Adaptive precision LLL and Potential-LLL reductions with Interval arithmetic

Thomas Espitau[1,2] and Antoine Joux[2,3]

[1] Ecole Normale Supérieure de Cachan,
[2] Sorbonne Universités, UPMC Univ Paris 06, LIP6,
[3] Chaire de Cryptologie de la Fondation de l'UPMC.
Thomas.Espitau@lip6.fr, antoine.joux@m4x.org

Abstract. Lattice reduction is fundamental in computational number theory and in computer science, especially in cryptography. The celebrated Lenstra–Lenstra–Lovász reduction algorithm (called LLL or L^3) has been improved in many ways through the past decades and remains one of the central tool for reducing lattice basis. In particular, its floating-point variants — where the long-integer arithmetic required by Gram–Schmidt orthogonalization is replaced by floating-point arithmetic — are now the fastest known. Yet, the running time of these floating-point versions is mostly determined by the precision needed to perform sound computations: theoretical lower bounds are large whereas the precision actually needed on average is much lower. In this article, we present an adaptive precision version of LLL and one of its variant Potential-LLL. In these algorithms, floating-point arithmetic is replaced by Interval Arithmetic. The certification property of interval arithmetic enables runtime detection of precision defects in numerical computations and accordingly, makes it possible to run the reduction algorithms with guaranteed nearly optimal precision. As such, these adaptive reduction algorithms run faster than the state-of-the-art implementations, while still being provable.

1 Introduction

Lattices are defined as additive discrete subgroups of $\mathbb{R}^n$, i.e. the integer span $L(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d}) = \bigoplus_{i=1}^{d} \mathbb{Z}\boldsymbol{b_i}$ of a linearly independent family of vectors $\boldsymbol{b_1}, \ldots, \boldsymbol{b_d}$ in $\mathbb{R}^n$. Such a family is called a *basis* of the lattice, and is not unique. Nevertheless, all the bases of a given lattice have the same number of elements, d, which is called the *dimension* of the lattice. Among the infinite number of different bases of a n-dimensional lattice with $n \geq 2$, some have interesting properties, such as having reasonably small vectors and low orthogonality defect. Finding such reduced bases is the goal of *lattice reduction* theory and has been crucial in several fields of computer science and mathematics, especially in cryptology. For instance, lattices have been used to break many public-key cryptosystems in the last decades, including knapsack cryptosystems [11] or RSA in specific settings thanks to Coppersmith's method [3].

The problem of finding good bases goes back to the early works of Lagrange and Gauss, for dimension two lattices, and the introduction of the Gauss — even though first introduced by Lagrange — algorithm. This procedure can be seen as a 2-dimensional extension of the well-known Euclid algorithm for computing the greatest common divisor of two integers. In 1850, Hermite published the first reduction algorithm for arbitrary dimension[4]. A century later, in 1982, Lenstra, Lenstra and Lovász designed the *LLL algorithm* [12], with the polynomial factorization problem as an application, after the celebrated work of Lenstra on integer programming [13]. This algorithm is a milestone in the history of lattice reduction algorithms, being the first algorithm whose running-time is polynomial for arbitrary dimension. This work has been improved in multiple ways by Kaltofen [9], Schnorr [20], and Gama and Nguyen [5] among others, decreasing the time complexity or improving the quality of the reduction.

Interval arithmetic is a representation of reals by intervals — whose endpoints are floating-point numbers — that contain them. Arithmetic operations, in particular the basic operations $+, -, \times, \div$ can be redefined in this context. The main interest of this representation lies in its *certification* property: if real numbers are represented by intervals, the interval resulting from the evaluation of an algebraic expression contains the exact value of the evaluated expression.

If the birth of Lattice Reduction is well-dated, it is not the case of Interval Arithmetic. For some authors, it has been introduced by R. Moore in 1962 in his PhD thesis [14]. For others it can be dated back to 1958 in an article of T. Sunaga [25] which describes an algebraic interpretation of the lattice of real intervals, or even sooner in 1931 as a proposal in the PhD thesis [27] of R.C. Young at Cambridge. Nonetheless its development and industrial applications have to wait for 1980, and the momentum of U. Kulisch in Karlsruhe, leading IBM to develop a specific instruction set and a compiler natively integrating Interval Arithmetic. Its main asset — calculating directly on sets — is nowadays used to deterministically determine the global extrema of a continuous function [19] or determining the zeroes of a function and proving their existence [8]. Another application of Interval Arithmetic is to be able to detect lack of precision at run-time of numerical algorithms, thanks to the guarantees it provides on computations.

This last application can lead to the design of adaptive precision numerical algorithms. In the present paper, we propose to transform the celebrated *LLL* algorithm into an adaptive precision version, leading to design a *provable* reduction algorithm which runs faster than the state-of-the-art standard FpLLL of D. Sthele et al. [23]. We also apply this technique to a stronger variant of *LLL*, called Potential-LLL, first introduced by Fontain et al. in [4], with similar results.

Organization of the paper. After reminders on lattices and on the *LLL* algorithm and its variant *Potential-LLL* in Section 2, we present Interval Arithmetic and the adaptive LLL variant in Section 4. Finally we discuss experimental results and running times in Section 5.

[4] *Stricto sensu*, he presented two algorithms, one for proving the existence of the so-called Hermite constant which bounds the length of the shortest vector of a lattice, and the other which allows to find basis with low orthogonality defect.

2 Preliminaries

In the sequel, we consider a lattice L of basis $\mathfrak{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ embedded in the canonical Euclidean vector space $\mathbb{R}^n$.

2.1 Lattices and reductions notions

We first define the *Gram matrix*, or *Gramian*, $G(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ of the vectors $\boldsymbol{b_1}, \ldots, \boldsymbol{b_d}$ as the matrix of their inner products $(\langle \boldsymbol{b_i}, \boldsymbol{b_j} \rangle)_{1 \leq i,j \leq d}$.

Any two bases of a non-trivial lattice are related to each other by an unimodular matrix[5]. Consequently, the determinant of the Gram matrix is independent of the choice of the basis. The square root of this constant is called the *volume* of the lattice.

For $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ a family of vectors in $\mathbb{R}^n$ we define by π_k the orthogonal projection onto the orthogonal complement of $\mathrm{Span}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_{k-1}})$. In particular, π_0 is the identity function. These projection operators are convenient to define reduction properties.

Gauss reduction. Before introducing more general notions of reduction, let us start with lattices.in dimension 2. We consider a two dimensional lattice L, and one of its basis $(\boldsymbol{u}, \boldsymbol{v})$. This basis is called *Gauss-reduced* if both of the following conditions are fulfilled:

$$\begin{cases} \|\boldsymbol{u}\| \leq \|\boldsymbol{v}\| \\ |\langle \boldsymbol{u}, \boldsymbol{v} \rangle| \leq \|\boldsymbol{u}\|^2/2 \end{cases}$$

The Gauss algorithm takes an arbitrary basis of the lattice and reduces it greedily in a manner similar to Euclid's GCD algorithm. More precisely, this process iteratively reduces the length of the longest vector of the basis, and continues until the vector being reduced remains larger than the other one. The

[5] A matrix $M \in M_d(\mathbb{Z})$ is said to be unimodular if $\det(M) \in \{-1, 1\}$.

description is fully given in Algorithm 1. Its termination relies on the decrease of the square of the norm of the shortest vector in the current pair $(\boldsymbol{u}, \boldsymbol{v})$.

Algorithm 1: Gauss reduction.

```
Input: Initial basis $(\boldsymbol{u}, \boldsymbol{v})$.
Result: A reduced basis of the lattice.
if $\|\boldsymbol{u}\| < \|\boldsymbol{v}\|$ then
    Swap $\boldsymbol{u}$ and $\boldsymbol{v}$;
end
repeat
    $\lambda \leftarrow \lfloor \langle \boldsymbol{u}, \boldsymbol{v} \rangle / \|\boldsymbol{v}\|^2 \rceil$;
    $\boldsymbol{u} \leftarrow \boldsymbol{u} - \lambda \cdot \boldsymbol{v}$;
    Swap $\boldsymbol{u}$ and $\boldsymbol{v}$;
until $\|\boldsymbol{u}\| \leq \|\boldsymbol{v}\|$;
Output $(\boldsymbol{u}, \boldsymbol{v})$;
```

It is easy to see that the first vector of the reduced basis is the shortest non-zero lattice vector. Indeed, let take a non-zero vector $\boldsymbol{x}$ in the lattice, and decompose it in the output basis $(\boldsymbol{u}, \boldsymbol{v})$:

$$\boldsymbol{x} = a \cdot \boldsymbol{u} + b \cdot \boldsymbol{v}$$

for integers $a, b \neq (0,0)$. We then have by the *Gauss reduction* hypothesis that:

$$\begin{aligned} \|\boldsymbol{x}\|^2 &= a^2\|\boldsymbol{u}\|^2 + 2ab\langle \boldsymbol{u}, \boldsymbol{v} \rangle + b^2\|\boldsymbol{v}\|^2 \\ &\geq \underbrace{(a^2 - |ab| + b^2)}_{>0 \text{ since } (a,b) \in \mathbb{N} - \{(0,0)\}} \|\boldsymbol{u}\|^2 \end{aligned}$$

In addition, the second vector reaches the second-minimum of the lattice[6]. Before getting deeper in the generalization of the Gauss reduction for higher dimension lattices, we introduce the notion of *Gram-Schmidt orthogonalization.*

Gram-Schmidt orthogonalization. We recall that we are considering the lattice spanned by the independent vectors $\mathfrak{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ and B its associated matrix. Their Gram-Schmidt orthogonalization $\mathfrak{B}^* = (\boldsymbol{b_1^*}, \ldots, \boldsymbol{b_d^*})$ is the family of vectors defined inductively by :

$$\begin{cases} \boldsymbol{b_1^*} := \boldsymbol{b_1} \\ \boldsymbol{b_i^*} := \boldsymbol{b_i} - \displaystyle\sum_{j=0}^{i-1} M_{i,j} \boldsymbol{b_j^*} \end{cases}$$

where $M_{i,j} = \frac{\langle \boldsymbol{b_i}, \boldsymbol{b_j^*} \rangle}{\|\boldsymbol{b_j^*}\|^2}$. Since the $M_{i,j}$ have no reason to be integers in general, the vectors $\boldsymbol{b_j^*}$ do not necessarily belong to the lattice L, but to the ambient space

[6] The k-th minimum of the lattice is defined as the smallest positive real number λ_k such that there exists at least one set of k linearly independent vectors which each vector of norm at most λ_k.

$\mathbb{R}^n$. We emphasize the fact that for every $1 \leq i \leq d$, we have $\mathrm{Span}(\boldsymbol{b_1}, \cdots, \boldsymbol{b_i}) = \mathrm{Span}(\boldsymbol{b_1^*}, \cdots, \boldsymbol{b_i^*})$ Moreover, one notices that, using the previously projections operators π_i, we have by definition:

$$\pi_i(\boldsymbol{b_i}) = \boldsymbol{b_i}^*$$

The whole construction is given in pseudo-code as Algorithm 2.

Algorithm 2: Gram-Schmidt orthogonalization.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$.
Result: Basis $\mathfrak{B}^*$, transformation matrix $M$.
for $i = 1$ to $d$ do
    $\boldsymbol{b_i^*} \leftarrow \boldsymbol{b_i}$;
    for $j = 1$ to $i - 1$ do
        $M_{i,j} \leftarrow \frac{\langle \boldsymbol{b_i}, \boldsymbol{b_j^*} \rangle}{\|\boldsymbol{b_j^*}\|^2}$;
        $\boldsymbol{b_i^*} \leftarrow \boldsymbol{b_i^*} - M_{i,j} \boldsymbol{b_j^*}$;
    end
end
```

Reduction in Higher dimension. Let us come back to lattice reduction. A natural question is whether the simplicity of the Gauss reduction extends naturally to higher dimensions. Unfortunately, this is not the case, and the various generalizations of this notion do not coincide. For instance, trying to generalize the fact that in dimension 2 a reduced basis reaches the first and second minima is not possible. Nonetheless, in 1982, Lenstra, Lenstra and Lovász [12] proposed a notion called *LLL-reduction* conjointly with a polynomial time algorithm. Their reduction notion is defined as follows:

Definition 1 (LLL reduction). *A basis $\mathfrak{B}$ of a lattice is said to be δ-**LLL-reduced** for a certain parameter $1/4 < \delta \leq 1$, if the following conditions are satisfied:*

$$\forall i < j, \quad |\langle \boldsymbol{b_j}, \pi_i(\boldsymbol{b_i}) \rangle| \leq \frac{\|\pi_i(\boldsymbol{b_i})\|^2}{2} \quad \textit{(Size-Reduction condition)} \tag{1}$$

$$\forall i, \quad \delta \cdot \|\pi_i(\boldsymbol{b_i})\|^2 \leq \|\pi_i(\boldsymbol{b_{i+1}})\|^2 \quad \textit{(Lovász condition)} \tag{2}$$

Using the Gram-Schmidt orthogonalization, we can rewrite this definition in a more algorithmic way, using the vectors $\boldsymbol{b_i^*}$.

Definition 2 (LLL reduction (algorithmic version)). *A basis $\mathfrak{B}$ of a lattice is said to be δ-**LLL-reduced** for a certain parameter $1/4 < \delta \leq 1$, if the following conditions are satisfied:*

$$\forall i < j, \quad |\langle \boldsymbol{b_j}, \boldsymbol{b_i}^* \rangle| \leq \frac{\|\boldsymbol{b_i}^*\|^2}{2} \quad \textit{(Size-Reduction condition)} \tag{3}$$

$$\forall i, \quad \delta \cdot \|\boldsymbol{b_i}^*\|^2 \leq \left(\|\boldsymbol{b_{i+1}}^*\|^2 + \frac{\langle \boldsymbol{b_{i+1}}, \boldsymbol{b_i^*} \rangle}{\|\boldsymbol{b_i}^*\|^2} \right) \quad \textit{(Lovász condition)} \qquad (4)$$

where the vectors $\boldsymbol{b_i^*}$ result from the Gram-Schmidt orthogonalization of the basis $\mathfrak{B}$.

We can now introduce two lattice reduction algorithms which both run in polynomial time: the celebrated *LLL* algorithm and one of its variants, the Potential-*LLL* [4].

3 *LLL* algorithm and its Potential-*LLL* variant

3.1 LLL reduction

The algorithm of Lenstra-Lenstra-Lovász [12], also denoted as *LLL* or L^3 is a combination of the Gauss reduction algorithm for 2-dimensional lattices and the Gram-Schmidt orthogonalization process. In a nutshell, this algorithm applies one iteration of the Gauss reduction to the lattice generated by $\pi_i(\boldsymbol{b_i})$ and $\pi_i(\boldsymbol{b_{i+1}})$. All the operations in this projected lattice are actually lifted to the vectors $\boldsymbol{b_i}$ and $\boldsymbol{b_{i+1}}$ themselves in order to act directly on the basis manipulated by the algorithm. Furthermore, integer linear combinations are performed between these vectors and the $\boldsymbol{b_1}, \dots, \boldsymbol{b_{i-1}}$ in order to reduce the orthogonality defect of the resulting basis.

More precisely, two kind of unimodular operations are performed on a vector of the current basis: *exchange steps* and integer linear combinations with other vectors of the basis, called *translations*. Before getting an in-depth view of the details of these steps within the algorithm, we use the Gram-Schmidt orthogonalization to provide a convenient matrix representation of the quantities used by *LLL*.

Indeed, the Gram-Schmidt orthogonalization on $\mathfrak{B}$ leads to the QR-decomposition of B into $B^* \cdot M$ where B^* is the matrix representing $\mathfrak{B}^*$, and M is the matrix of coefficients $M_{i,j}$. Thus, by considering the Gram matrix associated to the basis[7], one gets:

$$G = M^T \cdot B^{*T} \cdot B^* \cdot M = M^T D M$$

for a diagonal matrix D, since the rows B^* are by construction orthogonals. Denoting by R the matrix $D \cdot M$, we have:

$$G = R^T \cdot M = M^T \cdot R.$$

Translations and Size Reduction steps. The goal of the translation steps is to make the off-diagonals coefficients of M lie in $[-\frac{1}{2}, \frac{1}{2}]$. To perform such a reduction, one iteratively translates each $\boldsymbol{b_i}$ along $\boldsymbol{b_j}$ for each $j < i$, by substracting $\lceil M_{i,j} \rfloor$ times the vector $\boldsymbol{b_j}$ to $\boldsymbol{b_i}$. Geometrically speaking, it ensures that each of

[7] We recall that $G = B^T B$ by definition.

the $\boldsymbol{b_i}$ are not too far from the corresponding $\boldsymbol{b_i^*}$ and has low orthogonality defect with the predecessors: $(\boldsymbol{b_1}, \dots, \boldsymbol{b_{i-1}})$. This full procedure is called *size-reduction* and is described in pseudo-code in Algorithm 3. Note that size-reduction leaves the orthogonalized vectors b_i^* unchanged.

Algorithm 3: The size reduction algorithm, Sred.

```
Input: Initial basis $(\boldsymbol{b_1}, \dots, \boldsymbol{b_d})$, index $1 \leq k \leq d$
Result: Modified basis with $\boldsymbol{b_k}$ size-reduced.
for $i = k-1$ downto 1 do
    $\boldsymbol{b_k} \leftarrow \boldsymbol{b_k} - \lfloor M_{k,i} \rceil \cdot \boldsymbol{b_i}$;
    Update matrices $M$ and $R$ accordingly.
end
```

Exchanges and Lovász condition. In order to act on the $\boldsymbol{b_i^*}$, exchange steps are performed. Without loss of generality, assume that $\boldsymbol{b_i}$ and $\boldsymbol{b_{i+1}}$ are exchanged. We denote by $\hat{M}$ the state of matrix M after this exchange and by $\hat{\boldsymbol{b}}_{\boldsymbol{i}}^*$ the updated GSO vectors. Setting the vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ as the orthogonal projection of respectively $\boldsymbol{b_i}$ and $\boldsymbol{b_{i+1}}$ by π_i. Before the exchange we have by definition of an orthogonal projection:

$$\boldsymbol{b_i^*} = \pi_i(\boldsymbol{b_i}) = \boldsymbol{u} \quad \text{and} \quad \boldsymbol{b_{i+1}^*} = \boldsymbol{v} - \underbrace{\frac{\langle \boldsymbol{u}, \boldsymbol{v} \rangle}{\|\boldsymbol{u}\|^2}}_{=M_{i+1,i}} \boldsymbol{u}.$$

And after the exchange:

$$\hat{\boldsymbol{b}}_{\boldsymbol{i}}^* = \boldsymbol{v} \quad \text{and} \quad \hat{\boldsymbol{b}}_{\boldsymbol{i+1}}^* = \boldsymbol{u} - \underbrace{\frac{\langle \boldsymbol{v}, \boldsymbol{u} \rangle}{\|\boldsymbol{v}\|^2}}_{=\hat{M}_{i+1,i}} \boldsymbol{v}.$$

By invariance of the determinant of the sublattice $\operatorname{Span}(\pi_i(\boldsymbol{b_i}), \pi_i(\boldsymbol{b_{i+1}}))$, we have:

$$\|\hat{\boldsymbol{b}}_{\boldsymbol{i}}^*\|^2 = \|\boldsymbol{b_{i+1}^*}\|^2 + M_{i+1,i}^2 \|\boldsymbol{b_i^*}\|^2$$
$$\|\hat{\boldsymbol{b}}_{\boldsymbol{i+1}}^*\|^2 = \frac{\|\boldsymbol{b_i^*}\|^2 \|\boldsymbol{b_{i+1}^*}\|^2}{\|\hat{\boldsymbol{b}}_{\boldsymbol{i}}^*\|^2}.$$

Thus, in order to ensure getting a LLL-reduced basis, one performs the exchanges whenever the Lovász condition is not fulfilled, that is if:

$$\delta \cdot \|\boldsymbol{b_i}^*\|^2 > \left(\|\boldsymbol{b_{i+1}}^*\|^2 + \frac{\langle \boldsymbol{b_{i+1}}, \boldsymbol{b_i^*} \rangle}{\|\boldsymbol{b_i}^*\|^2} \right),$$

which can be rewritten using the coefficients of the previously defined R:

$$\delta R_{i,i} > R_{i+1,i+1} + M_{i+1,i}^2 R_{i,i}.$$

The whole LLL algorithm is given is Algorithm 4.

Algorithm 4: The LLL algorithm.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$
Input: Parameter $\delta \in ]1/4, 1[$.
Result: A reduced basis.
Compute $B^*$ and $M$ with the GSO Algorithm 2;
$R \leftarrow (G \cdot (M)^{-1})^T$;
$k \leftarrow 2$;
while $k \leq d$ do
    Apply length reduction SRed($k$);
    if $\delta R_{k-1,k-1} \leq R_{k,k} + M_{k,k-1}^2 R_{k-1,k-1}$ then
        $k \leftarrow k + 1$;
    else
        Swap $\boldsymbol{b_k}$ and $\boldsymbol{b_{k-1}}$;
        Update $M$, $B^*$, $R$ accordingly ;
        $k \leftarrow \max(k-1, 1)$;
    end
end
Output $(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$
```

A simple optimization can be made in the way we write the algorithm: one can perform directly the insertion as far as possible, instead of letting the algorithm performs successive exchanges in a row. This tweak is described in Algorithm 5.

Termination and decrease of the potential. The soundness the algorithm is direct. Moreover, it terminates in polynomial time when $\delta < 1$. A classical argument relies on the study of the quantity

$$\mathcal{P}(\mathfrak{B}) = \prod_{i=0}^{d} \|\boldsymbol{b_i^*}\|^{2(d-i+1)},$$

called *potential* associated to the basis $\mathfrak{B}$. The potential $\mathcal{P}(\mathfrak{B})$ can be seen as the product of the volume of increasing sublattices; explicitly, given $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ a basis of a lattice L, we consider the increasing chain of sublattices $L_1 \subset L_2 \subset \cdots \subset L_d = L$ where $L_i = \mathrm{Span}(\boldsymbol{b_1}, \ldots, \boldsymbol{b_i})$. Since $\mathrm{vol}(L_i) = \prod_{j=0}^{i} \|\boldsymbol{b_j^*}\|^2$, we get:

$$\mathcal{P}(\mathfrak{B}) = \prod_{i=0}^{d} \underbrace{\mathrm{vol}(L_i)}_{\in \mathbb{N}^*}.$$

Algorithm 5: The LLL algorithm, with insertion directly as deeply as possible.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$
Input: Parameter $\delta \in ]1/4, 1[$.
Result: A reduced basis.
Compute $B^*$ and $M$ with the GSO Algorithm 2;
$R \leftarrow (G \cdot (M)^{-1})^T$;
$k \leftarrow 2$;
while $k \leq d$ do
    Apply length reduction SRed($k$);
    $k' \leftarrow k$;
    while $k \geq 2$ and $\delta R_{k-1,k-1} > R_{k',k'} + \sum_{i=k-1}^{k'-1} M_{k',i}^2 R_{i,i}$ do
        $k \leftarrow k - 1$;
    end
    if $k \neq k'$ then
        Insert $\boldsymbol{b_{k'}}$ at pos $k$ and update $M, R, B^*$ accordingly[a];
    end
    $k \leftarrow k + 1$;
end
Output $(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$
```

[a] By "Insert $\boldsymbol{b_i}$ at pos j for $j < i$" we mean updating the basis to$(\boldsymbol{b_1}, \ldots, \boldsymbol{b_{j-1}}, \boldsymbol{b_i}, \boldsymbol{b_j}, \boldsymbol{b_{j+1}}, \ldots, \boldsymbol{b_{j-1}}, \boldsymbol{b_{i+1}}, \ldots, \boldsymbol{b_d})$.

Thus, $\mathcal{P}(\mathfrak{B})$ is a non-zero integer. In addition this potential decreases by a factor at least δ^{-1} at each exchange step and is left unchanged by other operations. Indeed:

- Either a linear combination on $\boldsymbol{b_k}$ of the previous vectors is performed and the chain of sublattices (L_j) is not altered.
- Or an exchange is done and the situation gets slightly more complicated. Assume without loss of generality that $\boldsymbol{b_k}$ and $\boldsymbol{b_{k-1}}$ are exchanged. Then, none of the $k-2$ first sublattices $(L_j)_{j<k-1}$ and none of the trailing ones $(L_j)j_{\geq}k$ are modified. However, the sublattice L_{k-1} changes, and so does its volume. Since by construction the exchange occurs if $\|\boldsymbol{b_{k-1}^*}\|$ shrinks by a factor at least δ^{-1}, the volume of L_{k-1} is also decreased by the same factor.

Thus the number of exchange steps — and consequently of iterations — is bounded by $O(\log(\|B\|_\infty)d^2)$ where B is the matrix of the initial basis[8].

Running-time of *LLL*. As the cost of a loop iteration is of $O(dn)$ arithmetic operations on *rational* coefficients $M_{i,j}$ and $R_{i,i}$, of length at most $O(d\log(\|B\|_\infty))$, the total cost in term of arithmetic operations is loosely bounded by $O(nd^5\log^3(\|B\|_\infty))$. By being more precise in the majoration of the bit-length of the integers appearing

[8] We recall that $\|B\|_\infty = \max_{1\leq i\leq j\leq d} |B_{i,j}|$.

in *LLL*, this analysis can be improved, such as in [10] where Kaltofen bound the complexity by $O(\frac{d^4 n \log^2(\|B\|_\infty)}{d+\log(\|B\|_\infty)} \cdot M(d+\log(\|B\|_\infty)))$ where $M(k)$ denotes the complexity of the multiplication of two integers of bit-length at most k. We emphasize once again on the fact that *for now* all computations are performed in an *exact manner*, with rationals coefficients represented as a quotient of large integers.

3.2 The Potential-LLL

Potential and potential reduction. In the analysis of the running-time of LLL, we introduced the the quantity $\mathcal{P}(\mathfrak{B}) = \prod_{i=0}^{d} \|\|\boldsymbol{b_i^*}\|^{2(d-i+1)}$, called the *potential* associated to the basis $\mathfrak{B}$. In particular, we proved that each exchange step of *LLL* leads to a decrease of the potential by a factor δ^{-1}. A natural generalization is then to design a reduction algorithm which directly aims at decreasing the potential, by performing, like in the original LLL algorithm, exchanges and translations. In this variation proposed in [4], the exchange condition is slightly relaxed but still guaranties a decrease of the potential.

Before going further in the presentation of this algorithm, let us describe the action of a sequence of insertions on the potential of a lattice.

Lemma 1. *Let $\mathfrak{B} = (\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$ a basis of a given lattice, and two integers i, j such as $1 \leq i < j \leq d$. Let $\mathfrak{B}'$ the basis obtained by inserting $\boldsymbol{b_j}$ to the left of $\boldsymbol{b_i}$. Then:*

$$\mathcal{P}(\mathfrak{B}') = \mathcal{P}(\mathfrak{B}) \cdot \prod_{k=i}^{j-1} \frac{\|\pi_k(\boldsymbol{b_j})\|^2}{\|\pi_k(\boldsymbol{b_k})\|^2}$$

Using directly conditions on the reduction of the potential instead of looking at the classical Lovász condition leads to a very similar running-time analysis as in *LLL* and therefore ensure a polynomial time reduction algorithm.

All these considerations lead to the *Potential-LLL* reduction.

Definition 3 (Potential reduction). *A basis $\mathfrak{B}$ of a lattice is said to be δ-**Potential-reduced** for a certain parameter $1/4 < \delta \leq 1$, if the following conditions are satisfied:*

$$\forall i < j, \quad |\langle \boldsymbol{b_j}, \boldsymbol{b_i}^* \rangle| \leq \frac{\|\boldsymbol{b_i}^*\|^2}{2} \quad \textit{Size-reduction condition} \tag{5}$$

$$\forall 1 \leq i < j \leq d, \quad \delta \cdot \mathcal{P}(\mathfrak{B}) \leq \mathcal{P}(\mathfrak{B}') \quad \textit{Lovász-potential condition} \tag{6}$$

where $\mathfrak{B}'$ is the basis resulting by the insertion of $\boldsymbol{b_j}$ to the left of $\boldsymbol{b_i}$.

Clearly, if a basis $\mathfrak{B}$ is δ-Potential Reduced, then it is also δ-LLL reduced.

The Potential-LLL algorithm. In [4], Fontein et al. introduced a polynomial-time reduction algorithm based on the decrease of the potential. We describe here a greedy variant[9] of this algorithm.

During the execution of the algorithm, the first $l-1$ vectors are always potential-reduced. Then, a sequence of test is performed to detect if there exists a position $1 \leq i \leq l-1$ at which the insertion of the l-th vector creates a diminution of the potential. If so, the vector $\boldsymbol{b_l}$ is inserted right before the vector $\boldsymbol{b_i}$ and other vectors are let unchanged.

The activation condition of these exchanges steps is then the comparison of potentials — the current potential of the basis and the potential of the basis after an exchange — and thanks to Lemma 1, we only need to compute the quotient of the two involved potentials. This quotient can be computed efficiently. The same analysis as in the description of LLL ensures that

$$\frac{\pi_j(\boldsymbol{b_\ell})}{\pi_j(\boldsymbol{b_j})} = \frac{\|\boldsymbol{b_\ell}^*\|^2 + \sum_{i=j}^{\ell-1} M_{\ell,i}^2 \cdot \|\boldsymbol{b_i^*}\|^2}{\|\boldsymbol{b_j^*}\|^2},$$

allowing an efficient inductive computation. The whole description of the procedure is given in Algorithm 6.

Complexity analysis. Once again, the soundness is direct. Since the *potential* appears in the analysis of LLL, we can also use it here. In the Potential-LLL algorithm, the situation is even simpler, since an insertion is done if there is a decrease of a factor at least δ^{-1}. As such, the number of exchanges remains unchanged: $O(d\log(\|B\|_\infty))$; therefore the number of iterations is now: $O(d^2\log(\|B\|_\infty))$. The computational cost of one loop now splits in two parts: the update of the GSO coefficients and the computations of the quotients of potentials, leading to $O(nd+d^2)$ operations. We then get a total complexity complexity in $O(nd^6\log^3(\|B\|_\infty))$, using naive multiplication on $O(\log(\|B\|_\infty)d)$ bits integers.

3.3 From exact rational arithmetic to the L^2 floating-point algorithm

Since the behavior of both algorithms is the same with respect to floating-point approximation techniques, we focus on the case of the original LLL.

Rational representation of GSO coefficients and Integral-LLL. The total cost of the LLL algorithm is dominated by the computation of the GSO coefficients, whose numerators and denominators are growing. A bottleneck when performing computations with such representations of rational values is the

[9] The main difference lies in the choice of the insertion to perform. In [4] the authors choose the insertion realizing the maximal diminution of the potential, whereas ours greedily inserts in the first position where a decrease occurs.

Algorithm 6: The Potential-LLL algorithm.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$
Input: Parameter $\delta \in ]1/4, 1[$.
Result: A reduced basis.
Compute $B^*$ and $M$ with the GSO Algorithm 2;
$R \leftarrow (G \cdot (M)^{-1})^T$;
$k \leftarrow 2$;
while $k \leq d$ do
    Apply length reduction SRed($k$);
    Q $\leftarrow 1$;
    for $\ell = k - 1$ downto 1 do
        Q $\leftarrow$ Q $\cdot \frac{\|b_k^*\|^2 + \sum_{i=\ell}^{k-1} M_{k,i}^2 \cdot \|\boldsymbol{b_i^*}\|^2}{\|\boldsymbol{b_\ell^*}\|^2}$;
        if $\delta > Q$ then
            Insert $\boldsymbol{b_k}$ at pos $\ell$ and update matrices $M, R$ and $B^*$;
            k $\leftarrow \ell$;
            break;
        end
    end
end
Output $(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$
```

necessity to perform repeated GCD procedures. A first idea to overcome this problem is to avoid the use of denominators by multiplying all the quantities involved by carefully chosen integers. This algorithm was introduced by De Weger in [26]. While slightly more efficient in practice, the algorithm has the same asymptotic behavior remains unchanged.

Using floating points approximations. However, it is remarkable that the norm of these rational values remains small, and naturally leads to using approximations of the desired quantities, instead of computing with them in an exact manner. Translating directly the LLL algorithm with floating point approximations leads to severe drawbacks: first, the whole algorithm might not terminate, and even if it does, the output basis is not any longer guaranteed to be LLL-reduced.

The seminal *provable* floating-point version of the algorithm is due to Schnorr in [21], leading to a complexity of $O(d^3 n \log(\|B\|_\infty) \cdot M(d + \log(\|B\|_\infty)))$. Using naive multiplication, this cost is however still cubic in $\log(\|B\|_\infty)$. Number theory libraries and packages contain heuristic variants of the fp-version of Schnorr and Euchner [22], like NTL [1].

Numerous variants are built on this first version of Schnorr and Euchner, to eventually lead to the now widely used and fastest provable floating-point variant of Nguyen-Stehlé L^2 [17], implemented in the library *FpLLL* [23].

3.4 The L^2 algorithm

The L^2 algorithm is a variant of Schnorr-Euchner version of *LLL* [22]. By contrast with *LLL*, L^2 computes the GSO coefficients on the fly as they are needed instead of doing a full orthogonalization at the start. It also uses a lazy size reduction inspired by the *Cholesky factorization algorithm*. More precisely, the GSO coefficients are obtained by computing the $R_{i,j}$ inductively with the formula:

$$R_{i,j} = G_{i,j} - \sum_{k=1}^{j-1} M_{j,k} \cdot R_{i,k}.$$

Then $M_{i,j}$ is obtained simply the quotient as $\frac{R_{i,j}}{R_{j,j}}$. Furthermore, the actual algorithm computes the quantities $s_j^{(i)} = \|\boldsymbol{b_i}\|^2 - \sum_{k=1}^{j-1} M_{j,k} \cdot R_{i,k}$. for all $1 \leq j \leq i$, leading to a simple reformulation of the Lovász condition:

$$\delta \cdot R_{k-1,k-1} \leq s_{k-1}^{(k)}.$$

The Lazy Size-Reduction procedure is fully given in Algorithm 7 and the L^2 algorithm in Algorithm 8.

These optimizations improve the running time of the lattice reduction to $O(d^4 n(d + \log(\|B\|_\infty)) \log(\|B\|_\infty))$, while still being provable. Similarly putting together L^2 and Potential-*LLL* yields an algorithm of complexity: $O(d^5 n(d + \log(\|B\|_\infty)) \log(\|B\|_\infty))$

Algorithm 7: The lazy size reduction algorithm, LazyRed.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$
Result: Basis $\mathfrak{B}^*$, transformation matrices $M$, $R$, $s$.
for $j = 1$ to $k - 1$ do
    $R_{i,j} \leftarrow G_{i,j}$;
    for $i = 1$ to $j - 1$ do
        $M_{i,j} \leftarrow R_{k,j} / R_{j,j}$;
    end
end
$s_1^{(k)} \leftarrow \|\boldsymbol{b_k}\|^2$;
for $j = 2$ to $k$ do
    $s_j^{(k)} \leftarrow s_{j-1}^{(k)} - M_{k,j-1} \cdot R_{k,j-1}$;
end
$R_{i,i} \leftarrow s_i^{(i)}$;
```

Precision required. The precision required by Algorithm 8 is $d \cdot \log_2(\frac{(1+\eta)^2}{(\delta-\eta)^2} + \epsilon) + o(d)$ bits for any $\epsilon > 0$, i.e. almost linear in the dimension of the lattice.

Algorithm 8: The L^2 Algorithm.

```
Input: Initial basis $(\boldsymbol{b_1}, \dots, \boldsymbol{b_d})$
Input: Parameter $\delta \in ]1/4, 1[$.
Result: A reduced basis.
Compute $G = G(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$ in exact integer arithmetic;
$R_{1,1} \leftarrow G_{1,1}$;
$k \leftarrow 2$; ;
while $k \leq d$ do
    Apply length reduction LazyRed($k$);
    $k' \leftarrow k$;
    while $k \geq 2$ and $\delta R_{k-1,k-1} > s_{k-1}^{k'}$ do
        $k \leftarrow k - 1$;
    end
    $R_{k,k} \leftarrow s_k^{k'}$;
    if $k \neq k'$ then
        for $i = 1$ to $k - 1$ do
            $M_{k,i} \leftarrow M_{k',i}$;
            $R_{k,i} \leftarrow R_{k',i}$;
        end
        Insert $\boldsymbol{b_{k'}}$ at pos $k - 1$ and update matrices $M, R$;
    end
    $k \leftarrow k + 1$;
end
Output $(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$
```

However, as presented in [16], it appears experimentally that, even though this bound is sharp, the number of bits required *on average* is much lower.

Hence, the challenge would be, to manage to detect at run-time whether the precision is sufficient or not to soundly perform the computation. Then it would be possible to dynamically adapt the precision of the current computation and only work with a quasi-optimal precision. All of this can be achieved using *Interval Arithmetic*.

4 Adaptive precision reduction algorithms with interval arithmetic

4.1 A primer on Interval Arithmetic

In this section, we present briefly Interval Arithmetic and focus on its interaction with floating point approximation of reals. For more details on Interval Arithmetic, the interested reader can consult a more extensive reference, such as [15].

Bird's eye view on Interval Arithmetic. Interval arithmetic is a representation of reals by intervals that contain them. For instance, one can specify that a value x is given with an error ϵ by considering the interval $[x-\epsilon, x+\epsilon]$, or even manipulating the transcendent constant π as the interval $[3.14, 3.15]$. Interval arithmetic is crucial the context of numerical computations, where reals can only be represented at finite precision.

Indeed, trade-offs must be found between the precision and range of the reals representable, such as in *floating-point* representation. As in the previous examples, Interval Arithmetic can be used to handle exact reals, now by manipulating intervals of length 2^{-n} where n is the number of bits of precision and bounds represented by floating-point numbers at precision n. More precisely, if we denote by $\lfloor x \rfloor_n$ and $\lceil x \rceil_n$ respectively the largest floating-point number below x and the lowest floating-point number above x written with n bits, the tightest representation of x is the interval $I_n(x) = [\lfloor x \rfloor_n, \lceil x \rceil_n]$.

Towards an algebra of Intervals. In the following, we denote by $\underline{x}$ an interval, by $\underline{x}^-$ — resp. $\underline{x}^+$ — its lower value — resp. its greatest value —, that is to say: $\underline{x} = [\underline{x}^-, \underline{x}^+]$. We can now define abstractly the arithmetic on intervals:

Definition 4. *Let $\bowtie$ be a binary operation — resp. f be a function —, then the result $\underline{x} \bowtie \underline{y}$ of the operation between the intervals $\underline{x}$ and $\underline{y}$ — resp $f(\underline{x})$, result of the application of f — is the smallest interval, in the sense of inclusion, containing*

$$\{x \bowtie y | (x,y) \in \underline{x} \times \underline{y}\} \qquad \textit{— resp. } \{f(x) | x \in \underline{x}\}\textit{—}.$$

In the context of actual computations, requiring the equality in the above definition is in most case illusory, since reals can not be represented exactly. Yet, only the inclusion of the resulting interval in the last defined set is required to ensure the most desired property of interval arithmetic: the most certification of computations.

Interval Arithmetic and Floating-point approximations. Explicitly, when approximating reals with floating point representations, basic arithmetic operations transpose directly into the formulae of the Figure 1.

$$\begin{aligned}
[\underline{x}^-,\underline{x}^+] + [\underline{y}^-,\underline{y}^+] &= [\underline{x}^- +^- \underline{y}^-,\underline{x}^+ +^+ \underline{y}^+]\\
[\underline{x}^-,\underline{x}^+] - [\underline{y}^-,\underline{y}^+] &= [\underline{x}^- -^- \underline{y}^-,\underline{x}^+ -^+ \underline{y}^+]\\
[\underline{x}^-,\underline{x}^+] \times [\underline{y}^-,\underline{y}^+] &= [\min{}^-(\rho),\max{}^+(\rho)] \qquad \text{where} \quad \rho = \underline{x}^-\underline{y}^-,\underline{x}^+\underline{y}^-,\underline{x}^-\underline{y}^+,\underline{x}^+\underline{y}^+\\
[\underline{x}^-,\underline{x}^+]^{-1} &= \left[\min{}^-(\frac{1}{\underline{x}^+},\frac{1}{\underline{x}^-}),\max{}^+(\frac{1}{\underline{x}^+},\frac{1}{\underline{x}^-})\right]
\end{aligned}$$

$+^+$, $+^-$ are here respectively the $+$ operator with higher approximation and with lower approximation. Same goes for the $-^+, -^-, \min^-, \max^+$ operators.

Fig. 1: Basic arithmetic operators in Interval Arithmetic

As a result, if numbers are represented by intervals, the interval resulting from the evaluation of an algebraic expression **contains** the exact value of the evaluated expression. More precisely for any family $(x_i)_{1 \leq i \leq n}$ of reals, and $(\underline{x}_i)_{1 \leq i \leq n}$ intervals such as for each i, $x_i \in \underline{x}_i$, then

$$f(x_1, \ldots, x_n) \in f(\underline{x}_1, \ldots, \underline{x}_N),$$

for any algebraic expression f. Therefore the results given in Interval Arithmetic are *certified*. For instance, one can certify whether a scalar x, represented by $\underline{x}$ is greater or equal than a value δ by ensuring that $\underline{x}^- \geq \lceil \delta \rceil_n$. If this test fails, one is only informed that the interval is too large to certify the desired property. If Interval Arithmetic is set with floating-point number with given precision, the failure of such a test allows to conclude that the precision chosen is not sufficient to assert the inequality. More precisely, in order to track down these precision defects in a comparison between x and δ, where x is represented by the interval $\underline{x}$ three cases must be handled:

- Either $\underline{x}^+ \leq \lfloor \delta \rfloor_n$, which certifies that $x \leq \delta$.
- Either $\underline{x}^- \geq \lceil \delta \rceil_n$, which certifies that $x \geq \delta$.
- Or $\delta \in \underline{x}$, and then the precision chosen is not sufficient to conclude.

In short, Interval Arithmetic used in such a way allows to detect a lack of precision at runtime of a numerical algorithm.

4.2 Interval arithmetic L^2 algorithm

This described certification property of Interval Arithmetic allows to run the LLL — more precisely its L^2 variant — algorithm at a chosen precision and thus to detect whether this choice is sufficient to achieve a provable reduction. As a result it is possible to iteratively increase the precision until the computation succeeds, and ensure to use an optimal — or at least quasi-optimal — number of bits.

Description of Adaptive-LLL. Starting with an initial precision p_0, the reduction procedure is launched with intervals of the form I_{p_0} to represent approximation of the Gram-Schmidt coefficients; instead of regular floating-point approximations. If a precision defect is detected by the technique of Section 4.1, then, the computation is directly aborted. In that case, the precision is increased and a whole new reduction procedure is called on the current basis; we can indeed start from the pre-reduced basis obtained just before aborting, which is *at least* closer to a LLL-reduced basis than the initial one. If no precision defect is detected, then the certification property and the provability of the reduction used ensures that the final basis is LLL-reduced.

The modifications made to the base algorithm, in our case L^2, are thus using interval representation for floating-point variables and transforming the conditional statements with inequalities between floating point values by the tests described in Section 4.1. In the case where the test returns a detection of a precision defect, the reduction stops and return an error. As such, we change the code from line 7 to 9 in Algorithm 8 into:

```
continue ← True;
while k $\leq$ 2 do
ret ← $(\underline{\delta} \cdot \underline{R_{k-1,k-1}} > \underline{s^{k'}_{k-1}})$
if ret = True then
  k ← k-1;
elif ret = False then
  break;
else
  return ErrorPrecision;
end;
end;
```

where a test between two intervals $\underline{a}$ and $\underline{b}$

$(\underline{a} > \underline{b})$

returns either a boolean value if the result can be certified thanks to interval Arithmetic, or $\perp$ if not. We call this modified version with return code, $\tilde{L}_2$.

The whole strategy is fully described in Algorithm 9. Thanks to the great similarities between the LLL algorithm and its Potential-LLL variant, the same techniques can be straightly used to design an adaptive Potential-LLL.

Remarks on time complexity of Adaptive-LLL. Once again, thanks to the similarities of analysis between LLL and Potential-LLL, we only continue the

Algorithm 9: The Adaptive-LLL algorithm.

```
Input: Initial basis $(\boldsymbol{b_1}, \ldots, \boldsymbol{b_d})$
Input: Parameter $\delta \in ]1/4, 1[$.
Result: A reduced basis.
prec $\leftarrow$ 64; succeed $\leftarrow$ 0;
repeat
    precr $\leftarrow$ prec$\times$2;
    if $\tilde{L}^2(\mathfrak{B}) \neq$ ErrorPrecision then
        succeed $\leftarrow$ True;
    end
until succeed = False;
Output $(\boldsymbol{b_1}, \cdots, \boldsymbol{b_d})$
```

study in the case of LLL. Depending on the library used, the cost of Interval Arithmetic with floating-point representation of bounds of intervals is up to fourth time the cost of classical floating-point arithmetic. Therefore the complexity of the inner loop remains unchanged with regards to the L^2 complexity. However, since the complexity of fp-multiplication is surlinear and the precision growth is geometric, the total cost of the adaptive-LLL is asymptotically dominated by the last iteration of the loop. This ensures that the total complexity of adaptive-LLL is the complexity of the L^2 algorithm using a approximation by a factor less than 2 of the optimal precision needed for the computation. In practice, for average dimensions it appears that the computational cost of the first iterations lies between 20% and 40% of the total cost.

Since the minimal precision required by the L^2 algorithm can be expressed as $d \cdot \log_2(\frac{(1+\eta)^2}{(\delta-\eta)^2} + \epsilon) + o(d)$. For usual choices of parameters, δ close to 1 and η close to $1/2$, it reduces roughly to $1.6 \cdot d + o(d)$. This last bound is tight in the sense that some lattices require such a complexity to be reduced by the L^2 algorithm. However, in [16], Nguyen and Stehlé presented an experimental heuristic on the precision required by L^2 to safely perform its computation:

Heuristic 41. Let δ be close to 1 and η be close to $1/2$. For almost every lattice, with a precision of $0.25\Delta d + o(d)$ bits for the fp-calculations, the L^2 algorithm performs correctly when given almost any input basis.

Under this heuristic, the adaptive-LLL algorithm gains a constant factor on the proved version of L^2, corresponding to the gap between mandatory precision on the average and on the worst-case, while still being a provable reduction algorithm. This is also the case for the Adaptive version of Potential-LLL.

5 Experimental analysis of Adaptive-LLL and Adaptive-Potential-LLL

5.1 Random lattices, random basis

In order to get information on the average properties of the adaptive version of the reduction algorithms, it is natural to work with randomly drawn lattices.

Random lattices, from Siegel to Goldstein and Mayer. The natural notion of random lattice was introduced by Siegel [24]. The Siegel measure is defined as the — finite — projected measure on $G = SL_n(\mathbb{R})/SL_n(\mathbb{Z})$ of the Haar measure[10]of $SL_n(\mathbb{R})$. Since G identifies with the real lattices modulo scaling, the Siegel measure gives a natural probability measure on lattices, after rescaling.

A practical method to construct lattices from a distribution that converges to the uniform distribution was introduced by Goldstein and Mayer in [7]. This procedure consists in fixing a large prime p, then sampling $d-1$ integers $(x_2, \dots, x_d)$ independently uniformly at random, in the underlying set of the finite field $\mathbb{Z}/p\mathbb{Z}$ and finally consider the lattice spanned by the rows of:

$$\begin{pmatrix} p & 0 & 0 & \cdots & 0 \\ x_2 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_d & 0 & 0 & \cdots & 1 \end{pmatrix}$$

From Random lattices, to random basis. Once a particular lattice is sampled in that way, we then need to select uniformly at random a basis of this lattice. Yet, there are infinitely many of them and even though all of them are related by an unimodular transformation, any non-trivial finite measure exists on the set of bases, since there do not exist non-trivial *finite* measures on unimodular matrices. In [16], Nguyen and Stehlé introduced an heuristic method to sample a randomized basis. Starting from a given lattice L, their method consists in sampling N vectors in $L \cap [-B; B]^d$, for a given $B \gg det(L)^{1/n}$. Even if this family is with high probability linearly independent, its integer span is likely to be only a sublattice of L. Nonetheless, it is possible to lift this family to a basis of L, using Babai's nearest plane algorithm [2], transforming vectors of norm lower than B into vectors of norm lower than $\frac{\sqrt{n}B}{2}$. Plugging this method after the sampling of Goldstein and Mayer allows then to generate vectors of infinite norm dominated by $\frac{\sqrt{d}p}{2}$.

[10] This is the unique — up to scale — bi-invariant Haar measure, which can be seen of as the measure it inherits as a hypersurface embedded in $\mathbb{R}^{n^2}$.

On basis from Ideal lattices. Ideal lattices were first introduced to improve space and time complexity of cryptosystems, but they actually have a much wider range of use. For instance, ideal lattices appear in the design of the first fully homomorphic encryption scheme by Gentry in 2009 [6]. The growing development of this class of lattices justifies its use as standalone benchmarks to compare lattice reduction algorithms. Before presenting the generation algorithm, we start by some reminders on elementary algebraic number theory.

Let ζ be an algebraic number over the rationals, and denote by $\Pi \in \mathbb{Q}[X]$ its minimal polynomial. We denote by $\mathbb{K}$ its splitting field $\mathbb{Q}[X]/(\Pi)$. One can define the integers $\mathcal{O}_{\mathbb{K}}$ of this number field as the elements whose minimal polynomial is monic with integers coefficients. Obviously $\mathbb{Z}[X]/(\Pi) \subset \mathcal{O}_{\mathbb{K}}$ but in general, the equality does not hold. Yet, as a finite-rank sub-module of the number field, there exists a finite family $(\zeta_i)_{i \in I}$ such that $\mathcal{O}_{\mathbb{K}} \cong \bigoplus_{i \in I} \mathbb{Z} \cdot \zeta_i$.

An integer ideal of $\mathbb{K}$ is an ideal of $\mathcal{O}_{\mathbb{K}}$, that is, an additive subgroup of $\mathcal{O}_{\mathbb{K}}$ which is stable by multiplication by any integer element. Integers ideals can be embedded in the canonical Hermitian space $\mathbb{C}^n$ in order to induce a natural geometry on it. The embedding is a multi-evaluation of the polynomial representing an element $g \in \mathbb{Q}[X]/(\Pi) \cong \mathbb{K}$ on the basis of $\mathbb{K}$. More precisely, the embedding family $\sigma = (\sigma_1, \cdots, \sigma_n)$ is defined as the $\mathbb{Q}$-automorphisms such that:

$$\begin{aligned} \sigma_i \colon \mathbb{Q}[X]/(\Pi) &\to \mathbb{C} \\ g &\mapsto g(\zeta_i). \end{aligned}$$

The embedding $\sigma(I)$ of an integer ideal I is then an Euclidean lattice.

Since number theoretical algorithms are more efficient for cyclotomic rings, we focus on that specific algebraic class (cyclotomic polynomials, cyclotomic fields and the ideals of cyclotomic integers rings).

We denote by Φ_m the *m-th cyclotomic polynomial*, that is the unique irreducible polynomial dividing $X^m - 1$ which is not dividing any of the $X^k - 1$ for $k < m$. Its roots are the n-th primitive roots of the unity. Cyclotomic polynomials can be written in closed form as:

$$\Phi_n(x) = \prod_{1 \leq k \leq n \wedge \gcd(k,n)=1} \left(x - e^{2i\pi\frac{k}{n}} \right)$$

The n-th cyclotomic field $\mathbb{Q}(\zeta_n)$ is obtained by adjoining a primitive n-th root of unity ζ_n to the rational numbers. As such, $\mathbb{Q}(\zeta_n)$ is isomorphic to the splitting field $\mathbb{Q}[X]/(\Phi_m)$ and in this specific case, its integer ring is precisely $\mathbb{Z}[X]/(\Phi_m)$.

The generation algorithm used is similar to the one of [18], which introduces the *Ideal Lattice Challenge*. We present the complete algorithm in Figure 10, which generates ideal lattices as embedding of ideals of cyclotomic integers rings.

Algorithm 10: Ideal Lattice generation.

```
Result: Basis $\mathfrak{B}$ of an ideal lattice of dim $d$
$\Phi_n(x) \leftarrow$ n-th cyclotomic polynomial.;
$d \leftarrow deg(\Phi_n)$;$det \leftarrow \text{rand}_s(2^{10d})$;
$det \leftarrow det - (det - 1 \text{mod } n)$;
repeat
    $det \leftarrow det + n$;
until $det$ is prime;
$g \leftarrow 1$;
repeat
    $g \leftarrow g + 1$; $\alpha \leftarrow g^{(det-1)/n}$;
until $\alpha$ is a root of unity in $\Phi_n(x)$;
$B = \begin{pmatrix} det & 0 \cdots 0 \\ -\alpha & 1 \quad \vdots \\ \vdots & \ddots \vdots \\ -\alpha^d & \quad 1 \end{pmatrix}$
```

		Maximum gain	Average Gain	Minimum Gain
Random Lattices	A-LLL	49%	35%	13%
	A-Pot-LLL	52%	37%	15%
Ideal Lattices	A-LLL	39%	25%	9%
	A-Pot-LLL	48%	29%	12%

Table 1: Comparison of the gain of running time between adaptive and non-adaptive reduction algorithms.

5.2 Experimental results

We now examine how the adaptive versions of LLL and of Potential-LLL compare to their original versions from the state-of-the-art implementations in $FpLLL$ [23]. Two series of extensive experiments have been realized: on random lattices from the Goldstein and Meyer class and on ideal lattices drawn with the Algorithm 10. Comparison between Adaptive-LLL — resp. Adaptive-Potential-LLL — and L^2 — resp. Potential-LLL — is given in Figure 2 — resp. Figure 3 — for Goldstein-Meyer lattices and in Figure 4 for ideal lattices — resp. Figure 5 —. We emphasize on the fact that the times are given in logarithmic scale to avoid a flattening of the lowest values in small dimensions.

As predicted, the running time of the adaptive reductions is lower than the running time of floating-point versions, of a constant factor which on average lies between 25% and 40%. A comparison of the gain between the different algorithms is given in Table 1. In medium dimension ($d > 150$), the gain on random lattices is almost 50%.

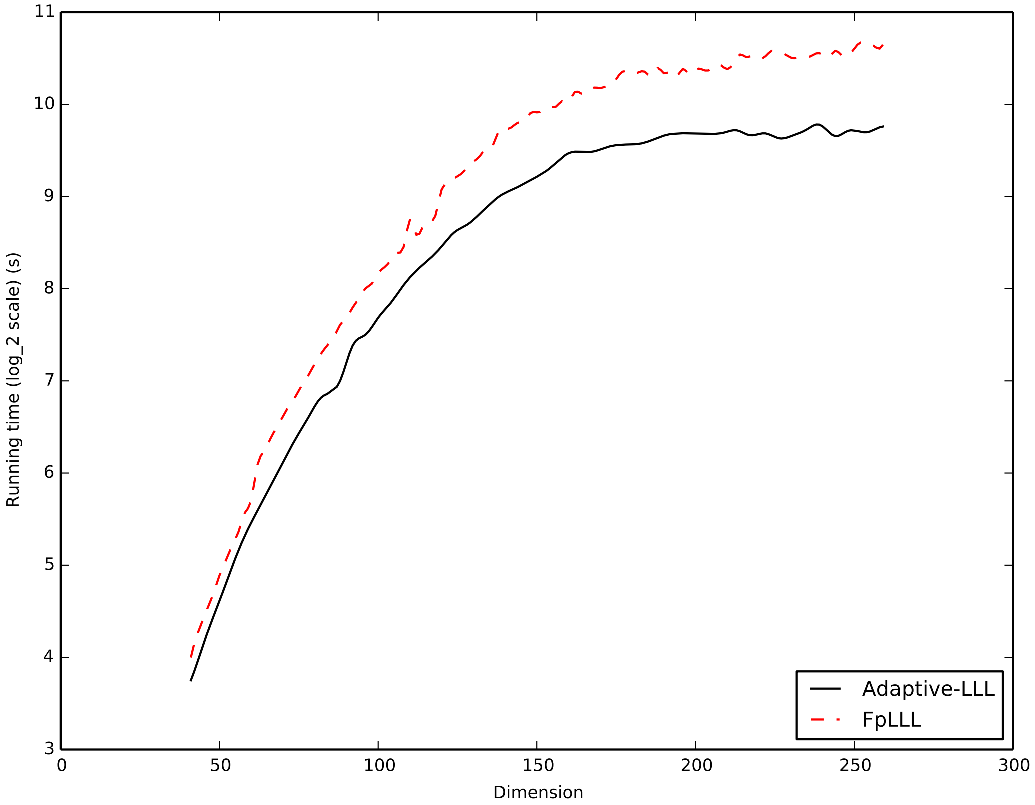

Fig. 2: Time benchmark for LLL/Adaptive-LLL on random lattices.

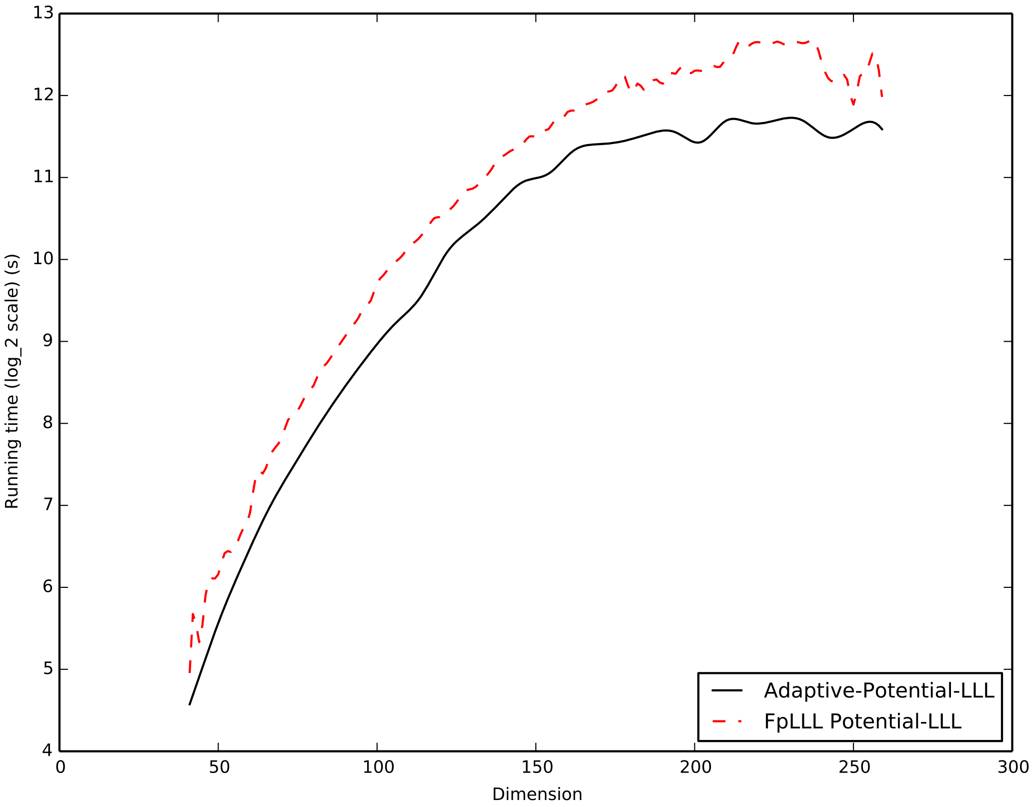

Fig. 3: Time benchmark for Potential-LLL/Adaptive-Potential-LLL on random lattices.

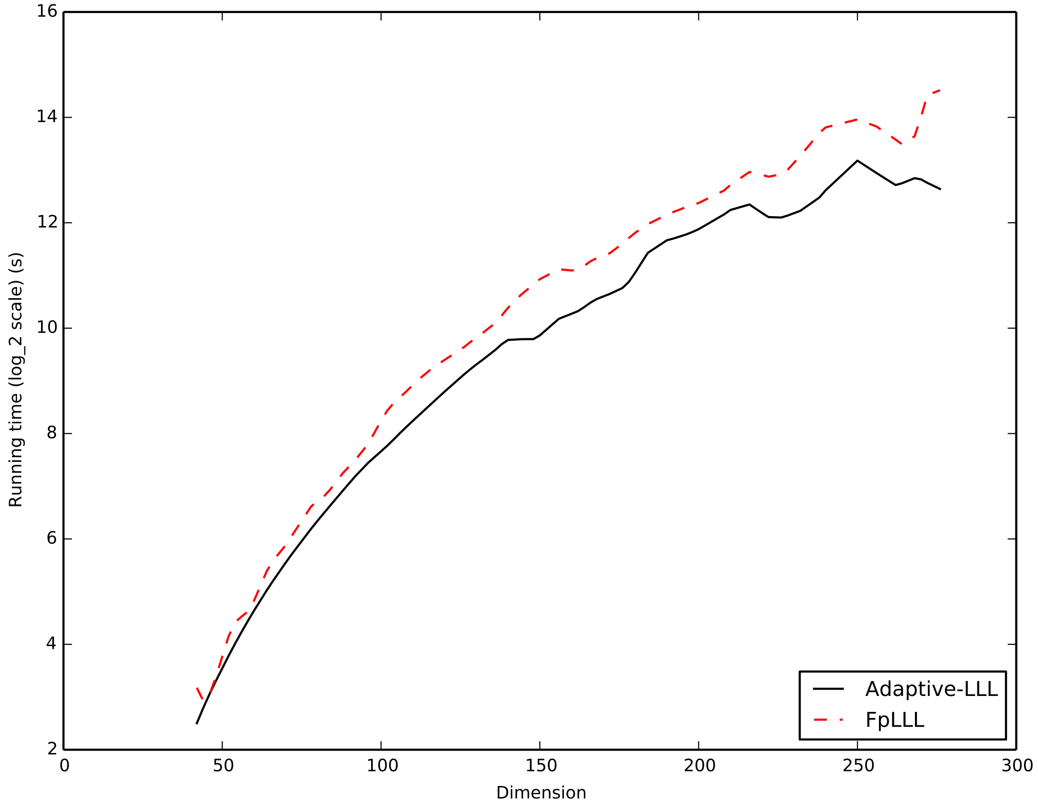

Fig. 4: Time benchmark for LLL/Adaptive-LLL on ideal lattices.

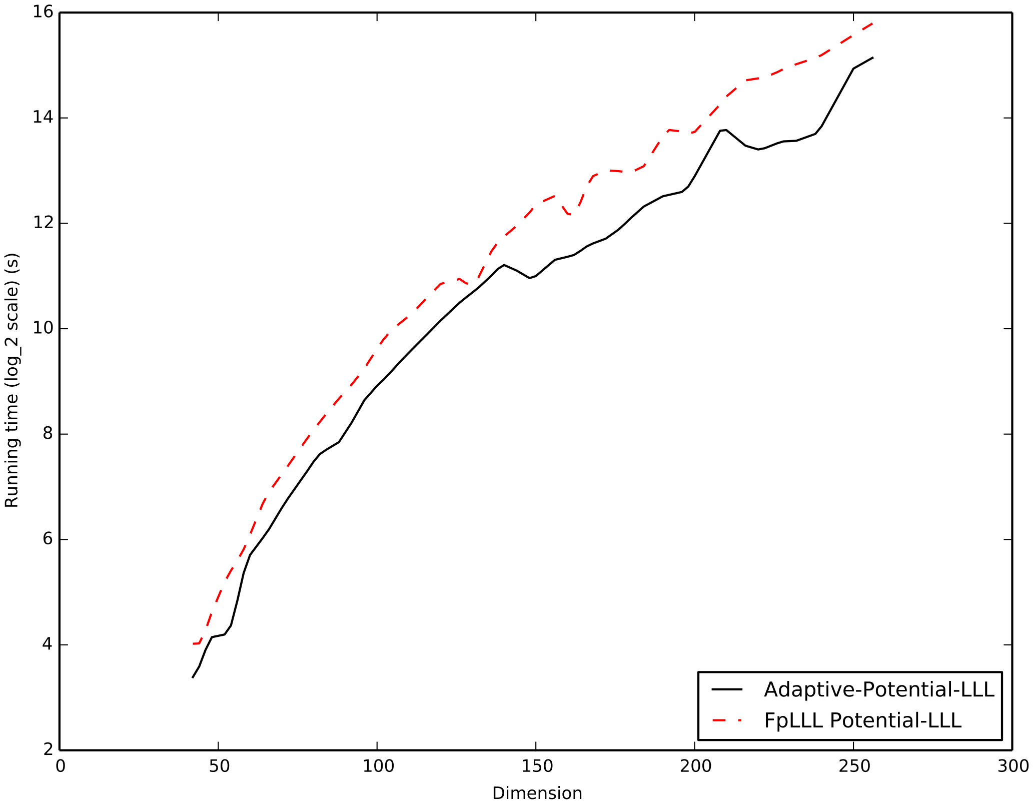

Fig. 5: Time benchmark for Potential-LLL/Adaptive-Potential-LLL on ideal lattices.

Bibliography

[1] M. Albrecht, S. Bai, D. Cadé, X. Pujol, and D. Stehlé. fplll-4.0, a floating-point LLL implementation. Available at http://perso.ens-lyon.fr/damien.stehle.

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

[3] D. Boneh et al. Twenty years of attacks on the RSA cryptosystem. *Notices of the AMS*, 46(2):203–213.

[4] F. Fontein, M. Schneider, and U. Wagner. A polynomial time version of LLL with deep insertions. *CoRR*, abs/1212.5100, 2012.

[5] N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 207–216, 2008.

[6] C. Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009*, pages 169–178, 2009.

[7] D. Goldstein and A. Mayer. On the equidistribution of hecke points. *Forum Mathematicum.*, 15(2):165–189, 2006.

[8] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis: with examples in parameter and state estimation, robust control and robotics.* Springer Verlag, 2001.

[9] E. Kaltofen. On the complexity of finding short vectors in integer lattices. In *Proc. EUROCAL '83*, volume 162, pages 236–244, 1983.

[10] E. Kaltofen. On the complexity of finding short vectors in integer lattices. In J. A. van Hulzen, editor, *Computer Algebra, EUROCAL '83, European Computer Algebra Conference*, volume 162 of *Lecture Notes in Computer Science*, pages 236–244. Springer, 1983.

[11] J. C. Lagarias and A. M. Odlyzko. Solving Low-density Subset Sum Problems. *J. ACM*, 32(1):229–246, 1985.

[12] A. Lenstra, H. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[13] H. Lenstra. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983.

[14] R. Moore. *Interval Arithmetic and automatic error analysis in digital computing.* PhD thesis, Stanford, 1962.

[15] R. E. Moore. *Methods and Applications of Interval Analysis.*

[16] P. Q. Nguyen and D. Stehlé. LLL on the Average. In F. Hess, S. Pauli, and M. E. Pohst, editors, *Algorithmic Number Theory, 7th International Symposium, ANTS-VII,*, volume 4076 of *Lecture Notes in Computer Science*, pages 238–256. Springer, 2006.

[17] P. Q. Nguyen and D. Stehlé. An LLL Algorithm with Quadratic Complexity. *SIAM J. of Computing*, 39(3):874—-903, 2009.

[18] T. Plantard and M. Schneider. Creating a Challenge for Ideal Lattices. Cryptology ePrint Archive, Report 2013/039, 2013.

[19] H. Ratschek and J. Rokne. *New Computer Methods for Global Optimization.* Halsted Press, New York, NY, USA, 1988.

[20] C. Schnorr. A Hierarchy of Polynomial Time Lattice Basis Reduction Algorithms. *Theor. Comput. Sci.*, 53:201–224, 1987.

[21] C. Schnorr. A More Efficient Algorithm for Lattice Basis Reduction. *J. Algorithms*, 9(1):47–62, 1988.

[22] C. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66:181–199, 1994.

[23] V. Shoup. NTL 9.8.1, a library for doing number theory. Available at `http://www.shoup.net/ntl/`.

[24] C. L. Siegel. A mean value theorem in geometry of numbers. *Annals of Mathematics*, 46(2):340–347, 1945.

[25] T. Sunaga. Theory of an interval algebra and its application to numerical analysis. *Japan J. Indust. Appl. Math.*, 26, 10 2009.

[26] B. M. M. D. Weger. Solving exponential Diophantine equations using lattice basis reduction algorithms. *J. NUMBER THEORY*, 26:325–367, 1987.

[27] R. C. Young. *The algebra of many-values quantities.* PhD thesis, Cambridge, 1931.