 + 2\log d - \log\min\left(\varepsilon, \eta - \frac{1}{2}\right) + d\,(C + \log \rho)$, the algorithm of Figure 5 provides a correct output and the returned $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s, and $\bar{s}_j^{(\kappa)}$'s are those that would have been returned by the CFA. Furthermore, if $\ell = O(d)$, the execution finishes in $O\left(\frac{n}{d}(d + \log B)(d + \log M) \cdot \mathcal{M}(d)\right)$ bit operations.*

Proof. We start by the correctness properties of the algorithm. At the last iteration of the main loop, we have $|\bar{\mu}_{\kappa,j}| \leq \bar{\eta}$ for all $j < \kappa$. By using the second set of inequalities of Theorem 3, we obtain

$$\begin{aligned}\max_{j<\kappa}|\mu_{\kappa,j}| &\leq \bar{\eta} + d\rho^{d-1}2^{-\ell+4}\\ &\leq \frac{1+2^{-\ell-1}}{2}\left(\diamond(\eta)+\frac{1}{2}\right)+d\rho^{d-1}2^{-\ell+4}\\ &\leq \frac{1+2^{-\ell-1}}{2}\left(\eta+2^{-\ell-1}+\frac{1}{2}\right)+d\rho^{d-1}2^{-\ell+4}\\ &\leq \frac{\eta+1/2}{2}+2^{-\ell}+d\rho^{d-1}2^{-\ell+4}\\ &\leq \frac{\eta+1/2}{2}+d\rho^{d}2^{-\ell+5}.\end{aligned}$$

With the hypothesis on ℓ, we obtain that $d\rho^d 2^{-\ell+5} \leq 2^{-6}\left(\eta-\frac{1}{2}\right)$. This gives us that $\frac{|\langle \mathbf{b}'_\kappa, \mathbf{b}_i^*\rangle|}{\|\mathbf{b}_i^*\|^2} \leq \eta$ for all $j<\kappa$.

We now consider the impact on M of one iteration of the main loop. Let M_1 be the "new M" after the loop iteration. Theorem 4 and the hypothesis on ℓ give the inequalities

$$M_1 \leq \frac{1}{2}+d\rho^d(M+1)2^{-\ell+8} \leq \frac{1}{2}+\frac{\varepsilon}{8}+2^{-Cd}\frac{\varepsilon}{8}M,$$

where $\varepsilon=\eta-1/2$. The kth iterate of M is thus smaller than $\frac{1/2+\varepsilon/8}{1-\varepsilon/8}+2^{-Cdk}M$. Therefore, within $1+\frac{1}{Cd}O\left(-\log\varepsilon+\log M\right)$ loop iterations, we have that the current value of M is $\leq \frac{1/2+\varepsilon/8}{1-\varepsilon/8}\left(1+\frac{\varepsilon}{128}\right)$. Theorem 3 implies that at that moment we have $\max_{j<\kappa}|\bar{\mu}_{\kappa,j}| \leq \frac{1/2+\varepsilon/8}{1-\varepsilon/8}\left(1+\frac{\varepsilon}{64}\right)$. Since $\bar{\eta} \geq \frac{1}{2}+\frac{\varepsilon}{2}-2^{-\ell} \geq \frac{1}{2}+\frac{\varepsilon}{2}-\frac{\varepsilon}{128}$, we then have $\max_{j<\kappa}|\bar{\mu}_{\kappa,j}| \leq \bar{\eta}$. We conclude that the number of loop iterations is $1+\frac{1}{Cd}O\left(-\log\varepsilon+\log M\right)$.

Suppose now that $\ell=O(d)$. Theorem 4 gives that the cost of one loop iteration is bounded by $O\left(n(d+\log B)\mathcal{M}(d)\right)$, since during the execution of the algorithm, the entries of the Gram matrix remain integers of length bounded by $O(d+\log B)$. The fact that we have additional vectors in the basis (namely $\mathbf{b}_{\kappa+1},\ldots,\mathbf{b}_d$) is taken into account in the complexity bound. Finally, the overall cost of the algorithm is bounded by $O(n(d+\log B)(1+\frac{\log M}{d})\cdot\mathcal{M}(d))$. ∎

4.3. Application to the L^2 algorithm. The correctness of the L^2 algorithm directly follows from the following theorem.

THEOREM 6. *Let (δ,η) be a valid pair (like in Theorem 2) and $\rho=\frac{(1+\eta)^2+\varepsilon}{\delta-\eta^2}$ with $\varepsilon\in(0,1/2]$. Let $C>0$ be a constant. Let $(\mathbf{b}_1^{(1)},\ldots,\mathbf{b}_d^{(1)})$ in $\mathbb{Z}^n$ be a lattice basis given as input to the L^2 algorithm. For any loop iteration t, let $(\mathbf{b}_1^{(t)},\ldots,\mathbf{b}_d^{(t)})$ denote the current basis at the beginning of the tth iteration. Assume that ℓ satisfies $d^2\rho^d 2^{-\ell+10+Cd} \leq \min\left(\varepsilon,\eta-\frac{1}{2},1-\delta\right)$. Then the following hold:*

1. *For any t, the truncated basis $(\mathbf{b}_1^{(t)},\ldots,\mathbf{b}_{\kappa(t)-1}^{(t)})$ is L^3-reduced with factor pair (δ,η).*
2. *For any $1\leq i\leq d$, we have $\max_{j\leq i}\|\mathbf{b}_j^{(t)*}\| \leq \max_{j\leq i}\|\mathbf{b}_j^{(1)*}\|$ and $\|\mathbf{b}_i^{(t)}\| \leq \sqrt{d}\cdot\max_{j\leq i}\|\mathbf{b}_i^{(1)}\|$.*

Proof. We show the result by induction on t. Clearly, all these properties are valid for $t=1$, since $\kappa(1)=2$. Assume now that we are performing the tth loop iteration.

We show that Lovász's tests work as desired. Recall that the vector $\mathbf{b}_\kappa$ is swapped with the vector $\mathbf{b}_i$ for $i < \kappa$ if and only if for any $j \in [\![i, \kappa-1]\!]$ we have $\bar{s}_j^{(\kappa)} \leq \diamond\left(\bar{\delta} \cdot \bar{r}_{j,j}\right)$. First of all, let us analyze the relevance of the quantity $\diamond\left(\bar{\delta} \cdot \bar{r}_{j,j}\right)$. We have

$$\left|\bar{\delta} - \frac{\delta+1}{2}\right| \leq \frac{1}{2}\left(2^{-\ell-1}(\delta+1) + (1+2^{-\ell-1}) \cdot |\diamond(\delta) - \delta|\right) \leq 2^{-\ell}.$$

The inequalities above and Theorem 3 give us that

$$\begin{aligned}\left|\diamond\left(\bar{\delta} \cdot \bar{r}_{j,j}\right) - \frac{\delta+1}{2} r_{j,j}\right| &\leq 2^{-\ell-1}\frac{\delta+1}{2} r_{j,j} + (1+2^{-\ell-1})\left|\bar{\delta} \cdot \bar{r}_{j,j} - \frac{\delta+1}{2} r_{j,j}\right| \\ &\leq 2^{-\ell-1} r_{j,j} + (1+2^{-\ell-1})\left|\bar{\delta} - \frac{\delta+1}{2}\right| \bar{r}_{j,j} \\ &\quad + (1+2^{-\ell-1})\frac{\delta+1}{2}\left|r_{j,j} - \bar{r}_{j,j}\right| \\ &\leq 2^{-\ell-1} r_{j,j} + (1+2^{-\ell-1})2^{-\ell}\left(r_{j,j} + |r_{j,j} - \bar{r}_{j,j}|\right) \\ &\quad + (1+2^{-\ell-1})\left|r_{j,j} - \bar{r}_{j,j}\right| \\ &\leq 2^{-\ell+1} r_{j,j} + (1+2^{-\ell})(1+2^{-\ell-1}) d\rho^{d-1} 2^{-\ell+2} r_{j,j} \\ &\leq d\rho^d 2^{-\ell+3} r_{j,j}.\end{aligned}$$

Assume now that the vector $\mathbf{b}_\kappa$ is swapped with the vector $\mathbf{b}_i$. The study above and Theorem 3 give us that

$$s_i^{(\kappa)}(1 - d2^{-\ell}) \leq r_{i,i}\left(\frac{\delta+1}{2} + d\rho^d 2^{-\ell+3} + d\rho^{d-1} 2^{-\ell+9}\right) \leq r_{i,i}\left(\frac{\delta+1}{2} + d\rho^d 2^{-\ell+9}\right).$$

Therefore, since $d^2\rho^d 2^{-\ell+9} \leq 1 - \delta$, if the vector $\mathbf{b}_\kappa$ is swapped with the vector $\mathbf{b}_i$, then they would have also been swapped with the classical L^3 algorithm with the factor $\frac{\delta+7}{8} \in (\delta, 1)$. Oppositely, assume that the vectors $\mathbf{b}_\kappa$ and $\mathbf{b}_i$ are not swapped. Theorem 3 gives

$$s_i^{(\kappa)}(1 + d2^{-\ell}) \geq r_{i,i}\left(\frac{\delta+1}{2} - d\rho^d 2^{-\ell+3} - d\rho^{d-1} 2^{-\ell+9}\right) \geq r_{i,i}\left(\frac{\delta+1}{2} - d\rho^d 2^{-\ell+9}\right).$$

As a consequence, if the vectors $\mathbf{b}_\kappa$ and $\mathbf{b}_i$ are not swapped, they would not have been swapped with the classical L^3 algorithm with the factor δ. This gives the first statement of the theorem.

For the second statement, observe that during a swap between the vectors $\mathbf{b}_\kappa$ and $\mathbf{b}_{\kappa-1}$, we have

1. $\left\|\mathbf{b}_{\kappa-1}^{*new}\right\| \leq \left\|\mathbf{b}_{\kappa-1}^{*old}\right\|$ because of Lovász's condition,
2. $\left\|\mathbf{b}_{\kappa}^{*new}\right\| \leq \left\|\mathbf{b}_{\kappa-1}^{*old}\right\|$ because the vector $\mathbf{b}_\kappa^{*new}$ is an orthogonal projection of the vector $\mathbf{b}_{\kappa-1}^{*old}$,

which gives the first part of the second statement. Finally, if the vector $\mathbf{b}_i^{(t)}$ appears during the execution of the algorithm and is size-reduced, we have

$$\left\|\mathbf{b}_i^{(t)}\right\|^2 \leq d \cdot \max_{j \leq i}\left\|\mathbf{b}_j^{(t)*}\right\|^2 \leq d \cdot \max_{j \leq i}\left\|\mathbf{b}_j^{(1)*}\right\|^2 \leq d \cdot \max_{j \leq i}\left\|\mathbf{b}_j^{(1)}\right\|^2.$$

This proves the second part of the statement for all $i < \kappa(t)$. If $i \geq \kappa(t)$, we consider the largest $t' < t$ such that $\kappa(t'+1) - 1 = i$. The loop iteration t' was the one during which the vector $\mathbf{b}_i^{(t)}$ was created. If t' does not exist, this vector is an initial vector and the result is obvious. Otherwise the vector $\mathbf{b}_i^{(t)} = \mathbf{b}_{\kappa(t'+1)-1}^{(t'+1)}$ is size-reduced at the $(t'+1)$th loop iteration. Thus we have $\|\mathbf{b}_i^{(t)}\|^2 \leq d \cdot \max_{j \leq \kappa(t'+1)-1} \|\mathbf{b}_j^{(1)}\|^2$. This ends the proof of the theorem. $\square$

5. Complexity analysis of the $\mathbf{L^2}$ algorithm. In section 4.3, we showed that an accuracy of $O(d)$ bits suffices to check Lovász's conditions. For any Lovász test, the index κ either increases or decreases by one, and when it decreases, the quantity $\prod_{i=1}^{d} \|\mathbf{b}_i^*\|^{2(d-i+1)}$ decreases by a factor of at least $\frac{\delta+7}{8} < 1$. It is a standard L^3 argument that this quantity is actually an integer (it is a product of squared volumes of integer lattices) and is initially bounded by $B^{O(d^2)}$. But during the execution of the algorithm, the difference between the numbers of decreases and increases of κ is $O(d)$, so there are $O(d^2 \log B)$ loop iterations.

In this section, we prove the claimed complexity bound $O(d^4 n \log B(d + \log B))$. This is done by generalizing a cascade phenomenon appearing in the analysis of Euclid's gcd algorithm.

5.1. Analyzing Euclid's gcd algorithm. As mentioned in the introduction, the L^3 algorithm can be viewed as a high-dimensional generalization of Euclid's algorithm to compute gcds. But this analogy is not completely satisfying: indeed, Euclid's algorithm has a quadratic complexity bound without fast integer arithmetic, whereas the L^3 algorithm is cubic for any fixed dimension without fast integer arithmetic.

Recall Euclid's algorithm: given as input two integers $r_0 > r_1 > 0$, it successively computes the quotients q_i and remainders r_i defined by

$$q_i = \lfloor r_{i-1}/r_i \rfloor \quad \text{and} \quad r_{i+1} = r_{i-1} - q_i r_i,$$

until $r_{\tau+1} = 0$ for some loop iteration τ. Then r_τ is the gcd of the integers r_0 and r_1. It is well known that the remainders decrease at least geometrically, so that the number of Euclidean divisions is $\tau = O(\log r_0)$. A very naive analysis of Euclid's algorithm states that the algorithm performs $O(\log r_0)$ arithmetic operations on integers of lengths bounded by $O(\log r_0)$, so that the overall cost is bounded by $O(\log^3 r_0)$. A well-known more subtle analysis notices that the cost of computing q_i and r_{i+1} without fast integer arithmetic is bounded by $O(\log r_{i-1} \cdot (1+\log q_i)) = O(\log r_0 \cdot (1 + \log r_{i-1} - \log r_i))$. Summed over all the steps, all but two terms "$\log r_i$" vanish, leading to the classical quadratic complexity bound.

Unfortunately, this amortized analysis of Euclid's algorithm cannot be extended to the standard L^3 algorithm, because the GSO coefficients stay big. In Euclid's algorithm, the numbers get smaller and smaller: in L^3, the basis vectors get shorter and shorter, but there still remain very large GSO coefficients. Surprisingly, we will show that the amortized analysis of Euclid's algorithm can be extended to the L^2 algorithm, and this would not be possible without a floating-point precision independent of $\log B$. The main difficulty is to generalize the cancellation of all but a very few terms in the sum of the costs of consecutive loop iterations. In Euclid's algorithm, this cancellation is trivial because two consecutive costs compensate each other directly. The phenomenon is much more complicated in higher dimension: the cost of the tth iteration will not necessarily be balanced by the cost of the previous $(t-1)$th iteration but by the cost of the t'th iteration for some $t' < t$. Special care must be taken so that the t''s do not collide.

5.2. An amortized analysis in arbitrary dimension. We now complete the proof of Theorem 2. We already know that the number of loop iterations is $\tau = O(d^2 \log B)$. Thanks to Theorem 5, we also know that the tth loop iteration costs $O(dn(d+\log B)(d+\log M(t)))$ bit operations, where $M(t) = \max_{j<\kappa(t)} |\mu^{(t)}_{\kappa(t),j}|$. By analogy with Euclid's gcd algorithm, we make terms cancel out in the sum over the loop iterations of the "$\log M(t)$'s." For this purpose, we define the index $\alpha(t)$ as the smallest swapping index since the last time the index κ was at least $\kappa(t)$ and 1 if this is the first time we have $\kappa = \kappa(t)$.

LEMMA 7. *Let t be a loop iteration. Let $\varphi(t) = \max(t' | t' < t, \kappa(t') \geq \kappa(t))$ if it exists and* 1 *otherwise, and let $\alpha(t) = \min(\kappa(t'), t' \in [\![\varphi(t), t-1]\!]) - 1$ if $t \neq 1$ and* 1 *otherwise. Then we have*

$$\log M(t) \leq O(d) + \log \left\| \mathbf{b}^{(t)}_{\kappa(t)} \right\| - \log \left\| \mathbf{b}^{(t)}_{\alpha(t)} \right\| .$$

Proof. Between the loop iterations $\varphi(t)$ and t, the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}$ remain unchanged, and because of the size-reductions, each vector created during these loop iterations is size-reduced with respect to the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{\alpha(t)-1}$. This includes the vector $\mathbf{b}^{(t)}_{\kappa(t)}$. As a consequence, by using the fact that $(\mathbf{b}^{(t)}_1, \ldots, \mathbf{b}^{(t)}_{\kappa(t)-1})$ is L^3-reduced (thanks to Theorem 6), we have

$$\begin{aligned} M(t) = \max_{i<\kappa(t)} |\mu_{\kappa(t),i}| &= \max\left(\max_{i<\alpha(t)} |\mu_{\kappa(t),i}|, \max_{i \in [\![\alpha(t),\kappa(t)-1]\!]} |\mu_{\kappa(t),i}| \right) \\ &\leq \max\left(\eta, \max_{i \in [\![\alpha(t),\kappa(t)-1]\!]} \frac{\left\| \mathbf{b}^{(t)}_{\kappa(t)} \right\|}{\left\| \mathbf{b}^{(t)*}_{i} \right\|} \right) \\ &\leq \eta + (\delta - \eta^2)^{-(\kappa(t)-\alpha(t))/2} \frac{\left\| \mathbf{b}^{(t)}_{\kappa(t)} \right\|}{\left\| \mathbf{b}^{(t)*}_{\alpha(t)} \right\|} \\ &\leq \eta + \sqrt{d}(\delta - \eta^2)^{-d/2} \frac{\left\| \mathbf{b}^{(t)}_{\kappa(t)} \right\|}{\left\| \mathbf{b}^{(t)}_{\alpha(t)} \right\|}, \end{aligned}$$

since $\|\mathbf{b}^{(t)}_{\alpha(t)}\| \leq \sqrt{d} \cdot \max_{i \leq \alpha(t)} \|\mathbf{b}^{(t)*}_i\| \leq \sqrt{d}(\delta - \eta^2)^{-\alpha(t)/2} \|\mathbf{b}^{(t)*}_{\alpha(t)}\|$. ☐

We will subdivide the sum of the $\log M(t)$'s over the successive loop iterations into $O(d)$ subsums according to the value of the index $\kappa(t)$:

$$\sum_{t \leq \tau} \left(d + \log \left\| \mathbf{b}^{(t)}_{\kappa(t)} \right\| - \log \left\| \mathbf{b}^{(t)}_{\alpha(t)} \right\| \right) \leq \tau d + \sum_{k=2}^{d} \sum_{t, \kappa(t)=k} \left(\log \left\| \mathbf{b}^{(t)}_{k} \right\| - \log \left\| \mathbf{b}^{(t)}_{\alpha(t)} \right\| \right).$$

For each of these subsums, we keep $k-1$ positive terms and $k-1$ negative terms and make the others vanish in a progressive cancellation. Terms proportional to d can appear from such cancellations, but they will be dominated by the above "τd." The crucial point to do this is the following.

LEMMA 8. *Let $k \in [\![2, d]\!]$ and $t_1 < \cdots < t_k$ be loop iterations of the L^2 algorithm such that for any $j \leq k$, we have $\kappa(t_j) = k$. Then there exists $j < k$ with*

$$d(\delta - \eta^2)^{-d} \left\| \mathbf{b}^{(t_j)}_{\alpha(t_j)} \right\| \geq \left\| \mathbf{b}^{(t_k)}_{k} \right\| .$$

To prove this result, we need the following technical fact.

LEMMA 9. *Let T and j be integers such that $\kappa(T) \geq j \geq \kappa(T+1)$. We have*

$$\max_{i \leq j} \left\| \mathbf{b}_i^{(T+1)*} \right\| \leq \max_{i<j} \left\| \mathbf{b}_i^{(T)*} \right\| \quad \text{and} \quad \max_{i \leq j} \left\| \mathbf{b}_i^{(T+1)} \right\| \leq \sqrt{d} \cdot \max_{i<j} \left\| \mathbf{b}_i^{(T)} \right\|.$$

Proof. If $i \leq j$ is different from $\kappa(T+1)-1$, then the vector $\mathbf{b}_i^{(T+1)}$ is a vector $\mathbf{b}_{i'}^{(T)}$ for some $i' \leq j-1$. Thanks to the size-reduction and to Theorem 6, it suffices to show that $\|\mathbf{b}_{\kappa(T+1)-1}^{(T+1)*}\| \leq \max_{i<j} \|\mathbf{b}_i^{(T)*}\|$. Since the Lovász test has failed for the loop iteration T and the index $\kappa(T+1)-1$, we have $\|\mathbf{b}_{\kappa(T+1)-1}^{(T+1)*}\|^2 \leq \delta \cdot \|\mathbf{b}_{\kappa(T+1)-1}^{(T)*}\|^2$, which implies

$$\left\| \mathbf{b}_{\kappa(T+1)-1}^{(T+1)*} \right\| \leq \left\| \mathbf{b}_{\kappa(T+1)-1}^{(T)*} \right\|. \qquad \square$$

Proof of Lemma 8. We choose $j = \max(i < k, \alpha(t_i) \geq i)$. This definition is valid because the maximized set is not empty (it contains $i = 1$). Since $\alpha(t_k) < k$ and $\kappa(t_k) = \kappa(t_{k-1}) = k$, there exists a first loop iteration $T_k \in [\![t_{k-1}, t_k - 1]\!]$ such that $\kappa(T_k) \geq k \geq \kappa(T_k+1)$. Because of Theorem 6 (for the first inequality) and Lemma 9 (for the second inequality), we have

$$\left\| \mathbf{b}_k^{(t_k)} \right\| \leq \sqrt{d} \cdot \max_{i \leq k} \left\| \mathbf{b}_i^{(T_k+1)} \right\| \leq d \cdot \max_{i \leq k-1} \left\| \mathbf{b}_i^{(T_k)} \right\|.$$

By definition of T_k, the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ do not change between the loop iterations t_{k-1} and T_k. Since the vectors $\mathbf{b}_1^{(t_{k-1})}, \ldots, \mathbf{b}_{k-1}^{(t_{k-1})}$ are L^3-reduced, we have

$$\left\| \mathbf{b}_k^{(t_k)} \right\| \leq d \cdot \max_{i \leq k-1} \left\| \mathbf{b}_i^{(t_{k-1})} \right\| \leq d(\delta - \eta^2)^{-d/2} \cdot \max_{i \leq k-1} \left\| \mathbf{b}_i^{(t_{k-1})*} \right\|.$$

If $j = k-1$, we have the result because in this case

$$\max_{i \leq k-1} \left\| \mathbf{b}_i^{(t_{k-1})*} \right\| \leq (\delta - \eta^2)^{-d/2} \left\| \mathbf{b}_{\alpha(t_{k-1})}^* \right\| \leq (\delta - \eta^2)^{-d/2} \left\| \mathbf{b}_{\alpha(t_{k-1})} \right\|.$$

Otherwise there exists a first loop iteration $T_{k-1} \in [\![t_{k-2}, t_{k-1} - 1]\!]$ such that $\kappa(T_{k-1}) \geq k-1 \geq \kappa(T_{k-1}+1)$. From Lemma 9, we have

$$\begin{aligned} \left\| \mathbf{b}_k^{(t_k)} \right\| &\leq d(\delta - \eta^2)^{-d/2} \cdot \max_{i \leq k-1} \left\| \mathbf{b}_i^{(T_{k-1}+1)*} \right\| \\ &\leq d(\delta - \eta^2)^{-d/2} \cdot \max_{i \leq k-2} \left\| \mathbf{b}_i^{(T_{k-1})*} \right\| \\ &\leq d(\delta - \eta^2)^{-d/2} \cdot \max_{i \leq k-2} \left\| \mathbf{b}_i^{(t_{k-2})*} \right\|. \end{aligned}$$

If $j = k-2$, we have the result; otherwise we go on constructing loop iterations T_i in a similar fashion to obtain the result. $\square$

It is now possible to complete the complexity analysis of the L^2 algorithm. Let $k \in [\![2, d]\!]$ and $t_1 < \cdots < t_{\tau_k} = \{t \leq \tau, \kappa(t) = k\}$. We extract from the global sum the terms corresponding to these loop iterations. Theorem 6 and the fact we are dealing with an integer lattice ensure that

$$\sum_{i=1}^{\tau_k} \log \frac{\left\| \mathbf{b}_k^{(t_i)} \right\|}{\left\| \mathbf{b}_{\alpha(t_i)}^{(t_i)} \right\|} \leq (k-1) \log(\sqrt{d} B) + \sum_{i=k}^{\tau_k} \log \left\| \mathbf{b}_k^{(t_i)} \right\| - \sum_{i=1}^{\tau_k - k+1} \log \left\| \mathbf{b}_{\alpha(t_i)}^{(t_i)} \right\|.$$

Lemma 8 helps to tightly bound the right-hand term above. First, we apply it with $t_1, \ldots, t_k$. This shows that there exists $j < k$ such that $\|\mathbf{b}_k^{(t_k)}\| \leq d(\delta - \eta^2)^{-d}\|\mathbf{b}_{\alpha(t_j)}^{(t_j)}\|$. The indices "$i = k$" in the positive sum and "$i = j$" in the negative sum cancel out, and a term "$-d\log(\delta - \eta^2) + \log d$" appears. Then we use Lemma 8 with t_{k+1} and the $k-1$ first t_i's that remain in the negative sum. It is easy to see that t_{k+1} is larger than any of them, so that we can have another "positive-negative" pair which cancels out in another "$-d\log(\delta - \eta^2) + \log d$." We perform this operation $\tau_k - k + 1$ times to obtain

$$\sum_{i=1}^{\tau_k} \log \frac{\left\|\mathbf{b}_k^{(t_i)}\right\|}{\left\|\mathbf{b}_{\alpha(t_i)}^{(t_i)}\right\|} \leq (k-1)\log(\sqrt{d}B) + \tau_k\left(-d\log(\delta - \eta^2) + \log d\right).$$

The fact that $\sum_k \tau_k = \tau$ finally gives

$$\sum_{t \leq \tau} (O(d) + \log M(t)) = O(\tau d + d^2 \log dB).$$

6. The L^2 algorithm for linearly dependent vectors. We show in this section that the L^2 algorithm can be extended to linearly dependent vectors. The modification resembles the one described in [38]. It can be used to find integer linear relations between linearly dependent lattice vectors. We add a new index ζ that has the following meaning: any vector of index $< \zeta$ is a zero vector, while any vector of index $\geq \zeta$ is nonzero. At any loop iteration of the algorithm, the vectors $\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_{\kappa-1}$ are L^3-reduced. The rest of the modified algorithm is the same as in the L^2 algorithm of Figure 6. The modified algorithm is given in Figure 9.

```
Input:  A valid pair $(\delta, \eta)$ like in Theorem 2, nonzero vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$,
and an fp-precision $\ell + 1$.
Output:  An L$^3$-reduced basis with factor pair $(\delta, \eta)$.
Variables:  An integral matrix $G$, floating-point numbers $\bar{r}_{i,j}$, $\bar{\mu}_{i,j}$, and $\bar{s}_i$.

1. Compute exactly $G = G(\mathbf{b}_1, \ldots, \mathbf{b}_d)$.
2. $\bar{\delta} \leftarrow \frac{\diamond(\diamond(\delta)+1)}{2}, \bar{r}_{1,1} \leftarrow \diamond(\langle \mathbf{b}_1, \mathbf{b}_1 \rangle), \kappa \leftarrow 2, \zeta \leftarrow 0$. While $\kappa \leq d$, do
3.    Size-reduce $\mathbf{b}_\kappa$ using the algorithm of Figure 5 for the vectors $\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_\kappa$.
It updates the floating-point GSO.
4.    $\kappa' \leftarrow \kappa$. While $\kappa \geq \zeta + 2$ and $\diamond(\bar{\delta} \cdot \bar{r}_{\kappa-1,\kappa-1}) \geq \bar{s}_{\kappa-1}^{(\kappa')}$, do $\kappa \leftarrow \kappa - 1$.
5.    For $i = \zeta + 1$ to $\kappa - 1$ do $\bar{\mu}_{\kappa,i} \leftarrow \bar{\mu}_{\kappa',i}, \bar{r}_{\kappa,i} \leftarrow \bar{r}_{\kappa',i}, \bar{r}_{\kappa,\kappa} \leftarrow \bar{s}_{\kappa}^{(\kappa')}$.
6.    Insert $\mathbf{b}_{\kappa'}$ right before $\mathbf{b}_\kappa$ and update $G$ accordingly.
7.    If $\mathbf{b}_\kappa = \mathbf{0}$, $\zeta \leftarrow \zeta + 1$.
8.    $\kappa \leftarrow \max(\zeta + 2, \kappa + 1)$.
9. Output $(\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_d)$.
```

FIG. 9. *The modified L^2 algorithm.*

The following theorem gives the bit complexity of the modified L^2 algorithm.

THEOREM 10. *Let (δ, η) be such that $1/4 < \delta < 1$ and $1/2 < \eta < \sqrt{\delta}$. Let $c > \log \frac{(1+\eta)^2}{\delta - \eta^2}$ be a constant. Given as input d vectors $(\mathbf{b}_1, \ldots, \mathbf{b}_d)$ in $\mathbb{Z}^n$ with $\max_i \|\mathbf{b}_i\| \leq B$, the modified L^2 algorithm of Figure 9 with $\ell = ck + o(k)$ outputs a (δ, η)-L^3-reduced basis of the lattice L generated by the $\mathbf{b}_i$'s in $O(n(k + \log B)(k^2 \log B + d(d-k)) \cdot \mathcal{M}(k))$*

bit operations, where $k = \dim L \leq d$. More precisely, if τ denotes the number of loop iterations, then the running time is $O(n(\tau + k \log kB)(k + \log B) \cdot \mathcal{M}(k))$.

Since most of the proof of this theorem is identical to the one of Theorem 2, we point out only the differences. Notice first that at any moment of the algorithm, there can be no more than k nonzero vectors of index smaller than κ: Theorem 6 remains valid, so that the vectors $\mathbf{b}_{\zeta+1}, \ldots, \mathbf{b}_{\kappa-1}$ are L^3-reduced, but L^3-reduced vectors must be linearly independent. This implies that in the modified version of Theorem 5, the complexity bound of the lazy size-reduction becomes

$$O\left(\frac{n}{k}(k + \log B)(k + \log M) \cdot \mathcal{M}(k)\right) \text{ bit operations.}$$

It remains to bound the number of loop iterations of the modified L^2 algorithm. For all $i \leq k$, let d_i be the product of the i first nonzero $\|\mathbf{b}_i^*\|^2$'s. The d_i's are positive integers since they are the determinants of the Gram matrices of subsets of the $\mathbf{b}_i$'s. We also define

$$D = \left(\prod_{i \leq \dim L} d_i\right) \cdot \left(\prod_{i, \mathbf{b}_i^* = \mathbf{0}} 2^i\right).$$

The quantity D is an integer that is initially bounded by $B^{O(k^2)} \cdot 4^{d(d-k)}$. We show that this quantity decreases by some constant factor each time there is a swap in the modified L^2 algorithm of Figure 9. This implies that there can be no more than $O(k^2 \log B + d(d-k))$ loop iterations.

Assume that the vectors $\mathbf{b}_\kappa$ and $\mathbf{b}_{\kappa-1}$ are not swapped. We consider three cases:

1. Both vectors $\mathbf{b}_{\kappa-1}^*$ and $\mathbf{b}_\kappa^*$ are nonzero. Then the right-hand side of the quantity D is constant, and the left-hand side decreases by a factor $\geq 1/\delta$: the first d_i's do not change because none of the terms of the corresponding products is changing; the last d_i's do not change either because the only terms of the corresponding products that change are $\|\mathbf{b}_{\kappa-1}^*\|$ and $\|\mathbf{b}_\kappa^*\|$, but their product remains the same; the remaining d_i has a single term that is changing, the term $\|\mathbf{b}_{\kappa-1}^*\|^2$, which is decreasing by a factor $\geq 1/\delta$.
2. If $\mathbf{b}_{\kappa-1}^* \neq \mathbf{0}$ and $\mathbf{b}_\kappa^* = \mathbf{0}$ and if the new vector $\mathbf{b}_{\kappa-1}^*$ is zero, after the swap we will have $\mathbf{b}_{\kappa-1}^* = \mathbf{0}$ and $\mathbf{b}_\kappa^* \neq \mathbf{0}$. More precisely, the new vector $\mathbf{b}_\kappa^*$ will exactly be the last vector $\mathbf{b}_{\kappa-1}^*$. As a consequence, the left-hand side of the quantity D is constant, while the right-hand side decreases by a factor 2.
3. If $\mathbf{b}_{\kappa-1}^* \neq \mathbf{0}$ and $\mathbf{b}_\kappa^* = \mathbf{0}$ and if the new vector $\mathbf{b}_{\kappa-1}^*$ is nonzero, then the new vector $\mathbf{b}_\kappa^*$ must be zero since the dimension of the lattice $L(\mathbf{b}_1, \ldots, \mathbf{b}_\kappa)$ shall remain constant. So the right-hand side of the quantity D is constant. Oppositely, all the d_i's above some threshold are decreasing by a factor $\geq 1/\delta$: none of the $\|\mathbf{b}_i^*\|^2$'s is changing except $\|\mathbf{b}_{\kappa-1}^*\|^2$, which decreases by a factor $\geq 1/\delta$.

The case $\mathbf{b}_{\kappa-1}^* = \mathbf{0}$ cannot occur, since at the loop iteration for which the vector $\mathbf{b}_{\kappa-1}$ would have been created and inserted at a rank $\kappa - 1 \geq \zeta + 1$, Lovász's condition between this vector and the previous one would not have been satisfied.

Remark. Like the L^3 algorithm, the L^2 algorithm and its modified version work on the underlying quadratic form. In particular, the modified L^2 algorithm of Figure 9 can easily be adapted to the case of possibly nondefinite and possibly nonpositive integer quadratic forms, by adding absolute values in the Lovász conditions (see [46] for more details).

Acknowledgments. We thank Richard Brent, Guillaume Hanrot, Florian Hess, Claus Peter Schnorr, Victor Shoup, and Paul Zimmermann for numerous discussions. We also thank the reviewers for their helpful comments.

REFERENCES

[1] A. Akhavi, *Worst-case complexity of the optimal LLL algorithm*, in Proceedings of the 2000 Latin American Theoretical Informatics (LATIN 2000), Lecture Notes in Comput. Sci. 1776, Springer, Berlin, 2000, pp. 355–366.

[2] L. Babai, *On Lovász lattice reduction and the nearest lattice point problem*, Combinatorica, 6 (1986), pp. 1–13.

[3] C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier, *PARI/GP Computer Package Version* 2, Université de Bordeaux I, Talence, France.

[4] D. Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices Amer. Math. Soc., 46 (1999), pp. 203–213.

[5] D. Boneh and G. Durfee, *Cryptanalysis of RSA with private key d less than* $n^{0.292}$, in Proceedings of Eurocrypt '99, Lecture Notes in Comput. Sci. 1592, Springer, Berlin, 1999, pp. 1–11.

[6] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer, Berlin, 1995.

[7] D. Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. Cryptology, 10 (1997), pp. 233–260.

[8] N. Gama, N. Howgrave-Graham, H. Koy, and P. Q. Nguyen, *Rankin's constant and blockwise lattice reduction*, in Advances in Cryptology, Proceedings of CRYPTO '06, Lecture Notes in Comput. Sci. 4117, Springer, Berlin, 2006, pp. 112–130.

[9] N. Gama and P. Q. Nguyen, *Finding short lattice vectors within Mordell's inequality*, in Proceedings of the 40th ACM Symposium on the Theory of Computing (STOC '08), ACM, New York, 2008.

[10] C. Gauss, *Disquisitiones Arithmeticae*, G. Fleischer, Leipzig, 1801.

[11] G. H. Golub and Charles F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.

[12] M. Grötschel, L. Lovász, and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer, Berlin, 1993.

[13] C. Hermite, *Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres*, J. Reine Angew. Math., 40 (1850), pp. 279–290.

[14] N. J. Higham, *The accuracy of floating point summation*, SIAM J. Sci. Comput., 14 (1993), pp. 783–799.

[15] N. A. Howgrave-Graham and N. P. Smart, *Lattice attacks on digital signature schemes*, Des. Codes Cryptogr., 23 (2001), pp. 283–290.

[16] Ieee, *ANSI/IEEE Standard* 754–1985 *for Binary Floating-Point Arithmetic*; reprinted in ACM SIGPLAN Notices, 22 (1987), pp. 9–25.

[17] E. Kaltofen, *On the complexity of finding short vectors in integer lattices*, in Proceedings of EUROCAL'83, Lecture Notes in Comput. Sci. 162, Springer, Berlin, 1983, pp. 236–244.

[18] H. Koy and C. P. Schnorr, *Segment LLL-reduction of lattice bases*, in Cryptography and Lattices, Proceedings of CALC'01, Lecture Notes in Comput. Sci. 2146, Springer, Berlin, 2001, pp. 67–80.

[19] H. Koy and C. P. Schnorr, *Segment LLL-reduction of lattice bases with floating-point orthogonalization*, in Cryptography and Lattices, Proceedings of CALC'01, Lecture Notes in Comput. Sci. 2146, Springer, Berlin, 2001, pp. 81–96.

[20] J. C. Lagarias and A. M. Odlyzko, *Solving low-density subset sum problems*, J. Assoc. Comput. Mach., 32 (1985), pp. 229–246.

[21] L. Lagrange, *Recherches d'arithmétique*, Nouv. Mém. Acad., Berlin, 1773.

[22] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*, SIAM, Philadelphia, 1995.

[23] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 515–534.

[24] H. W. Lenstra, Jr., *Integer Programming with a Fixed Number of Variables*, Technical report 81-03, Mathematisch Instituut, Universiteit van Amsterdam, Amsterdam, The Netherlands, 1981.

[25] H. W. Lenstra, Jr., *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.

[26] H. W. Lenstra, Jr., *Flags and lattice basis reduction*, in Proceedings of the Third European Congress of Mathematics, Vol. 1, Birkhäuser, Basel, 2001, pp. 37–51.

[27] LIDIA, *A Library for Computational Number Theory*, http://www-jb.cs.uni-sb.de/LiDIA/linkhtml/lidia/lidia.html.

[28] MAGMA, *The Magma Computational Algebra System for Algebra, Number Theory and Geometry*, http://magma.maths.usyd.edu.au/magma/.

[29] D. MICCIANCIO AND S. GOLDWASSER, *Complexity of Lattice Problems: A Cryptographic Perspective*, Kluwer Internat. Ser. Engrg. Comput. Sci. 671, Kluwer Academic Publishers, Boston, MA, 2002.

[30] P. NGUYEN, *Cryptanalysis of the Goldreich-Goldwasser-Halevi cryptosystem from Crypto* '97, in Proceedings of the 19th IACR Cryptology Conference (Crypto '99), Lecture Notes in Comput. Sci. 1666, Springer, Berlin, 1999, pp. 288–304.

[31] P. NGUYEN AND J. STERN, *Cryptanalysis of the Ajtai-Dwork cryptosystem*, in Proceedings of the 18th IACR Cryptology Conference (Crypto '98), Lecture Notes in Comput. Sci. 1462, Springer, Berlin, 1998, pp. 223–242.

[32] P. Q. NGUYEN AND I. E. SHPARLINSKI, *The insecurity of the digital signature algorithm with partially known nonces*, J. Cryptology, 15 (2002), pp. 151–176.

[33] P. Q. NGUYEN AND D. STEHLÉ, *Low-dimensional lattice basis reduction revisited (extended abstract)*, in Proceedings of the 6th International Algorithmic Number Theory Symposium (ANTS-VI), Lecture Notes in Comput. Sci. 3076, Springer, Berlin, 2004, pp. 338–357.

[34] P. Q. NGUYEN AND D. STEHLÉ, *Floating-point LLL revisited*, in Advances in Cryptology—Proceedings of EUROCRYPT '05, Lecture Notes in Comput. Sci. 3494, Springer, Berlin, 2005, pp. 215–233.

[35] P. Q. NGUYEN AND D. STEHLÉ, *LLL on the average*, in Proceedings of the 7th International Algorithmic Number Theory Symposium (ANTS-VII), Lecture Notes in Comput. Sci. 4076, Springer, Berlin, 2006, pp. 238–256.

[36] P. Q. NGUYEN AND J. STERN, *The two faces of lattices in cryptology*, in Proceedings of CALC '01, Lecture Notes in Comput. Sci. 2146, Springer, Berlin, 2001, pp. 146–180.

[37] A. M. ODLYZKO, *The rise and fall of knapsack cryptosystems*, in Cryptology and Computational Number Theory, Proc. Sympos. Appl. Math. 42, AMS, Providence, RI, 1990, pp. 75–88.

[38] M. POHST, *A modification of the LLL reduction algorithm*, J. Symbolic Comput., 4 (1987), pp. 123–127.

[39] C. P. SCHNORR, *A hierarchy of polynomial lattice basis reduction algorithms*, Theoret. Comput. Sci., 53 (1987), pp. 201–224.

[40] C. P. SCHNORR, *A more efficient algorithm for lattice basis reduction*, J. Algorithms, 9 (1988), pp. 47–62.

[41] C. P. SCHNORR, *Fast LLL-type lattice reduction*, Inform. Comput., 204 (2006), pp. 1–25.

[42] C. P. SCHNORR AND M. EUCHNER, *Lattice basis reduction: Improved practical algorithms and solving subset sum problems*, Math. Programming, 66 (1994), pp. 181–199.

[43] A. SCHÖNHAGE, *Factorization of univariate integer polynomials by Diophantine approximation and improved basis reduction algorithm*, in Proceedings of the 1984 International Colloquium on Automata, Languages and Programming (ICALP 1984), Lecture Notes in Comput. Sci. 172, Springer, Berlin, 1984, pp. 436–447.

[44] A. SCHÖNHAGE, *Fast reduction and composition of binary quadratic forms*, in Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC'91), ACM, New York, 1991, pp. 128–133.

[45] V. SHOUP, *Number Theory C++ Library (NTL)*, http://www.shoup.net/ntl/.

[46] D. SIMON, *Solving quadratic equations using reduced unimodular quadratic forms*, Math. Comp., 74 (2005), pp. 1531–1543.

[47] D. STEHLÉ, *Floating-point LLL: Theoretical and practical aspects*, in Proceedings of LLL + 25, P. Q. Nguyen and B. Vallée, eds., Springer, Berlin, to appear.

[48] A. STORJOHANN, *Faster Algorithms for Integer Lattice Basis Reduction*, Technical report, ETH Zürich, Zürich, Switzerland, 1996.

[49] THE SPACES PROJECT, *MPFR, a LGPL-Library for Multiple-Precision Floating-Point Computations with Exact Rounding*, http://www.mpfr.org/.

[50] J. H. WILKINSON, *The Algebraic Eigenvalue Problem*, The Clarendon Press, Oxford University Press, New York, 1988.

[51] C. K. YAP, *Fast unimodular reduction: Planar integer lattices*, in Proceedings of the 1992 Symposium on the Foundations of Computer Science (FOCS 1992), IEEE Computer Society, Los Alamitos, CA, 1992, pp. 437–446.