Floating-Point LLL Revisited

Phong Q. Nguyễn[⋆1] and Damien Stehlé[2]

[1] CNRS/École normale supérieure, DI, 45 rue d'Ulm, 75005 Paris, France.
Phong.Nguyen@di.ens.fr – http://www.di.ens.fr/~pnguyen/

[2] Univ. Nancy 1/LORIA, 615 rue du J. Botanique, 54602 Villers-lès-Nancy, France.
Damien.Stehle@loria.fr – http://www.loria.fr/~stehle/

Abstract. The Lenstra-Lenstra-Lovász lattice basis reduction algorithm (LLL or L^3) is a very popular tool in public-key cryptanalysis and in many other fields. Given an integer d-dimensional lattice basis with vectors of norm less than B in an n-dimensional space, L^3 outputs a so-called L^3-reduced basis in polynomial time $O(d^5 n \log^3 B)$, using arithmetic operations on integers of bit-length $O(d \log B)$. This worst-case complexity is problematic for lattices arising in cryptanalysis where d or/and $\log B$ are often large. As a result, the original L^3 is almost never used in practice. Instead, one applies floating-point variants of L^3, where the long-integer arithmetic required by Gram-Schmidt orthogonalisation (central in L^3) is replaced by floating-point arithmetic. Unfortunately, this is known to be unstable in the worst-case: the usual floating-point L^3 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 L^3 which provably outputs L^3-reduced bases in polynomial time $O(d^4 n(d + \log B) \log B)$. This is the first L^3 algorithm whose running time (without fast integer arithmetic) provably grows only quadratically with respect to $\log B$, like the well-known Euclidean and Gaussian algorithms, which it generalizes.
Keywords: LLL, L^3, Lattice Reduction, Public-Key Cryptanalysis.

1 Introduction

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ be linearly independent vectors in $\mathbb{R}^n$ with $n \geq d$: often $n = d$ or $n = O(d)$. We denote by $L[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d] = \left\{ \sum_{i=1}^{d} x_i \boldsymbol{b}_i \mid x_i \in \mathbb{Z} \right\}$ the set of all integer linear combinations of the $\boldsymbol{b}_i$'s. This set is called a *lattice* and $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ 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 [12, 8]), particularly in cryptology (see [30, 24]). This problem is known as

⋆ The work described in this article has in part been supported by the Commission of the European Communities through the IST program under contract IST-2002-507932 ECRYPT.

lattice reduction and can intuitively be viewed as a vectorial generalisation of gcd computations.

The first breakthrough in lattice reduction dates back to 1981 with Lenstra's celebrated work on integer programming [20, 21], which was, among others, based on a novel lattice reduction technique (which can be found in the preliminary version [20] of [21]). Lenstra's reduction technique was only polynomial-time for fixed dimension, which was however sufficient in [20]. This inspired Lovász to develop a polynomial-time variant of the algorithm, which reached a final form in the seminal paper [19] where Lenstra, Lenstra and Lovász applied it to factor rational polynomials in polynomial time (back then, a famous problem), from which the name LLL or L^3 comes. Further refinements of the L^3 algorithm were later proposed, notably by Schnorr [33, 34]. Reduction algorithms (in particular L^3) have arguably become the most popular tool in public-key cryptanalysis (see the survey [30]). In the past twenty-five years, they have been used to break many public-key cryptosystems, including knapsack cryptosystems [31], RSA in particular settings [9, 7, 6], DSA and similar signatures in particular settings [14, 26], etc.

Given as input an integer d-dimensional lattice basis whose n-dimensional vectors have norm less than B, L^3 outputs a so-called L^3-reduced basis in time $O(d^5 n \log^3 B)$ without fast integer arithmetic, using arithmetic operations on integers of bit-length $O(d \log B)$. 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 [9], we may need to reduce a 64-dimensional lattice with vectors having RSA-type coefficients (1024-bit), in which case the complexity becomes "$d^5 n \log^3 B = 2^{66}$ ". As a result, the original L^3 algorithm is seldom used in practice. Instead, one applies floating-point (*fp*) variants, where the long-integer arithmetic required by Gram-Schmidt orthogonalisation (which plays a central role in L^3) is replaced by floating-point arithmetic (*fpa*) on much smaller numbers. The use of *fpa* in L^3 goes back to the early eighties when L^3 was used to solve low-density knapsacks [17]. Unfortunately, *fpa* may lead to stability problems, both in theory and practice, especially when the dimension increases: the running time of *fp* variants of L^3 such as Schnorr-Euchner's [36] is not guaranteed to be polynomial nor 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 [29, 25] of lattice-based cryptosystems, which led to improvements in Shoup's NTL library [41].

There is however one provable *fp*-variant of L^3, due to Schnorr [34], which significantly improves the worst-case complexity. Schnorr's variant outputs an approximate L^3-reduced basis in time $O(d^3 n \log B (d + \log B)^2)$, using $O(d + \log B)$ precision *fp* numbers. However, this algorithm is mostly of theoretical interest and is not implemented in any of the main computational libraries [41, 22, 4, 1]. This can be explained by at least three reasons: it is not clear which *fpa*-model is used, the algorithm is difficult to describe, and the hidden complexity constants are rather large. More precisely, the required precision of *fp* numbers in [34] seems to be higher than $12d + 7\log_2 B$.

Our Results. We present the L^2 algorithm, a new and simple *fp*-variant of L^3 in a standard *fpa*-model, which provably outputs approximate L^3-reduced bases in polynomial time. More precisely, its complexity is $O(d^4 n(d + \log B)\log B)$ using only a $(d\log_2 3)$-bit precision, which is independent of $\log B$. This is the first L^3 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 provable L^3 algorithms known. This improvement is significant for lattices where $\log B$ is larger than d, for example those arising from minimal polynomials [8] and Coppersmith's technique [9]. Interestingly, L^3 can be viewed as a generalisation of the famous Euclidean and Gaussian algorithms whose complexities are quadratic, not cubic like the original L^3. This arguably makes L^2 closer to the Euclid algorithm.

	L^3[19]	Schnorr[34]	L^2
Required precision	$O(d\log B)$	$> 12d + 7\log_2 B$	$d\log_2 3 \approx 1.58d$
Complexity	$O(d^5 n\log^3 B)$	$O(d^3 n(d+\log B)^2\log B)$	$O(d^4 n(d+\log B)\log B)$

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 [3] to *fpa* 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 [28] with 100-bit vectors which makes NTL's LLL_FP [41] (an improved version of [36]) loop forever, which contradicts [16] where it is claimed that double precision is sufficient in [36] 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 dimension much higher than 55, due perhaps to the well-known experimental fact that for such input bases, L^3 outputs better bases than for the worst-case. 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 [27], which is quadratic in low dimension thanks to another cascade. The cascade analysis is made possible by the efficiency of our *fp*-variant of Babai's algorithm, and cannot be adapted to the standard L^3 algorithm. Besides, our tight bound on Babai's algorithm may be of independent interest. For instance, in Micciancio's variant [23] of the GGH cryptosystem [10], Babai's algorithm is used to decrypt.

Related Work. Much work [38, 34, 42, 15, 16, 35] 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 [44, 39] for dimension two). 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 orthogonalisation could be very unstable, it was suggested in the late nineties to use well-known alternative techniques (see [18, 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 [16], where a theorem from [18] is incorrectly applied. At the moment, it is therefore not clear how to exploit known results on Givens rotations and Householder reflections to improve L^3 theoretically. This is why L^2 only uses a process 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. Additional information (such as complete proofs of technical lemmata and experimental results) will be given in the journal version of the present work.

REMARKS. Like L^3, the L^2 algorithm works in fact with the underlying quadratic form and can therefore be used to reduce positive definite integer quadratic forms. It can be checked (see the full version) that L^2 can be extended to linearly dependent vectors, leading to what is to our knowledge the fastest algorithm known to construct a lattice basis from a generating set. Schnorr pointed out that the required precision in L^2 can be slightly decreased by using a better summation algorithm than ours (e.g. a tree-like algorithm), which may be interesting if one is restricted to a fixed precision. For the sake of simplicity, we keep a basic summation algorithm.

2 Background

NOTATION. All logarithms are in base 2. Let $\|.\|$ and $\langle ., . \rangle$ 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, without fast integer arithmetic [40]. Our *fpa*-model is a smooth extension of the IEEE-754 standard [2], as provided by NTL [41] and MPFR [32]. With an ℓ-bit working precision, a *fp*-number is of the form $x = \pm m_x \cdot 2^{e_x}$ where the mantissa $m_x \in [1/2, 1)$ is ℓ-bit long and the exponent e_x is an integer. We expect all four basic *fp*-operations to be correctly rounded: the returned value $\diamond(a \text{ op } b)$ for op $\in \{+, -, /, *\}$ is a closest *fp*-number to $(a \text{ op } b)$. In our complexity analysis, we do not consider the cost of the arithmetic on the exponents: it can be checked that the exponents are

integers of length $O(\log(d + \log B))$, so that the cost is indeed negligible.
We recall basic notions from algorithmic geometry of numbers (see [24]).

Gram matrix. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ be vectors. Their *Gram matrix* $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$ is the $d \times d$ symmetric matrix $(\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle)_{1 \leq i,j \leq d}$ formed by all the inner products.

Lattice volume. A lattice L has infinitely many lattice bases when $\dim(L) \geq 2$. Any two bases are related to each other by a unimodular matrix (integral matrix of determinant ± 1), and therefore the determinant of the Gram matrix of a basis only depends on the lattice. The square root of this determinant is the *volume* $\mathrm{vol}(L)$ (or *determinant*) of the lattice.

Gram-Schmidt orthogonalisation. Let $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ be linearly independent vectors. The *Gram-Schmidt orthogonalisation* (GSO) $[\boldsymbol{b}_1^*, \ldots, \boldsymbol{b}_d^*]$ is the orthogonal family defined recursively as follows: $\boldsymbol{b}_i^*$ is the component of $\boldsymbol{b}_i$ orthogonal to the linear span of $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$. We have $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \boldsymbol{b}_j^*$ where $\mu_{i,j} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2$. For $i \leq d$ we let $\mu_{i,i} = 1$. The lattice L spanned by the $\boldsymbol{b}_i$'s satisfies $\mathrm{vol}(L) = \prod_{i=1}^{d} \|\boldsymbol{b}_i^*\|$. The GSO family depends on the order of the vectors. If the $\boldsymbol{b}_i$'s are integer vectors, the $\boldsymbol{b}_i^*$'s and the $\mu_{i,j}$'s are rational.

QR **factorisation.** The GSO corresponds to the "R" part of the $Q \cdot R$ factorisation of the matrix representing the basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$, where Q is an orthogonal matrix ($Q \cdot Q^t = Q^t \cdot Q = Id$) and R is lower triangular. If $R = (r_{i,j})$, for any i we have $r_{i,i} = \|\boldsymbol{b}_i^*\|^2$ and for any $i \geq j$ we have $r_{i,j} = \mu_{i,j} \|\boldsymbol{b}_j^*\|^2$. 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 *fp* calculations, it is useful to have all these variables to minimize the number of arithmetic operations and thus the precision loss.

Size-reduction. A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ 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 i-th vector $\boldsymbol{b}_i$ is *size-reduced* if $|\mu_{i,j}| \leq \eta$ for all $j < i$. Size-reduction usually refers to $\eta = 1/2$, but it is essential for L^2 to allow larger η.

L^3-reduction. A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ is L^3-reduced with factor (δ, η) where $1/4 < \delta \leq 1$ and $1/2 \leq \eta < \sqrt{\delta}$ if the basis is size-reduced with factor η and if its GSO satisfies the $(d-1)$ Lovász conditions $(\delta - \mu_{\kappa,\kappa-1}^2) r_{\kappa-1,\kappa-1} \leq r_{\kappa,\kappa}$ (or equivalently $\delta \|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^* + \mu_{\kappa,\kappa-1} \boldsymbol{b}_{\kappa-1}^*\|^2$), which implies that the GSO vectors never drop too much. Such bases have useful properties (see [19, 8, 24]), like providing approximations to the shortest vector problem and the closest vector problem. In particular, their first vector is relatively short: $\|\boldsymbol{b}_1\| \leq (\delta - \eta^2)^{-(d-1)/4} \mathrm{vol}(L)^{1/d}$. L^3-reduction usually refers to the factor $(3/4, 1/2)$ initially chosen in [19]. But the closer δ and η are respectively to 1 and $1/2$, the shorter $\boldsymbol{b}_1$ should be. In practice, one usually selects $\delta \approx 1$ and $\eta \approx 1/2$, so that $\|\boldsymbol{b}_1\| \leq (4/3)^{(d-1)/4} \mathrm{vol}(L)^{1/d}$ approximately. 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. 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, and whether or not $\eta = 1/2$ can be achieved in quadratic time like L^2. The case $(\delta, \eta) = (1, 1/2)$ is closely related to a notion of reduction invented by Hermite [13] in the language of quadratic forms.

The L^3 algorithm. The L^3 algorithm [19] is described in Fig. 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 $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}]$ is L^3-reduced. At each loop iteration, κ is either incremented or decremented: the loop stops when κ reaches the value $d+1$, in which case the entire basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ is L^3-reduced. L^3 performs two kinds of operations: swaps of consecutive vectors and Babai's

Input: A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ 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. κ:=2. While $\kappa \leq d$ do
3. Size-reduce $\boldsymbol{b}_\kappa$ using Babai's algorithm (Fig. 3), which updates the GSO.
4. $\kappa' := \kappa$. While $\kappa \geq 2$ and $\delta r_{\kappa-1,\kappa-1} \geq r_{\kappa',\kappa'} + \sum_{i=\kappa-1}^{\kappa'-1} \mu_{\kappa',i}^2 r_{i,i}$, do $\kappa := \kappa - 1$.
5. For $i = 1$ to $\kappa - 1$, $\mu_{\kappa,i} := \mu_{\kappa',i}$.
6. Insert $\boldsymbol{b}_{\kappa'}$ right before $\boldsymbol{b}_\kappa$.
7. $\kappa := \kappa + 1$.
8. Output $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$.

Fig. 2. The L^3 Algorithm.

nearest plane algorithm [3] (see Fig. 3), which uses at most d translations of the form $\boldsymbol{b}_\kappa := \boldsymbol{b}_\kappa - m\boldsymbol{b}_i$, where m is some integer and $i < \kappa$. Swaps are used to achieve Lovász's conditions, while Babai's algorithm is used to size-reduce vectors. We explain Steps 4–7: if Lovász's condition is satisfied, nothing happens in Steps 5 and 6, and κ is incremented like in classical descriptions of the L^3 algorithm. Otherwise, Step 4 finds the right index to insert $\boldsymbol{b}_\kappa$, thus collecting successive failures of Lovász's test.

Input: A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$, its GSO and an index κ.
Output: The basis with $\boldsymbol{b}_\kappa$ size-reduced and the updated GSO.
1. For $i = \kappa - 1$ downto 1 do
2. $\boldsymbol{b}_\kappa := \boldsymbol{b}_\kappa - \lceil \mu_{\kappa,i} \rfloor \boldsymbol{b}_i$.
3. For $j = 1$ to i do
4. $\mu_{\kappa,j} := \mu_{\kappa,j} - \lceil \mu_{\kappa,i} \rfloor \mu_{i,j}$.

Fig. 3. Babai's algorithm to size-reduce $\boldsymbol{b}_\kappa$, so that $|\mu_{\kappa,i}| \leq 1/2$ for all $i < \kappa$.

If L^3 terminates, it is clear that the output basis is L^3-reduced. What is less clear *a priori* is why L^3 has a polynomial-time complexity. A standard argument shows that each swap decreases the quantity $\Delta = \prod_{i=1}^{d} \|\boldsymbol{b}_i^*\|^{2(d-i+1)}$ by at least a factor $\delta < 1$, while $\Delta \geq 1$ because the $\boldsymbol{b}_i$'s are integer vectors and Δ can be viewed as a product of squared volumes of lattices spanned by some of the $\boldsymbol{b}_i$'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 Fig. 2 without fast integer arithmetic is $O((d^2 \log B) \cdot dn \cdot (d \log B)^2)) = O(d^5 n \log^3 B)$.

L^3 with *fpa*. The cost of L^3 is dominated by 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 *fp* approximations. 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 no longer necessarily decreases at each swap. And if ever the algorithm 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 *fp*-L^3 was the one of Schnorr [34], which simulates the behavior of L^3 using *fp*-approximations of the coefficients of the inverse matrix of the $\mu_{i,j}$'s: it computes a $(0.95, 0.55)$-L^3-reduced basis. The number of loop iterations and the number of arithmetic operations (in each iteration) remain the same as L^3: only the cost of each arithmetic operation related to the GSO decreases. Instead of handling integers of length $O(d \log B)$, [34] uses *fp*-numbers with $O(d + \log B)$-bit long mantissæ (with large hidden constants, as mentioned in the introduction), which decreases the worst-case complexity of L^3 to $O(d^4 \log B(d + \log B)^2)$. This is still cubic in $\log B$. Because this algorithm is mostly of theoretical interest, the main number theory computer packages [41, 22, 4] only implement heuristic *fp*-variants of L^3 à la Schnorr-Euchner [36] which suffer from stability problems in high dimension.

3 The L^2 Algorithm

We now describe the L^2 algorithm, which is a natural *fp*-variant of L^3. The main principle is to keep sufficiently good *fp*-approximations of the GSO coefficients during the execution of the algorithm. Accuracy is crucial for size-reduction and for checking Lovász's conditions. If one is not careful, swaps and translations may decrease the accuracy to the point of having meaningless *fp*-values.

3.1 Gram-Schmidt Computations

It is important for L^2 to have accurate formulas for the computation of the GSO coefficients. In [36], the following recursive formulas are used:

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

In these formulas, the inner products $\langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle$ are computed in *fpa*, which leads to a potential inaccuracy of $2^{-\ell} \|\boldsymbol{b}_i\| \|\boldsymbol{b}_j\|$, with the following drawback: to ensure that the basis returned by L^2 is size-reduced, absolute error bounds on the $\mu_{i,j}$'s are required; if the error is therefore larger than $2^{-\ell} \|\boldsymbol{b}_i\| \|\boldsymbol{b}_j\|$, the precision ℓ must be $\Omega(\log B)$ in the worst case. The analyses of [34, 35] do not tackle this issue.

We use slightly different formulas by introducing the quantity $r_{i,j} = \mu_{i,j}\|\boldsymbol{b}_j^*\|^2 = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle$ for all $i \geq j$:

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

Accuracy is improved because the inner products are extracted from the exact Gram matrix and because each term of the sum now only requires one multiplication instead of two. For $i = j$, the first formula is $r_{i,i} = \|\boldsymbol{b}_i\|^2 - \sum_{k=1}^{i-1} \mu_{i,k} \cdot r_{i,k}$, which suggests to define $s_j = \|\boldsymbol{b}_i\|^2 - \sum_{k=1}^{j-1} \mu_{i,k} \cdot r_{i,k}$ for all $1 \leq j \leq i$, so that $\|\boldsymbol{b}_i^*\|^2 = r_{i,i} = s_i$. The quantities s_i will be useful to check consecutive Lovász's conditions. Indeed, Lovász's condition $(\delta - \mu_{\kappa,\kappa-1}^2)\|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^*\|^2$ can be rewritten as $\delta\|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^*\|^2 + \mu_{\kappa,\kappa-1}^2\|\boldsymbol{b}_{\kappa-1}^*\|^2$,i.e.,

$$\delta r_{\kappa-1,\kappa-1} \leq s_{\kappa-1}.$$

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

$$\delta r_{\kappa-2,\kappa-2} \leq s_{\kappa-2}.$$

Thus, storing the s_j's enables 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 $r_{i,j}$, $\mu_{i,j}$ and s_j is summarized in the so-called Cholesky Factorisation Algorithm (CFA) of Fig. 4. Of course, because one

Input: The Gram matrix of $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$.
Output: All the $r_{i,j}$'s, $\mu_{i,j}$'s and s_j's.
1. For $i = 1$ to d do
2. For $j = 1$ to i do
3. $r_{i,j} := \langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle$,
4. For $k = 1$ to $j - 1$ do $r_{i,j} := r_{i,j} - r_{i,k}\mu_{j,k}$,
5. $\mu_{i,j} := r_{i,j}/r_{j,j}$.
6. $s_0 := \|\boldsymbol{b}_d\|^2$. For $j = 1$ to d do $s_j := s_{j-1} - \mu_{d,j} r_{d,j}$.
7. $r_{d,d} := s_d$.

Fig. 4. The Cholesky Factorisation Algorithm (CFA)

uses *fpa*, the exact values are unknown. Instead, one computes *fp*-approximations $\bar{r}_{i,j}$, $\bar{\mu}_{i,j}$ and $\bar{s}_i$. Steps 4–6 are performed in the following way:

$$\bar{r}_{i,j} := \diamond\left(\bar{r}_{i,j} - \diamond(\bar{r}_{i,k} \cdot \bar{\mu}_{j,k})\right),\ \bar{\mu}_{i,j} := \diamond\left(\bar{r}_{i,j}/\bar{r}_{j,j}\right) \text{ and } \bar{s}_j := \diamond\left(\bar{s}_{j-1} - \diamond(\bar{\mu}_{d,j} \cdot \bar{r}_{d,j})\right).$$

We will not use CFA directly in L^2. Instead, we will use parts of it during the execution of the algorithm: because the orthogonalisation is performed vector by vector, there is no need recomputing everything from scratch if the $r_{i,j}$'s and $\mu_{i,j}$'s are already known for i and j below some threshold. Notice that the $r_{i,j}$'s can be updated at the same location, except for $r_{d,d}$ because the different s_j's need being returned to check Lovász's conditions. The CFA will prove useful to estimate in Section 4 the precision required to guarantee the correctness of L^2.

3.2 An Iterative Floating-Point Version of Babai's Algorithm

The core of L^2 is an iterative *fp*-version of Babai's nearest plane algorithm, described in Fig. 5. Instead of size-reducing $\boldsymbol{b}_\kappa$ at once like in Fig. 3, our *fp*-version of Babai performs an iterative process using parts of the CFA algorithm of Fig. 4. Here, the x_i's of Babai's algorithm are computed progressively: the most significant bits in the first loop iteration, then more bits in the second iteration, and so on. This has two nice properties: first it terminates and gives correct results, and, more importantly, it makes L^2 efficient because very few bits of the GSO are required. At Step 4, $\bar{\eta} = \frac{\eta+1/2}{2} \in (1/2, \eta)$ is used instead

Input: A factor $\eta > 1/2$, a *fp*-precision ℓ, an integer κ, a basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$, $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$, and *fp* numbers $\bar{r}_{i,j}$ and $\bar{\mu}_{i,j}$'s for $j \leq i < \kappa$.
Output: *fp* numbers $\bar{r}_{\kappa,j}$, $\bar{\mu}_{\kappa,j}$ and $\bar{s}_j$ for $j \leq \kappa$, $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}, \boldsymbol{b}'_\kappa, \boldsymbol{b}_{\kappa+1}, \ldots, \boldsymbol{b}_d]$, and $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}, \boldsymbol{b}'_\kappa, \boldsymbol{b}_{\kappa+1}, \ldots, \boldsymbol{b}_d)$ where $\boldsymbol{b}'_\kappa = \boldsymbol{b}_\kappa - \sum_{i<\kappa} x_i \boldsymbol{b}_i$ for some integers x_i's and: $|\langle \boldsymbol{b}'_\kappa, \boldsymbol{b}^*_i \rangle| \leq \eta \|\boldsymbol{b}^*_i\|^2$ for any $i < \kappa$.
1. $\bar{\eta} := \frac{\eta+1/2}{2}$. Repeat
2. Compute the $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s, $\bar{s}_j$'s with Steps 2–7 of the CFA with "$i = \kappa$".
3. For $i = \kappa - 1$ downto 1 do
4. If $|\bar{\mu}_{\kappa,i}| \geq \bar{\eta}$, then $X_i := \lfloor \bar{\mu}_{\kappa,i} \rceil$, else $X_i := 0$,
5. For $j = 1$ to $i - 1$, $\bar{\mu}_{\kappa,j} := \diamond\left(\bar{\mu}_{\kappa,j} - \diamond(X_i \cdot \bar{\mu}_{i,j})\right)$.
6. Update $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ and $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$, according to $\boldsymbol{b}_\kappa := \boldsymbol{b}_\kappa - \sum_{i=1}^{\kappa-1} X_i \boldsymbol{b}_i$.
7. Until all X_i's are zero.

Fig. 5. The Iterative Babai Nearest Plane Algorithm

of η to take into account the fact that $\mu_{\kappa,i}$ is known only approximately. At Step 6, it suffices to update the scalar products $\langle \boldsymbol{b}_i, \boldsymbol{b}_\kappa \rangle$ for $i \leq d$ according to the following relations:

$$\|\boldsymbol{b}'_\kappa\|^2 = \|\boldsymbol{b}_\kappa\|^2 + \sum_{j \neq \kappa} x_j^2 \|\boldsymbol{b}_j\|^2 - 2 \sum_{j \neq \kappa} x_j \langle \boldsymbol{b}_j, \boldsymbol{b}_\kappa \rangle + 2 \sum_{j \neq \kappa, i \neq \kappa} x_i x_j \langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle$$

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

3.3 Main Results

A description of L^2 is given in Fig. 6. There is no need keeping approximations of all the GSO coefficients: because L^2 is iterative, it suffices to have approximations up to the threshold κ. Notice that the cost of the first step is bounded by $O(d^2 n \log^2 B)$ and is thus negligible compared to the rest of the reduction. At Step 4, $\bar{\delta} = \frac{\delta+1}{2} \in]\delta, 1[$ is used instead of δ to take into account the fact that $\bar{r}_{\kappa-1,\kappa-1}$ and $\bar{s}_{\kappa-1}$ are known only approximately. The main result of this paper is the following:

Theorem 1. *Let (δ, η) 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 $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$*

Input: A valid pair (δ, η) like in Th. 1, a basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ and a *fp*-precision ℓ.
Output: An L^3-reduced basis with factor pair (δ, η).
Variables: A matrix G, two $d \times d$ *fp*-matrices $(\bar{r}_{i,j})$ and $(\bar{\mu}_{i,j})$, a *fp*-vector $\bar{s}$.
1. Compute exactly $G = G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$.
2. $\bar{\delta} := \frac{\delta+1}{2}$, $\bar{r}_{1,1} := \diamond(\langle \boldsymbol{b}_1, \boldsymbol{b}_1 \rangle)$, $\kappa := 2$. While $\kappa \leq d$, do
3. Size-reduce $\boldsymbol{b}_\kappa$ using the algorithm of Fig. 5. It updates the *fp*-GSO.
4. $\kappa' := \kappa$. While $\kappa \geq 2$ and $\bar{\delta}\bar{r}_{\kappa-1,\kappa-1} \geq \bar{s}_{\kappa-1}$, do $\kappa := \kappa - 1$.
5. For $i = 1$ to $\kappa - 1$ do $\bar{\mu}_{\kappa,i} := \bar{\mu}_{\kappa',i}$, $\bar{r}_{\kappa,i} := \bar{r}_{\kappa',i}$, $\bar{r}_{\kappa,\kappa} := \bar{s}_\kappa$.
6. Insert $\boldsymbol{b}_{\kappa'}$ right before $\boldsymbol{b}_\kappa$ and update G accordingly.
7. $\kappa := \kappa + 1$.
8. Output $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$.

Fig. 6. The L^2 algorithm.

in $\mathbb{Z}^n$ with $max_i \|\boldsymbol{b}_i\| \leq B$, the L^2 algorithm of Fig. 6 with precision $\ell = cd + o(d)$ outputs a (δ, η)-L^3-reduced basis in time $O(d^4 n \log B(d + \log B))$. More precisely, if τ denotes the number of loop iterations, then the running time is $O(dn(\tau + d \log dB)(d + \log B))$.

Let us make a few remarks. First, L^2 decreases the complexity bound $O(d^3 n \log B(d + \log B)^2)$ of [34] by a factor $\frac{d+\log B}{d}$. We can choose δ arbitrarily close to 1 and η arbitrarily close to $1/2$, so that the coefficient $c > \log \frac{(1+\eta)^2}{\delta - \eta^2}$ becomes arbitrarily close to $\log 3 < 1.585$. The additional statement 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 $\tau = O(d \log B)$ instead of the usual $O(d^2 \log B)$. Finally, the $o(d)$ term in the condition $\ell = c \cdot d + o(d)$ can be made effective and may be used backwards: if we perform calculations with a fixed number of bits, the correctness will be guaranteed up to a computable dimension.

4 Correctness of the L^2 Algorithm

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

4.1 Accuracy of Gram-Schmidt Computations

In general, the classical Gram-Schmidt algorithm is known to have very poor numerical stability [5, 11, 18, 43]. However, it must be stressed that in the context of L^3, 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 roughly 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 2. *Let (δ, η) be a valid factor pair like in Th. 1. Let $\rho = \frac{(1+\eta)^2}{\delta - \eta^2}$. Let $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ in $\mathbb{Z}^n$ be a d-dimensional lattice basis whose Gram matrix is* *given as input to the CFA algorithm from Fig. 4. Suppose* $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}]$ *is* (δ, η)*-*L^3*-reduced. In the case of* fpa *with a precision* ℓ *satisfying* $d\rho^d 2^{-\ell+2} \leq 1$*, the* fp*-numbers returned by the CFA of Fig. 4 satisfy the following equations: for all* $j \leq i < d$*,*

$$|\bar{r}_{i,j} - r_{i,j}| \leq d\rho^{j-1} 2^{-\ell+4} \cdot r_{j,j} \quad \text{and} \quad |\bar{\mu}_{i,j} - \mu_{i,j}| \leq d\rho^{j-1} 2^{-\ell+6}.$$

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

$$|\bar{r}_{d,j} - r_{d,j}| \leq d\rho^{j-1} M 2^{-\ell+4} \cdot r_{j,j} \quad \text{and} \quad |\bar{\mu}_{d,j} - \mu_{d,j}| \leq d\rho^{j-1} M 2^{-\ell+6}.$$

Finally, if $\boldsymbol{b}_d$ *is* η*-size-reduced with respect to* $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}]$*, then for any* $j \leq d$*:*

$$|\bar{s}_j - s_j| \leq d\rho^{j-1} 2^{-\ell+7} \cdot r_{j,j} + d 2^{-\ell} \cdot s_j.$$

The second set of inequalities is useful for the analysis of Babai's algorithm, while the last set provides guarantees when checking Lovász's conditions. We now give a sketch of the proof of Theorem 2 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 $err_j = \max_{i<d} \frac{|\bar{r}_{i,j} - r_{i,j}|}{r_{j,j}}$, i.e., the error on $r_{i,j}$ relative to $r_{j,j}$, and we estimate its growth as j increases. Obviously $err_1 \leq 2^{-\ell} \max_{i<d} \frac{|\langle \boldsymbol{b}_i, \boldsymbol{b}_1 \rangle|}{\|\boldsymbol{b}_1\|^2} \leq 2^{-\ell}$, because of size-reduction. We now choose $j \in [2, d-1]$. The result for $i = d$ can be derived from the proof for $i \leq d-1$, intuitively by replacing "$\boldsymbol{b}_d$" by "$\frac{1}{M}\boldsymbol{b}_d$" in it. Because of Step 5, for any $i < d$ and any $k < j$:

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

where we neglected low-order terms and used the fact that $|r_{i,k}| \leq \left(\frac{1}{2} + \epsilon\right) \|\boldsymbol{b}_k\|^2$, which comes from size-reduction. This implies that:

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

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

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

by using Lovász's conditions. This last inequality finally gives $err_j \leq (3+\epsilon)^j \cdot err_1 \leq (3+\epsilon)^j 2^{-\ell}$, since we have $\left(\frac{4}{3} + \epsilon\right)\left(\frac{5}{4} + \epsilon + 1\right) \approx 3 + \epsilon$. □

The bound in Theorem 2 seems to be tight, at least in practice: the classical Gram-Schmidt algorithm or the CFA become experimentally inaccurate with a precision $\leq d \log 3$ bits for certain bases. Consider indeed the L^3-reduced lattice basis given by the rows of the following $d \times d$ matrix L:

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

To obtain an integral lattice, one can multiply L by a large scaling factor and round its entries. By definition, this matrix is already L^3-reduced. With double precision calculations, i.e., with 53-bit mantissæ, the error on the $\mu_{i,j}$'s becomes significant (higher than 0.5) in dimension 35. These bases show the tightness of our $\log 3 \cdot d$ bound. By inserting a suitable random vector to such a basis, we were able to make the LLL_FP routine of NTL loop forever in dimension 55 (see [28]). This invalidates the claim of [36, 37, 15] which states that double precision suffices for lattices of dimension up to ≈ 250 using classical Gram-Schmidt.

4.2 Accuracy of Babai's Nearest Plane Algorithm

To estimate the accuracy of the iterative *fp*-version of Babai's algorithm given in Fig. 5 and used in L^2, we first study a simpler *fp*-version described in Fig. 7.

Input: A *fp*-precision ℓ, a basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$, $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$
and *fp* numbers $\bar{r}_{i,j}$'s and $\bar{\mu}_{i,j}$'s for $j \le i < d$.
Output: $x_1, \ldots, x_{d-1} \in \mathbb{Z}$ and $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}, \boldsymbol{b}'_d)$, where $\boldsymbol{b}'_d = \boldsymbol{b}_d - \sum_{i<d} x_i \boldsymbol{b}_i$.
1. Compute the $\bar{\mu}_{d,j}$'s for $j < d$ with Steps 2–7 of the CFA with "$i = d$".
2. $\bar{\eta} := \frac{\eta + 1/2}{2}$. For $i = d-1$ downto 1 do
3. If $|\bar{\mu}_{d,i}| \ge \bar{\eta}$, then $x_i := \lfloor \bar{\mu}_{d,i}^{(i+1)} \rceil$, else $x_i := 0$,
4. For $j = 1$ to $i-1$ do $\bar{\mu}_{d,j} := \diamond\, (\bar{\mu}_{d,j} - \diamond(x_i \cdot \bar{\mu}_{i,j}))$.
5. Compute $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}, \boldsymbol{b}'_d)$ from $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}, \boldsymbol{b}_d)$.

Fig. 7. Babai's Nearest Plane Algorithm

We use Theorem 2 to show stability properties of Babai's algorithm:

Theorem 3. *Let (δ, η) be a valid reduction factor (like in Th. 1) and $\rho = \frac{(1+\eta)^2}{\delta - \eta^2}$. Let $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ in $\mathbb{Z}^n$ be a d-dimensional lattice basis given as input to the algorithm of Fig. 7, and $B = max_i \|\boldsymbol{b}_i\|$. Suppose that $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}]$ 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 with working precision ℓ. Let $M = max_j |\mu_{d,j}|$. If ℓ satisfies $d^2 \rho^d 2^{-\ell+2} \le 1$, the algorithm of Fig. 7 finds integers $x_1, \ldots, x_{d-1}$ such that for any $i < d$:*

$$|x_i| \le 2(1+\eta)^{d-1-i}(M+1) \quad \text{and} \quad \frac{|\langle \boldsymbol{b}'_d, \boldsymbol{b}^*_i \rangle|}{\|\boldsymbol{b}^*_i\|^2} \le \bar{\eta} + d\rho^d (M+1) 2^{-\ell+4}.$$

Moreover it works in time $O(dn\ell(d + \log B))$ as long as $\ell = O(d + \log B)$.

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), this result 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 the Euclid 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.

The iterative Babai algorithm from Fig. 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 Babai steps, each one making $\log M$ decrease by $\Omega(d)$, until we reach $M \leq \eta$. This strategy is somewhat similar to the Babai routine of the floating-point L^3 algorithm of NTL [41], which repeatedly applies Babai's algorithm until nothing happens.

The iterative Babai algorithm will use a precision $\ell = \left(\log \frac{(1+\eta)^2}{\delta-\eta^2} + C\right) d + o(d)$ with an arbitrary $C > 0$. The CFA with working precision ℓ gives the input $\bar{r}_{i,j}$'s and $\bar{\mu}_{i,j}$'s, which by Theorem 2 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 Babai's algorithm in one single step, but Theorem 3 gives that their approximations suffice to make $M = \max_{i<\kappa} \frac{|\langle \boldsymbol{b}_\kappa, \boldsymbol{b}_i^* \rangle|}{\|\boldsymbol{b}_i^*\|^2}$ decrease by $\approx Cd$ bits. By making $O\left(1 + \frac{\log M}{d}\right)$ such calls to Babai's algorithm, size-reduction can be achieved.

Theorem 4. *Let (δ, η) be a valid pair (like in Th. 1) and $\rho = \frac{(1+\eta)^2}{\delta-\eta^2}$. Let $C > 0$ be a constant. Let $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ be a d-dimensional lattice basis in $\mathbb{Z}^n$ given as input to the algorithm of Fig. 7, and $B = max_i \|\boldsymbol{b}_i\|$. Suppose that $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}]$ is (δ, η)-L^3-reduced and that the given $\bar{r}_{i,j}$'s, $\bar{\mu}_{i,j}$'s are those that would have been returned by the CFA with precision ℓ. Let $M = max_{j<\kappa} |\mu_{\kappa,j}|$. If ℓ satisfies $d^2 \rho^d 2^{-\ell+6+Cd} \leq \eta - \frac{1}{2}$, the algorithm of Fig. 5 provides a correct output and the returned $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s, $\bar{s}_j$'s are those that would have been returned by the CFA with precision ℓ. Moreover, if $\ell = O(d)$, then the running time is $O(dn(d + \log B)(d + \log M))$.*

Proof. We start by the correctness properties of the algorithm. At the last iteration of the main loop, the computed X_j's are all zero, which implies that nothing happens during Steps 3–6. This gives that for any $j < \kappa$, $|\bar{\mu}_{\kappa,j}| \leq \eta^-$, from which we derive $\frac{|\langle \boldsymbol{b}'_\kappa, \boldsymbol{b}_i^* \rangle|}{\|\boldsymbol{b}_i^*\|^2} \leq \eta$, by using Theorem 3 and the hypothesis on ℓ. This also gives the correctness of the returned $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s and $\bar{s}_j$'s.

We now consider the impact of one iteration of the main loop on M. Let M_1 be the "new M" after the loop iteration. Theorem 3 and the hypothesis on ℓ give the inequality:

$$M_1 \leq \bar{\eta} + d\rho^d (1 + M) 2^{-\ell+4} \leq \frac{1/2 + 2\eta}{3} + 2^{-Cd} M.$$

As a consequence, there can be at most $1 + \frac{1}{Cd}(\log \frac{\eta - 1/2}{3} + \log M)$ loop iterations.

Theorem 3 gives that the cost of one loop iteration is bounded by $O(d^2 n(d + \log B))$, because 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 $\boldsymbol{b}_{\kappa+1}, \ldots, \boldsymbol{b}_d$) is taken into account in the complexity bound. Finally, the overall cost of the algorithm is bounded by:

$$O\left(d^2 n(d + \log B)\left(1 + \frac{\log M}{d}\right)\right) = O(dn(d + \log B)(d + \log M)). \qquad \square$$

4.3 Application to $\mathbf{L^2}$

We now prove the correctness of L^2. To do this, we show the following:

Theorem 5. *Let $[\boldsymbol{b}_1^{(0)}, \ldots, \boldsymbol{b}_d^{(0)}]$ in $\mathbb{Z}^n$ be a lattice basis given as input to the L^2 algorithm. For any loop iteration t, let $[\boldsymbol{b}_1^{(t)}, \ldots, \boldsymbol{b}_d^{(t)}]$ denote the current basis at the beginning of the t-th iteration. We have:*

1. *For any $i \leq \kappa(t)-1$, $\boldsymbol{b}_i^{(t)}$ is η-size-reduced, and $[\boldsymbol{b}_1^{(t)}, \ldots, \boldsymbol{b}_{\kappa(t)-1}^{(t)}]$ is L^3-reduced with factor pair (δ, η).*
2. *For any $i \leq d$, $max_{j\leq i}\|\boldsymbol{b}_j^{(t)*}\| \leq max_{j\leq i}\|\boldsymbol{b}_j^{(0)*}\|$ and $\|\boldsymbol{b}_i^{(t)}\| \leq \sqrt{d} max_{j \leq i}\|\boldsymbol{b}_i^{(0)}\|$.*

Proof. Clearly, all these properties are valid for $t = 0$, and size-reduction comes from Theorem 4. Assume now that we are performing the t-th loop iteration.

We now show that Lovász's tests work as desired. Recall that $\boldsymbol{b}_\kappa$ is swapped with $\boldsymbol{b}_i$ for $i < \kappa$ if and only if for any $j \in [i, \kappa-1]$ we have $\bar{s}_j \leq \bar{\delta}\bar{r}_{j,j}$. Assume first that $\boldsymbol{b}_\kappa$ is swapped with $\boldsymbol{b}_j$. Theorem 2 gives that:

$$s_j(1 - d2^{-\ell}) \leq \bar{r}_{j,j}(\bar{\delta} + d\rho^{j-1}2^{-\ell+8}).$$

Therefore, as soon as $d\rho^d 2^{-\ell+5} \leq 1-\delta$, if a swap is done in L^2 then Lovász's condition was not fulfilled for the factor $\frac{2-\delta}{3} \in (\delta, 1)$. On the opposite, assume that $\boldsymbol{b}_\kappa$ and $\boldsymbol{b}_j$ are not swapped. Theorem 2 gives:

$$s_j(1 + d2^{-\ell}) \geq r_{j,j}(\bar{\delta} - d\rho^{j-1}2^{-\ell+8}).$$

Thus, as soon as $d\rho^d 2^{-\ell+4} \leq 1-\delta$, if there is no swap, then Lovász's condition was fulfilled for δ. This gives the first statement for the new loop iteration. For the second statement, observe that during a swap between $\boldsymbol{b}_k$ and $\boldsymbol{b}_{k-1}$:

- $\|\boldsymbol{b}_{k-1}^{*new}\| \leq \|\boldsymbol{b}_{k-1}^{*old}\|$ because of Lovász's condition,
- $\|\boldsymbol{b}_k^{*new}\| \leq \|\boldsymbol{b}_{k-1}^{*old}\|$ because $\boldsymbol{b}_k^{*new}$ is an orthogonal projection of $\boldsymbol{b}_{k-1}^{*old}$,

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

$$\|\boldsymbol{b}_i^{(t)}\|^2 \leq d \cdot \max_{j\leq i}\|\boldsymbol{b}_j^{(t)*}\|^2 \leq d \cdot \max_{j\leq i}\|\boldsymbol{b}_j^{(0)*}\|^2 \leq d \cdot \max_{j\leq i}\|\boldsymbol{b}_j^{(0)}\|^2.$$

This proves the second part for $i < \kappa(t)$. If $i \geq \kappa(t)$, we consider the largest $t' < t$ such that $\kappa(t'+1) - 1 = i$. The iteration t' was the one which created $\boldsymbol{b}_i^{(t)}$. If t' does not exist, this vector is an initial vector and the result is obvious. Otherwise $\boldsymbol{b}_i^{(t)} = \boldsymbol{b}_{\kappa(t'+1)-1}^{(t'+1)}$ is size-reduced at the $(t'+1)$-th iteration. Thus we have $\|\boldsymbol{b}_i^{(t)}\|^2 \leq d \cdot \max_{j\leq\kappa(t'+1)-1}\|\boldsymbol{b}_j^{(0)}\|^2$. Since κ cannot increase by more than 1 in a single iteration, we must have $i \geq \kappa(t'+1) - 1$, which ends the proof.

5 Complexity Analysis of the $\mathbf{L^2}$ Algorithm

We now prove the complexity statement of Theorem 1.

5.1 On the Number of Loop Iterations of L^2

In Section 4.3, we showed that the accuracy suffices to check Lovász's conditions. For any Lovász test, either κ increases or decreases by one and when it decreases, the quantity $\prod_{i=1}^{d} \|\boldsymbol{b}_i^*\|^{2(d-i)}$ decreases by a factor of at least $\frac{3}{2\delta+1} > 1$. 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 exactly d, so there are at most $O(d^2 \log B)$ loop iterations.

In the rest of this section we show how to achieve the complexity bound $O(d^4 n \log B(d + \log B))$. This is done by generalizing a cascade phenomenon which appears in the analyses of the Euclidean and Gaussian algorithms.

5.2 Analyses of the Euclidean Algorithm

As mentioned in the introduction, the L^3 algorithm can be viewed as a high-dimensional generalisation of the Euclidean algorithm to compute gcds. But this analogy is incomplete: the Euclidean algorithm has a quadratic complexity bound, whereas the L^3 algorithm is cubic for any fixed dimension. In some sense, the analysis of the standard L^3 algorithm corresponds to a naive analysis of the Euclidean algorithm which also gives a cubic complexity. Recall the Euclidean algorithm: given as input two numbers $r_0 > r_1 > 0$, Euclid successively computes the quotients q_i and remainders r_i defined by $q_i = \lfloor r_{i-1}/r_i \rfloor$ and $r_{i+1} = r_{i-1} - q_i r_i$, until $r_{\tau+1} = 0$ for some τ. Then r_τ is the gcd of 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 naive analysis of the Euclidean 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 $O(\log r_{i-1} \cdot \log q_i) = O(\log r_0 \cdot (1 + \log r_{i-1} - \log r_i))$. Summed over all the steps, all but two terms "$\log r_i$" vanish, leading to the classical quadratic complexity bound.

This improved Euclidean analysis cannot unfortunately be extended to the standard L^3, because the GSO coefficients are too big. The bit-length of the numerator and denominator of most GSO coefficients is $O(d \log B)$: computing the exact GSO of an L^3-reduced basis already takes cubic time. Surprisingly, the improved Euclidean analysis can be extended to L^2: this would not be possible without a working 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 the Euclidean and Gaussian algorithms, 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 t-th 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.

5.3 A Cascade in Arbitrary Dimension

In this subsection we complete the proof of Theorem 1. We already know that the number of loop iterations is $\tau = O(d^2 \log B)$. The t-th loop iteration costs $O(dn(d+\log B)(d+\log M(t)))$ where $M(t) = \max_{j<\kappa(t)} |\mu_{\kappa(t),j}(t)|$. By analogy with the Euclidean 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 κ was at least $\kappa(t)$.

Lemma 1. *Let t be a loop iteration. Let $\phi(t) = max(t' < t \mid \kappa(t') \geq \kappa(t))$ if it exists and 1 otherwise, and let $\alpha(t) = min(\kappa(t') \mid t' \in [\phi(t), t)) - 1$. Then we have $\log M(t) \leq d + \log \|\boldsymbol{b}_{\kappa(t)}^{(t)}\| - \log \|\boldsymbol{b}_{\alpha(t)}^{(t)}\|$.*

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

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

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 absorbed in "$O(\tau d)$". The crucial point to do this is the following:

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

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

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

Lemma 3. *Let T and j be integers such that $\kappa(T) \geq j \geq \kappa(T+1)$. We have:*

$$max_{i\leq j} \|\boldsymbol{b}_{i}^{(T+1)*}\| \leq max_{i<j} \|\boldsymbol{b}_{i}^{(T)*}\| \quad and \quad max_{i\leq j} \|\boldsymbol{b}_{i}^{(T+1)}\| \leq \sqrt{d} \cdot max_{i<j} \|\boldsymbol{b}_{i}^{(T)}\|.$$

It is now possible to finish the complexity analysis. Let $k \in [2, d]$ and $t_1 < \ldots < t_{\tau_k} = \{t \leq \tau \mid k(t) = k\}$. We extract from the global sum the terms corresponding to these loop iterations. Theorem 5 and the fact we are dealing with an integer lattice ensures that:

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

Lemma 2 helps to tighly 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 $\|\boldsymbol{b}_{k}^{(t_k)}\| \leq d(\delta - \eta^2)^{-d/2} \|\boldsymbol{b}_{\alpha(t_j)}^{(t_j)}\|$. The indices "$i = k$" in the positive sum and "$i = j$" in the negative sum cancel out and a term "$\frac{d}{2}\log(\delta - \eta^2)^{-1} + \log d$" appears. Then we use Lemma 2 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 a "$\frac{d}{2}\log(\delta-\eta^2)^{-1}+\log d$". We perform this operation $\tau_k - k + 1$ times, to obtain:

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

The fact that $\sum_k \tau_k = \tau$ finally gives $\sum_{t\leq\tau}(d+\log M(t)) = O(\tau d + d^2\log(dB))$.

Acknowledgments

We thank Richard Brent, Guillaume Hanrot, Claus Peter Schnorr and Paul Zimmermann for numerous discussions and anonymous referees for comments which improved the presentation of the results.

References

1. LIDIA 2.1.3. A C++ library for computational number theory. `http://www.informatik.tu-darmstadt.de/TI/LiDIA/`.
2. IEEE 754. IEEE standard for binary floating-point arithmetic.
3. L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.
4. C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. PARI/GP computer package version 2. Université de Bordeaux I. `http://pari.math.u-bordeaux.fr/`
5. Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.
6. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.
7. D. Boneh and G. Durfee. Cryptanalysis of RSA with private key d less than $n^{0.292}$. In *Proc. of Eurocrypt '99*, volume 1592 of *Lecture Notes in Computer Science*, pages 1–11. Springer-Verlag, 1999.
8. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag, 1995. Second edition.
9. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. of Cryptology*, 10(4):233–260, 1997.
10. O. Goldreich, S. Goldwasser, and S. Halevi. Public-key cryptosystems from lattice reduction problems. In *Proc. of Crypto '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag, 1997.
11. G. Golub and C. van Loan. *Matrix Computations*. Johns Hopkins Univ. Press, 1996.
12. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer-Verlag, 1993.
13. C. Hermite. Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres, deuxième lettre. *J. Reine Angew. Math.*, 40:279–290, 1850. Also available in pp122–135 of the first volume of Hermite's complete works, published by Gauthier-Villars.
14. N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Design, Codes and Cryptography*, 23:283–290, 2001.
15. H. Koy and C. P. Schnorr. Segment LLL-reduction of lattice bases. In *Proc. of CALC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 67–80. Springer-Verlag, 2001.
16. H. Koy and C. P. Schnorr. Segment LLL-reduction with floating point orthogonalization. In *Proc. of CALC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 81–96. Springer-Verlag, 2001.
17. J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the Association for Computing Machinery*, January 1985.
18. C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems*. SIAM, 1995.
19. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:513–534, 1982.
20. H. W. Lenstra, Jr. Integer programming with a fixed number of variables. Technical report, Mathematisch Instituut, Universiteit van Amsterdam, April 1981. Report 81-03.

21. H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
22. Magma. The Magma computational algebra system for algebra, number theory and geometry. `http://www.maths.usyd.edu.au:8000/u/magma/`.
23. D. Micciancio. Improving lattice-based cryptosystems using the Hermite normal form. In *Proc. of CALC '01*, volume 2146 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.
24. D. Micciancio and S. Goldwasser. *Complexity of lattice problems: A cryptographic perspective*. Kluwer Academic Publishers, Boston, 2002.
25. P. Q. Nguyễn. Cryptanalysis of the Goldreich-Goldwasser-Halevi Cryptosystem from Crypto '97. In *Proc. of the 19th Cryptology Conference (Crypto '99)*, volume 1666 of *Lecture Notes in Computer Science*, pages 288–304. IACR, Springer-Verlag, 1999.
26. P. Q. Nguyễn and I. E. Shparlinski. The insecurity of the Digital Signature Algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, 2002.
27. P. Q. Nguyễn and D. Stehlé. Low-dimensional lattice basis reduction revisited (extended abstract). In *Proc. of ANTS-VI*, volume 3076 of *Lecture Notes in Computer Science*, pages 338–357. Springer-Verlag, 2004. Journal version in preparation.
28. P. Q. Nguyên and D. Stehlé. A 55-dimensional lattice which makes NTL [41]'s LLL_FP (with $\delta = 0.99$) loop forever. Available at `http://www.loria.fr/~stehle/FPLLL.html`
29. P. Q. Nguyễn and J. Stern. Cryptanalysis of the Ajtai-Dwork Cryptosystem. In *Proc. of the 18th Cryptology Conference (Crypto '98)*, volume 1462 of *Lecture Notes in Computer Science*, pages 223–242. IACR, Springer-Verlag, 1998.
30. P. Q. Nguyễn and J. Stern. The two faces of lattices in cryptology. In *Cryptography and Lattices – Proc. CALC '01*, volume 2146 of *Lecture Notes in Computer Science*, pages 146–180. Springer-Verlag, 2001.
31. A. M. Odlyzko. The rise and fall of knapsack cryptosystems. In *Cryptology and Computational Number Theory*, volume 42 of *Proc. of Symposia in Applied Mathematics*, pages 75–88. A.M.S., 1990.
32. The SPACES Project. MPFR, a LGPL-library for multiple-precision floating-point computations with exact rounding. `http://www.mpfr.org/`.
33. C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Th. Computer Science*, 53:201–224, 1987.
34. C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. of algorithms*, 9(1):47–62, 1988.
35. C. P. Schnorr. Fast LLL-type lattice reduction. Unpublished draft available at `http://www.mi.informatik.uni-frankfurt.de/research/papers.html`, October 2004.
36. C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. In *Proc. of FCT '91*, volume 591 of *Lecture Notes in Computer Science*, pages 68–85. Springer-Verlag, 1991.
37. C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–199, 1994.
38. A. Schönhage. Factorization of univariate integer polynomials by diophantine aproximation and an improved basis reduction algorithm. In *Proc. of ICALP '84*, Lecture Notes in Computer Science, pages 436–447. Springer-Verlag, 1984.
39. A. Schönhage. Fast reduction and composition of binary quadratic forms. In *Proc. of ISSAC '91*, pages 128–133. ACM Press, 1991.
40. A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
41. V. Shoup. NTL, Number Theory C++ Library. `http://www.shoup.net/ntl/`.
42. A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical report, ETH Zurich, 1996.
43. J. H. Wilkinson. *The algebraic eigenvalue problem*. Oxford University Press, New-York, 1988.
44. C. K. Yap. Fast unimodular reduction: Planar integer lattices. In *Proc. of the 33rd Annual Symposium on Foundations of Computer Science*, IEEE, pages 437–446, 1992.