Towards faster polynomial-time lattice reduction

Paul Kirchner[1], Thomas Espitau[2], and Pierre-Alain Fouque[1]

[1] IRISA/Inria, Rennes Univ., France, {paul.kirchner,pierre-alain.fouque}@irisa.fr
[2] NTT Secure Platform Laboratories, Tokyo, Japan, t.espitau@gmail.com

Abstract. The LLL algorithm is a polynomial-time algorithm for reducing d-dimensional lattice with exponential approximation factor. Currently, the most efficient variant of LLL, by Neumaier and Stehlé, has a theoretical running time in $d^4 \cdot B^{1+o(1)}$ where B is the bitlength of the entries, but has never been implemented. This work introduces new asymptotically fast, parallel, yet *heuristic*, reduction algorithms with their optimized implementations. Our algorithms are recursive and fully exploit fast matrix multiplication. We experimentally demonstrate that by carefully controlling the floating-point precision during the recursion steps, we can reduce euclidean lattices of rank d in time $\tilde{O}(d^\omega \cdot C)$, i.e., almost a constant number of matrix multiplications, where ω is the exponent of matrix multiplication and C is the log of the condition number of the matrix. For cryptographic applications, C is close to B, while it can be up to d times larger in the worst case. It improves the running-time of the state-of-the-art implementation `fplll` by a multiplicative factor of order $d^2 \cdot B$. Further, we show that we can reduce structured lattices, the so-called knapsack lattices, in time $\tilde{O}(d^{\omega-1} \cdot C)$ with a progressive reduction strategy. Besides allowing reducing huge lattices, our implementation can break several instances of Fully Homomorphic Encryption schemes based on large integers in dimension 2,230 with 4 millions of bits.

1 Introduction

Lattice reduction and cryptanalysis. Lattice reduction is of the utmost importance in public-key cryptanalysis, as testified, for instance, by the extensive survey of Joux and Stern [40]. Indeed, many cryptographic problems are solved by constructing an appropriate lattice and retrieving one of its short vectors. Some standard examples include knapsack problems [46,48,40], breaking linear congruential generators [28,69]), Coppersmith attack [19] against RSA modulus by retrieving small roots of univariate polynomials over $\mathbb{Z}/N\mathbb{Z}$ or bivariate polynomials over $\mathbb{Z}$, or even attacks against the initial versions of the NTRU cryptosystem [20,32,4]. Yet, its field of applications extends way beyond cryptography, as lattice reduction is a cornerstone of many number theoretical algorithms, allowing factoring polynomials over $\mathbb{Z}[X]$ [50], finding integer relations [37], solving simultaneous diophantine approximation problems [45].

Essentially, lattice reduction means finding a short and nearly orthogonal basis to a lattice Λ (represented as a $\mathbb{Z}$-basis). For many applications, finding a

small non-zero lattice vector, i.e., solving the (approximate) Short Vector Problem (SVP), shall suffice. Since the work of Minkowski, we know that there exists a vector with euclidean norm smaller than $\sqrt{d}(\operatorname{vol}\Lambda)^{\frac{1}{d}}$, but the proof is not constructive. Nonetheless, the LLL algorithm, introduced in 1982 by Lenstra, Lenstra, and Lovász [50] retrieves vector within an exponential factor to the shortest vector of a lattice of dimension d in time $\mathrm{O}\big(d^6B^3\big)$ where B is the bitsize of the input representation. One can also prove that the norm of the first vector of an LLL-reduced basis is less than $\big(\sqrt{4/3}\big)^{\frac{d-1}{2}}(\operatorname{vol}\Lambda)^{\frac{1}{d}}$. The approximation factor $(\|b_1\|/\operatorname{vol}\Lambda)^{1/d}$ is called the *root Hermite factor* (RHF), with b_1 a short vector. Later, Schnorr developed a hierarchy of algorithms to reach better RHF in $\beta^{\frac{1}{2\beta}}$ in time $2^{\mathrm{O}(\beta)}$ for large β [62,64]. This family leads to a polynomial-time algorithm with a RHF $2^{\frac{\log\log d}{\log d}}$ [47]. Gama and Nguyen introduced the slide reduction to give an effective take on Mordell's inequality and further improve the RHF [29]. In an orthogonal direction, following Haståd and Lagarias, Seysen proposed a variant of LLL aiming at simultaneously reduces the primal and dual basis [67]. He defines a new reduceness measure which is closely related to the condition number of the matrix [51].

Related work. The two most singular characteristics of lattices appearing in the cryptographic setting are their high dimension and the large bitsize of their matrix representation. As such, the reduction of cryptanalytically relevant lattices is a computationally intensive challenge. While the original LLL implementation works with exact arithmetic on rational entries, Schnorr proposed in 1988 to replace it with floating-point arithmetic [63], significantly improving its efficiency. Since 1996, Shoup maintains a heuristic yet very efficient version in the NTL library with fine control of the float-point precision. This code has been routinely used for more than a decade to break cryptographic schemes. Later, Nguyen and Stehlé precisely analyzed and decreased the asymptotic complexity to $\mathrm{O}\big(d^5(d+B)B\big)$ in [57], a.k.a. the quadratic LLL or L^2 algorithm. This algorithm has been then implemented in `fpLLL` [3], which is the current state-of-the-art open-source implementation of LLL. However, despite many *theoretical* improvements to reduce the complexity to quasi-linear in the bitsize using recursive local computation techniques [65,44,58,55] and some attempts [13,61,11] to use only the most significant bits, the practical complexity of the best implementation available remains in $\mathrm{O}\big(d^4B^2\big)$. As such, it struggles to reduce lattices with large entries in high dimensions. Consequently, cryptographers still assess their concrete parameters using L^2 as a reference for LLL.

Thus, from a cryptanalytical standpoint, it is interesting to have a fast implementation of lattice reduction (with a controlled approximation factor) even though this algorithm might rely on some heuristics. Since lattice-based cryptography is becoming a strong contender for post-quantum cryptography and offers many interesting functionalities to cryptography, such as efficient Fully Homomorphic Encryption (FHE), new algorithms and implementations of lattice reduction have been designed to give better security estimate for lattice-based cryptography. Some improvements mainly target the BKZ algorithm since

it allows to finely adjust the approximation factor [29,30,35,14,53,5]. Others are heuristic and improve sieving technique for solving SVP, use a reduction technique in the lattice dimension [6], exploit subfield structure and symmetries in structured lattices [60,41], or use the tensor core architecture of GPU [26]. Some of them with sieving SVP-oracle [6] are used to perform the security estimation of signatures and KEM, where the dimension are generally lying between 512 and 1024. However, FHE schemes over the integers use extremely large integers (several millions of bits) and high dimensional lattices (typically of a few thousand dimensions), but can be broken with high approximation factors. To deal with such settings, faster algorithms are required, in particular with complexity quasi-linear in the bitsize and not much more costly than matrix multiplication.

Our Contributions. To improve the running time of lattice reduction algorithms, we propose to exploit parallelism with many cores, make full use of computer's cache using block matrix implementation [34], and use a low precision while still controlling the approximation factor at the same time. Our implementation we describe allows reducing lattice in dimensions up to 2,000 with entries of up to millions of bits, which is intractable otherwise.

Our proposal of lattice reduction is a LLL-type algorithm, i.e., using a size-reduction procedure jointly, together with many passes of a rank-2 reduction subprocess. The design rationale is to exploit fast block matrix operations and locality of operations. To do so, we use a block variant of the Cholesky factorization algorithm for computing the QR-decomposition [34]. We replace the size-reduction with a block variant of Seysen's size-reduction, which can be thought of as a rounded version of the multiplication by the inverse of the $\mathbf{R}$ factor of the QR-decomposition. To our knowledge, this algorithm has not been used since 1993. Contrary to the textbook LLL, we do not swap vectors when the Lovász condition is not fulfilled, but we fully reduce the 2-dimensional corresponding projected sublattice, using Schönhage's algorithm. The global design is recursive, as was proposed before by Koy and Schnorr [44] with Segmented LLL and by Neumaier and Stehlé [55]. However, in this work, we do not recurse on overlapping blocks but on separate ones; a technique proposed by Villard to achieve parallelism [73] with even and odd steps, also used recently in [41].

As all the computations are conducted in floating-point arithmetic, a systematic caveat concerns the precision required for computing the correct result. We claim and experimentally verify that on average, it decreases exponentially with the recursion depth as shown in section 4.1, allowing to reduce the overall complexity by a factor d. Additionally, we handle matrice multiplications in the Fourier domain to compute with large numbers. We conjecture and experimentally verify a complexity of approximately $d^\omega \cdot C / \log C$, where ω is the exponent of matrix multiplication, and C is the logarithm of the condition number of the input matrix. We highlight that typically, the complexity of lattice reduction depends on the bitlength B of the input, instead of C. For cryptographic applications (Coppersmith and knapsack-type lattice amid others), C is close to B, while it can be up to d times larger in the worst case. It is well-known that a row-wise diagonal dominant matrice has a condition number bounded by a

constant times the ratio beteen the largest diagonal entry and the smallest one, so the logarithm of the condition number will be close to the size of the entries.

Additionally, section 5 shows that one can reduce knapsack lattice in a time approximately equal to the reduction of a random lattice with a bitsize reduced by a factor of d. Such a phenomenon is already known for some algorithms like fplll [68, 1.5.3], noted [56], and exploited [72]. We present a reduction between the two problems with this property. The idea is to iteratively double the number of columns reduced, and reduce the bitsize of the other ones. It has been implemented and tested.

The complexity of our algorithms can be analyzed in an arithmetic cost model with an analysis similar to [35] (sandpile model) for LLL with even and odd pass as in [42]. However, the specificity of our algorithm is to consider the precision. Without such attention, it would have been impossible to reduce high dimensional lattice with so many bits. In an exact arithmetic cost model, the complexity would have been comparable to previous algorithms, which is not the case in practice. Such heuristic algorithms is interesting, even without a full analysis, to assess the security of cryptographic instances. For instance, many FHE schemes over the integers base their security on the complexity of the best algorithm. However, a rigorous proof of the algorithm with the precision is highly technical in a numerical computational model and escape us so far. Consequently, we decided to present only all the ingredients of our implementation with its applications in this paper and postpone a proof for future work.

Regarding the applications, we first show in section 6 that our implementation is much faster than fplll with a factor between 30 and 45 on single-thread in all dimensions tractable by fplll. However, our implementation can exploit multicore processors and reduce lattices in much higher dimensions. Consequently, we run it on matrices of dimensions a few thousand and inputs of millions of bits, as reported in table 1. As a result, we attack many instances FHE over the integers to illustrate the efficiency of our code and evaluate its running time on large inputs. For these examples, the wall-clock time is six orders of magnitude smaller than the (estimated) cost of fplll. We broke knapsack instances from [21] in dimension 2,230 with 4.26 millions of bits in 22h with 18 cores, while the security level was evaluated at 2^{62}. We also broke NTRU instances with overstretched parameters proposed in [31] in 5h (resp. 10 days) in dimension 2560 (resp. 3086) with 111 (resp. 883) bits and RHF $2^{0.1105}$ ($2^{0.018}$, equivalent to BKZ-25). In practice, at the bottom of the recursion tree, we use a small BKZ to improve the approximation factor, whilst not altering too much the running time.

2 Background

2.1 Notations and conventions

The capitals $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$ refer to the ring of integers, the field of rational and real. Given a real number x, its integral rounding denoted by $\lfloor x \rceil$ returns its closest integer. The logarithms are log for the binary one and ln for the natural one.

Matrix and norms. We denote by $\mathbb{Q}^{d\times d}$ the space of square matrices of size d over $\mathbb{Q}$, $\mathrm{GL}_d(\mathbb{Q})$ its group of invertible. We use bold fonts for matrices and denote the elementary matrix transformations by $\mathbf{T}_{i,j}(\lambda)$ and $\mathbf{D}_i(\lambda)$ for respectively the transvection (or shear mapping) and the dilatation of parameter λ. We use $\mathrm{Diag}(x_1,\ldots,x_d)$ to refers to a diagonal matrix of elements $x_1,\ldots,x_d$. We generalize this definition to block matrices and overload it to the extraction of the diagonal of a given matrix. A *triangular unipotent* or *unitriangular matrix* is a triangular matrix with ones on the diagonal. We extend the product for any pair of matrices $(\mathbf{A},\mathbf{B})$: for every matrix $\mathbf{C}$ with compatible size with $\mathbf{A}$ and $\mathbf{B}$, we set: $(\mathbf{A},\mathbf{B})\cdot\mathbf{C} = (\mathbf{AC},\mathbf{BC})$. We adopt the usual conventions for submatrix extraction: for any matrix $\mathbf{M} = (m_{i,j}) \in \mathbb{Q}^{d\times d}$ and $1 \leqslant u < v \leqslant d, 1 \leqslant w < x \leqslant d$, define the submatrix $\mathbf{M}[u:v, w:x] = (m_{i,j})_{u\leqslant i\leqslant v, w\leqslant j\leqslant x}$, while $\mathbf{M}_i$ refers to the i-th column of $\mathbf{M}$. For a vector v (resp. matrix $\mathbf{A} = (a_{i,j})_{1\leqslant i,j\leqslant d}$), we denote by $\|v\|$ (resp. $\|\mathbf{A}\|$) the Frobenius norm, i.e., $\|\mathbf{A}\| = \sqrt{\sum_{1\leqslant i,j\leqslant d} a_{i,j}^2}$. The condition number of an invertible matrix $\mathbf{M}$ measures how much the output value of the matrix can change for a small change in the input. It is defined as $\kappa(\mathbf{M}) = \|\mathbf{M}\|\|\mathbf{M}^{-1}\|$ and allows to compute the precision needed during the computation. We deal with block decomposition of matrices, with block of half-dimension. For matrices of odd dimension $2k+1$, the upper-left block to be of dimension $k+1$ and the bottom-right one of dimension k.

Computational setting. We use the standard model in algorithmic theory, i.e., the word-RAM with unit cost and logarithmic size register (see [52, Section 2.2] for a comprehensive description). The number of bits in the register is w and the precision during the computation by p. All computations with rational/real values are conducted in floating-point, unless stated otherwise. For a non-negative integer d, we set $\omega(d)$ to be the exponent of matrix multiplication of $d\times d$ matrices. If the dimension d is clear from context we might omit it and write simply $\mathrm{O}(d^\omega)$ for this complexity. We can assume that this exponent is not too close to 2, in particular $\omega(d) > 2 + 1/\log(d)$. Due to the conflict with Laudau's small omega notation, we use $\boldsymbol{\omega}$ for the latter symbol.

2.2 Lattices and LLL reduction

Definition 1 (Lattice). *A d-dimensional (real) lattice $\Lambda \subseteq \mathbb{R}^d$ is the set of integer linear combinations $\sum_{i=1}^d b_i\mathbb{Z}$ of some linearly independent vectors $(b_i)_{1\leqslant i\leqslant d}$.*

The finite family $(b_1,\ldots,b_d) \in \Lambda$ is called a *basis* of Λ. Every basis has the same number of elements called the *rank* of the lattice. A measure of the density of the lattice is its *(co)volume*, defined to be the volume of the torus $\mathbb{R}^d/\Lambda$, which corresponds to the square root of the Gram-determinant of any basis $(b_1,\ldots,b_d)$:

$$\mathrm{vol}\,\Lambda = \sqrt{\det(\langle b_i, b_j\rangle)_{1\leqslant i,j\leqslant d}}.$$

Two different bases of a lattice Λ are related by a *unimodular transformation*, i.e., a linear transformation represented by an element of $\mathrm{GL}_d(\mathbb{Z})$, the set of

$d \times d$ integer-valued matrices of determinant ± 1. In essence, algorithms acting on lattice bases are sequences of unimodular transformations. Among these procedures, reduction algorithms are of the utmost importance. They aim at finding congenial classes of bases, which are *quasi-orthogonal* and with controlled norms. Fundamental constant associated to any rank d lattice Λ are its successive minima $\lambda_1, \ldots, \lambda_d$. The ith minimum $\lambda_i(\Lambda)$ is the radius of the smallest sphere centered in the origin containing i linearly independent lattice vectors.

Orthogonalization, QR-decomposition. Let $\mathbf{B} = (b_1, \ldots, b_d)$ a family of linearly independent vectors. Let π_i the orthogonal projection on $(b_1, \ldots, b_{i-1})^\perp$, with the convention that $\pi_1 = \mathrm{Id}$. The Gram-Schmidt orthogonalization process is an algorithmic method for orthogonalizing $\mathbf{B}$ while preserving the increasing chain of subspaces $(\bigoplus_{j=1}^{i} b_j \mathbb{R})_{1 \leqslant i \leqslant d}$. It constructs the orthogonal set $\mathbf{B}^* = (\pi_1(b_1), \ldots, \pi_d(b_d))$. For notational simplicity we refer generically to the orthogonalized vectors by b_i^* for $\pi_i(b_i)$. The computation of $\mathbf{B}^*$ can be done inductively as follows: for all $1 \leqslant i \leqslant d$, $b_i^* = b_i - \sum_{j=1}^{i-1} \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} b_j^*$. Collect the family $\mathbf{B}$ in a matrix also denoted by the same notation and set $\mathbf{R}_{i,j} = \frac{\langle b_j, b_i^* \rangle}{\|b_i^*\|}$ and $\mathbf{Q} = \left[\frac{b_1^*}{\|b_1^*\|} \middle| \ldots \middle| \frac{b_d^*}{\|b_d^*\|} \right]$. Then, we have $\mathbf{B} = \mathbf{QR}$, with $\mathbf{Q}$ being an orthogonal matrix and $\mathbf{R}$ being upper triangular. This is the QR-decomposition of $\mathbf{B}$. In the following, we work with the $\mathbf{R}$ part only, so that we present the computation of this matrix in the pseudo-code **Orthogonalize** below. We omit considerations on the required fp-precision here, to just focus on the core ideas of the algorithms.

Algorithm 1 — Orthogonalize

Input Basis $\mathbf{B}$

Output $\mathbf{R}$ part of QR-decomposition

for $i = 1$ **to** d **do**
  for $j = i - 1$ **to** 1 **do**
    $\mathbf{Q}_i \leftarrow b_i - \frac{\langle b_i, \mathbf{Q}_j \rangle}{\langle \mathbf{Q}_j, \mathbf{Q}_j \rangle} \mathbf{Q}_j$
  end for
end for
return $\mathbf{R} = \left(\frac{\langle \mathbf{Q}_i, b_j \rangle}{\|\mathbf{Q}_i\|} \right)_{1 \leqslant i \leqslant j \leqslant d}$

Algorithm 2 — Size-Reduce

Input Basis $\mathbf{B}$, $\mathbf{R}$ part of QR-decomposition

Output Tranformation of SR basis

$\mathbf{U} = \mathbf{Id}_d$
for $i = 1$ **to** d **do**
  for $j = i - 1$ **to** 1 **do**
    $(U, R) \leftarrow (U, R) \cdot \mathbf{T}_{i,j} \left(- \left\lceil \frac{\mathbf{R}[i,j]}{\mathbf{R}[i,i]} \right\rfloor \right)$
  end for
end for
return $\mathbf{U}$

Size-reduction of a family of vectors. Let Λ be a rank d lattice given by a basis $\mathbf{B} = (b_1, \ldots, b_d)$, we might want to use the Gram-Schmidt process. However, since the quotients $\frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}$ are not integral in general, the vectors b_i^* may not lie in Λ. The size-reduction process instead approximates the result of the Gram-Schmidt process by rounding to a nearest integer: each vector b_i is replaced by $b_i - \sum_{j=1}^{i-1} \left\lceil \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle} \right\rfloor b_j$. The whole process takes time $\mathrm{O}(d^5 B^2)$ when

the input matrix $\mathbf{B}$ is of dimension $d \times d$ with B-bit entries. This process is called *Size-reduction* and corresponds to the following iterative algorithm[3] **Size-reduce**.

2.3 The LLL reduction algorithm

Lenstra, Lenstra, and Lovász [50] proposed a notion called LLL-*reduction* and a polynomial-time algorithm that computes an LLL-reduced basis from an arbitrary basis of the same lattice. Their reduction notion is formally defined as follows (presented directly in an algorithmic way with the QR-decomposition):

Definition 2 (LLL reduction). *A basis* $\mathbf{B}$ *of a lattice, admitting the decomposition* $\mathbf{B} = \mathbf{QR}$*, is said to be* δ-LLL*-reduced for* $1/4 < \delta \leqslant 1$*, if the following two conditions are satisfied:*

$$\forall i < j, \quad |\mathbf{R}[i,j]| \leqslant \frac{1}{2}|\mathbf{R}[i,i]| \quad \textit{(Size-Reduction condition)} \tag{1}$$

$$\forall i, \quad \delta \left\| \begin{pmatrix} \mathbf{R}_{j,j} \\ 0 \end{pmatrix} \right\|^2 \leqslant \left\| \begin{pmatrix} \mathbf{R}_{j,j+1} \\ \mathbf{R}_{j+1,j+1} \end{pmatrix} \right\|^2 \quad \textit{(Lovász condition).} \tag{2}$$

The length of vectors and orthogonality defect is related to the parameter δ:

Proposition 1. *Let* $1/4 < \delta \leqslant 1$ *be an admissible* LLL *parameter. Let* $(b_1, \ldots, b_d)$ *a* δ-LLL *reduced basis of rank-*d *lattice* Λ*. Then for any* $1 \leqslant k \leqslant d$*:*

$$\operatorname{vol}(b_1, \ldots, b_k) \leqslant (\delta - 1/4)^{-\frac{(d-k)k}{4}} \operatorname{vol} \Lambda^{\frac{k}{d}}.$$

In particular, we have that $\mathbf{R}_{i,i} \leqslant (\delta - 1/4)^{-1} \mathbf{R}_{i+1,i+1}$.

We recall that $\operatorname{vol}(\Lambda) = \det(\mathbf{B}) = \prod_{i=1}^{d} \mathbf{R}_{i,i}$ and the log-potential is defined as $\Pi(\mathbf{B}) = \sum_{i=1}^{d} (d-i) \log(\mathbf{R}_{i,i})$. For $k = 1$ and $\delta = 1$, the Hermite approximation factor defined as $\|b_1\| / \det(\mathbf{B})^{1/d}$, will be $(4/3)^{(d-1)/4}$. To find a basis entailing the LLL conditions, it suffices to iteratively modify it at any index violating one of these conditions. This process yields the simplest version of the LLL algorithm. However, we choose to present a different take on this algorithm, closer to the algorithms we introduce later. The first remark is that for a given $1 \leqslant j \leqslant d-1$, the LLL-reduceness conditions correspond to saying that the basis

$$\begin{pmatrix} \mathbf{R}_{j,j} & \mathbf{R}_{j,j+1} \\ 0 & \mathbf{R}_{j+1,j+1} \end{pmatrix}$$

is Gauss–reduced. The global strategy given in algorithm 3 to reduce a lattice consists of iteratively applying a reduction procedure in rank 2 to projected sublattices, naturally using the Gauss reduction algorithm [35]. We start by reducing the sublattice spanned by b_1, b_2, then the projection onto the orthogonal

[3] We choose to present it using the matrix $\mathbf{R}$ and yielding the unimodular transformation matrix, for consistency with the description of our algorithms in section 3.

subspace to b_1 sublattice spanned by b_2, b_3 and so on. When we hit the end of the basis, this iteration restarts afresh until no more progress is achieved.

We replace the outermost **while** loop by a for loop of a fixed number ρ of iterations. This parameter is set to be sufficiently large to ensure the reducedness of the output (using a dynamical system analysis à la [35] after $\mathrm{O}(d^2 \log B)$ rounds, a vector within the LLL quality bound is discovered).

We will use a slight generalization of the LLL-reduction notion. In particular, a LLL-reduced basis satisfying the Lovász conditions, is a Siegel reduced basis.

Definition 3 (Siegel reduction). *The Siegel reduction problem consists in, given an integer matrix* $\mathbf{A}$ *of dimension* d *with* $\|\mathbf{A}\|, \|\mathbf{A}^{-1}\| \leqslant 2^B$*, outputting a matrix* $\mathbf{AU}$ *with* $\mathbf{U}$ *a unimodular integer matrix such that with* $\mathbf{QR} = \mathbf{AU}$ *the QR-decomposition, we have for all* i*:* $\mathbf{R}_{i,i} \leqslant 2\mathbf{R}_{i+1,i+1}$.

Algorithm 3 — Reduction

Input : Initial basis $\mathbf{B} = (b_1, \ldots, b_d)$
Output : A δ-LLL-reduced basis

while $\mathbf{B}$ *is not* LLL*-reduced* **do**
  $\mathbf{R} \leftarrow$ **Orthogonalize**$(\mathbf{B})$
  $\mathbf{U}_i \leftarrow$ **Size-Reduce**$(\mathbf{R})$
  $(\mathbf{B}, \mathbf{R}) \leftarrow (\mathbf{B}, \mathbf{R}) \cdot \mathbf{U}_i$
  for $j = 1$ **to** d **do**
    $\mathbf{B}' \leftarrow \mathbf{R}[j : j+1, j : j+1]$
    $\mathbf{U}' \leftarrow$**Gauss**$(\mathbf{B}')$
    $(\mathbf{U}_i, \mathbf{B}) \leftarrow (\mathbf{U}_i, \mathbf{B}) \cdot \mathrm{Diag}(\mathbf{Id}_{j-1}, \mathbf{U}', \mathbf{Id}_{d-j-1})$
  end for
end while
return $\prod_{i=1}^{\rho} \mathbf{U}_i$ // ρ is the number of passes

2.4 Matrices Representation

A matrix $\mathbf{A}$ is represented as $\mathbf{A}'2^e$ where $\mathbf{A}'$ is an integer matrix and $e \leqslant 0$. The quantity $\log(\|\mathbf{A}'\|)$ is the *precision* of the matrix. The standard algorithm for multiplying matrices with large entries consists in transforming the integers in $\mathbf{A}$ and $\mathbf{B}$ into polynomials of degree bounded by $\mathrm{O}\left(\frac{p+w}{w}\right)$ (p is the precision and w the number of bits in registers), and computing their evaluations on roots of unity. The matrices of evaluations are then multiplied, and an inverse Fourier transform gives the product of the matrix of polynomials. Carries are then computed to obtain $\mathbf{AB}$. Matrices can be multiplied quickly using the FFT:

Theorem 1. *Given* $\mathbf{A}$ *and* $\mathbf{B}$ *two integer matrices of dimension* d *with* $\log(\|\mathbf{A}\| + \|\mathbf{B}\|) = p$*, the product* $\mathbf{AB}$ *can be computed in time* $O\left(d^\omega \frac{p+w}{w} + d^2 \frac{p}{w} \log\left(2 + \frac{p}{w}\right)\right)$.

2.5 Fast inversion of unitriangular matrices

We eventually conclude this preliminary section by introducing a natural recursive algorithm to invert unitriangular matrices—working with floating-point approximation. It is a direct application of the computation of Schur's complement in the case of a block triangular matrix, i.e., the observation that:

$$\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ 0 & \mathbf{D} \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{C}\mathbf{D}^{-1} \\ 0 & \mathbf{D}^{-1} \end{pmatrix}.$$

As both $\mathbf{A}$ and $\mathbf{D}$ are unitriangular, this inversion formula translates naturally in a recursive algorithm. Its base case corresponds to inverting a one dimensional unitriangular matrix, that is (1), which is its own inverse. The corresponding pseudo-code is given in **Invert**. Its complexity is easily analyzed to be asymptotically the cost of a matrix multiplication, as the dominant step of each recursive call is the computation of the complement $-\mathbf{A}^{-1}\mathbf{C}\mathbf{D}^{-1}$.

Algorithm 4 — Invert

Input : A unitriangular matrix $\mathbf{M}$
Output : A fp-approximation of $\mathbf{M}^{-1}$

1 **if** $dim(\mathbf{M}) = 1$ **then**
2 | **return** (1)
3 **end if**
4 $\begin{pmatrix} \mathbf{A} & \mathbf{C} \\ 0 & \mathbf{D} \end{pmatrix} \leftarrow \mathbf{M}$ // with dimension almost halved
5 $\mathbf{A}' \leftarrow$ **Invert**$(\mathbf{A})$; $\mathbf{D}' \leftarrow$ **Invert**$(\mathbf{D})$ // fp-approximations of A^{-1}, D^{-1}
6 $\mathbf{S} \leftarrow -\mathbf{A}'\mathbf{C}\mathbf{D}'$ // Computed in floating-point
7 **return** $\begin{pmatrix} \mathbf{A}' & \mathbf{S} \\ 0 & \mathbf{D}' \end{pmatrix}$

We provide the precise analysis of this inversion. It can be extended to *triangular* matrices, and we consider that **Invert** also computes their inverse.

Lemma 1. *Given an integral unitriangular matrix* $\mathbf{M}$ *of dimension* d*, with both* $\|\mathbf{M}\|, \|\mathbf{M}^{-1}\| \leqslant 2^p$ *and* $p \geqslant w + \log(d)$*,* ***Invert*** *returns a matrix* $\mathbf{M}'$ *such that* $\|\mathbf{M}' - \mathbf{M}^{-1}\| \leqslant 2^{-p}$ *with a running time of* $O\left(\frac{d^\omega p}{w} + d^2 p\right)$.

Proof. We set a working precision $p' = 1 + 3p + \lceil \log d \rceil = \mathrm{O}(p)$, and by induction on d, let us prove that

$$\|\mathbf{M}'^{-1} - \mathbf{M}\| \leqslant 2\sqrt{d}2^{-p'}.$$

The case $d = 1$ is straightforward, so that we now deal with inductive case $d > 1$. Let $\mathbf{E}$, $\delta\mathbf{A}$ and $\delta\mathbf{D}$ be matrices such that the top-right part of $\mathbf{M}'$ is $-\mathbf{A}'\mathbf{C}\mathbf{D}' + \mathbf{E}$, $\mathbf{A}'^{-1} = \mathbf{A} + \delta\mathbf{A}$, and $\mathbf{D}'^{-1} = \mathbf{D} + \delta\mathbf{D}$. Consequently, we

get: $\mathbf{M}'^{-1} - \mathbf{M} = \begin{pmatrix} \delta\mathbf{A} & -\mathbf{A}'^{-1}\mathbf{E}\mathbf{D}'^{-1} \\ 0 & \delta\mathbf{D} \end{pmatrix}$. We can guarantee that $\|\mathbf{E}\| \leqslant 2^{-p'-2p}$ with a computation with intermediary bitsize $\mathrm{O}(p')$. This leads to our intermediary result. Now let $\mathbf{M}'^{-1} = \mathbf{M} + \mathbf{F}$, so $\mathbf{M}' = (\mathbf{M}(\mathbf{Id} + \mathbf{M}^{-1}\mathbf{F}))^{-1} = (\mathbf{Id} + \mathbf{M}^{-1}\mathbf{F})^{-1}\mathbf{M}^{-1}$ and $\|\mathbf{M}' - \mathbf{M}^{-1}\| \leqslant \|\mathbf{M}^{-1}\|\|(\mathbf{Id} + \mathbf{M}^{-1}\mathbf{F})^{-1} - \mathbf{Id}\| \leqslant 2^{-p}$. The complexity comes from the matrix multiplication with words of size w.

3 Fast reduction of Euclidean lattices

This section is devoted to the description of our block recursive lattice reduction algorithm. In the following, let us fix a Euclidean lattice Λ of rank d, described by a basis collected in a rational matrix $\mathbf{B}$ in the canonical basis of $\mathbb{R}^d$. We generically denote by $\mathbf{R}$ the R-part of the QR-decomposition of this matrix. We recall that computations are conducted in floating-point arithmetic. However, for the sake of readability and ease of presentation, we defer the issue of the necessary precision to section 4.

We turn to a detailed breakdown of the essential parts of the algorithm. Each of the following subsections details and refers to the corresponding lines of algorithm 5, **Reduce**.

Algorithm 5 — Reduce

Parameter : Relaxation factor $\alpha, \varepsilon > 0$, number of rounds ρ, number of blocks D, $d' = d/D$ block size.
Input : Basis $\mathbf{B} \in \mathbb{Z}^{d\times d}$ of the lattice Λ
Output : A unimodular transformation $\mathbf{U} \in \mathbb{Z}^{d\times d}$, $\mathbf{UB}$ reduced.

1 **if** $d = 2$ **then return** **Schonhage**$(\mathbf{B})$
2 **for** $i = 1$ **to** ρ **do**
3 $\quad \mathbf{R} \leftarrow$ **Block-Cholesky**$(\mathbf{B}^T\mathbf{B})$
4 $\quad \mathbf{U}_i \leftarrow$ **Size-Reduce**$(\mathrm{Diag}(\mathbf{R})^{-1} \cdot \mathbf{R})$
5 $\quad (\mathbf{B}, \mathbf{R}) \leftarrow (\mathbf{B}, \mathbf{R}) \cdot \mathbf{U}_i$
6 $\quad$ **for** $j = 1 + (i \mod 2)$ **to** $D/2$ **by step of** 2 **do**
7 $\quad\quad V_1 \leftarrow \mathrm{vol}(\mathbf{R}[(j-1)d' + 1 : jd', (j-1)d' + 1 : jd'])$
8 $\quad\quad V_2 \leftarrow \mathrm{vol}(\mathbf{R}[jd' + 1 : (j+1)d' - 1, jd' + 1 : (j+1)d'])$
9 $\quad\quad$ **if** $V_1 \geqslant 2^{2(1+\varepsilon)\alpha(d')^2} V_2$ **then**
10 $\quad\quad\quad \mathbf{U}' \leftarrow$ **Reduce**$(\mathbf{R}[jd' : (j+2)d' - 1, jd' : (j+2)d' - 1])$
11 $\quad\quad\quad (\mathbf{U}_i, \mathbf{B}) \leftarrow (\mathbf{U}_i, \mathbf{B}) \cdot \mathrm{Diag}(\mathbf{Id}_{jd'}, \mathbf{U}', \mathbf{Id}_{d-3jd'})$
12 $\quad\quad$ **end if**
13 $\quad$ **end for**
14 **end for**
15 **return** $\prod_{i=1}^{\rho} \mathbf{U}_i$ // The product is computed from the end

3.1 Base case: plane lattices [Line 1]

As in all variants of the LLL algorithm, the *base case* of the reduction boils down to the two-dimensional case, usually handled by the celebrated Lagrange-Gauss reduction or some equivalent transformations. For instance, in the original LLL algorithm, truncated steps of Lagrange-Gauss reduction are conducted on two-dimensional projections of shape $\pi_i(b_i)\mathbb{Z} \oplus \pi_i(b_{i+1})\mathbb{Z}$.

For the sake of efficiency, we adapt Schönhage's algorithm [66], as in the algorithms of [41,35], to reduce these plane lattices. This algorithm is an extension to the bidimensional case of the so-called `half-GCD` algorithm [54], likewise that Gauss' algorithm is a bidimensional generalization of the classical Euclid's GCD. The original algorithm of Schönhage only deals with the reduction of binary quadratic forms but can be straightforwardly adapted to reduce lattices, as well as returning the corresponding unimodular transformation matrix. In the following, we denote by **Schonhage** this modified procedure. Its complexity is *quasilinear* in the size of its input (which is to be compared with the *quadratic* complexity of the classical Gauss reduction).

3.2 Outer iteration [Line 2]

To reduce the lattice Λ, we adopt an iterative strategy to progressively modify the basis: for $\rho > 0$ steps, a reduction pass over the current basis is performed, ρ being a parameter set to optimize the complexity of the whole algorithm while still ensuring the reduceness of the basis. We defer the choice of this constant for the moment. This global iterative scheme is similar to the *terminating* variants of the BKZ algorithm, for instance as in [36] or [53], where a polynomial number of rounds is fixed to reduce the input.

3.3 Orthogonalization via Block-Cholesky decomposition [Line 6]

Gram-Schmidt Orthogonalization is a preliminary step of every LLL-type algorithms, as it computes the so-called *Gram-Schmidt vectors* of the basis, which are ubiquitous in the definition of the reduction itself. On symmetric matrices as the Gram-matrix $\mathbf{B}^T\mathbf{B}$ of the basis, one computes the *Cholesky factorization*, which given a symmetric positive-definite matrix $\mathbf{G}$, the factorization asserts the existence (and unicity) of an upper triangular matrix $\mathbf{R}$ such that $\mathbf{G} = \mathbf{R}^T\mathbf{R}$ which is the some $\mathbf{R}$ in the QR decomposition of $\mathbf{B}$ since $\mathbf{G} = \mathbf{B}^T\mathbf{B} = \mathbf{R}^T\mathbf{R}$.

We use here a *recursive block variant* of the Cholesky factorization algorithm, allowing to compute a floating-point approximation of the matrix $\mathbf{R}$, whose running time is heuristically the cost of a matrix multiplication. It relies heavily on the **Invert** procedure introduced in Section 2.5.

Remark 1. Block computations of decompositions seems to be folklore in numerical algebra (see, for instance, the complete monograph of Higham [39] for multiple variants of block orthogonalization, such as modified Gram-Schmidt, Householder transformations, ...), but oddly, we were unable to find a proper reference to the block Cholesky factorization.

The decomposition is as follows, given as input a symmetric matrix $\mathbf{G}$. We start by block splitting it (with blocks of half size): $\mathbf{G} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{pmatrix}$, where $\mathbf{A}, \mathbf{C}$ are also symmetric. Its Schur complement $\mathbf{S} = \mathbf{C} - \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}$ is then also symmetric. Suppose that we know the factorization of the $\mathbf{A}$ and $\mathbf{S}$ in say: $\mathbf{A} = \mathbf{R}_A^T\mathbf{R}_A$ and $\mathbf{S} = \mathbf{R}_S^T\mathbf{R}_S$. Then, we set $\mathbf{R} = \begin{pmatrix} \mathbf{R}_A & \mathbf{R}_A^{-T}\mathbf{B} \\ 0 & \mathbf{R}_S \end{pmatrix}$. This matrix is indeed the Cholesky factorization of $\mathbf{G}$, as ensured by the following computation:

$$\begin{aligned} \mathbf{R}^T\mathbf{R} &= \begin{pmatrix} \mathbf{R}_A^T & 0 \\ \mathbf{B}^T\mathbf{R}_A^{-1} & \mathbf{R}_S^T \end{pmatrix} \cdot \begin{pmatrix} \mathbf{R}_A & \mathbf{R}_A^{-T}\mathbf{B} \\ 0 & \mathbf{R}_S \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{R}_A^T\mathbf{R}_A & \mathbf{R}_A^T\mathbf{R}_A^{-T}\mathbf{B} \\ \mathbf{B}^T\mathbf{R}_A^{-1}\mathbf{R}_A & \mathbf{B}^T\mathbf{R}_A^{-1}\mathbf{R}_A^{-T}\mathbf{B} + \mathbf{R}_S^T\mathbf{R}_S \end{pmatrix} = \begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{pmatrix}, \end{aligned}$$

since $\mathbf{B}^T\mathbf{R}_A^{-1}\mathbf{R}_A^{-T}\mathbf{B} + \mathbf{R}_S^T\mathbf{R}_S = \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B} + \mathbf{C} - \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B} = \mathbf{C}$ by definition of the Schur complement.

This derivation yields a direct recursive algorithm, whose base case corresponds to the unidimensional instance, i.e., $\mathbf{G} = (g)$, admitting the trivial decomposition $\mathbf{G} = (\sqrt{g})^T(\sqrt{g})$. This observation yields the procedure stated in pseudocode in algorithm 6 **Block-Cholesky**, computing a floating-point approximation of the Cholesky decomposition.

Algorithm 6 — Block-Cholesky

Input : A positive-definite symmetric matrix $\mathbf{G}$
Output : A fp-approx. of a triangular matrix $\mathbf{R}$ s.t. $\mathbf{R}^T\mathbf{R} = \mathbf{G}$

1 **if** $dim(\mathbf{G}) = 1$ **then return** $\sqrt{\mathbf{G}}$
2 $\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{C} \end{pmatrix} \leftarrow \mathbf{G}$ // with blocks of half-dimension
3 $\mathbf{R}_A \leftarrow$ **Block-Cholesky**$(\mathbf{A})$
4 $\mathbf{R}'_A \leftarrow$ **Invert**$(\mathbf{R}_A)$
5 $\mathbf{A}' \leftarrow \mathbf{R}'^T_A\mathbf{R}'_A$
6 $\mathbf{R}_S \leftarrow$ **Block-Cholesky**$(\mathbf{C} - \mathbf{B}^T\mathbf{A}'\mathbf{B})$
7 **return** $\begin{pmatrix} \mathbf{R}_A & \mathbf{R}'^T_A\mathbf{B} \\ 0 & \mathbf{R}_S \end{pmatrix}$

3.4 Size-reduction [Line 4]

As in the LLL algorithm, a size-reduction operation is conducted at each step of the reduction. It allows to control the size of the coefficients and ensures that the running time remains polynomial. However, in our case, we lean on a Seysen-like

reduction to perform this operation [67]. Our recursive procedure allows to size-reduce a unitriangular matrix (in our case, the matrix $\mathrm{Diag}(\mathbf{R})^{-1}\mathbf{R}$) in roughly the time of matrix multiplication.

We start from the classical observation that the usual size-reduction process is a discretized version of the iterative Gram-Schmidt process (which is a way of computing the QR-decomposition of a matrix). Over the triangular matrix $\mathbf{R}$, it corresponds to make iteratively the extra diagonal elements as close as possible to 0. However, instead of using an iterative process, we use a lattice reduction algorithm with block matrix operations.

Algorithm 7 — Size-Reduce

Input : A unitriangular matrix $\mathbf{T}$
Output : An integer unitriangular matrix $\mathbf{U}$, $\mathbf{TU}$ reduced

1 **if** $dim(\mathbf{T}) = 1$ **then return** (1)
2 $\begin{pmatrix}\mathbf{A} & \mathbf{C}\\ 0 & \mathbf{D}\end{pmatrix} \leftarrow \mathbf{R}$ // with half dimension
3 $\mathbf{U}_1 \leftarrow$ **Size-Reduce**$(\mathbf{A})$
4 $\mathbf{U}_2 \leftarrow$ **Size-Reduce**$(\mathbf{D})$
5 $\mathbf{A}' \leftarrow$ **Invert**$(\mathbf{AU}_1)$
6 $\mathbf{W} \leftarrow \lfloor \mathbf{A}'\mathbf{C}\mathbf{U}_2 \rceil$
7 **return** $\begin{pmatrix}\mathbf{U}_1 & -\mathbf{U}_1\mathbf{W}\\ 0 & \mathbf{U}_2\end{pmatrix}$

Let us start with a unitriangular matrix $\mathbf{R}$, split in block of half dimension: $\begin{pmatrix}\mathbf{A} & \mathbf{C}\\ 0 & \mathbf{D}\end{pmatrix}$. Assume for the moment that both unitriangular submatrices $\mathbf{A}$ and $\mathbf{D}$ are already size-reduced. Then, set

$$\mathbf{U} = \begin{pmatrix}\mathbf{Id} & -\lfloor \mathbf{A}^{-1}\mathbf{C} \rceil\\ 0 & \mathbf{Id}\end{pmatrix},$$

which is unimodular as its diagonal elements are all 1. Its action on $\mathbf{R}$ gives by elementary computation: $\mathbf{RU} = \begin{pmatrix}\mathbf{A} & \mathbf{C} - \mathbf{A}\lfloor \mathbf{A}^{-1}\mathbf{C} \rceil\\ 0 & \mathbf{D}\end{pmatrix}$ and the top-right part is of the same magnitude as $\mathbf{A}$. The inverse of $\mathbf{RU}$ is

$$\begin{pmatrix}\mathbf{A}^{-1} & -(\mathbf{A}^{-1}\mathbf{C} - \lfloor \mathbf{A}^{-1}\mathbf{C} \rceil)\mathbf{D}^{-1}\\ 0 & \mathbf{D}^{-1}\end{pmatrix}$$

and the top-right part is of the same magnitude as $\mathbf{D}^{-1}$, ensuring that the norm of this block is controlled. The translation of this process in pseudocode yields algorithm 7 **Size-reduce**. Note that this algorithm is presented as yielding the *transformation* matrix instead of the reduced matrix, to be consistent with the presentation of **Reduce** (see proof in Appendix A).

Theorem 2. *Given a d-dimensional unitriangular matrix $\mathbf{T}$ such that $\|\mathbf{T}\|$ and $\|\mathbf{T}^{-1}\| \leqslant 2^p$ and $p \geqslant w + \log(d)^2 d$, the algorithm **Size-Reduce** returns an integral unitriangular matrix $\mathbf{U}$ with $\|\mathbf{U}\| \leqslant 2^{O(p)}$ such that $\|\mathbf{TU}\|$, $\|(\mathbf{TU})^{-1}\| \leqslant d^{\lceil \log d \rceil}$ with a running time of $O\left(\frac{d^\omega p}{w} + d^2 p\right)$.*

3.5 Step reduction subroutine [Lines 3-13]

From parallel design of LLL... Let us now describe the step reduction pass, occuring once the size-reduction operation has been performed. As observed in section 2, the LLL algorithm reduces lattice reduction to the reduction of rank two lattices (more precisely, iteratively reduce *orthogonally projected* rank-2 sublattices). A first idea would be to use the same paradigm here and pass over the current basis in a sequence of reduction of projected planar lattices. However, on the contrary to the standard LLL or BKZ-2 algorithms, remark that we are not forced to proceed progressively along the basis, but that we can reduce $\lfloor d/2 \rfloor$ independent (non-overlapping) rank-2 lattices at each step, namely the $(\pi_{2i}(b_{2i}\mathbb{Z} \oplus b_{2i+1}\mathbb{Z}))_{1 \leqslant i \leqslant d/2}$ and then, $(\pi_{2i+1}(b_{2i+1}\mathbb{Z} \oplus b_{2i+2}\mathbb{Z}))_{0 \leqslant i \leqslant d/2}$. This design enables an efficient parallel implementation which reduces sublattices simultaneously, in the same way that the classical LLL algorithm can be parallelized [73,38]. This technique can also be thought of as a parallelized BKZ [53] or slide-reduction [1] with blocksize 2.

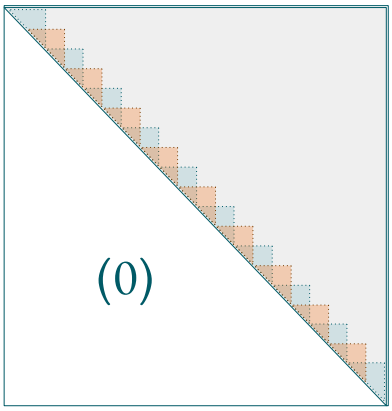

Fig. 1: Illustration of the parallel step reduction on the R-part of the QR-decomposition. Green 2×2 blocks are simultaneously reduced on odd steps and orange ones are reduced on even steps. This strategy is similar to [38].

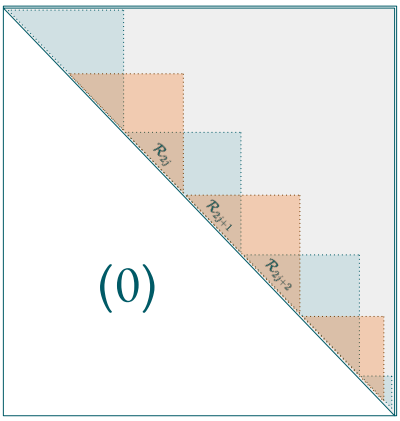

Fig. 2: The block process on the R-part of the basis. Green blocks are *recursively* reduced on odd steps and oranges one are reduced on even steps.

...to recursive block design A bottleneck with this strategy is that each round needs (at least) a matrix multiplication to be updated. Using a dynamical system analysis similar to [35], such a reduction would require ρ rounds to be a $\Omega(d^2)$ to ensure an LLL approximation factor. This implies a dependency in the running time which would be at *least* quartic in the dimension d. However, one can notice that each round only makes *local* modifications on the basis. As a result, we propose to use a small number D of blocks, and let a round recursively reduces consecutive pairs of blocks of dimension $\frac{d}{D}$. In this setting, the dynamical system analysis of [35] shows that a $\mathrm{O}\big(D^2 \log C\big)$ bound on the number of iterations ρ is now adequate.

Let us denote by R'_j the extracted submatrix $(R_{a,b})_{(j-1)d'<a,b\leqslant jd'}$, with $d' = d/D$. The lattice $\mathcal{R}'_j$ spanned by R'_j is the projection of $\Lambda_j = \bigoplus_{(j-1)d'<a\leqslant jd'} b_a\mathbb{Z}$ over the orthogonal space to the first $(j-1)d'$ vectors $(b_1, \ldots, b_{(j-1)d'-1})$. The step reduction subprocess simultaneously (and recursively) calls the reduction of all the *shifted* sublattices $\left(\mathcal{R}'_{2j} \oplus \mathcal{R}'_{2j+1}\right)_{1\leqslant j<\lceil \frac{D}{2} \rceil}$. Then the same is done on the sublattices $\left(\mathcal{R}'_{2j+1} \oplus \mathcal{R}'_{2j+2}\right)_{0\leqslant j<\lceil \frac{D}{2} \rceil}$ to enable the reduction of cross blocks. This step reduction is then restarted for ρ rounds as indicated in section 3.2.

On the volumetric Siegel condition. Remark the use of relaxation parameters $\varepsilon, \alpha > 0$, acting on the approximation factor of the reduction. As an avatar of the relaxation factor δ of LLL, they allow a slight tradeoff between the running time and the overall reduction quality. It is an equivalent of the Siegel condition between blocks: instead of recursively calling the reduction every time on the blocks $\mathcal{R}_{2j} \oplus \mathcal{R}_{2j+1}$, we only do it if the volume of the left block $\mathcal{R}_{2j}$ is sufficiently larger than the one of the right block $\mathcal{R}_{2j+1}$. We *do not* perform a recursive reduction if the slope between the blocks is *already small enough*.

In practice, these values are dependent on the depth of recursion to optimize the global running time. Section 4 addresses this technicality more thoroughly.

4 Complexity estimation and supporting experiments

We now turn to the fine-tuning of the implementation and describe some optimization tricks used. We backed up our choices by supporting experiments and eventually devise an *empirical estimate* of the bit-complexity of our algorithm.

4.1 Needed Precision

Since the implementation of the algorithm is done using floating-point arithmetic, we need to set a precision which is sufficient to handle the internal values during the computation. To do so, we set:

$$p = \log \frac{\max_i \mathbf{R}[i,i]}{\min_i \mathbf{R}[i,i]},$$

where the $\mathbf{R}[i,i]$ encodes the norm of the Gram-Schmidt vectors. As in floating-point variants of LLL [63,57,44,55], it is straightforward that a $O(p)$ is sufficient to handle the computation. However, the remaining question is the evolution of this quantity within the recursive calls. Indeed, as we have more and more recursive

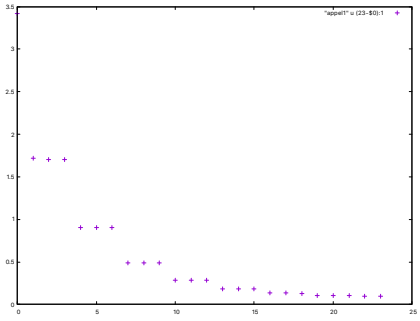

Fig. 3: Abscissa corresponds to the iteration time and ordinates corresponds to the value $\log(\max_i \mathbf{R}[i,i]/\min_i \mathbf{R}[i,i])$. As predicted by heuristic 1 the corresponding graph presents an exponential decay.

calls of the reduction algorithm on projected lattices of smaller dimensions, we would like to reduce them with a limited precision to get an overall faster reduction.

The analysis of [55] bounds the number of rounds, and reaches a complexity in $d^3C^{1+o(1)}$ with exact arithmetic (C is the log of the condition number), while the non-optimized algorithm [35] uses $\Omega(d^3 \log C)$ local reductions. Consequently, to decrease the complexity of the reduction, we have to reduce the precision in the local operations. The justification of this fact comes from that *in practice*, the values of $\mathbf{R}[i,i]$ decrease roughly exponentially in i both in the input and the output matrices. To define our heuristic, we rely on the notion of *slope* of the basis, which is the opposite of the slope of the linear regression of the log of the norm the Gram-Schmidt vectors. Under the Geometric Series Assumption (GSA), this corresponds to the usual geometric decay factor. Heuristic 1 says that we reduce the slope, i.e. the logarithm potential $\Pi(\mathbf{B}) = \sum_{i=1}^{d}(d-i)\log(\mathbf{R}_{i,i})$. We consider that we have access to an oracle which reduces with a slope parameter of α, namely $\mathbf{R}[i,i]/\mathbf{R}[i+1,i+1] \approx 2^{2\alpha}$. The matrix returned will have a slope parameter of $(1+\varepsilon)\alpha$ for $0 < \varepsilon < 1/2$.

Heuristic 1 *If ρ is even, and $\frac{\rho}{2}(2D-1) \geqslant D^3$, then the slope decreases exponentially quickly towards $(1+\varepsilon)\alpha$, with rate $1 - O\big(\frac{1}{D^2}\big)$.*

Remark 2. For a smaller ρ, we would have several leaves in the recursion tree, which would be negligible compared to d^3, making it unlikely to reduce the lattice by a significant amount. These values come from an heuristic analysis.

Figure 3 shows the evolution of the slope on a lattice of dimension 1024 where the phenomenon is observable. heuristic 1 has been tested on various types of lattices (Knapsacks, NTRU-like) in dimensions from 128 to 2048 without failing.

4.2 On the choice of the relaxation parameter ε and its relation to the global complexity

To finely tune our parameters, we need to estimate the decrease of the potential at each recursive call. Using heuristic 1 at any moment in the recursion, when called with a lattice of rank d and working precision $p = \log \frac{\max_i \mathbf{R}_{i,i}}{\min_i \mathbf{R}_{i,i}}$, such that

$$\prod_{i=1}^{d/2} \mathbf{R}_{i,i} > 2^{(1+\varepsilon)\alpha d^2/2} \prod_{i=d/2+1}^{d} \mathbf{R}_{i,i}, \qquad \text{(condition)}$$

the output basis has a log-potential reduced by at least $\Omega(d^2 p\varepsilon)$. Calling a recursive reduction only when the condition is fulfilled allows the callee to reduce the slope by a factor of roughly $1+\varepsilon$. If this is actually done, the potential is reduced by $\Omega(\varepsilon\big(\frac{d}{D}\big)^2 p')$ where p' is the precision used by the callee. The complexity of the callee, if not already in a leaf, and outside of its recursive calls is in

$$\mathrm{O}\left(D^2\left((d/D)^\omega(p'/w+1) + (d/D)^2 p' \frac{\log p'}{w}\right)\right).$$

Keeping only the first term and assuming $p' > w$, we get that the complexity per unit reduction in potential should behave in

$$\mathrm{O}\big(D^2(d/D)^{\omega-2}w^{-1}\varepsilon^{-1}\big).$$

This suggests minimizing D, so that we set $D = 2$ and $\rho = 6$; and also we deduce that most of the complexity is at low-depth. While the global complexity is minimized for $D = 2$, considering a larger D leads to better running time when using multithreading (higher number of blocks can be treated in parallel).

If we write d_i and ε_i for their values at depth i, we obtain that the global approximation factor is the one at the leaf multiplied by $\exp(\sum_i \varepsilon_i)$. Also, the main term in the complexity is proportional to $\sum_i d_i^{\omega-2}\varepsilon_i^{-1}$. Thus, we want ε_i proportional to $d_i^{1-\omega/2}$. If we want $\sum_i \varepsilon_i = \Theta(\delta)$, we get

$$\varepsilon_i = \delta(\omega(d) - 2)(d/d_i)^{1-\omega(d_i)/2}.$$

Summing the complexity at all depths, we see that the main term becomes:

$$\mathrm{O}\left(\frac{d^\omega C}{w(\omega-2)^2\delta}\right) \quad \text{for any} \quad \delta = \mathrm{O}\left(\frac{1}{\omega-2}\right).$$

4.3 Using small-dimension fast enumeration in the leaves

Since almost all the complexity concentrates at low recursive depth, we can allocate more time in the leaves of the recursion tree to improve the quality of the reduction without altering much the global complexity. In practice, this means stopping the recursion before reaching rank-2 sublattices and using a stronger reduction process than LLL on these (higher dimensional) leaves.

Some instances of stronger algorithms are the BKZ-type family, which are parameterized by a block size β, and have a complexity exponential in β [2]. This family includes Schnorr's original BKZ algorithm, Terminated-BKZ with less rounds [35], the self-dual BKZ [53] or pressed-BKZ [7]—which is particularly good for low β. If the dimension at the leaf d_l is significantly larger than $\beta \log \beta$, the famous Geometric Series Assumption states that the Gram-Schmidt norms of the reduced basis are well approximated by a geometric series of rate $2^{\Theta\left(\frac{\beta}{\log \beta}\right)}$.

We can assume that the basis was already reduced with a constant slope 2α, so that the potential will overall decrease only by $\mathrm{O}\big(d^3\big)$. At each leaf, we can use a constant ε_{l-1} and thus expect the log-potential to decrease by at least $\Omega(d_l^3 \frac{\log \beta}{\beta})$. The number of calls is therefore

$$\mathrm{O}\left(\frac{d^3\beta}{d_l^3 \log \beta}\right) = \mathrm{O}\left(\frac{d^3}{\beta^2}\right)$$

so we can choose any β smaller than $\Omega((\omega-2)\log C)$.

4.4 Complexity estimation

The sketch of analysis conducted previously let us conjecture that the complexity should have a dominant term in $d^\omega C$. We plot the *single-thread* running time on lattices with dimension $d = 2n$ generated by the columns of the following matrix

$$\begin{pmatrix} q\mathbf{Id}_n & \mathbf{A} \\ 0 & \mathbf{Id}_n \end{pmatrix}$$

with $\mathbf{A}$ sampled uniformly modulo q, and $C = \log(q) \approx n4^{k-1}$ for k from 0 (green) to 3 (blue). The slope of the reduced matrix is $2\alpha \approx 0.065$ (RHF$= 2^{0.032} = 1.02$).

To confirm this hypothesis, we perform a linear regression on the log/log data of the running time in function of the input dimension (ranging from 128 to 2048). The regression reveals a slope of 3.5, that is a complexity in $\mathrm{O}\big(d^{2.5}C\big)$ as C is linear in d. Given the noise generated by the inherent complexity of the program, its libraries, and the complex processor architecture, this experiment seems to validate our conjectural complexity. Each line corresponds to experiments made with matrices with bitsize bounded by (dimension K) [from green (lower) line with $K = 1/4$ to blue (upper) line $K = 16$]. We propose the following complexity for our algorithm, using the small BKZ-enumeration in the leaves.

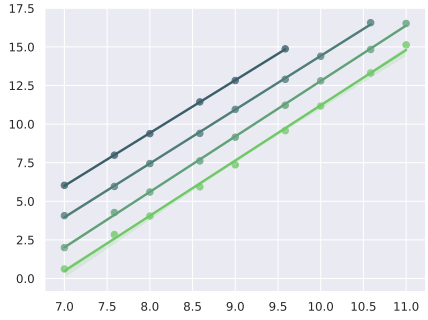

Fig. 4: Log-Log representation of the running time (in seconds) for increasing dimension, with constant C/d on each line.

Analysis 1 *Let* $\mathbf{A}$ *be a matrix of dimension* d *with integer entries, with* $\kappa(\mathbf{A}) \leqslant 2^C$ *such that* $C \geqslant d/\log d$. ***Reduce****(*A*) returns the transformation matrix to a basis of* $\mathbf{A}\mathbb{Z}^d$ *having its first vector of norm bounded by*

$$\max\Big(\sqrt{d}, 2^{O\left(d(\omega-2)\frac{\log\log C}{\log C}\right)}\Big)\,\mathrm{vol}\,A^{\frac{1}{d}}$$

Further, the heuristic *running time is*

$$O\bigg(d^\omega \cdot \frac{C}{(\omega-2)^2\log C} + d^2 C\log C + \frac{d^2 C}{(\omega-2)^2}\bigg).$$

Remark 3. The values come from a heuristic analysis that we do not develop.

- *In practice*, the entire basis is reduced at the end of the algorithm (as LLL algorithm gives a reduced basis with controlled decay of the Gram-Schmidt).

- When ω is bounded away from 2, and C is not extremely large ($C = 2^{o(d)}$), the complexity simplifies to $\mathrm{O}\left(d^\omega \cdot \frac{C}{\log C}\right)$.
- It is better to first reduce with a large δ (say $\min(\log(C/d), \frac{1}{\omega-2})$), and progressively reduce the slope by decreasing δ by a constant, so that the precision used is exponentially decreasing. For $C > d2^{1/(\omega-2)}$, we obtain a heuristic complexity of:

$$\mathrm{O}\left(d^\omega \cdot \frac{C}{(\omega-2)\log C} + d^2 C \log C\right).$$

- The dependency in the second term of the complexity (term in $d^2 C \log C$) comes as a direct consequence of the complexity of the Schonhäge algorithm.

The implementation mixes multiple machine representation as it needs to manage efficiently both large and small matrices, with a large range in bit-sizes. On the one hand, the "large matrices", e.g. with of dimension greater than 80 and of coefficient represented on few hundreds bits, are represented in the Fourier domain, that is to say by a collection of complex matrices, one for each evaluation point. The complex matrices are with double-precision floating-point coordinates. Large integers are transformed into polynomials, with between 14 and 16 bits per coefficient.

On the other hand, small matrices (dimension lower than 80) and with small bitsize are represented with an array of `MPFR` values [27]. A reduction of small matrix with at most 300 bits is computed by repeatedly reducing matrices with at most 39 bits, which are in turned reduced using blocks of dimension 12. These matrices of dimension 12 and with at most 20 bits are reduced with the quadratic L^2 [57] procedure.

Finally, matrices where p is small (around 30) and dimension up to 400 are treated in double precision, thanks to the use of the Householder QR-decomposition and the Seysen size-reduction.

5 Reduction of structured knapsack-like

In this section, we present a progressive strategy to *provably* speed-up the reduction of *almost* triangular matrices. Combined with the reduction of section 3, it gives a heuristic reduction process which estimated running time is essentially a $\mathrm{O}\left(d^{\omega-1}\frac{C}{\log C} + Cd\log d\right)$. The general idea is that a knapsack-like matrix of dimension d and with log condition number C can be reduced as quickly as a matrix of dimension d and condition number $2^{C/d}$. As this effect was already known for some algorithms like `fplll` [68, 1.5.3], noted [56], and used in [72], we aim at giving a general framework to encompass this observation.

5.1 Setting

Definition 4 (Almost triangular matrix). *A matrix* $\mathbf{B}$ *with* d *columns and* $O(d)$ *rows is said to be (asymptotically)* almost triangular *if* $\mathbf{B}_{i,j} = 0$ *for any* $i \geqslant O(j)$*, with a uniform constant.*

In order to analyze our strategy we also require the matrices to be well conditioned in the following sense:

Definition 5 (Knapsack-like matrix). *Let $\mathbf{B} \in \mathbf{Z}^{d\times d}$ be an almost triangular matrix and set $C \geqslant d^2$ such that $\lambda_k(\mathbf{C}) \leqslant 2^{C/k}$, for all matrices $\mathbf{C}$ whose columns are a subset of those of $\mathbf{B}$ of dimension k. Set $\mathbf{R}$ to be the R-factor of the QR-decomposition of $\mathbf{B}$. We say that $\mathbf{B}$ is $C-$*knapsack-like *if furthermore $\|\mathbf{R}^{-1}\| \leqslant 2^{C/d}$ and $|\mathbf{R}_{i,j}| \leqslant 2^{C/i}$ for all i, j.*

Remark 4. The conditions detailed in the previous definition seems apparently strong but such matrices are actually widespread, as corresponding to generic instances of so-called *knapsack* problems or searching integer relations. In practice, one can easily computationally check that these matrices, as well as Hermite Normal Form matrices with decreasing round pivots verify the assumptions with a reasonably small B.

5.2 Iterative reduction strategy

Hypothesis 1 *In all of the following suppose that we have access to a lattice reduction oracle* ***red-Oracle****, whose output is a transition matrix to a* Siegel-reduced *and* size-reduced *basis. Its running time on a $d \times d$ matrix of condition number bounded by C is denoted by $T(d, C)$.*

The progressive reduction consists in reducing the first $k = 2^i$ columns of $\mathbf{B}$, for all successive powers of two until reaching d. At step $1 \leqslant i \leqslant \lfloor \log d \rfloor$, we use the—now reduced—first k vectors to size-reduce the remaining columns before concatenating them to the current basis and pursuing the reduction. Hence, the bitsize of the whole matrix is reduced for each i before being actively used in the lattice reduction oracle **red-Oracle**.

Formally, define inductively a family of matrices $\mathbf{B}_i$ which represents the state of the matrix $\mathbf{B}$ computed in the i-th iteration.

Initialization: $\mathbf{B}_0 = \mathbf{B}$.

Induction: Let $i > 0$, and suppose that $\mathbf{B}_i$ is known. We start by reducing *only* the first $k = 2^i$ vectors using **red-Oracle** of $\mathbf{B}_i$ and denote by $\mathbf{B}'_i$ the result. Define $\mathbf{Q}_i\mathbf{R}_i$ to be the QR-decomposition of $\mathbf{B}'_i[:1, k]$. Then, remark that for any x being a column of $\mathbf{B}'_i$ not in the span of $\mathbf{B}'_i[:1, k]$, we can reduce its bitsize by replacing it by $x - \mathbf{B}'_j\lfloor \mathbf{R}_j^{-1}\mathbf{Q}_j^T x \rceil$ for increasing $1 \leqslant j \leqslant k$. Such a size-reduction can be computed on all the columns of $\mathbf{B}_i[k+1 : d]$ simultaneously using a single matrix multiplication and call $\mathbf{C}$ the corresponding vectors. Eventually set $\mathbf{B}_{i+1}$ to the concatenation $\big[\mathbf{B}'[:1, k] \mid \mathbf{C}\big]$.

The corresponding pseudo-code is given in algorithm 8.

Algorithm 8 — Reduction of Knapsack-like lattices

Parameter : Reduction oracle **red-Oracle**
Input : Matrix $\mathbf{B} \in \mathbb{Z}^{d\times d}$
Output : A reduced basis of $\mathbf{B}\mathbb{Z}$

$k, i \leftarrow 1, 0$
$\mathbf{B}_0 \leftarrow \mathbf{B}$
while $k < d$ **do**
  $k \leftarrow 2k$; $i \leftarrow i + 1$
  $\mathbf{B}_i[:1,k] \leftarrow \mathbf{B}_{i-1}[:1,k]\cdot$**red-Oracle**$(\mathbf{B}_{i-1}[:1,k])$
  $\mathbf{R}_i \leftarrow$ **Block-Cholesky**$(\mathbf{B}_i[:1,k]^T\mathbf{B}[:1,k])$
  $\mathbf{Q}_i \leftarrow \mathbf{B}_i[:1,k]\cdot\mathbf{R}_i^{-1}$
  $\mathbf{B}_{i+1}[1:k,k+1:d] \leftarrow \mathbf{B}_i[1:k,k+1:d] -$
    $\mathbf{B}_i\cdot\left\lfloor\mathbf{R}_i^{-1}\cdot\mathbf{Q}_i^T\cdot\mathbf{B}_i[1:k,k+1:d]\right\rceil$
end while

5.3 Complexity analysis

We now present the complexity analysis of the algorithm presented, under the hypothesis made on the lattice reduction oracle. For readability, we defer the proof to the full version. The following lemma entails that the condition number of the input of the reduction oracle is sufficiently small.

Lemma 2. *Let* $\mathbf{B}$ *a rank* d *almost triangular matrix which is* C*-knapsack-like. For any index* $0 \leqslant i \leqslant \lceil \log d \rceil$*, set* $\mathbf{B}_i$ *to be the matrix computed by the execution of algorithm 8 on* $\mathbf{B}$*. Denote by* $\mathbf{Q}_i\mathbf{R}_i = \mathbf{B}_i[,1:2^i]$ *the QR-decomposition of the matrix of* 2^i *first columns of* $\mathbf{B}_i$*. We get* $\|\mathbf{R}_i\|, \|\mathbf{R}_i^{-1}\| = 2^{O\left(d+C2^{-i}\right)}$ *for all* i*.*

From this we have:

Theorem 3. *Let* $\mathbf{B}$ *a rank* d *almost triangular matrix which is* C*-well conditioned,* $C \geqslant d^2$*. We can Siegel-reduce it in time*

$$O\left(\sum_{i=1}^{\log d} T(2^i, O(C2^{-i})) + \frac{d^{\omega-1}}{\omega-2}\cdot\frac{C}{\log C} + dC\log d\right).$$

Remark 5.
- One can use such a procedure to quickly search a putative minimal polynomial; the knapsack-like condition is however not guaranteed.
- The setting of theorem 3 includes both modular and integer knapsacks.
- Assuming algorithm of section 3 has heuristically the right properties (which is the case in all of our extensive experiments), the complexity of the reduction of knapsack like matrices then becomes:

$$\mathrm{O}\left(\frac{d^{\omega-1}}{(\omega-2)^2}\cdot\frac{C}{\log C} + dC\log C\right).$$

6 Applications

Lattice reduction algorithms have numerous applications in mathematics and computer science. We survey here the impact of the implementation our algorithm, starting with cryptanalysis. In particular, we can reduce lattices of dimension in the thousands and with millions of bits. We recall that the Gram-Schmidt norms in the output basis are expected to decrease geometrically with rate $2^{2\alpha}$ so that the Hermite factor in dimension d is $2^{\alpha d}$.

For all the presented experiments, we use an Intel CPU E5-2695 v4 with 18 cores running at 2.10GHz processors; and 768 GiB of RAM. Some SSD swap was slightly used in the largest computation. For comparison with older timings, we used a machine with an Intel i7-8650U with 4 cores at 1.9 GHz. The program was compiled with Intel's libraries and compiler with the standard `-Ofast` low-level optimization flag.

6.1 Comparison with state of the art

We start this section by a comparison with the state-of-the-art implementation of `fplll`. Its complexity is $\mathrm{O}\left(d^4B^2\right)$ in the general case, and its heuristic complexity[4] is $\mathrm{O}\left(d^2B^2\right)$ for knapsack matrices, as reported in [68, 1.5.3]. When $d \geqslant 220$, its practical efficiency drops sharply due to the need of multiprecision computations. The following table presents a running time comparison with `fplll`, in *single-threaded* mode, on classical types of lattices namely knapsack and NTRU matrices. On the all instances, our implementation is sensibly faster than `fplll`.

Type	Dimension d	Bitsize B	fplll	This work
Knapsack	128	100 000	88 min	6 min
	256	10 000	134 min	4 min
	384	10 000	388 min	13 min
NTRU	256	80	24 min	3 min
	384	70	431 min	10 min
	512	70	1392 min	33 min

6.2 Fully Homomorphic Encryption over the integers

FHE scheme was first designed by Gentry [33] using number theoretical tools in 2009. Soon after, an equivalent system was presented, using only integer arithmetic [70], and is based on a distant relative [17] of the celebrated Learning

[4] This estimation comes from the observation that dB swaps are performed, each taking d^2 operations on B/d bits. There are B/d reduction steps for each new vector, each takes d^2 operations on B bits.

With Error (LWE) problem in dimension one. More precisely, given an integer secret $|s| \leqslant 2^\eta$ (typically a prime), this problem aims at retrieving s from given samples x_i of the form $a_i s + e_i$ where $0 \leqslant a_i \leqslant 2^\gamma/|s|$ and $|e_i| \leqslant 2^\rho$ are sampled uniformly and independently. The parameters verify $\gamma \gg \eta \gg \rho$.

A natural lattice reduction attack consists in collecting d samples $x = (x_i)_{1 \leqslant i \leqslant d}$ and building the matrix $\mathbf{X} = \begin{pmatrix} x_1, \ldots, x_d \\ \mathbf{Id}_d \end{pmatrix}$. The volume of the lattice $\mathcal{X}$ spanned by the columns of $\mathbf{X}$ is $\sqrt{1 + \sum_{i=1}^d x_i^2} \approx 2^\gamma$. Hence, lattice reduction with root Hermite factor 2^α can be used to construct a non-zero vector $y \in \mathbb{Z}^d$ such that $\|y\|, |\langle x, y\rangle| \leqslant 2^{\gamma/d+\alpha d}$. Indeed, any vector in this lattice is of the form $(\langle x, y\rangle, y)$, so that its squared norm is the sum of two contributions: $\|y\|^2 + |\langle x, y\rangle|^2$. It now suffices to remark that norm of a vector found by reduction is smaller than the normalized covolume $2^{\gamma/d}$ times the root Hermite factor $2^{\alpha d}$.

By plugging back the definition of the (x_i), we have $\langle x, y\rangle = s\langle a, y\rangle + \langle e, y\rangle$ where $a = (a_1, \ldots, a_d)$, $e = (e_1, \ldots, e_d)$. Assuming $2^{\gamma/d+\alpha d} \leqslant 2^{\eta-\rho}/\sqrt{d}$, the Cauchy-Schwarz inequality implies that $\langle a, y\rangle = 0$. This is enough to break the scheme; if the $(d-1)$ first vectors of the basis have this length, then the last one must be proportional to a (and is $\pm a$ if the entries are coprime). The first $d-1$ first vectors are orthogonal to a and are independent, so the last one must be proportional to a since a is in the lattice orthogonal to these vectors. The optimal d – for maximizing α – is therefore close to $\sqrt{\gamma/\alpha}$, leading to the condition $\alpha \leqslant \frac{(\eta-\rho)^2}{4\gamma}$.

A part of the original paper [70] considers security against polynomial-time adversaries, so that they obtain the condition $\gamma = \boldsymbol{\omega}(\eta^2 \log \lambda)$ for a "security parameter" of λ. Another part of the original paper [70, Section 6.3], and almost all follow-ups [24,25,22,15,21,23], consider however security against adversaries able to do 2^λ operations. However, the condition was copied without change, which possibly explains why a large α was chosen in several implementations[5].

As the lattice reduction algorithm can easily reach $\alpha = 0.04$, this means we can use a smaller d, close to $\frac{\gamma}{\eta-\rho}$ for many instances than the d in the table, which is the dimension where α is maximal. In the instance where $\gamma = 1.02 \cdot 10^6$ and $\eta - \rho = 376$, we used $d = 3600$ so that $\alpha = 0.024$ was needed, and we used a pressed-BKZ-19 in the leaves. This choice was made due to memory concerns.

While the large problems are clearly quite difficult, even the largest instances of the table seem to be within range of (motivated!) academic attackers, with terabytes of SSD memory and perhaps around 2^{65} flop.

6.3 Overstretched NTRU

It is well-known since the work of Albrecht *et al.* [4], Cheon *et al.* [16] and Kirchner and Fouque [43] that an NTRU scheme with a very large modulus q compared to the dimension of the lattice is prone to attacks. However, these cases often happen in NTRU-based Homomorphic encryption schemes such as

[5] Surprisingly, many $\log \lambda$ were "rounded up" to λ in the parameter choices.

Scheme	λ	$\gamma/10^6$	$\eta - \rho$	α	d	Algorithm	Running time
[24]	42	0.16	1072	1.8	299	LLL	5 min
	52	0.86	1608	0.75	1070	LLL	55 min
	62	4.2	2144	0.273	3918	LLL	1030 min
	72	19	2613	0.089	14543	LLL	-
[25]	42	0.061	322	0.43	379	LLL	3 min
	52	0.27	370	0.12	1460	LLL	29 min
	62	1.02	376	0.0347	5426	LLL	27 hours
	72	2.2	420	0.02	10476	BKZ-20	-
[22]	42	0.27	929	0.8	290	LLL	13 min
	52	1.1	924	0.2	2380	LLL	176 min
	62	4.2	919	0.051	9140	LLL	-
[15]	52	0.9	1517	0.64	1186	LLL	74 min
	62	4.6	2072	0.24	4440	LLL	1382 min
	72	21	2627	0.082	15988	LLL	-
[21]	52	1	1797	0.82	1104	LLL	67 min
	62	4.26	1987	0.24	4288	LLL	1322 min
	72	18.7	2189	0.0643	17043	LLL	-
	80	63.7	2353	0.0218	54117	BKZ-20	-

Table 1: Examples of schemes attacked, and corresponding reduction algorithm required to break.

YASHE [12] or LTV schemes. In 2019, a homomorphic scheme has been proposed by Genise *et al.* [31] with a similar variant of this problem, hoping that the overstretched NTRU only works in algebraic setting using ring of polynomials. Some parameters proposed for performances evaluations have been broken in [49] also showing that the assumptions used is flawed. Here, we break comparable parameters showing that the proposed parameters only achieve a low security level. In [59], Pataki and Tural showed that the volume of any r-dimensional sublattice L' of a lattice L is larger than the product of the r smaller Gram-Schmidt. Kirchner and Fouque combined this result with the fact that in any $2d$-dimensional NTRU lattices, there is a sublattice of dimension d and volume roughly the size of secret-key to the power d, one can deduce that if the volume

of the secret key sublattice is of size about the product of the d smaller Gram-Schmidt, it is possible to recover the secret key.

The optimal d is around $\frac{\log(q)}{4\alpha}$, which corresponds to a volume close to $2^{\log(q)^2/16\alpha}$. The scheme of [31] chooses entries in $\mathbf{F}, \mathbf{G}$ as integer Gaussians of standard deviation $\sigma = \sqrt{r/\pi}$ where r is the dimension of their lattice. We can restrict the lattice reduction to the middle $2d$ square matrix, but the volume is conserved. A more precise estimate consists in using the volume of the sublattice, projected orthogonally to the first r vectors of the reduced basis [43]. We expect the i-th Gram-Schmidt norm of the projected basis to be around $\sqrt{r+1-i}\sigma$, so that the volume can be computed with Stirling's formula $\left(\prod_{i=1}^{r}(r+1-i)\sigma^2\right)^{1/2} \approx \left(\frac{\sqrt{r}\sigma}{\sqrt{\mathrm{e}}}\right)^r$. Overall we obtain $2^{\log(q)^2/16\alpha} \approx \left(\frac{r}{\sqrt{\pi \mathrm{e}}}\right)^r$, from which we can find the α required. The first one necessitates roughly 2^{20} calls

Dimension r	$\log q$	α	Effective dimension	Algorithm	Time
1024	42	0.01274	1648	BKZ-101	-
4096	111	0.01799	3086	BKZ-25	233 hours
32768	883	0.1105	2560	LLL	263 min

Table 2: Experiments for overstretched NTRU problems. Dimension is the actual dimension of the problem, and effective dimension refers to the dimension required in practice to mount the attack.

to a SVP in dimension 101, and each call currently needs 2^{11} *core*-seconds [6], this translates into a year of computation on our machine. Alternatively, each call can be computed in 2^2 seconds with a GPU [26]. A pressed-BKZ of dimension 29 was used for the second one.

6.4 Miscellaneous

Integral relations Another use of lattice reduction is the discovery small linear integer relation between reals. It actually corresponds to the setting of section 6.2, where 2^η corresponds the norm of the relation, and γ the precision used to represent the reals. Then clearly, $\gamma \approx d\eta + d^2\alpha$ is enough to perform a search by reduction. In 2001, Bailey and Broadhurst believed [8] that their computation with $\gamma \approx 166000$ and $d = 110$ was the largest performed. It took 44 hours, on 32 CPUs of a Cray-T3E (300 MHz). We report this takes 5 minutes on a laptop, or 600 times fewer cycles. As the task is identical (for large α) to breaking the integer homomorphic schemes, the running time for bigger examples can be found in the previous subsections.

Univariate polynomial factorization Yet another application is factoring univariate polynomials [71,10] over the integers. The first step is to factor modulo some prime, and the number of factors n is the dimension of the modular vectorial knapsack we have to solve, namely we have to find very short vectors in the lattice generated by $\begin{pmatrix} q\mathbf{Id}_r & \mathbf{A} \\ 0 & \mathbf{Id}_n \end{pmatrix}$. The precision q, and number of coordinates r can essentially be freely chosen. For random polynomials, n is typically very small (e.g. logarithmic) and lattice reduction is not the bottleneck; but it can be as large as half the degree. Our choice is to take r small, say $n/\log n$, and then $r \log q \approx \alpha n^2$ allows (heuristically) to obtain the last Gram-Schmidt norms larger than n^2. Then, this restricts the solutions of the knapsack – known to be shorter than this – to the first few vectors of the reduced basis. At this point, one can recover the factors, and *prove* that they are irreducible. Taking $n = 256$, we get a solution in two minutes on one core of our laptop instead of ten with a 1 GHz Athlon [9]; for $n = 512$ it takes 25 minutes instead of 500. For ω bounded away from 2, with $\alpha = \mathrm{O}\left(\frac{\log\log n}{\log n}\right)$ the heuristic asymptotical complexity is

$$\mathrm{O}\left(\frac{n^{\omega+1}\log\log n}{\log^2 n}\right).$$

7 Conclusion and open questions

In this work, we introduced a recursive lattice reduction algorithm, whose heuristic complexity is equivalent to a few matrix multiplications. This algorithm and the heuristics used to complete the complexity analysis have been thoroughly tested and applied to reduce lattices of very large dimension. The implementation takes advantage of fast matrix multiplications, and fast Fourier transforms.

This work raises several questions. First of all, the analysis we are making is so far heuristic and empirical. It is possible to get a provable result by mitigating the complexity, in particular it seems difficult to be able to formally prove the heuristic on the decrease of the needed precision, even though this fact is easily checkable in practice. Reaching a *provable bound* in $d^\omega C$ is an open and interesting problem, and our algorithm is a first step in this direction.

References

1. D. Aggarwal, J. Li, P. Q. Nguyen, and N. Stephens-Davidowitz. Slide reduction, revisited - filling the gaps in SVP approximation. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*.
2. A. Becker, L. Ducas, N. Gama, and T. Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. *SODA 2016*, pages 10–24.
3. M. Albrecht, S. Bai, D. Cadé, X. Pujol, and D. Stehlé. fplll-5.0, a floating-point LLL implementation. Available at http://perso.ens-lyon.fr/damien.stehle, 2017.
4. M. R. Albrecht, S. Bai, and L. Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In

M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178.
5. M. R. Albrecht, S. Bai, P.-A. Fouque, P. Kirchner, D. Stehlé, and W. Wen. Faster enumeration-based lattice reduction: Root hermite factor $k^{1/(2k)}$ time $k^{k/8+o(k)}$. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 186–212.
6. M. R. Albrecht, L. Ducas, G. Herold, E. Kirshanova, E. W. Postlethwaite, and M. Stevens. The General Sieve Kernel and New Records in Lattice Reduction. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 717–746.
7. S. Bai, D. Stehlé, and W. Wen. Measuring, simulating and exploiting the head concavity phenomenon in BKZ. In T. Peyrin and S. Galbraith, editors, *ASIACRYPT 2018, Part I*, volume 11272 of *LNCS*, pages 369–404.
8. D. Bailey and D. Broadhurst. Parallel integer relation detection: Techniques and applications. *Mathematics of Computation*, 70(236):1719–1736, 2001.
9. K. Belabas. A relative van Hoeij algorithm over number fields. *Journal of Symbolic Computation*, 37(5):641–668, 2004.
10. K. Belabas, M. van Hoeij, J. Klüners, and A. Steel. Factoring polynomials over global fields. *Journal de théorie des nombres de Bordeaux*, 21(1):15–39, 2009.
11. J. Bi, J.-S. Coron, J.-C. Faugère, P. Q. Nguyen, G. Renault, and R. Zeitoun. Rounding and chaining LLL: Finding faster small roots of univariate polynomial congruences. In H. Krawczyk, editor *PKC 2014*, volume 8383 *LNCS*.
12. J. W. Bos, K. E. Lauter, J. Loftus, and M. Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In *IMACC*, pages 45–64, 2013.
13. J. A. Buchmann. Reducing lattice bases by means of approximations. In L. M. Adleman and M. A. Huang, editors, *ANTS-I*, volume 877 of *LNCS*, pages 160–168. 1994.
14. Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In D. H. Lee and X. Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 1–20.
15. J. H. Cheon, J.-S. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 315–335.
16. J. H. Cheon, M. Hhan, and C. Lee. Cryptanalysis of middle lattice on the over-stretched NTRU problem for general modulus polynomial. Cryptology ePrint Archive, Report 2017/484, 2017. http://eprint.iacr.org/2017/484.
17. J. H. Cheon and D. Stehlé. Fully homomorphic encryption over the integers revisited. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 513–536, 2015.
18. H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1993.
19. D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.
20. D. Coppersmith and A. Shamir. Lattice attacks on NTRU. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 52–61. May 1997.
21. J.-S. Coron, T. Lepoint, and M. Tibouchi. Practical multilinear maps over the integers. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493, 2013.
22. J.-S. Coron, T. Lepoint, and M. Tibouchi. Scale-invariant fully homomorphic encryption over the integers. In H. Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 311–328, 2014.

23. J.-S. Coron, T. Lepoint, and M. Tibouchi. New multilinear maps over the integers. In R. Gennaro and M. J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 267–286, 2015.
24. J.-S. Coron, A. Mandal, D. Naccache, and M. Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 487–504, 2011.
25. J.-S. Coron, D. Naccache, and M. Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 446–464, 2012.
26. L. Ducas, M. Stevens, and W. van Woerden. Advanced lattice sieving on GPUs, with tensor cores. Cryptology ePrint Archive, Report 2021/141, 2021. https://eprint.iacr.org/2021/141.
27. L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software (TOMS)*, 33(2):13, 2007.
28. A. M. Frieze, R. Kannan, and J. C. Lagarias. Linear congruential generators do not produce random sequences. In *25th FOCS*, pages 480–484. IEEE Computer Society Press.
29. N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In R. E. Ladner and C. Dwork, editors, *40th ACM STOC*, pages 207–216.
30. N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 257–278.
31. N. Genise, C. Gentry, S. Halevi, B. Li, and D. Micciancio. Homomorphic encryption for finite automata. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 473–502.
32. C. Gentry. Key recovery and message attacks on NTRU-composite. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 182–194.
33. C. Gentry. Fully homomorphic encryption using ideal lattices. In M. Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press.
34. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition, 1996.
35. G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In P. Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 447–464.
36. G. Hanrot, X. Pujol, and D. Stehlé. Terminating BKZ. Cryptology ePrint Archive, Report 2011/198, 2011. http://eprint.iacr.org/2011/198.
37. J. Håstad, B. Just, J. C. Lagarias, and C. Schnorr. Polynomial time algorithms for finding integer relations among real numbers. *SIAM J. Comput.*, 18(5):859–881, 1989.
38. C. Heckler and L. Thiele. Complexity analysis of a parallel lattice basis reduction algorithm. *SIAM Journal on Computing*, 27(5):1295–1302, 1998.
39. N. J. Higham. *Accuracy and stability of numerical algorithms*, volume 80. Siam, 2002.
40. A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185.
41. P. Kirchner, T. Espitau, and P.-A. Fouque. Fast reduction of algebraic lattices over cyclotomic fields. In D. Micciancio and T. Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 155–185.

42. P. Kirchner, T. Espitau and P.-A. Fouque. Algebraic and Euclidean Lattices: Optimal Lattice Reduction and Beyond. Cryptology ePrint Archive, Report 2019/1436, 2019. https://eprint.iacr.org/2019/1436.
43. P. Kirchner and P.-A. Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In J.-S. Coron and J. B. Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 3–26.
44. H. Koy and C. Schnorr. Segment LLL-reduction of lattice bases. In J. H. Silverman, editor, *Cryptography and Lattices, International Conference, CaLC 2001*, volume 2146 of *LNCS*, pages 67–80. 2001.
45. J. C. Lagarias. The computational complexity of simultaneous diophantine approximation problems. In *23rd FOCS*, pages 32–39. IEEE Comp. Soc. Press. 1982.
46. J. C. Lagarias. Knapsack public key cryptosystems and diophantine approximation. In D. Chaum, editor, *CRYPTO'83*, pages 3–23. Plenum Press, New York, USA.
47. J. C. Lagarias, H. W. L. Jr., and C. Schnorr. Korkin-Zolotarev bases and successive minima of a lattice and its reciprocal lattice. *Comb.*, 10(4):333–348, 1990.
48. J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. In *24th FOCS*, pages 1–10. IEEE Computer Society Press, Nov. 1983.
49. C. Lee and A. Wallet. Lattice analysis on MiNTRU problem. Cryptology ePrint Archive, Report 2020/230, 2020. https://eprint.iacr.org/2020/230.
50. A. K. Lenstra, H. W. J. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.
51. G. Maze. Some inequalities related to the Seysen measure of a lattice, 2010.
52. K. Mehlhorn and P. Sanders. *Algorithms and data structures: The basic toolbox*. Springer Science & Business Media, 2008.
53. D. Micciancio and M. Walter. Practical, predictable lattice basis reduction. In M. Fischlin and J.-S. Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 820–849.
54. N. Möller. On Schönhage's Algorithm and Subquadratic Integer GCD Computation. *Mathematics of Computation*, 77(261):589–607, 2008.
55. A. Neumaier and D. Stehlé. Faster LLL-type Reduction of Lattice Bases. In *ISSAC*, pages 373–380, 2016.
56. P. Q. Nguyen and D. Stehlé. LLL on the average. In F. Hess, S. Pauli, and M. E. Pohst, editors, *ANTS*, volume 4076 of *LNCS*, pages 238–256. 2006.
57. P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, 2009.
58. A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. In *43rd STOC*, pages 403–412. ACM, 2011.
59. G. Pataki and M. Tural. Lattice determinants in reduced bases. Arxiv:0804.4014, 2008. https://arxiv.org/abs/0804.4014.
60. A. Pellet-Mary, G. Hanrot, and D. Stehlé. Approx-SVP in ideal lattices with pre-processing. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 685–716.
61. Saruchi, I. Morel, D. Stehlé, and G. Villard. LLL reducing with the most significant bits. In K. Nabeshima, K. Nagasaka, F. Winkler, and Á. Szántó, editors, *ISSAC*, pages 367–374. ACM, 2014.
62. C. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.
63. C. Schnorr. A more efficient algorithm for lattice basis reduction. *J. Algorithms*, 9(1):47–62, 1988.
64. C. Schnorr. Block reduced lattice bases and successive minima. *Comb. Probab. Comput.*, 3:507–522, 1994.

65. A. Schönhage. Factorization of univariate integer polynomials by diophantine aproximation and an improved basis reduction algorithm. In J. Paredaens, editor, *ICALP*, volume 172 of *LNCS*, pages 436–447. 1984.
66. A. Schönhage. Fast Reduction and Composition of Binary Quadratic Forms. In *ISSAC*, pages 128–133, ACM, 1991.
67. M. Seysen. Simultaneous reduction of a lattice basis and its reciprocal basis. *Combinatorica*, 13(3):363–376, 1993.
68. D. Stehlé. Floating-point LLL: theoretical and practical aspects. In *The LLL Algorithm*, pages 179–213. 2009.
69. J. Stern. Secret linear congruential generators are not cryptographically secure. In *28th FOCS*, pages 421–426. IEEE Computer Society Press, Oct. 1987.
70. M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43.
71. M. Van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number theory*, 95(2):167–189, 2002.
72. M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. In *LATIN*, volume 6034 of *LNCS*, pages 539–553. 2010.
73. G. Villard. Parallel Lattice Basis Reduction. In *ISSAC*, pages 269–277, 1992.

A Proof of Theorem 2

Proof. We use a precision $p' = \mathrm{O}\big(p + \log(d)^2\big) = \mathrm{O}(p)$. We prove by induction on d that $\|\mathbf{TU}\|, \|(\mathbf{TU})^{-1}\| \leqslant d^{\lceil \log d \rceil}$. Initialization is clear, so that we now assume that $d > 1$. We have by direct computation that $\mathbf{TU}$ is

$$\begin{pmatrix} \mathbf{AU}_1 & \mathbf{CU}_2 - \mathbf{AU}_1\mathbf{W} \\ 0 & \mathbf{DU}_2 \end{pmatrix}.$$

The top-right matrix is $\mathbf{AU}_1((\mathbf{AU}_1)^{-1}\mathbf{CU}_2 - \mathbf{W})$ and we have, by setting that $\mathbf{A}' - (\mathbf{AU}_1)^{-1} = \delta\mathbf{A}$:

$$\|(\mathbf{AU}_1)^{-1}\mathbf{CU}_2 - \mathbf{W}\| \leqslant \|\delta\mathbf{ACU}_2\| + \|\mathbf{A}'\mathbf{CU}_2 - \mathbf{W}\|.$$

The first term is bounded by $2^{\mathrm{O}(p)}\|\delta\mathbf{A}\|$ and the second by $2d/3$. We choose the precision so that the first term is at most $1/3$ and the result follows directly, as $\|\mathbf{AU}_1\|, \|\mathbf{CU}_2\| \leqslant d^{\lceil \log d \rceil - 1}$.

Next, the matrix $(\mathbf{TU})^{-1}$ is equal to

$$\begin{pmatrix} (\mathbf{AU}_1)^{-1} & -(\mathbf{AU}_1)^{-1}(\mathbf{CU}_2 - \mathbf{AU}_1\mathbf{W})(\mathbf{DU}_2)^{-1} \\ 0 & (\mathbf{DU}_2)^{-1} \end{pmatrix}.$$

The top-right matrix is $((\mathbf{AU}_1)^{-1}\mathbf{CU}_2 - \mathbf{W})(\mathbf{DU}_2)^{-1}$. The first term was already bounded above by d, and $\|(\mathbf{DU}_2)^{-1}\| \leqslant d^{\lceil \log d \rceil - 1}$ and this gives the result.

Finally, we have $\|\mathbf{U}\| = \|\mathbf{T}^{-1}\mathbf{TU}\| \leqslant \|\mathbf{T}^{-1}\|\|\mathbf{TU}\| \leqslant 2^p d^{\lceil \log d \rceil}$.

Remark 6. It is mandatory to have $\mathbf{T}$ well-conditioned if we want a $\mathbf{U}$ which is not much larger than $\mathbf{T}$. This is also true for other variants of LLL (including fplll): outputting the transition matrix may lead to a slow-down by a factor of d.