Floating-Point LLL Revisited

Phong Q. Nguyen, Damien Stehlé

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Floating-Point LLL Revisited

Phong Q. Nguyen, Damien Stehlé

N° 5387

Novembre 2004

THÈME 2

ISSN 0249-6399 ISRN INRIA/RR--5387--FR+ENG

INRIA
LORRAINE

Floating-Point LLL Revisited

Phong Q. Nguyen, Damien Stehlé

Thème 2 — Génie logiciel
et calcul symbolique
Projet Spaces

Rapport de recherche n° 5387 — Novembre 2004 — 28 pages

Abstract: Everybody knows the Lenstra-Lenstra-Lovász lattice basis reduction algorithm (LLL), which has proved invaluable in public-key cryptanalysis and in many other fields. Given an integer d-dimensional lattice basis which vectors have norms smaller than B, LLL outputs a so-called LLL-reduced basis in time $O(d^6 \log^3 B)$, 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 LLL is almost never used in practice. Instead, one applies floating-point variants of LLL, where the long-integer arithmetic required by Gram-Schmidt orthogonalisation (central in LLL) is replaced by floating-point arithmetic. Unfortunately, this is known to be unstable in the worst-case: the usual floating-point LLL is not even guaranteed to terminate, and the output basis may not be LLL-reduced at all. In this article, we introduce the LLL^2 algorithm, a new and natural floating-point variant of LLL which provably outputs LLL-reduced bases in polynomial time $O(d^5(d+\log B)\log B)$. This is the first LLL algorithm which running time provably grows only quadratically with respect to $\log B$ without fast integer arithmetic, like the famous Gaussian and Euclidean algorithms. The growth is cubic for all other LLL algorithms known.

Key-words: LLL, Lattice reduction, Complexity, Public-Key Cryptanalysis.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy (France)
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

LLL flottant revisité

Résumé : Tout le monde connaît l'algorithme de Lenstra, Lenstra et Lovász (LLL) pour réduire les bases de réseaux Euclidiens, qui s'est avéré fort utile pour les cryptanalyses de cryptosystèmes à clé publique et dans de nombreux autres domaines. Étant donnée une base à coefficients entiers d'un réseau de dimension d avec des vecteurs de normes plus petites que B, LLL calcule une base LLL-réduite en temps polynomial $O(d^6 \log^3 B)$, en utilisant des opérations arithmétiques sur des entiers de taille $O(d \log B)$. Cette complexité dans le cas le pire est problématique pour les réseaux que l'on rencontre en cryptanalyse où d et/ou $\log B$ sont souvent très grands. Par conséquent, l'algorithme LLL original n'est presque jamais utilisé en pratique. À la place, on se sert de variantes flottantes de LLL, où l'arithmétique entière sur de grands nombres requise par le procédé d'orthogonalisation de Gram-Schmidt (central dans LLL) est remplacée par de l'arithmétique flottante. Malheureusement, ce procédé est connu comme étant instable numériquement dans le cas le pire: il se peut que l'algorithme LLL flottant classique ne termine pas, et que même s'il termine, la base renvoyée ne soit pas du tout LLL-réduite. Dans cet article, nous introduisons l'algorithme LLL^2, qui est une variante nouvelle et naturelle de LLL flottant qui renvoie toujours des bases LLL-réduites en temps polynomial $O(d^5(d+\log B)\log B)$. Il s'agit de la première variante de LLL dont le temps d'éxécution croisse seulement de façon quadratique en $\log B$ sans utiliser de l'arithmétique rapide, comme c'est le cas pour les célèbres algorithmes d'Euclide et de Gauss. La complexité est au moins cubique pour toutes les autres variantes connues de LLL.

Mots-clés : LLL, réduction des réseaux, complexité, cryptanalyse de cryptosystèmes à clé publique.

1 Introduction

Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ be linearly independent vectors in $\mathbb{R}^n$ with $n = O(d)$. The set of all integer linear combinations of the $\boldsymbol{b}_i$'s, denoted by $L[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d] = \left\{\sum_{i=1}^{d} x_i \boldsymbol{b}_i \mid x_i \in \mathbb{Z}\right\}$, is called a *lattice*, and $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ is said to be 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 [10, 6]), particularly in cryptology (see [26, 22]). This problem is known as *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 [17, 18], which was, among others, based on a novel lattice reduction technique (which can be found in the preliminary version [17] of [18]). Lenstra's reduction technique was only polynomial-time for fixed dimension, which was however sufficient in [17]. This inspired Lovász to develop a polynomial-time variant of the algorithm, which reached a final form in the seminal paper [16] where Lenstra, Lenstra and Lovász applied it to factor rational polynomials in polynomial time (back then, a famous problem), from which the name LLL comes. Further refinements of the LLL algorithm were later proposed, notably by Schnorr [29, 30]. LLL and other reduction algorithms have arguably become the most popular tool in public-key cryptanalysis (see [26]). In the past twenty-five years, they have been used to break many public-key cryptosystems, including knapsack cryptosystems [27], RSA in particular settings [7, 5, 4], DSA and similar signatures in particular settings [11, 24], most lattice-based cryptosystems [25, 23], *etc.*

Given as input an integer d-dimensional lattice basis of vectors with norms smaller than B, LLL outputs a so-called LLL-reduced basis in time $O(d^6 \log^3 B)$, 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 [7], we may need to reduce a 64-dimensional lattice with vectors having RSA-type coefficients (1024-bit), in which case the complexity becomes "$d^6 \log^3 B = 2^{66}$ ". As a result, the original LLL algorithm is seldom used in practice. Instead, one applies floating-point (*fp*) variants of LLL, where the long-integer arithmetic required by Gram-Schmidt orthogonalisation (which plays a

central role in LLL) is replaced by floating-point arithmetic (*fpa*) on much smaller numbers. The use of *fpa* in LLL goes back to the early eighties when LLL was used to solve low-density knapsacks [14]. Unfortunately, *fpa* may lead to stability problems, both in theory and practice, especially when the dimension increases: the running time of *fp* variants of LLL such as Schnorr-Euchner's [32] is not guaranteed to be polynomial nor even finite, and the output basis may not be LLL-reduced at all. This phenomenon is well-known to LLL practitioners. For instance, experimental problems arose during the cryptanalyses [25, 23], which led to considerable improvements in Shoup's NTL library [36].

There is however one provable *fp*-variant of LLL, due to Schnorr [30], which significantly improves LLL's worst-case complexity. Schnorr's variant outputs an approximate LLL-reduced basis in time $O(d^4 \log B(d + \log B)^2)$, 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 [36, 20, 2, 19]. 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 [30] is higher than $12d+7 \log B$.

Our Results. We present the LLL^2 algorithm, a new and simple *fp*-variant of LLL in a standard *fpa*-model à la IEEE-754, which provably outputs LLL-reduced bases in polynomial time. LLL^2 comes in two flavors. The first one matches the complexity $O(d^4 \log B(d + \log B)^2)$ of [30], but it is much simpler and has better constants: namely, the required precision decreases from $12d + 7 \log B$ to $1.812d + \log B$. The second version of LLL^2 is even more interesting: its complexity is $O(d^5(d + \log B) \log B)$ using only a $1.6d$-bit precision, that is, independently of $\log B$. This is the first LLL which running time grows only quadratically with respect to $\log B$ (hence the name LLL^2), whereas the growth is cubic for all other provable LLL algorithms known. Interestingly, LLL can be viewed as a generalisation of the famous Gaussian and Euclidean algorithms which complexities are quadratic, not cubic like the original LLL. This arguably makes LLL^2 the most natural version of LLL.

The new LLL^2 algorithm is based on several improvements, both in the LLL 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 [1] to *fpa* in order to stabilize the so-called size-reduction process extensively used in LLL. We give tight bounds on the accuracy of Gram-Schmidt computations to prove the correctness of LLL^2. The analysis leads 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 [36] (an improved version of [32]) loop forever, which contradicts [13] where it was claimed that double precision was sufficient in [32] to LLL-reduce lattices up to dimension 250. 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 is made possible by a tight bound on the complexity of our *fp*-variant of Babai's algorithm, which may be of independent interest. For instance, in Micciancio's variant [21] of the GGH cryptosystem [8], Babai's algorithm is used to decrypt, which may be computationally expensive.

RELATED WORK. Much work [34, 30, 37, 12, 13, 31] has been devoted to improve the complexity of LLL, but none improves the $\log^3 B$ factor. Rather, they focus on the exponent of d in the complexity. We expect most of these improvements to be adaptable to LLL^2.

Floating-point stability has long been a mysterious issue in LLL. 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 [15, 9]) 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 Givens rotations or Householder reflections 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 by Koy and Schnorr in [13] (and also in an early version of [31]), where a theorem from [15] is incorrectly applied. At the moment, it is therefore not clear how to exploit known results on Givens rotations and Householder reflections to improve LLL theoretically. This is why LLL^2 only uses a process close to classical Gram-Schmidt.

ROAD MAP. The paper is organized as follows. In Section 2 we provide necessary background on lattices and LLL. We describe the new LLL^2 algorithm in Section 3. Section 4 proves the correctness of LLL^2, while Section 5 analyzes its complexity. Additional information (such as complete proofs of technical lemmata) is provided in appendix.

2 Background

NOTATIONS. 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 [35]. Our *fpa*-model is a smooth extension of the IEEE-754 standard, as provided for example in NTL [36] and MPFR [28]. With a 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 implemented: the returned value $o(a \text{ op } b)$ for $\text{op} \in \{+, -, /, *\}$ is the 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 assume the reader is familiar with basic algorithmic geometry of numbers [22].

Gram matrix. Let $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d$ be vectors. We denote by $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$ their Gram matrix, i.e. 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 some 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* $\text{vol}(L)$ (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 an orthogonal family defined recursively as follows: $\boldsymbol{b}_i^*$ is the component of $\boldsymbol{b}_i$ orthogonal to the subspace spanned by $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{i-1}$. We have $\boldsymbol{b}_i^* = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \mu_{i,j} \boldsymbol{b}_j^*$ where $\mu_{i,j} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle / \|\boldsymbol{b}_j^*\|^2$. The lattice L spanned by the $\boldsymbol{b}_i$'s satisfies $\text{vol}(L) = \prod_{i=1}^{d} \|\boldsymbol{b}_i^*\|$. 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 in general rational.

QR factorisation. The GSO can be seen as 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 (i.e. $Q \cdot Q^t = Q^t \cdot Q = Id$) and R is lower triangular. This decomposition can be computed e.g. with the Gram-Schmidt algorithm as described above. If $R = (r_{i,j})$, for any i we have $r_{i,i} = \|\boldsymbol{b}_i^*\|^2$ and for any $i > j$ 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 ideal arithmetic, but in the context of our *fp* calculations, it is useful to have them all 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 $j < i \leq d$. An individual vector $\boldsymbol{b}_i$ is size-reduced if $|\mu_{i,j}| \leq \eta$ for all $j < i$. Size reduction usually refers to $\eta = 1/2$ and is typically achieved by successively size-reducing individual vectors by increasing index i.

LLL-reduction. A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ is LLL-reduced with factor (δ, η) for $1/4 < \delta \leq 1$ 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 $(\delta - \mu_{i,i-1}^2)\|\boldsymbol{b}_{i-1}^*\|^2 \leq \|\boldsymbol{b}_i^*\|^2$, which means that the GSO vectors never drop too much. Such bases have several useful properties (see [16, 6, 22]), the following one in particular: the first basis vector is relatively short, namely:

$$\|\boldsymbol{b}_1\| \leq \beta^{(d-1)/4} \mathrm{vol}(L)^{1/d} \text{ , where } \beta = 1/(\delta - \eta^2).$$

LLL-reduction usually refers to the factor $(3/4, 1/2)$ because this was the choice considered in the original paper [16]. But the closer δ and η are respectively to 1 and 1/2, the more reduced the basis is. The classical LLL algorithm obtains in polynomial time a basis reduced with factor $(\delta, 1/2)$ where δ can be arbitrarily close to 1. The new LLL^2 algorithm will achieve a factor of $(0.998, 0.501)$.

The LLL algorithm. The basic LLL algorithm [16] is described in Figure 1. It computes a LLL-reduced basis in an iterative fashion: there is an index κ such that at any stage of the algorithm, the truncated basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}]$ is LLL-reduced. At each loop iteration, κ is either incremented or decremented: the loop stops when κ eventually reaches the value $d+1$, in which case the entire basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ is already LLL-reduced. LLL uses two kinds of operations: swaps of consecutive vectors

Input: A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ and $\delta \in]1/4, 1[$.
Output: A LLL-reduced basis with factor $(\delta, 1/2)$.
1. Compute the GSO, i.e. all the $\mu_{i,j}$'s and $\|\boldsymbol{b}_i^*\|^2$'s.
2. κ:=2. While $\kappa \leq d$ do
3. Size-reduce $\boldsymbol{b}_\kappa$ using Babai's algorithm (Fig. 2), which updates the GSO.
4. If $(\delta - \mu_{\kappa,\kappa-1}^2)\|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^*\|^2$, then κ:=$\kappa + 1$.
5. Else swap $\boldsymbol{b}_{\kappa-1}$ and $\boldsymbol{b}_\kappa$, update the GSO and set κ:=max$(\kappa - 1, 2)$.
6. Output $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$.

Fig. 1. The basic LLL Algorithm.

and Babai's nearest plane algorithm [1] (see Figure 2), which does 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 conditions, while Babai's algorithm is used to size-reduce vectors. When a swap occurs, there is no need size-reducing the newly swapped vector, because it is already size-reduced: this fact will be implicitly used in LLL^2.

If LLL terminates, it is clear that the output basis is LLL-reduced. What is less clear *a priori* is why LLL has 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 multiplicative factor δ, whereas $\Delta \geq 1$ because the $\boldsymbol{b}_i$'s are integer vectors. This allows

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

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

to show that the number of swaps (and therefore the number of loop iterations) is $O(d^2 \log B)$ 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 $O(d^2)$ arithmetic operations on the GSO coefficients $\mu_{i,j}$ and $\|\boldsymbol{b}_i^*\|^2$ which are rational numbers with numerator and denominator of bit-length $O(d \log B)$. Thus, the total complexity of the LLL algorithm described in Fig.1 without fast integer arithmetic is $O((d^2 \log B) \cdot d^2 \cdot (d \log B)^2)) = O(d^6 \log^3 B)$.

LLL with floating-point arithmetic. The cost of the basic LLL 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 representation of the GSO coefficients by floating-point 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 upper bound the complexity of LLL no longer necessarily decreases at each swap. And if ever the algorithm terminates, the output basis may not be LLL-reduced at all, due to potential inaccuracy in the GSO coefficients. Prior to this work, the only provable *fp*-LLL was the one of Schnorr [30], which simulates the behavior of LLL using *fp*-approximations of the coefficients of the inverse matrix of the $\mu_{i,j}$ matrix: it computes a LLL-reduced basis with factor $(0.95, 0.55)$. The number of loop iterations and the number of arithmetic operations (in each loop) remain the same as LLL: only the cost of each arithmetic operation is decreased. Instead of handling integers of bit-length $O(d \log B)$, [30] 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 LLL to $O(d^4 \log B(d + \log B)^2)$, which is still cubic in $\log B$. Because this algorithm is mostly of theoretical interest, the main computer packages [36, 20, 2] only implement heuristic *fp*-variants of LLL à la Schnorr-Euchner [32] which suffer from potential stability problems.

3 The LLL^2 Algorithm

We now describe the LLL^2 algorithm, which is a natural *fp*-variant of LLL. The basic strategy is to keep good *fp*-approximations of the GSO coefficients and to make sure that sufficient accuracy is preserved during the execution of the algorithm. There is no need keeping an approximation for all the GSO coefficients: because LLL is iterative, it suffices to have approximations up to the threshold κ. Accuracy is very important 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 losing stability.

3.1 Gram-Schmidt Computations

It is important for LLL^2 to have accurate formulas for the computation of GSO coefficients. In [32], the following recursive formulas were used:

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

In this formula, the inner products $\langle \boldsymbol{b}_k, \boldsymbol{b}_j \rangle$ and $\|\boldsymbol{b}_k\|^2$ were computed in *fpa*, which leads to a potential inaccuracy of $2^{-\ell}\|\boldsymbol{b}_k\|\|\boldsymbol{b}_j\|$, which has the following drawback: to ensure the basis returned by the LLL^{2}algorithm is size-reduced, absolute error bounds on the $\mu_{k,j}$'s are needed; therefore, if the error is larger than $2^{-\ell}\|\boldsymbol{b}_k\|\|\boldsymbol{b}_j\|$, the precision ℓ must be $\Omega(\log B)$. The analyses of [30], and [31] for the Householder orthogonalisation do not work around this point. We use slightly different formulas by introducing for any $k \geq j$ the quantity $r_{k,j} = \mu_{k,j} \cdot \|\boldsymbol{b}_j^*\|^2 = \langle \boldsymbol{b}_k, \boldsymbol{b}_j^* \rangle$:

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

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: the fewer arithmetic operations, the less the precision loss. For $k = j$, the first formula is $r_{k,k} = \|\boldsymbol{b}_k\|^2 - \sum_{i=1}^{k-1} \mu_{k,i} \cdot r_{k,i}$, which suggests to define $s_{k,j} = \|\boldsymbol{b}_k\|^2 - \sum_{i=1}^{j-1} \mu_{k,i} \cdot r_{k,i}$ for all $1 \leq j \leq k$, so that $\|\boldsymbol{b}_k^*\|^2 = r_{k,k} = s_{k,k}$. The quantities $s_{k,i}$ will be useful to accurately check Lovász's conditions. Indeed, Lovász condition $(\delta - \mu_{\kappa,\kappa-1}^2)\|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^*\|^2$ can be rewritten as $\delta\|\boldsymbol{b}_{\kappa-1}^*\|^2 \leq \|\boldsymbol{b}_\kappa^*\|^2 + \mu_{\kappa,\kappa-1}^2\|\boldsymbol{b}_{\kappa-1}^*\|^2$, that is,

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

If ever that condition is not satisfied, LLL would swap $\boldsymbol{b}_{\kappa-1}$ and $\boldsymbol{b}_\kappa$, and the new Lovász condition to be checked would be:

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

Thus, storing the values $s_{\kappa,j}$'s enables us to check consecutive Lovász conditions (when consecutive swaps occur) without any additional cost since they appear in the calculation of $r_{k,k}$. The computation of $r_{k,j}$, $\mu_{k,j}$ and $s_{k,j}$ is summarized in the so-called Cholesky Factorisation Algorithm (CFA) of Figure 3. Of course, because

Input: The Gram matrix of $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$.
Output: All the $r_{i,j}$'s and $\mu_{i,j}$'s, along with the $s_{d,j}$'s.
1. For $i = 1$ to d do
2. For $j = 1$ to i do
3. $r_{i,j}^{(0)} := \langle \boldsymbol{b}_i, \boldsymbol{b}_j \rangle$.
4. For $k = 1$ to $j - 1$ do
5. $r_{i,j}^{(k)} := r_{i,j}^{(k-1)} - r_{j,k}^{(k-1)} \cdot \mu_{i,k}$.
6. If $i > j$, $\mu_{i,j} := r_{i,j}^{(j-1)} / r_{j,j}^{(j-1)}$.
7. Output $r_{i,j} := r_{i,j}^{(j-1)}$ for $i \geq j$.
8. Output $\mu_{i,j}$ for $i > j$.
9. Output $s_{d,j} := r_{d,d}^{(j-1)}$ for any j.

Fig. 3. The Cholesky Factorisation Algorithm (CFA)

one uses *fpa*, the exact values are unknown to the algorithm: instead, one computes *fp*-approximations $\bar{r}_{i,j}$, $\bar{\mu}_{i,j}$ and $\bar{s}_{k,i}$. Steps 5 and 6 are performed in the following way:

$$\bar{r}_{i,j}^{(k)} := o\left(\bar{r}_{i,j}^{(k-1)} - o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{i,k}\right)\right) \quad \text{and} \quad \bar{\mu}_{i,j} := o\left(\bar{r}_{i,j}^{(j-1)} / \bar{r}_{j,j}^{(j-1)}\right).$$

We will not use CFA directly in LLL^2. Instead, we will use parts of the CFA 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 also that the $r_{i,j}$'s can be updated "in place", except for $r_{d,d}$ because the different $r_{d,d}^{(j)} = s_{d,j}$ need being returned to check Lovász conditions. The whole CFA will prove useful to estimate in Section 4 how many precision bits are required to guarantee the correctness of LLL^2.

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

The core of the LLL^2 algorithm is an iterative *fp*-version of Babai's nearest plane algorithm, which is described in Figure 4. Instead of size-reducing $\boldsymbol{b}_\kappa$ at once like

Input: A working 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 $\bar{r}_{\kappa,j}$'s for $j \leq \kappa$, $\bar{\mu}_{\kappa,j}$'s and $\bar{s}_{\kappa,j}$ for $j < \kappa$, $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1}, \boldsymbol{b}'_\kappa, \boldsymbol{b}_{\kappa+1}, \ldots, \boldsymbol{b}_d)$, 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: $|\langle \boldsymbol{b}'_\kappa, \boldsymbol{b}^*_i \rangle| \leq 0.501 \cdot \|\boldsymbol{b}^*_i\|^2$ for any $i < \kappa$.

1. Repeat
2. Compute the $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s and $\bar{s}_{\kappa,j}$'s with Steps 2–6 of the CFA with "$i = \kappa$".
3. For $i = \kappa - 1$ downto 1 do
4. If $|\bar{\mu}_{\kappa,i}| \geq 0.5005$, then X_i:=$\lfloor \bar{\mu}_{\kappa,i} \rceil$, else X_i:=0,
5. For $j = 1$ to $i - 1$, $\bar{\mu}_{\kappa,j}$:=$o(\bar{\mu}_{\kappa,j} - o(X_i \cdot \bar{\mu}_{i,j}))$.
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. 4. The Iterative Babai Nearest Plane Algorithm

the original Babai algorithm of Figure 2, our *fp*-version of Babai applies an iterative process using parts of the CFA algorithm of Figure 3. 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 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 LLL^2 is given in Figure 5. By selecting particular values of ℓ, we

Input: A basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ and a working precision ℓ for *fpa*.
Output: A LLL-reduced basis with factor $(0.998, 0.501)$.
Variables: An integer matrix G, two $d \times d$ *fp*-matrices $(\bar{r}_{i,j})$ and $(\bar{\mu}_{i,j})$, a *fp*-vector $\bar{s}$.

1. Compute the Gram matrix $G = G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$.
2. $\bar{r}_{1,1}$:=$o(\langle \boldsymbol{b}_1, \boldsymbol{b}_1 \rangle)$, κ:=2. While $\kappa \leq d$, do
3. Size-reduce $\boldsymbol{b}_\kappa$ using the algorithm of Figure 4. It updates the approximate GSO.
4. κ':=κ. While ($\kappa \geq 2$ and $0.999\bar{r}_{\kappa-1,\kappa-1} \geq \bar{s}_{\kappa',\kappa-1}$), do κ:=$\kappa - 1$.
5. For $i = 1$ to $\kappa - 1$, $\bar{\mu}_{\kappa,i}$:=$\bar{\mu}_{\kappa',i}$, $\bar{r}_{\kappa,i}$:=$\bar{r}_{\kappa',i}$. $\bar{r}_{\kappa,\kappa}$:=$\bar{s}_{\kappa',\kappa}$.
6. Update G according to the insertion of $\boldsymbol{b}_{\kappa'}$ right before $\boldsymbol{b}_\kappa$.
7. κ:=$\kappa + 1$.
8. Output $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$.

Fig. 5. The LLL^2 algorithm.

obtain different LLL^2 algorithms. Notice that the cost of the first step is bounded

by $O(d^3 \log^2 B)$ and is thus negligible as regard to the LLL-reduction itself. Steps 4–7 require some explaination: if Lovász's condition is satisfied, nothing happens in Steps 5 and 6, and κ is incremented like in the basic LLL algorithm. Otherwise, the aim of Step 4 is to find the right index of insertion of $\boldsymbol{b}_\kappa$, therefore batching successive failures of Lovász's condition in the basic LLL algorithm. The main results of this paper can be summarized as follows. The first result achieves the same result as [30], using a simpler method and better constants:

Theorem 1. *Given as input a d-dimensional lattice basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ in $\mathbb{Z}^d$ such that $\|\boldsymbol{b}_i\| \leq B$ for any i, the algorithm LLL^2 of Figure 5 with working precision $\ell = 1.812d + \log B + o(d)$ outputs a LLL-reduced basis of factor $(0.998, 0.501)$ in time $O(d^4 \log B(d + \log B)^2)$.*

The second result provides a quadratic LLL algorithm:

Theorem 2. *Given as input a d-dimensional lattice basis $[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d]$ in $\mathbb{Z}^d$ such that $\|\boldsymbol{b}_i\| \leq B$ for any i, the algorithm LLL^2 of Figure 5 with working precision $\ell = 1.6d + o(d)$ outputs a LLL-reduced basis of factor $(0.998, 0.501)$ in time $O(d^5 \log B(d + \log B))$.*

The rest of the paper is devoted to proving those theorems.

4 Correctness of the LLL^2 Algorithm

To guarantee the correctness of LLL^2, we need to estimate the accuracy of *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 [3, 9, 15, 38]. However, it should be stressed that in the context of LLL, 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 LLL-reduced. In this particular case, because a LLL-reduced basis is nearly orthogonal, the following result shows that a working precision of $(\log 3 + \epsilon)d \approx 1.585 \cdot d$ bits is sufficient for the CFA. Although the result holds for any $\epsilon > 0$, for the sake of simplicity, we prove it for $\log 3.03$ instead of $\log 3 + \epsilon$.

Theorem 3. *Let $\rho = 3.03$. Suppose that the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1})$ is LLL-reduced with factor $(0.998, 0.501)$. In the case of floating-point arithmetic with a working*

precision ℓ satisfying $d^2\rho^{d+10}2^{-\ell} \leq 0.001$, the CFA algorithm from Figure 3 computes in time $O(d^3\ell^2)$ fp-numbers satisfying the following equations. For any $j \leq i < d$:

$$|\bar{r}_{i,j} - r_{i,j}| \leq d\rho^{j+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2 \quad \textit{and} \quad |\bar{\mu}_{i,j} - \mu_{i,j}| \leq 1.503d\rho^{j+10}2^{-\ell}.$$

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

$$|\bar{r}_{d,j} - r_{d,j}| \leq d\rho^{j+10}M2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2 \quad \textit{and} \quad |\bar{\mu}_{d,j} - \mu_{d,j}| \leq 1.503d\rho^{j+10}M2^{-\ell}.$$

Finally, if $M \leq 0.501$ then for any $j \leq d$: $|\bar{s}_{d,j} - s_{d,j}| \leq d\rho^{d+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2 + d2^{1-\ell}|s_{d,j}|$.

The second set of inequalities is useful for the analysis of Babai's nearest plane algorithm, while the last set provides guarantees when testing Lovász's conditions on the approximate $r_{i,i}$'s. We now give a sketch of the proof of Theorem 3: a complete proof can be found in appendix. Most of the accuracy loss comes from Step 5, which amplifies the error. We define $err_j = \max_{i<d} \frac{|\bar{r}_{i,j} - r_{i,j}|}{\|\boldsymbol{b}_j^*\|^2}$, which is the error on $r_{i,j}$ relatively to $\|\boldsymbol{b}_j^*\|^2$, and we try to analyze its potential growth with j. 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 the size-reduction property of the LLL-reduction. We choose now $j \in [|2, d-1|]$. 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 6, we have for any $i < d$ and any $k < j$:

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

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 property. This implies that:

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

where we also neglected low-order terms and used size-reduction twice. As a consequence,

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

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

The bound in Theorem 3 is in some sense tight: it is possible to observe in practice that a *fp*-LLL using either the classical Gram-Schmidt algorithm or the Cholesky factorisation algorithm requires a precision of $\Omega(d)$ bits. Indeed, the proof which we just sketched suggests to consider the already reduced lattice given by the rows of the $d \times d$ random matrix L defined by:

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

To obtain an integral lattice, one can multiply L by a large scaling factor and round its entries. Notice that this matrix is indeed LLL-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 40. This type of lattices show the tightness of our $\log 3 \cdot d$ bound. By adding a suitable random vector to such a basis, we were able to make the *fp*-LLL routine of NTL[1] loop forever in dimension 55. This invalidates the claim of [32, 33, 12] 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 Figure 4 and used in LLL[2], we first study a simpler *fp*-version described in Figure 6. Theorem 3 can be used to show stability properties of the Babai algorithm from

Input: A working precision ℓ, $G(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_d)$ and *fp* numbers $\bar{r}_{i,j}$'s and $\bar{\mu}_{i,j}$'s for $j \leq i < d$.
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 2–6 of the CFA with "$i = d$", call them the $\bar{\mu}^{(d)}_{d,j}$'s.
2. For $i = d-1$ downto 1 do
3. If $|\bar{\mu}^{(i+1)}_{d,i}| \geq 0.5005$, then $x_j := \lfloor \bar{\mu}^{(i+1)}_{d,i} \rceil$, else $x_i := 0$.
4. For $j = 1$ to $i-1$ do
5. $\bar{\mu}^{(i)}_{d,j} := o(\bar{\mu}^{(i+1)}_{d,j} - o(x_i \cdot \bar{\mu}_{i,j}))$.
6. Compute the Gram matrix of $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1}, \boldsymbol{b}'_d)$.

Fig. 6. Babai's Nearest Plane Algorithm

Figure 6 in the case of floating-point arithmetic:

[1] FP_LLL with Lovász factor 0.99

Theorem 4. *Suppose the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1})$ is LLL-reduced with factor $(0.998, 0.501)$, 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 $M = max_j|\mu_{d,j}|$ and $M' = 1.001M + 0.5005$. If ℓ satisfies $d^2\rho^{d+15}2^{-\ell} \leq 0.001$, the algorithm of Figure 6 finds integers $x_1, \ldots, x_{d-1}$ such that for any $i < d$:*

$$|x_i| \leq 1.503^{d-i+10}M' \quad \textit{and} \quad \frac{|\langle \boldsymbol{b}'_d, \boldsymbol{b}^*_i \rangle|}{\|\boldsymbol{b}^*_i\|^2} \leq 0.5005 + d^2\rho^{d+15}M'2^{-\ell}.$$

Moreover it performs $O(d^2\ell(d + \log B))$ bit operations, as long as $\ell = O(d + \log B)$.

The proof is given in appendix. Notice that by using the relation $\log M = O(d+\log B)$ (coming from the fact that the $d-1$ first vectors are LLL-reduced), this result implies that taking $\ell = O(d+\log B)$ is sufficient to make the $|\mu_{d,i}|$'s smaller than 0.501. 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 like an overkill because $O(d+\log M)$ bits suffice and M is usually far smaller than B. In the case of the Euclidean algorithm, the analogy is that the quotients would be computed by using all the bits of the remainders, instead of the most significant ones.

The iterative Babai algorithm from Figure 4 is a way to work around the difficulty that M cannot be bound tightly 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 $M \leq 0.501$. This strategy is somewhat similar to the Babai routine of the floating-point LLL algorithm of NTL, in which one applies repeatedly Babai's algorithm until nothing happens.

The iterative Babai will use a working precision $\ell = (\log 3.03 + C)d + o(d)$ for an arbitrary $C > 0$. The CFA with working precision d 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 Babai's algorithm in one single step, but Theorem 4 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, the size-reduction for κ can be achieved. In practice, the optimal choice for C depends on the considered lattice: the higher C, the fewer loop iterations, but the higher arithmetic cost. If the coefficients x_i's are guessed small, it is tempting to choose a small C.

Theorem 5. *Let $c > 12$. Suppose the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1})$ is LLL-reduced with factor $(0.998, 0.501)$, 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 $M = max_{j<\kappa}|\mu_{\kappa,j}|$. If ℓ satisfies*

$d^2\rho^{d+15}2^{c-\ell} \leq 0.001$, *the algorithm of Figure 4 provides a correct output in* $O(d^2(d+\log B)(d+\log M))$ *bit operations, as long as* $\ell = O(d)$. *Moreover, the returned* $\bar{r}_{\kappa,j}$*'s,* $\bar{\mu}_{\kappa,j}$*'s and* $\bar{s}_{\kappa,j}$*'s are those that would have been returned by the CFA with working precision* ℓ.

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 the correctness of the returned $\bar{r}_{\kappa,j}$'s, $\bar{\mu}_{\kappa,j}$'s and $\bar{s}_{\kappa,j}$'s. It also gives that for any $j < \kappa$, $|\bar{\mu}_{\kappa,j}| \leq 0.5005$, from which we derive $|\langle \boldsymbol{b}'_\kappa, \boldsymbol{b}^*_i \rangle| \leq 0.501 \cdot \|\boldsymbol{b}^*_i\|^2$, by using Theorem 4 and the hypothesis on ℓ.

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

$$M_1 \leq 0.5005 + d^2\rho^{d+15}(1.001M + 0.5005)2^{-\ell} \leq 0.5007 + 2^{1-c}M.$$

As a consequence, there can be at most $O\left(1 + \frac{\log M}{c-1}\right)$ loop iterations.

Suppose that we choose $c = \Theta(d)$. Theorem 4 also gives that the cost of one loop iteration is bounded by $O(d^3(d + \log B))$, 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 Gram matrix is taken into account in the complexity bound. Finally, the overall cost of the algorithm is bounded by $O\left(d^3(d+\log B)\left(1+\frac{\log M}{d}\right)\right) = O(d^2(d+\log B)(d+\log M))$. □

4.3 Application to LLL^2

In the *fp*-version of Babai's algorithm, a precision $\ell > \log M + 1.6d$ is required to ensure size-reduction, namely to make the $|\mu_{\kappa,i}|$'s smaller than 0.501. Since M cannot be known in advance, it is natural to replace it by an upper bound which guarantees that the working precision will suffice. For this purpose, we use the following inequalities:

$$M = \max_{j<\kappa} \frac{|\langle \boldsymbol{b}_\kappa, \boldsymbol{b}^*_i \rangle|}{\|\boldsymbol{b}^*_i\|^2} \leq \frac{\sqrt{d}B}{\min_{j<\kappa}\|\boldsymbol{b}^*_i\|} \leq \sqrt{d1.34^d}B,$$

because $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\kappa-1})$ is LLL-reduced and the basis is integral. As a result, $\ell = 1.812d + \log B + o(d)$ is sufficient. In this situation, the iterative Babai algorithm performs only two loop iterations: the first one size-reduces $\boldsymbol{b}_\kappa$, whereas the second one recomputes the $r_{\kappa,j}$'s, $\mu_{\kappa,j}$'s and $s_{\kappa,j}$'s more accurately. This shows that

size-reduction with 0.501 is achieved for the LLL^2 of Theorem 1. For the LLL^2 of Theorem 2, this follows directly from Theorem 5.

It remains to show that Lovász conditions are approximately satisfied. 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 $\bar{s}_{\kappa,j} \leq 0.999\bar{r}_{j,j}$, in which case $1.001 \cdot \bar{s}_{\kappa,j} \leq \bar{r}_{j,j}$ because $1/0.999 \geq 1.001$, which means that with our approximate knowledge of the Gram-Schmidt orthogonalisation, we think that $\boldsymbol{b}_\kappa$ is much shorter than $\boldsymbol{b}_j$ in the space projected orthogonally to $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{j-1})$. Because of Theorem 3, we obtain that:

$$1.001(1 - d2^{1-\ell})r_{\kappa,j} \leq (1 + d\rho^{d+11}2^{-\ell})r_{j,j},$$

therefore, as soon as $d^2\rho^{d+10}2^{-\ell} \leq 10^{-10}$, such a swap implies $(1+2^{-10})s_{\kappa,j} \leq \|\boldsymbol{b}_j^*\|^2$. Similarly, if $\boldsymbol{b}_\kappa$ is not swapped with $\boldsymbol{b}_i$, then $1.001 \cdot \bar{s}_{\kappa,i} \geq \bar{r}_{i,i}$, implying by Theorem 3 that $(1+2^{-9})s_{\kappa,i} \geq \|\boldsymbol{b}_i^*\|^2$. To sum up, after a loop iteration of the LLL^2 algorithm, $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_i, \boldsymbol{b}'_\kappa)$ is LLL-reduced with parameters 0.501 and 0.998, and κ is set to $i+1$.

We thus have shown that the basis output by the LLL^2 algorithms of Theorems 1 and 2 is LLL-reduced with factor $(0.998, 0.501)$.

5 Complexity Analysis of the LLL^2 Algorithm

5.1 Basic Properties and Proof of Theorem 1

Before proving the main two results, we need a set of simple properties satisfied by the LLL^2 algorithm, which we prove in appendix.

Theorem 6. *Let $[\boldsymbol{b}_1^{(0)}, \ldots, \boldsymbol{b}_d^{(0)}]$ a basis of of a lattice L given as input to the LLL^2. For any loop iteration t, $[\boldsymbol{b}_1^{(t)}, \ldots, \boldsymbol{b}_d^{(t)}]$ denotes the current basis at the beginning of the t-th loop iteration. We have:*

- *For any $i \leq \kappa(t) - 1$, $\boldsymbol{b}_i^{(t)}$ is size-reduced, and $[\boldsymbol{b}_1^{(t)}, \ldots, \boldsymbol{b}_{\kappa-1}^{(t)}]$ is LLL-reduced with factor $(0.998, 0.501)$.*
- *For any $i \leq d$, $max_{j \leq i}\|\boldsymbol{b}_j^{(t)*}\| \leq max_{j \leq i}\|\boldsymbol{b}_j^{(0)*}\|$.*
- *For any $i \leq d$, $\|\boldsymbol{b}_i^{(t)}\|^2 \leq d \cdot max_{j \leq i}\|\boldsymbol{b}_i^{(0)}\|^2$.*
- *Any vector appearing during the size-reduction of $\boldsymbol{b}_{\kappa(t)}$ is of length smaller than $2^{O(d)}B^{O(1)}$.*

In Section 4.3, we showed that the accuracy in the check of Lovász conditions was good. For any Lovász's 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 $1 + 2^{-10}$ by Section 4.3. It is a standard LLL argument that this quantity is actually an integer (it is a product of squared volumes of integer lattices), and is initially bounded by $B^{O(d^2)}$. Since in the overall execution of the algorithm the difference between the numbers of decreases and increases of κ is exactly d, there must be at most $O(d^2 \log B)$ loop iterations.

Theorem 5 now immediately gives the complexity $O(d^4 \log B(d + \log B)^2)$ announced in Theorem 1. The rest of this section is devoted to showing how to achieve the complexity $O(d^5 \log B(d + \log B))$ of Theorem 2. This is done by generalizing a cascade phenomenon which appears in the analysis of the Euclidean and the Gaussian algorithm.

5.2 A Naive Analysis of the Euclidean Algorithm

As mentioned in the introduction, the LLL algorithm can be viewed as a high-dimensional generalisation of the Euclidean algorithm to compute gcds. But one annoying fact argues against this analogy: the Euclidean algorithm has a quadratic complexity bound, whereas the LLL algorithm is cubic for any fixed dimension. We claim that the standard analysis of the LLL algorithm corresponds to a naive analysis of the Euclidean algorithm which gives a cubic complexity. The Euclidean algorithm works as follows: given as input two initial remainders $r_0 > r_1 > 0$, it builds the sequences of 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 $r_{\tau+1} = 0$. The r_i's are strictly decreasing integers and r_τ is the gcd of r_0 and r_1. It is well-known that the remainders decrease at least geometrically, which implies that $\tau = O(\log r_0)$. A naive analysis of the Euclidean algorithm states that the algorithm performs $O(\log r_0)$ arithmetic operations on integers of lengths bounded by $O(\log r_0)$, and that the overall cost is bounded by $O(\log^3 r_0)$. Of course, the correct analysis notices that the cost of computing q_i and r_{i+1} 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 in the computation, giving the classical quadratic $O(\log^2 r_0)$ complexity bound.

We show in the following subsection how to obtain such an analysis in the case of the LLL^2 algorithm. The difficulty essentially relies in the generalisation of the cancellation of all but a very few terms in the sum of the costs of consecutive loop iterations.

5.3 A Cascade in the LLL^2 Complexity Analysis

In this subsection we give the proof of the main theorem of the paper, that is Theorem 2.

We already know that the number of loop iterations is $\tau = O(d^2 \log B)$. The t-th loop iteration costs $O(d^2(d+\log B)(d+\log M(t)))$ where $M(t) = \max_{j<\kappa(t)} |\mu_{\kappa(t),j}(t)|$. By analogy with the Euclidean algorithm, we make terms cancel with each other in the sum over the loop iterations of the "$M(t)$'s". For this purpose, we define the index $\alpha(t)$ as the smallest swapping index since the last moment 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 $\alpha(t) = min(\kappa(t') \mid t' \in [|\phi(t), t[|) - 1$. Then we have $\log M(t) \leq d + \log \|\boldsymbol{b}_{\kappa(t)}^{(t)}\| - \log \|\boldsymbol{b}_{\alpha(t)}^{(t)}\|$.*

Proof. Between loop iterations $\phi(t)$ and t, $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\alpha(t)-1}$ remain unchanged and because of the size-reductions, each vector created during these iterations is size-reduced as regard to $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\alpha(t)-1}$. This includes $\boldsymbol{b}_{\kappa(t)}^{(t)}$. As a consequence, by using the fact that $(\boldsymbol{b}_1^{(t)}, \ldots, \boldsymbol{b}_{\kappa(t)-1}^{(t)})$ is LLL-reduced,

$$M(t) \leq 0.501 + \max_{i=\alpha(t)..(\kappa(t)-1)} |\mu_{\kappa(t),i}| \leq 0.501 + 1.34^{d/2} \|\boldsymbol{b}_{\kappa(t)}^{(t)}\| / \|\boldsymbol{b}_{\alpha(t)}^{(t)}\|.$$

□

We are to subdivide the sum of the "$M(t)$'s" into $O(d^2)$ subsums according to the value $\alpha(t)$ and $\kappa(t)$ at the loop iteration t:

$$\sum_{t \leq \tau} (d + \log M(t)) \leq d^3 \log B + \sum_{a=1}^{d-1} \sum_{k=a+1}^{d} \sum_{t \text{ s.t. } \alpha(t)=a, \kappa(t)=k} (\log \|\boldsymbol{b}_k^{(t)}\| - \log \|\boldsymbol{b}_a^{(t)}\|).$$

For each of these subsums, we keep $k-a$ positive terms and $k-a$ negative terms, and make the others vanish in a "$(k-a)$-shifted cascade". The crucial point to do this is the following:

Lemma 2. *Let $1 \leq a < k \leq d$ and $t_1 < \ldots < t_{k-a}$ be loop iterations of the LLL^2 algorithm such that for any $j \leq k-a$, $\kappa(t_j) = k$ and $\alpha(t_j) = a$. Then $\|\boldsymbol{b}_a^{(t_1)}\| \geq d^2 1.34^{2d} \|\boldsymbol{b}_k^{(t_{k-a})}\|$.*

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

Lemma 3. *Let T be a loop iteration and j an integer such that $\kappa(T) \geq j \geq \kappa(T+1)$. We have:*

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

Proof of Lemma 3: If $i \leq j$ differs from $\kappa(T+1)-1$, $\boldsymbol{b}_i^{(T+1)}$ is one the $\boldsymbol{b}_{i'}^{(T)}$'s for $i' \leq j-1$. From size-reduction, it suffices to show that $\|\boldsymbol{b}_{\kappa(T+1)-1}^{(T+1)*}\| \leq \max_{i\leq j-1}\|\boldsymbol{b}_i^{(T)*}\|$. Because Lovász's test failed at loop iteration T for $\kappa(T+1)-1$, we have $\|\boldsymbol{b}_{\kappa(T+1)-1}^{(T+1)*}\|^2 \leq \|\boldsymbol{b}_{\kappa(T+1)-1}^{(T)*}\|^2$. □

Proof of Lemma 2: By definition of α, for any $j \in [|2, k-a|]$, there exits a first loop iteration $T_j \in [|t_{j-1}, t_j|[$ such that $\kappa(T_j) \geq j \geq \kappa(T_j+1)$. Because of Theorem 6 and Lemma 3, we have:

$$\|\boldsymbol{b}_k^{(t_{k-a})}\|^2 \leq d\cdot\max_{i\leq k}\|\boldsymbol{b}_i^{(T_{k-a}+1)}\|^2 \leq d^2\cdot\max_{i\leq k-1}\|\boldsymbol{b}_i^{(T_{k-a})}\|^2.$$

By minimality of T_{k-a} the vectors $\boldsymbol{b}_1,\ldots,\boldsymbol{b}_{k-1}$ do not change in loop iterations t_{k-a-1} to T_{k-a}. Since $\kappa(t_{k-a-1}) = k$, these vectors are LLL-reduced, which gives that:

$$\|\boldsymbol{b}_k^{(t_{k-a})}\|^2 \leq d^2 1.34^d \max_{i\leq k-1}\|\boldsymbol{b}_i^{(t_{k-a-1})*}\|^2.$$

Theorem 6 and the first claim of Lemma 3 give that:

$$\begin{aligned}\max_{i\leq k-1}\|\boldsymbol{b}_i^{(t_{k-a-1})*}\|^2 &\leq \max_{i\leq k-1}\|\boldsymbol{b}_i^{(T_{k-a-1}+1)*}\|^2 \leq \max_{i\leq k-2}\|\boldsymbol{b}_i^{(T_{k-a-1})*}\|^2\\ &\leq \max_{i\leq k-2}\|\boldsymbol{b}_i^{(t_{k-a-2})*}\|^2\\ &\cdots\\ &\leq \max_{i\leq a}\|\boldsymbol{b}_i^{(t_1)*}\|^2\end{aligned}$$

The fact that $(\boldsymbol{b}_1^{(t_1)},\ldots,\boldsymbol{b}_a^{(t_1)})$ is LLL-reduced gives the result. □

It is now possible to finish the complexity analysis. Lemma 2 gives that:

$$\sum_{t\leq\tau}\log\left(\frac{\|\boldsymbol{b}_{\kappa(t)}^{(t)}\|}{\|\boldsymbol{b}_{\alpha(t)}^{(t)}\|}\right) \leq \sum_a\sum_k\left((k-a)\log(\sqrt{d}B) + d\cdot|\{t \mid \kappa(t)=k, \alpha(t)=a\}|\right),$$

which is $O(d^3\log B)$, because all the $\|\boldsymbol{b}_i^{(t)}\|$'s are smaller than $\sqrt{d}B$ and $\tau = O(d^2\log B)$.

References

1. L. Babai. On Lovász lattice reduction and the nearest lattice point problem. *Combinatorica*, 6:1–13, 1986.
2. C. Batut, K. Belabas, D. Bernardi, H. Cohen, and M. Olivier. PARI/GP computer package version 2. Université de Bordeaux I.
3. Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1996.
4. D. Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213, 1999.
5. 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.
6. H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, 1995. Second edition.
7. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *J. of Cryptology*, 10(4):233–260, 1997.
8. 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.
9. G. Golub and C. van Loan. *Matrix Computations.* Johns Hopkins Univ. Press, 1996.
10. M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization.* Springer-Verlag, 1993.
11. N. A. Howgrave-Graham and N. P. Smart. Lattice attacks on digital signature schemes. *Design, Codes and Cryptography*, 23:283–290, 2001.
12. 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.
13. 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.
14. J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the Association for Computing Machinery*, January 1985.
15. C. L. Lawson and R. J. Hanson. *Solving Least Squares Problems.* SIAM, 1995.
16. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:513–534, 1982.
17. H. W. Lenstra, Jr. Integer programming with a fixed number of variables. Technical report, Mathematisch Instituut, Universiteit van Amsterdam, April 1981. Report 81-03.
18. H. W. Lenstra, Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8(4):538–548, 1983.
19. Lidia. A library for computational number theory. `http://www-jb.cs.uni-sb.de/LiDIA/linkhtml/lidia/lidia.html`.
20. Magma. The Magma computational algebra system for algebra, number theory and geometry. `http://www.maths.usyd.edu.au:8000/u/magma/`.
21. 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.
22. D. Micciancio and S. Goldwasser. *Complexity of lattice problems: A cryptographic perspective.* Kluwer Academic Publishers, Boston, 2002.

23. P. Q. Nguyen. 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.
24. P. Q. Nguyen and I. E. Shparlinski. The insecurity of the Digital Signature Algorithm with partially known nonces. *Journal of Cryptology*, 15(3):151–176, 2002.
25. P. Q. Nguyen 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.
26. P. Q. Nguyen 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.
27. 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.
28. The Spaces Project. Mpfr, a LGPL-library for multiple-precision floating-point computations with exact rounding. `http://www.mpfr.org/`.
29. C. P. Schnorr. A hierarchy of polynomial lattice basis reduction algorithms. *Th. Computer Science*, 53:201–224, 1987.
30. C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *J. of algorithms*, 9(1):47–62, 1988.
31. C. P. Schnorr. Fast LLL-type lattice reduction. Unpublished draft available at `http://www.mi.informatik.uni-frankfurt.de/research/papers.html`, October 2004.
32. 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.
33. C. P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Math. Programming*, 66:181–199, 1994.
34. 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.
35. A. Schönhage and V. Strassen. Schnelle Multiplikation grosser Zahlen. *Computing*, 7:281–292, 1971.
36. V. Shoup. Number Theory C++ Library (NTL). See `http://www.shoup.net/ntl/`.
37. A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical report, ETH Zurich, 1996.
38. J. H. Wilkinson. *The algebraic eigenvalue problem.* Oxford University Press, New-York, 1988.

A Proof of Theorem 3

Correctness in the case of perfect arithmetic: Suppose first that we use a perfect arithmetic. We claim that for any $k < j \leq i$, $r_{i,j}^{(k)} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^{(k)} \rangle$, where $\boldsymbol{b}_j^{(k)}$ is the component of $\boldsymbol{b}_j$ that is orthogonal to $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_k$, i.e. $\boldsymbol{b}_j^{(k)} = \boldsymbol{b}_j - \sum_{l \leq k} \mu_{j,l} \boldsymbol{b}_l^*$. It is easy to see that:

$$r_{i,j}^{(k)} = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^{(k)} \rangle = \langle \boldsymbol{b}_i, \boldsymbol{b}_j^{(k-1)} \rangle - \mu_{j,k} \langle \boldsymbol{b}_i, \boldsymbol{b}_k^* \rangle = r_{i,j}^{(k-1)} - r_{j,k}^{(k-1)} \cdot \mu_{i,k},$$

because of the definitions of $\mu_{j,k}$ and $\mu_{i,k}$.

Proof for $i < d$: Suppose now that we use a floating-point arithmetic with a precision ℓ satisfying $d^2\rho^{d+10}2^{-\ell} \leq 0.001$. The main point is to bound the error made in the calculations of the $r_{i,j}^{(k)}$'s computed in Step 5. Recall that Step 5 is performed in the following way:

$$\bar{r}_{i,j}^{(k)} := o\left(\bar{r}_{i,j}^{(k-1)} - o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{i,k}\right)\right).$$

Let $j < d$. We show the following error bound by induction on $j \leq d-1$:

$$\text{For any } i \in [\![j, d-1]\!], \quad |\bar{r}_{i,j}^{(j-1)} - r_{i,j}^{(j-1)}| \leq d\rho^{j+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2. \qquad (*)$$

This is easy for $j = 1$ since the $\langle \boldsymbol{b}_i, \boldsymbol{b}_1 \rangle$'s are known exactly and since $|\langle \boldsymbol{b}_i, \boldsymbol{b}_1 \rangle| \leq 0.501\|\boldsymbol{b}_1\|^2$. Suppose now that $j \in [\![2, d-1]\!]$ and that the result holds for any $k \leq j-1$. By using the induction hypothesis, we have the following inequalities:

$$\begin{aligned}
|\bar{\mu}_{i,k} - \mu_{i,k}| &\leq 2^{-\ell}\frac{|\bar{r}_{i,k}^{(k-1)}|}{\bar{r}_{k,k}^{(k-1)}} + \frac{|\bar{r}_{i,k}^{(k-1)} - r_{i,k}^{(k-1)}|}{\bar{r}_{k,k}^{(k-1)}} + |r_{i,k}^{(k-1)}| \cdot \left|\frac{1}{r_{k,k}^{(k-1)}} - \frac{1}{\bar{r}_{k,k}^{(k-1)}}\right| \\
&\leq 2^{-\ell} + 2^{-\ell}\left|\frac{\bar{r}_{i,k}^{(k-1)}}{\bar{r}_{k,k}^{(k-1)}} - \frac{r_{i,k}^{(k-1)}}{r_{k,k}^{(k-1)}}\right| + \frac{r_{k,k}^{(k-1)}}{\bar{r}_{k,k}^{(k-1)}} d2^{-\ell}\rho^{k+10} + 0.501\frac{r_{k,k}^{(k-1)}}{\bar{r}_{k,k}^{(k-1)}} d\rho^{k+10}2^{-\ell} \\
&\leq 1.503 \cdot d\rho^{k+10}2^{-\ell},
\end{aligned}$$

because $\frac{r_{k,k}^{(k-1)}}{\bar{r}_{k,k}^{(k-1)}} \leq 1 + d\rho^{k+10}2^{-\ell} \leq 1.001$ and $2^{-\ell} < 3^{-10}$. As a consequence:

$$\begin{aligned}
&\left|o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{i,k}\right) - r_{j,k}^{(k-1)}\mu_{i,k}\right| \\
&\quad\leq 2^{-\ell}|\bar{r}_{j,k}^{(k-1)}| \cdot |\bar{\mu}_{i,k}| + |\bar{r}_{j,k}^{(k-1)}| \cdot |\bar{\mu}_{i,k} - \mu_{i,k}| + |\mu_{i,k}| \cdot \left|\bar{r}_{j,k}^{(k-1)} - r_{j,k}^{(k-1)}\right| \\
&\quad\leq 2^{-\ell}(0.501 + d\rho^{k+10}2^{-\ell})^2\|\boldsymbol{b}_k^*\|^2 + (0.501 + d\rho^{k+10}2^{-\ell}) \cdot 1.503 \cdot d\rho^{k+10}2^{-\ell}\|\boldsymbol{b}_k^*\|^2 \\
&\qquad + 0.501 \cdot d\rho^{k+10}2^{-\ell}\|\boldsymbol{b}_k^*\|^2,
\end{aligned}$$

because of the induction hypothesis and the fact that $\frac{|r_{j,k}^{(k-1)}|}{\|\boldsymbol{b}_k^*\|^2}, |\mu_{i,k}| \leq 0.501$,

$$\leq 1.256 \cdot d\rho^{k+10}2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^4,$$

by using the fact that $d^2\rho^{k+10}2^{-\ell} \leq 0.001$.

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

$$\begin{aligned}
&\leq 2^{-\ell}|\bar{r}_{i,j}^{(k-1)}| + \left|\bar{r}_{i,j}^{(k-1)} - r_{i,j}^{(k-1)}\right| + (0.502)^2 2^{-\ell}\|\boldsymbol{b}_k^*\|^2 \\
&\quad +1.256 \cdot d\rho^{k+10}(2^{-\ell} + 2^{-2\ell})\|\boldsymbol{b}_k^*\|^2 \\
&\leq 2^{-\ell}|r_{i,j}^{(k-1)}| + \left|\bar{r}_{i,j}^{(k-1)} - r_{i,j}^{(k-1)}\right|(1 + 2^{-\ell}) + 1.258 \cdot d\rho^{k+10}2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2 \ .
\end{aligned}$$

Notice now that for any k,

$$|r_{i,j}^{(k)}| \leq |\langle \boldsymbol{b}_i, \boldsymbol{b}_j^* \rangle| + \sum_{k'=k+1}^{j-1} |\mu_{j,k'}||\langle \boldsymbol{b}_i, \boldsymbol{b}_{k'}^* \rangle| \leq \|\boldsymbol{b}_j^*\|^2 \sum_{k'=k+1}^{j} 1.34^{j-k'} \leq 3 \cdot 1.34^{j-k} \cdot \|\boldsymbol{b}_j^*\|^2,$$

because the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_j)$ is LLL-reduced. By using the fact that $(1 + 2^{-\ell})^d \leq 1 + d2^{1-\ell}$, we have for any $k' < j$:

$$\begin{aligned}
\left|\bar{r}_{i,j}^{(k')} - r_{i,j}^{(k')}\right| &\leq 2^{-\ell}(1 + d2^{1-\ell}) \textstyle\sum_{k<k'} |\bar{r}_{i,j}^{(k)} - r_{i,j}^{(k)}| + 3 \cdot d1.34^{j-k'+1}(1 + d2^{1-\ell})\|\boldsymbol{b}_j^*\|^2 \\
&\quad +1.259 \cdot d2^{-\ell} \cdot \|\boldsymbol{b}_{k'}^*\|^2 \textstyle\sum_{k\leq k'} \rho^{k+10} 1.34^{k'-k} \\
&\leq 2^{1-\ell} \textstyle\sum_{k<k'} |\bar{r}_{i,j}^{(k)} - r_{i,j}^{(k)}| + 3 \cdot d1.34^{j-k'+2}\|\boldsymbol{b}_j^*\|^2 + \frac{1.259}{\frac{2}{1.34}-1} \cdot d\rho^{k'+11}2^{-\ell} \cdot \frac{\|\boldsymbol{b}_{k'}^*\|^2}{1.34} \\
&\leq 2^{1-\ell} \textstyle\sum_{k<k'} |\bar{r}_{i,j}^{(k)} - r_{i,j}^{(k)}| + 0.999 \cdot d\rho^{j+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2.
\end{aligned}$$

From this it is easy to show by induction on k' that:

$$\left|\bar{r}_{i,j}^{(k')} - r_{i,j}^{(k')}\right| \leq 0.999 \cdot d\rho^{j+10}(1 + k'2^{1-\ell})2^{-t} \cdot \|\boldsymbol{b}_j^*\|^2,$$

which directly implies $(*)$ for j, because $0.999 \cdot (1 + d2^{1-\ell}) \leq 1$.

Proof for $i = d$: Essentially, we "add M's" in the previous proof where they have to be added. We prove by induction on $j \leq d - 1$ that:

$$|\bar{r}_{d,j}^{(j-1)} - r_{d,j}^{(j-1)}| \leq d\rho^{j+10}M2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2.$$

The case $j = 1$ is obvious, so we suppose that $j \geq 2$ and that we know the result holds for any $k < j$. By proceeding as in the proof for $i < d$, we obtain the following inequalities:

$$\begin{aligned}
|\bar{\mu}_{d,k} - \mu_{d,k}| &\leq 2^{-\ell}\frac{|\bar{r}_{d,k}^{(k-1)}|}{\bar{r}_{k,k}^{(k-1)}} + \frac{|\bar{r}_{d,k}^{(k-1)} - r_{d,k}^{(k-1)}|}{\bar{r}_{k,k}^{(k-1)}} + |r_{d,k}^{(k-1)}| \cdot \left|\frac{1}{r_{k,k}^{(k-1)}} - \frac{1}{\bar{r}_{k,k}^{(k-1)}}\right| \\
&\leq 2^{-\ell}M + 2^{-\ell}\left|\frac{\bar{r}_{d,k}^{(k-1)}}{\bar{r}_{k,k}^{(k-1)}} - \frac{r_{d,k}^{(k-1)}}{r_{k,k}^{(k-1)}}\right| + 1.501(1 + d\rho^{k+10}2^{-\ell})d\rho^{k+10}M2^{-\ell} \\
&\leq 1.503d\rho^{k+10}M2^{-\ell},
\end{aligned}$$

which implies that:

$$\begin{aligned}
&\left|o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{d,k}\right) - r_{j,k}^{(k-1)}\mu_{d,k}\right| \\
&\quad\leq 2^{-\ell}|\bar{r}_{j,k}^{(k-1)}| \cdot |\bar{\mu}_{d,k}| + |\bar{r}_{j,k}^{(k-1)}| \cdot |\bar{\mu}_{d,k} - \mu_{d,k}| + |\mu_{d,k}| \cdot |\bar{r}_{j,k}^{(k-1)} - r_{j,k}^{(k-1)}| \\
&\quad\leq (M + 1.503d\rho^{k+10}M2^{-\ell})2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2 + 0.502 \cdot 1.503 \cdot d\rho^{k+10}M2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2 \\
&\qquad + d\rho^{k+10}M2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2,
\end{aligned}$$

by definition of M, because of $(*)$ and because of the induction hypothesis.

$$\leq 1.258 \cdot d\rho^{k+10}M2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2 \ ,$$

because we supposed that $d^2\rho^{d+10}2^{-\ell} \leq 0.001$.

$$\begin{aligned}
|\bar{r}_{d,j}^{(k)} - r_{d,j}^{(k)}| &\leq 2^{-\ell}|\bar{r}_{d,j}^{(k-1)}| + 2^{-\ell}\left|o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{d,k}\right)\right| + \left|\bar{r}_{d,j}^{(k-1)} - r_{d,j}^{(k-1)}\right| \\
&\quad + \left|o\left(\bar{r}_{j,k}^{(k-1)} \cdot \bar{\mu}_{d,k}\right) - r_{j,k}^{(k-1)}\mu_{d,k}\right| \\
&\leq 2^{-\ell}|\bar{r}_{d,j}^{(k-1)}| + \left|\bar{r}_{d,j}^{(k-1)} - r_{d,j}^{(k-1)}\right| + 0.501M2^{-\ell}\|\boldsymbol{b}_k^*\|^2 \\
&\quad + 1.258d\rho^{k+10}M(2^{-\ell} + 2^{-2\ell})\|\boldsymbol{b}_k^*\|^2 \\
&\leq 2^{-\ell}|r_{d,j}^{(k-1)}| + \left|\bar{r}_{d,j}^{(k-1)} - r_{d,j}^{(k-1)}\right|(1 + 2^{-\ell}) + 1.259d\rho^{k+10}M2^{-\ell} \cdot \|\boldsymbol{b}_k^*\|^2 \ .
\end{aligned}$$

Notice now that for any k,

$$|r_{d,j}^{(k)}| \leq |\langle \boldsymbol{b}_d, \boldsymbol{b}_j^* \rangle| + \sum_{k'<j} |\mu_{j,k'}||\langle \boldsymbol{b}_d, \boldsymbol{b}_{k'}^* \rangle| \leq M \sum_{k' \leq j} \|\boldsymbol{b}_{k'}^*\|^2 \leq 3 \cdot 1.34^{j-k}M \cdot \|\boldsymbol{b}_j^*\|^2,$$

since $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1})$ is LLL-reduced. For the same reason, by summing over k we obtain that:

$$\begin{aligned}
\left|\bar{r}_{d,j}^{(k')} - r_{d,j}^{(k')}\right| &\leq 2^{1-\ell}\textstyle\sum_{k<k'} |\bar{r}_{d,j}^{(k)} - r_{d,j}^{(k)}| + 3d1.334^{j-k'+2}M\|\boldsymbol{b}_j^*\|^2 \\
&\quad + 1.259dM2^{-\ell}\|\boldsymbol{b}_{k'}^*\|^2 \textstyle\sum_{k \leq k'} \rho^{k+10}1.34^{k'-k} \\
&\leq 2^{1-\ell}\textstyle\sum_{k<k'} |\bar{r}_{d,j}^{(k)} - r_{d,j}^{(k)}| + 3d1.34^{j-k'+2}M\|\boldsymbol{b}_j^*\|^2 + \\
&\quad \tfrac{1.259}{\frac{\rho}{1.34}-1}d\rho^{l+11}M2^{-\ell}\tfrac{\|\boldsymbol{b}_{k'}^*\|^2}{1.34}.
\end{aligned}$$

As a consequence, for any $k' < j < d$:

$$\left|\bar{r}_{d,j}^{(k')} - r_{d,j}^{(k')}\right| \leq 2^{1-\ell}\sum_{k<k'} |\bar{r}_{d,j}^{(k)} - r_{d,j}^{(k)}| + 0.999 \cdot d\rho^{j+10}M2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2.$$

From this it is easy to show by induction on k' that:

$$\left|\bar{r}_{d,j}^{(k')} - r_{d,j}^{(k')}\right| \leq 0.999 \cdot d\rho^{j+10}(1 + k'2^{1-\ell})M2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2,$$

which directly implies the induction claim for j, because $0.999 \cdot (1 + d2^{1-t}) \leq 1$.

Case where $\boldsymbol{b}_d$ is size-reduced as regard to $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1})$. Similarly, we have that:

$$\left|\bar{r}_{d,d}^{(j)} - r_{d,d}^{(j)}\right| \leq 2^{-\ell}|r_{d,d}^{(j-1)}| + \left|\bar{r}_{d,d}^{(j-1)} - r_{d,d}^{(j-1)}\right|(1+2^{-\ell}) + 1.259 \cdot d\rho^{j+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2.$$

By using the fact that $|r_{d,d}^{(j)}| \leq \|\boldsymbol{b}_d\|^2$, we obtain:

$$\begin{aligned}\left|\bar{r}_{d,d}^{(j)} - r_{d,d}^{(j)}\right| &\leq 2^{1-\ell}\sum_{k<j}|\bar{r}_{d,d}^{(k)} - r_{d,d}^{(k)}| + d2^{-\ell}\|\boldsymbol{b}_d\|^2 + 1.259d2^{-\ell}\|\boldsymbol{b}_j^*\|^2\sum_{k\leq j}\rho^{k+10}1.34^{j-k} \\ &\leq 2^{1-\ell}\sum_{k<j}|\bar{r}_{d,d}^{(k)} - r_{d,d}^{(k)}| + d2^{-\ell}\|\boldsymbol{b}_d\|^2 + 0.94 \cdot d\rho^{d+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2.\end{aligned}$$

From this it is easy to show by induction on j that:

$$\left|\bar{r}_{d,d}^{(j)} - r_{d,d}^{(j)}\right| \leq 0.94 \cdot d\rho^{d+10}(1+j2^{1-\ell})2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2 + d(1+j2^{1-\ell})2^{-\ell} \cdot \|\boldsymbol{b}_d\|^2.$$

This gives that $\left|\bar{r}_{d,d}^{(j)} - r_{d,d}^{(j)}\right| \leq 0.95d\rho^{d+10}2^{-\ell} \cdot \|\boldsymbol{b}_j^*\|^2 + d2^{1-\ell} \cdot \|\boldsymbol{b}_d\|^2$. The result follows from the fact that $\boldsymbol{b}_d$ is size-reduced and that $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{d-1})$ is LLL-reduced.

The complexity bound of the theorem is obvious. □

B Proof of Theorem 4

Let $\mu_{d,j}^{(i)} = \mu_{d,j} - \sum_{k=i}^{d-1} x_k\mu_{l,j}$. We show by decremental induction on $i \in [|1,d|]$ that for any $j < i$:

$$|\bar{\mu}_{d,j}^{(i)}| \leq 1.503^{d-i+10}M' - 0.5005 \quad \text{and} \quad |\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}| \leq (d+1-i)d\rho^{d+14}M'2^{-\ell}. \qquad (*)$$

Equation $(*)$ *implies the bound on the* $|x_i|$*'s* because $|x_i| \leq |\bar{\mu}_{d,i}^{(i+1)}| + \max(0.5005, 2^{-\ell}|\bar{\mu}_{d,i}^{(i+1)}|)$.

Equation $(*)$ *implies the bound on the* $\frac{\langle \boldsymbol{b}_d', \boldsymbol{b}_i^*\rangle}{\|\boldsymbol{b}_i^*\|^2}$*'s* because for any $i < d$:

$$\begin{aligned}\frac{|\langle \boldsymbol{b}_d', \boldsymbol{b}_i^*\rangle|}{\|\boldsymbol{b}_i^*\|^2} = |\mu_{d,i}^{(i+1)} - \lfloor\bar{\mu}_{d,i}^{(i+1)}\rceil| &\leq |\bar{\mu}_{d,i}^{(i+1)} - \mu_{d,i}^{(i+1)}| + |\bar{\mu}_{d,i}^{(i+1)} - \lfloor\bar{\mu}_{d,i}^{(i+1)}\rceil| \\ &\leq d^2\rho^{d+14}M'2^{-\ell} + \max(0.5005, 1.503^{d-i+9}M'2^{-\ell}) \\ &\leq 0.5005 + d^2\rho^{d+15}M'2^{-\ell}.\end{aligned}$$

Theorem 3 gives the base case of the induction. For $j < d$ we have:

$$\left|\bar{\mu}_{d,j}^{(d)} - \mu_{d,j}^{(d)}\right| \leq 1.503 d\rho^{j+10} M 2^{-\ell} \leq d\rho^{d+11} M 2^{-\ell}.$$

Moreover, $|\bar{\mu}_{d,j}^{(d)}| \leq M + d\rho^{d+11} M 2^{-\ell} \leq 1.001M$, because of the hypothesis on ℓ.

We now prove that $(*)$ *holds for* i *under the hypothesis that it holds for any* $i' > i$. Because $|\bar{\mu}_{d,i}^{(i+1)}| \leq 1.503^{d-i+9} M' - 0.5005$, we have that $|x_i| \leq 1.503^{d-i+10} M'$. By using Theorem 3, we obtain:

$$\begin{aligned}
|o(x_i \cdot \bar{\mu}_{i,j}) - x_i \mu_{i,j}| &\leq 2^{-\ell} |x_i||\bar{\mu}_{i,j}| + |x_i||\bar{\mu}_{i,j} - \mu_{i,j}| \\
&\leq 0.502 \cdot 1.503^{d-i+10} M' 2^{-\ell} + 1.503^{d-i+10} M' \cdot 1.503 d\rho^{j+10} 2^{-\ell} \\
&\leq 1.504 \cdot d 1.503^{d-i+10} \rho^{j+10} M' 2^{-\ell},
\end{aligned}$$

which gives that:

$$\begin{aligned}
|\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}| &\leq 2^{-\ell} |\bar{\mu}_{d,j}^{(i+1)}| + 2^{-\ell} |o(x_i \cdot \bar{\mu}_{i,j})| + |\bar{\mu}_{d,j}^{(i+1)} - \mu_{d,j}^{(i+1)}| + |o(x_i \cdot \bar{\mu}_{i,j}) - x_i \mu_{i,j}| \\
&\leq 1.503^{d-i+9} M' 2^{-\ell} + 1.504 \cdot d 1.503^{d-i+10} \rho^{j+10} M' (2^{-\ell} + 2^{-2\ell}) \\
&\qquad + 0.501 \cdot 1.503^{d-i+10} M' 2^{-\ell} + |\bar{\mu}_{d,j}^{(i+1)} - \mu_{d,j}^{(i+1)}| \\
&\leq |\bar{\mu}_{d,j}^{(i+1)} - \mu_{d,j}^{(i+1)}| + 1.506 \cdot d 1.503^{d-i+10} \rho^{j+10} M' 2^{-\ell}. \\
&\leq (d-i) d\rho^{d+14} M' 2^{-\ell} + d\rho^{d+14} M' 2^{-\ell},
\end{aligned}$$

by using the fact that $j \leq i - 1$ and $1.506 \cdot 1.503^{11} < \rho^5$. This gives the second part of Equation $(*)$ for i. For the first part, we use the fact that $|x_k| \leq 1.503^{d-k+9}(1 + 2^{-\ell}) M'$ for any $k \geq i$, so that we obtain for any $j < i$:

$$\begin{aligned}
|\bar{\mu}_{d,j}^{(i)}| &\leq |\bar{\mu}_{d,j}^{(i)} - \mu_{d,j}^{(i)}| + |\mu_{d,j}| + \sum_{k=i}^{d-1} |x_k||\mu_{k,j}| \\
&\leq M + 0.501 M' (1 + 2^{-\ell}) \sum_{k=i}^{d-1} 1.503^{d-k+9} + (d + 1 - i) d\rho^{d+14} M' 2^{-\ell} \\
&\leq M + 1.001 \cdot \frac{0.501}{0.503} \cdot (1.503^{d-i+10} - 1.503^{10}) M' + 0.001 M' \\
&\leq 1.503^{d-i+10} M' - 0.5.
\end{aligned}$$

We finally analyze the cost of the different arithmetic operations. Theorem 3 gives that the cost of Step 1 is bounded by $O(d^2 \ell^2)$. Compared to the cost of Step 5, the cost of Step 3 is negligible. There are $O(d^2)$ arithmetic operations during Steps 2–5, each

one having its cost dominated by the multiplication $o(x_i \cdot \bar{\mu}_{i,j})$. Because $x_i = \lfloor \bar{\mu}_{d,i}^{(i+1)} \rceil$, x_i is an integer that can be represented on $O(\ell)$ bits. As a consequence, the cost of Steps 2–5 is $O(d^2\ell^2)$. At Step 6, we have $O(d^2)$ arithmetic operations on integers. By using Equation $(*)$ and the relation: $|\langle \boldsymbol{b}_i, \boldsymbol{b}_d \rangle| \leq B(\|\boldsymbol{b}_d^*\|^2 + \sum_{i<d} \mu_{d,i}^2 \|\boldsymbol{b}_i^*\|^2)$, it is easy to see that all the involved scalar products remain below $dB^2 1.503^{d+1} M' \leq (B2^d)^{O(1)}$. The most expensive arithmetic operations are of the kind "$x_i \langle \boldsymbol{b}_d, \boldsymbol{b}_i \rangle$", and since the x_i's are of size $O(\ell)$, the cost of Step 6 can be bounded by $O(d^2\ell(d+\log B))$. This dominates the overall cost of the algorithm. □

C Proof of Theorem 6

Claim 1) has been proved in Section 4. Claim 4) follows from Theorem 5. We now consider Claim 2). During a swap, we have that:

- $\|\boldsymbol{b}_{\kappa-1}^{*new}\| \leq \|\boldsymbol{b}_{\kappa-1}^{*old}\|$ because of Lovász's condition and the analysis of Section 4,
- $\|\boldsymbol{b}_{\kappa}^{*new}\| \leq \|\boldsymbol{b}_{\kappa-1}^{*old}\|$ because $\boldsymbol{b}_{\kappa}^{*new}$ is an orthogonal projection of $\boldsymbol{b}_{\kappa-1}^{*old}$,
- $\|\boldsymbol{b}_{\kappa}^{*new}\| \geq \|\boldsymbol{b}_{\kappa}^{*old}\|$ because $\boldsymbol{b}_{\kappa}^{*old}$ is an orthogonal projection of $\boldsymbol{b}_{\kappa-1}^{*new}$,
- $\|\boldsymbol{b}_{\kappa}^{*new}\| \geq \|\boldsymbol{b}_{\kappa}^{*old}\|$ because $\|\boldsymbol{b}_{\kappa-1}^*\| \cdot \|\boldsymbol{b}_{\kappa}^*\|$ is constant.

Claim 2) directly follows from these facts and an induction on t. It implies that 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 Claim 3) for $i < \kappa(t)$. If $i \geq \kappa(t)$, we consider the last loop iteration $t' < t$ where the vector $\boldsymbol{b}_i^{(t)}$ was created. If t' does not exist, then this vector is an initial vector, and the result is obvious. Otherwise, we have $\boldsymbol{b}_i^{(t)} = \boldsymbol{b}_{\kappa(t'+1)-1}^{(t'+1)}$ which is size-reduced at the $(t'+1)$-th loop iteration. This finishes the proof of the theorem. □

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399