An LLL-Reduction Algorithm with Quasi-linear Time Complexity

[Extended Abstract] *

Andrew Novocin
andrew.novocin@ens-lyon.fr

Damien Stehlé
damien.stehle@ens-lyon.fr

Gilles Villard
gilles.villard@ens-lyon.fr

CNRS, ENS Lyon, INRIA, UCBL, U. Lyon
Laboratoire LIP, 46 Allée d'Italie, 69364 Lyon Cedex 07, France

ABSTRACT

We devise an algorithm, $\widetilde{\mathrm{L}}^1$, with the following specifications: It takes as input an arbitrary basis $B = (\mathbf{b}_i)_i \in \mathbb{Z}^{d\times d}$ of a Euclidean lattice L; It computes a basis of L which is reduced for a mild modification of the Lenstra-Lenstra-Lovász reduction; It terminates in time $O(d^{5+\varepsilon}\beta + d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$ where $\beta = \log \max \|\mathbf{b}_i\|$ (for any $\varepsilon > 0$ and ω is a valid exponent for matrix multiplication). This is the first LLL-reducing algorithm with a time complexity that is quasi-linear in β and polynomial in d.

The backbone structure of $\widetilde{\mathrm{L}}^1$ is able to mimic the Knuth-Schönhage fast gcd algorithm thanks to a combination of cutting-edge ingredients. First the bit-size of our lattice bases can be decreased via truncations whose validity are backed by recent numerical stability results on the QR matrix factorization. Also we establish a new framework for analyzing unimodular transformation matrices which reduce shifts of reduced bases, this includes bit-size control and new perturbation tools. We illustrate the power of this framework by generating a family of reduction algorithms.

Categories and Subject Descriptors

I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Euclidean lattice, LLL lattice basis reduction

*A preliminary full version [24] of this paper with all claim proofs can be found at
`http://prunel.ccsd.cnrs.fr/ensl-00534899`

STOC'11, June 6–8, 2011, San Jose, California, USA.

1. INTRODUCTION

We present the first lattice reduction algorithm which has complexity both quasi-linear in the bit-length of the entries and polynomial time overall for an input basis $B = (\mathbf{b}_i)_i \in \mathbb{Z}^{d\times d}$. This is the first progress on quasi-linear lattice reduction in nearly 10 years, improving Schönhage [29], Yap [33], and Eisenbrand and Rote [7] whose algorithm is exponential in d. Our result can be seen as a generalization of the Knuth-Schönhage quasi-linear GCD [13, 27] from integers to matrices. For solving the matrix case difficulties which relate to multi-dimensionality we combine several new main ingredients. We establish a theoretical framework for analyzing and designing general lattice reduction algorithms. In particular we discover an underlying structure on any transformation matrix which reduces shifts of reduced lattices; this new structure reveals some of the inefficiencies of traditional lattice reduction algorithms. The multi-dimensional difficulty also leads us to establish new perturbation analysis results for mastering the complexity bounds. The Knuth-Schönhage scalar approach essentially relies on truncations of the Euclidean remainders [13, 27] , while the matrix case requires truncating both the "remainder" and "quotient" matrices. We can use our theoretical framework to propose a family of new reduction algorithms, which includes a Lehmer-type sub-quadratic algorithm in addition to $\widetilde{\mathrm{L}}^1$.

In 1982, Lenstra, Lenstra and Lovász devised an algorithm, L^3, that computes reduced bases of integral Euclidean lattices (i.e., subgroups of a $\mathbb{Z}^d$) in polynomial time [16]. This typically allows one to solve approximate variants of computationally hard problems such as the Shortest Vector, Closest Vector, and the Shortest Independent Vectors problems (see [18]). L^3 has since proven useful in dozens of applications in a wide range including cryptanalysis, computer algebra, communications theory, combinatorial optimization, algorithmic number theory, etc (see [22, 6] for two recent surveys).

In [16], Lenstra, Lenstra and Lovász bounded the bit-complexity of L^3 by $\mathcal{O}(d^{5+\varepsilon}\beta^{2+\varepsilon})$ when the input basis $B = (\mathbf{b}_i)_i \in \mathbb{Z}^{d\times d}$ satisfies $\max \|\mathbf{b}_i\| \le 2^\beta$. For the sake of simplicity, we will only consider full-rank lattices. The current best algorithm for integer multiplication is Fürer's, which allows one to multiply two k-bit long integers in time $\mathcal{M}(k) = \mathcal{O}(k(\log k)2^{\log^* k})$. The analysis of L^3 was quickly refined by Kaltofen [11], who showed a $\mathcal{O}(d^5\beta^2(d+\beta)^\varepsilon)$ complexity bound. Schnorr [25] later proposed an algorithm of bit-complexity $\mathcal{O}(d^4\beta(d+\beta)^{1+\varepsilon})$, using approximate computations for internal Gram-Schmidt orthogonalizations. Some works have since focused on improving the complexity bounds with respect to the dimension d, including [28, 31, 14, 26], but they have not lowered the cost with respect to β (for fixed d). More recently, Nguyen and Stehlé devised L^2 [21], a variant of L^3 with complexity $\mathcal{O}(d^{4+\varepsilon}\beta(d+\beta))$. The latter bound is quadratic with respect to β (even with naive integer multiplication), which led to the name L^2. The same complexity bound was also obtained in [20] for a different algorithm, H-LLL, but with a simpler complexity analysis.

As a broad approximation, L^3, L^2 and H-LLL are generalizations of Euclid's greatest common divisor algorithm. The successive bases computed during the execution play the role of Euclid's remainders, and the elementary matrix operations performed on the bases play the role of Euclid's quotients. L^3 may be interpreted in such a framework. It is slow because it computes its "quotients" using all the bits from the "remainders" rather than the most significant bits: The cost of computing one Euclidean division in an L^3 way is $\mathcal{O}(\beta^{1+\varepsilon})$, leading to an overall $\mathcal{O}(\beta^{2+\varepsilon})$ bound for Euclid's algorithm. Lehmer [15] proposed an acceleration of Euclid's algorithm by the means of truncations. Since the ℓ most significant bits of the remainders provide the first $\Omega(\ell)$ bits of the sequence of quotients, one may: Truncate the remainders to precision ℓ; Compute the sequence of quotients for the truncated remainders; Store the first $\Omega(\ell)$ bits of the quotients into an $\Omega(\ell)$-bit matrix; Apply the latter to the input remainders, which are shortened by $\Omega(\ell)$ bits; And iterate. The cost gain stems from the decrease of the bit-lengths of the computed remainders. Choosing $\ell \approx \sqrt{\beta}$ leads to a complexity bound of $\mathcal{O}(\beta^{3/2+\varepsilon})$. In the early 70's, Knuth [13] and Schönhage [27] independently observed that using Lehmer's idea recursively leads to a gcd algorithm with complexity bound $\mathcal{O}(\beta^{1+\varepsilon})$. The above approach for the computation of gcds has been successfully adapted to two-dimensional lattices [33, 29, 5], and the resulting algorithm was then used in [7] to reduce lattices in arbitrary dimensions in quasi-linear time. Unfortunately, the best known cost bound for the latter is $\mathcal{O}(\beta^{1+\varepsilon}(\log\beta)^{d-1})$ for fixed d.

Our result. We adapt the Lehmer-Knuth-Schönhage gcd framework to the case of LLL-reduction. $\widetilde{\mathrm{L}}^1$ takes as input a non-singular $B \in \mathbb{Z}^{d\times d}$; terminates within $\mathcal{O}(d^{5+\varepsilon}\beta + d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$ bit operations, where $\beta = \log\max\|\mathbf{b}_i\|$; and returns a basis of the lattice $L(B)$ spanned by B which is LLL-reduced in the sense of Definition 1 given hereafter. (L^3 reduces bases for $\Xi = (3/4, 1/2, 0)$.) The time bound is obtained via an algorithm that can multiply two $d\times d$ matrices in $\mathcal{O}(d^\omega)$ scalar operations. (We can set $\omega \approx 2.376$ [4].) Our complexity improvement is particularly relevant for applications of LLL reduction where β is large. These include the recognition of algebraic numbers [12] and Coppersmith's method for finding the small roots of polynomials [3].

Definition 1 ([2, Def. 5.3]). *Let $\Xi = (\delta, \eta, \theta)$ with $\eta \in (1/2, 1)$, $\theta > 0$ and $\delta \in (\eta^2, 1)$. Let $B \in \mathbb{R}^{d\times d}$ be non-singular with QR factorization $B = Q\cdot R$ (i.e., the unique decomposition of B as a product of an orthogonal matrix and an upper triangular matrix with positive diagonal entries). The matrix B is Ξ-LLL-reduced if:*

- *for all $i < j$, we have $|r_{i,j}| \leq \eta r_{i,i} + \theta r_{j,j}$ (B is size-reduced);*
- *for all i, we have $\delta \cdot r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ (B is said to satisfy Lovász' conditions).*

Let $\Xi_i = (\delta_i, \eta_i, \theta_i)$ be valid LLL-parameters for $i \in \{1, 2\}$. We say that Ξ_1 is stronger than Ξ_2 and write $\Xi_1 > \Xi_2$ if $\delta_1 > \delta_2$, $\eta_1 < \eta_2$ and $\theta_1 < \theta_2$.

This modified LLL-reduction is as powerful as the classical one (note that by choosing (δ, η, θ) close to the ideal parameters $(1, 1/2, 0)$, the derived α tends to $2/\sqrt{3}$):

Theorem 1 ([2, Th. 5.4]). *Let $B \in \mathbb{R}^{d\times d}$ be (δ, η, θ)-LLL-reduced with R-factor R. Let $\alpha = \frac{\eta\theta+\sqrt{(1+\theta^2)\delta-\eta^2}}{\delta-\eta^2}$. Then, for all i, $r_{i,i} \leq \alpha \cdot r_{i+1,i+1}$ and $r_{i,i} \leq \|\mathbf{b}_i\| \leq \alpha^i \cdot r_{i,i}$. This implies that $\|\mathbf{b}_1\| \leq \alpha^{\frac{d-1}{2}} |\det B|^{1/d}$ and $\alpha^{i-d} r_{i,i} \leq \lambda_i \leq \alpha^i r_{i,i}$, where λ_i is the ith minimum of the lattice $L(B)$.*

$\widetilde{\mathrm{L}}^1$ and its analysis rely on two recent lattice reduction techniques (described below), whose contributions can be easily explained in the gcd framework. The efficiency of the fast gcd algorithms [13, 27] stems from two sources: Performing operations on truncated remainders is meaningful (which allows one to consider remainders with smaller bit-sizes), and the obtained transformations corresponding to the quotients sequence have small bit-sizes (which allows one to transmit at low cost the information obtained on the truncated remainders back to the genuine remainders). We achieve an analogue of the latter by *gradually feeding the input* to the reduction algorithm, and the former is ensured thanks to the modified notion of LLL-reduction which is *resilient to truncations*.

The main difficulty in adapting the fast gcd framework lies in the multi-dimensionality of lattice reduction. In particular, the basis vectors may have significantly differing magnitudes. This means that basis truncations must be performed vector-wise. (Column-wise using the matrix setting.) Also, the resulting unimodular transformation matrices (integral with determinant ± 1 so that the spanned lattice is preserved) may have large magnitudes, hence need to be truncated for being be stored on few bits.

To solve these dilemmas we focus on reducing bases which are a mere scalar shift from being reduced. We call this process *lift-reducing*, and it can be used to provide a family of new reduction algorithms. We illustrate in Section 2 that the general lattice reduction problem can be reduced to the problem of lift-reduction. Indeed, the LLL-reduction of B can be implemented as a sequence of lift-reductions by performing a Hermite Normal Form (HNF) computation on B beforehand. Note that there could be other means of seeding the lift-reduction process. Our lift-reductions are a generalization of recent gradual feeding algorithms.

Gradual feeding of the input. Gradual feeding was introduced by Belabas [1], Novocin, and van Hoeij [23, 10], in the context of specific lattice bases that are encountered while factoring rational polynomials (e.g., with the algorithm from [9]). Gradual feeding was restricted to reducing specific sub-lattices which avoid the above dimensionality difficulties. We generalize these results to the following. Suppose that we wish to reduce a matrix B with the property that $B_0 := \sigma_\ell^{-k} B$ is reduced for some k and σ_ℓ is the diagonal matrix $\mathrm{diag}(2^\ell, 1, \ldots, 1)$. If one runs L^3 on B directly then the structure of B_0 is not being exploited. Instead, the matrix B can be slowly reduced allowing us to control and understand the intermediate transformations: Compute the unimodular transform U_1 (with any reduction algorithm) such that $\sigma_\ell B_0 U_1$ is reduced and repeat until we have $\sigma_\ell^k B_0 U_1 \cdots U_k = B(U_1 \cdots U_k)$. Each entry of U_i and each entry of $U_1 \cdots U_i$ can be bounded sensitive to the shape of the lattice. Further we will illustrate that the bit-size of any entry of U_i can be made $\mathcal{O}(\ell+d)$ (see Theorems 2 and 4).

In addition, control over U gives us the ability to analyze the impact of efficient truncations on lift-reductions.

Truncations of basis matrices. In order to work on as few bits of basis matrices as possible during our lift-reductions, we apply column-wise truncations. A truncation of precision p replaces a matrix B by a truncated matrix $B + \Delta B$ such that $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-p}$ holds for all i, and only the most significant $p + \mathcal{O}(\log d)$ bits of every column of $B + \Delta B$ are allowed to be non-zero. Each entry of $B + \Delta B$ is an integer multiplied by some power of 2. (In the notation ΔB, Δ does not represent anything, i.e., the matrix ΔB is not a product of Δ and B.) A truncation is an efficiency-motivated column-wise perturbation. The following lemmata explain why we are interested in such perturbations.

Lemma 1 ([2, Se. 2], refined from [8]). *Let $p > 0$, $B \in \mathbb{R}^{d\times d}$ non-singular with R-factor R, and let ΔB with $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-p}$. If $cond(R) = \||R||R^{-1}|\|_2$ (using the induced norm) satisfies $c_0 \cdot cond(R) \cdot 2^{-p} < 1$ with $c_0 = 8d^{3/2}$, then $B+\Delta B$ is non-singular and its R-factor $R+\Delta R$ satisfies $\max \frac{\|\Delta \mathbf{r}_i\|}{\|\mathbf{r}_i\|} \leq c_0 \cdot cond(R) \cdot 2^{-p}$.*

Lemma 2 ([2, Le. 5.5]). *If $B \in \mathbb{R}^{d\times d}$ with R-factor R is (δ, η, θ)-reduced then $cond(R) \leq \frac{\rho+1}{\rho-1}\rho^d$, with $\rho = (1+\eta+\theta)\alpha$, with α as in Theorem 1.*

These results imply that a column-wise truncation of a reduced basis with precision $\Omega(d)$ remains reduced. This explains why the parameter θ was introduced in Definition 1, as such a property does not hold if LLL-reduction is restricted to $\theta = 0$ (see [30, Se. 3.1]).

Lemma 3 ([2, Co. 5.1]). *Let $\Xi_1 > \Xi_2$ be valid reduction parameters. There exists a constant c_1 such that for any Ξ_1-reduced $B \in \mathbb{R}^{d\times d}$ and any ΔB with $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-c_1 \cdot d}$, the matrix $B + \Delta B$ is non-singular and Ξ_2-reduced.*

As we will see in Section 3 (see Lemma 7) the latter lemmata will allow us to develop the gradual reduction strategy with truncation, which is to approximate the matrix to be reduced, reduce that approximation, and apply the unimodular transform to the original matrix, and repeat the process.

Lift-$\widetilde{\mathrm{L}}^1$. Our quasi-linear general lattice reduction algorithm, $\widetilde{\mathrm{L}}^1$, is composed of a sequence of calls to a specialized lift-reduction algorithm, Lift-$\widetilde{\mathrm{L}}^1$. Sections 2 and 4.4 shows the relationship between general reduction and lift-reduction via HNF. When we combine lift-reduction (gradual feeding) and truncation we see another difficulty which must be addressed. That is, lift-reducing a truncation of B_0 will not give the same transformation as lift-reducing B_0 directly; likewise any truncation of U weakens our reduction even further. Thus after working with truncations we must apply any transformations to a higher precision lattice and refine the result. In other words, we will need to have a method for strengthening the quality of a weakly reduced basis. Such an algorithm exists in [19] and we adapt it to performing lift-reductions in section 3.2. Small lift-reductions with this algorithm also become the leaves of our recursive tree. The Lift-$\widetilde{\mathrm{L}}^1$ algorithm in Figure 4 is a rigorous implementation of the pseudo algorithm in Figure 1: Lift-$\widetilde{\mathrm{L}}^1$ must refine current matrices more often than this pseudo algorithm to properly handle a specified reduction.

Inputs: B_0 reduced, and target lift ℓ.
Output: U_{small} such that $\sigma_\ell B_0 U_{\text{small}}$ is reduced.

1. Get $U_{1,\text{small}}$ from `pseudo-Lift-`$\widetilde{\mathrm{L}}^1$(`truncate`$(B_0), \ell/2$).
2. $B_1 := \sigma_{\ell/2} B_0 U_{1,\text{small}}$.
3. Get U from `refineReduction`(C).
4. Get $U_{2,\text{small}}$ from `pseudo-Lift-`$\widetilde{\mathrm{L}}^1$(`truncate`$(B_1 U), \ell/2$).
5. U_{small} :=`clean`$(U_{1,\text{small}} \cdot U \cdot U_{2,\text{small}})$.
6. Return U_{small}.

Figure 1: `pseudo-Lift-`$\widetilde{\mathrm{L}}^1$.

It could be noted that `clean` is stronger than mere truncation. It can utilize our new understanding of the structure of any lift-reducing U to provide an appropriate transformation which is well structured and efficiently stored.

Comments on the cost of $\widetilde{\mathrm{L}}^1$. The term $\mathcal{O}(d^{5+\varepsilon}\beta)$ stems from a series of β calls to H-LLL [20] or L^2 [21] on integral matrices whose entries have bit-lengths $\mathcal{O}(d)$. These calls are at the leaves of the tree of the recursive algorithm. An amortized analysis allows us to show that the total number of LLL switches performed summed over all calls is $\mathcal{O}(d^2\beta)$ (see Lemma 11). We recall that known LLL reduction algorithms perform two types of vector operations: Either translations or switches. The number of switches performed is a key factor of the complexity bounds. The H-LLL component of the cost of $\widetilde{\mathrm{L}}^1$ could be lowered by using faster LLL-reducing algorithms than H-LLL (with respect to d), but for our amortization to hold, they have to satisfy a standard property (see Section 3.2). The term $\mathcal{O}(d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$ derives from both the HNF computation mentioned above and a series of product trees of balanced matrix multiplications whose overall product has bit-length $\mathcal{O}(d\beta)$. Furthermore, the precise cost dependence of $\widetilde{\mathrm{L}}^1$ in β is $\mathcal{P}oly(d) \cdot \mathcal{M}(\beta)\log\beta$. We also remark that the cost can be proven to be $\mathcal{O}(d^{4+\varepsilon}\log|\det B| + d^{5+\varepsilon} + d^{\omega}(\log|\det B|)^{1+\varepsilon}) + \mathcal{H}(d,\beta)$, where $\mathcal{H}(d,\beta)$ denotes the cost of computing the Hermite normal form. Finally, we may note that if the size-reduction parameter θ is not considered as a constant, then a factor $\mathcal{P}oly(\log(1/\theta))$ is involved in the cost of the leaf calls.

Road-map. We construct $\widetilde{\mathrm{L}}^1$ in several generalization steps which, in the gcd framework, respectively correspond to Euclid's algorithm (Section 2), Lehmer's inclusion of truncations in Euclid's algorithm (Section 3) and the Knuth-Schönhage recursive generalization of Lehmer's algorithm (Section 4).

2. LIFT-REDUCTION

In order to enable the adaptation of the gcd framework to lattice reduction, we introduce a new type of reduction which behaves more predictively and regularly. In this new framework, called lift-reduction, we are given a reduced matrix B and a lifting target $\ell \geq 0$, and we aim at computing a unimodular U such that $\sigma_\ell BU$ is reduced (with $\sigma_\ell = \mathrm{diag}(2^\ell, 1, \ldots, 1)$). Lift-reduction can naturally be performed using any general purpose reduction algorithm, however we will design fast algorithms specific to lift-reduction in Sections 3 and 4. Lifting a lattice basis has a predictable impact on the $r_{i,i}$'s and the successive minima.

LEMMA 4. *Let B be non-singular and $\ell \geq 0$. If R (resp. R') is the R-factor of B (resp. $B' = \sigma_\ell B$), then $r'_{i,i} \geq r_{i,i}$ for all i and $\prod r'_{i,i} = 2^\ell \prod r_{i,i}$. Furthermore, if $(\lambda_i)_i$ (resp. $(\lambda'_i)_i$) are the successive minima of $L = L(B)$ (resp. $L' = L(B')$), then $\lambda_i \leq \lambda'_i \leq 2^\ell \lambda_i$ for all i.*

PROOF. The first statement is proven in [10, Le. 4]. For the second one, notice that $\prod r'_{i,i} = |\det B'| = 2^\ell |\det B| = 2^\ell \prod r_{i,i}$. We now prove the third statement. Let $(\mathbf{v}_i)_i$ and $(\mathbf{v}'_i)_i$ be linearly independent vectors in L and L' respectively with $\|\mathbf{v}_i\| = \lambda_i$ and $\|\mathbf{v}'_i\| = \lambda'_i$ for all i. For any i, we define $S'_i = \{\sigma_\ell \mathbf{v}_j, j \leq i\}$ and $S_i = \{\sigma_\ell^{-1} \mathbf{v}'_j, j \leq i\}$. These are linearly independent sets in L' and L respectively. Then for any i we have $\lambda_i \leq \max_{\|\cdot\|}(S_i) \leq \lambda'_i \leq \max_{\|\cdot\|}(S'_i) \leq 2^\ell \lambda_i$. ☐

We can now bound the entries of any matrix which performs lift-reduction.

LEMMA 5. *Let Ξ_1, Ξ_2 be valid parameters and α_1 and α_2 as in Theorem 1. Let $\ell \geq 0$, $B \in \mathbb{R}^{d\times d}$ be Ξ_1-reduced and U such that $C = \sigma_\ell BU$ is Ξ_2-reduced. Letting $\zeta_1 = (1+\eta_1+\theta_1)\alpha_1\alpha_2$, we have:*

$$\forall i,j:\ |u_{i,j}| \leq 4d^3\zeta_1^d \cdot \frac{r'_{j,j}}{r_{i,i}} \leq 2^{\ell+2} d^3 \zeta_1^{2d} \cdot \frac{r_{j,j}}{r_{i,i}},$$

where R (resp. R') is the R-factor of B (resp. C). In addition, if $V = U^{-1}$ and $\zeta_2 = (1+\eta_2+\theta_2)\alpha_2\alpha_1$:

$$\forall i,j:\ |v_{j,i}| \leq 2^{\ell+2} d^3 \zeta_2^d \cdot \frac{r_{i,i}}{r'_{j,j}} \leq 2^{\ell+2} d^3 \zeta_2^{2d} \cdot \frac{r_{i,i}}{r_{j,j}}.$$

PROOF. Let $B = QR$, $C = Q'R'$ be the QR-factorizations of B and C. Then

$$\begin{aligned} U &= R^{-1} Q^t \sigma_\ell^{-1} Q' R' \\ &= \mathrm{diag}(r_{i,i}^{-1}) \bar{R}^{-1} \left(Q^t \sigma_\ell^{-1} Q'\right) \bar{R}' \mathrm{diag}(r'_{j,j}), \end{aligned}$$

with $\bar{R} = R \cdot \mathrm{diag}(1/r_{i,i})$ and $\bar{R}' = R' \cdot \mathrm{diag}(1/r'_{j,j})$. From the proof of [2, Le. 5.5], we know that $|\bar{R}^{-1}| \leq 2((1+\eta_1+\theta_1)\alpha_1)^d T$, where $t_{i,j} = 1$ if $i \leq j$ and $t_{i,j} = 0$ otherwise. By Theorem 1, we have $|\bar{R}'| \leq (\eta_2 \alpha_2^{d-1} + \theta_2) T \leq 2\alpha_2^d T$ (using $\theta_2 \leq \alpha_2$ and $\eta_2 \leq 1$). Finally, we have $|Q|, |Q'| \leq M$, where $m_{i,j} = 1$ for all i, j. Using the triangular inequality, we obtain:

$$\begin{aligned} |U| &\leq 4\zeta^d \mathrm{diag}(r_{i,i}^{-1}) T M^2 T \mathrm{diag}(r'_{j,j}) \\ &\leq 4d^3 \zeta^d \mathrm{diag}(r_{i,i}^{-1}) M \mathrm{diag}(r'_{j,j}). \end{aligned}$$

Now, by Theorem 1 and Lemma 4, we have $r'_{j,j} \leq \alpha_2^{d-j} \lambda'_j \leq 2^\ell \alpha_2^{d-j} \lambda_j \leq 2^\ell \alpha_1^j \alpha_2^{d-j} r_{j,j}$, which completes the proof of the first statement.

For the second statement note that

$$V = \mathrm{diag}({r'}_{i,i}^{-1}) {\bar{R}'}^{-1} \left({Q'}^t \sigma_\ell Q\right) \bar{R} \mathrm{diag}(r_{j,j})$$

is similar to the expression for U in the proof of the first statement, except that σ_ℓ can increase the innermost product by a factor 2^ℓ. ☐

LLL-REDUCTION AS A SEQUENCE OF LIFT-REDUCTIONS. In the remainder of this section we illustrate that LLL-reduction can be achieved with an efficient sequence of lift-reductions. Lift-reduction is specialized to reducing a scalar-shift/lift of an already reduced basis. In Figure 2 we create reduced bases (of distinct lattices from the input lattice) which we use to progressively create a reduced basis for the input lattice. Here we use an HNF triangularization and scalar shifts to find suitable reduced lattice bases. We analyze the cost and accuracy of Figure 2 using a generic lift-reduction algorithm. The remainder of the paper can then focus on specialized lift-reduction algorithms which each use Figure 2 to achieve generic reduction. We note that other wrappers of lift-reduction are possible.

Recall that the HNF of a (full-rank) lattice $L \subseteq \mathbb{Z}^d$ is the unique upper triangular basis H of L such that $-h_{i,i}/2 \leq h_{i,j} < h_{i,i}/2$ for any $i < j$ and $h_{i,i} > 0$ for any i. Using [17, 32], it can be computed in time $O(d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$, where the input matrix $B \in \mathbb{Z}^{d\times d}$ satisfies $\max \|\mathbf{b}_i\| \leq 2^\beta$.

Inputs: LLL parameters Ξ; a non-singular $B \in \mathbb{Z}^{d\times d}$.
Output: A Ξ-reduced basis of $L(B)$.

1. $B := \mathrm{HNF}(B)$.
2. For k from $d-1$ down to 1 do
3. Let C be the bottom-right $(d-k+1)$-dimensional submatrix of B.
4. $\ell_k := \lceil \log_2(b_{k,k}) \rceil$, $C := \sigma_{\ell_k}^{-1} C$.
5. *Lift-reduction:* Find U' unimodular such that $\sigma_{\ell_k} CU'$ is Ξ-reduced.
6. Let U be the block-diagonal matrix $\mathrm{diag}(I, U')$.
7. Compute $B := B \cdot U$, reducing row i symmetrically modulo $b_{i,i}$ for $i < k$.
8. Return B.

Figure 2: Reducing LLL-reduction to lift-reduction.

Let H be the HNF of $L(B)$. At the end of Step 1, the matrix $B = H$ is upper triangular, $\prod b_{i,i} = |\det H| \leq 2^{d\beta}$, and the 1×1 bottom rightmost sub-matrix of H is trivially Ξ-reduced. In each iteration we Ξ-reduce a lower-right sub-matrix of B via lift-reduction (increasing the dimension with each iteration). This is done by augmenting the previous Ξ-reduced sub-matrix by a scaling down of the next row (such that the new values are tiny). This creates a C which is reduced and such that a lift-reduction of C will be a complete Ξ-reduction of the next largest sub-matrix of B. The column operations of the lift-reduction are then applied to rest of B with the triangular structure allowing us to reduce each remaining row modulo $b_{i,i}$. From a cost point of view, it is worth noting that the sum of the lifts ℓ_k is $\mathcal{O}(\log|\det H|) = \mathcal{O}(d\beta)$.

LEMMA 6. *The algorithm of Figure 2 Ξ-reduces B such that $\max \|\mathbf{b}_i\| \leq 2^\beta$ using*

$$\mathcal{O}(d^{\omega+1+\varepsilon}(\beta^{1+\varepsilon} + d)) + \sum_{k=d-1}^{1} \mathcal{C}_k$$

bit operations, where $\mathcal{C}_k$ is the cost of Step 5 for the specific value of k.

PROOF. We first prove the correctness of the algorithm. We let U_H be the unimodular transformation such that $H = BU_H$. For $k < d$, we let U'_k be the $(d-k+1)\times(d-k+1)$ unimodular transformation that reduces $\sigma_{\ell_k} C$ at Step 5 and U''_k be the unimodular transformation that reduces rows

$1 \leq i < k$ at Step 7. With input B the algorithm returns $B \cdot U_H \cdot \mathrm{diag}(I, U'_{d-1}) \cdot U''_{d-1} \ldots \cdot \mathrm{diag}(I, U'_2) \cdot U''_2 \cdot U'_1$. Since B is multiplied by a product of unimodular matrices, the output matrix is a basis of the lattice spanned by the columns of B.

We show by induction on k from d down to 1 that at the end of the $(d-k)$-th loop iteration, the bottom-right $(d-k+1)$-dimensional submatrix of the current B is Ξ-reduced. The statement is valid for $k = d$, as a non-zero matrix in dimension 1 is always reduced, and instanciating the statement with $k = 1$ ensures that the matrix resturned by the algorithm is Ξ-reduced. The non-trivial ingredient of the proof of the statement is to show that for $k < d$, the input of the *lift-reduction* of Step 5 is valid, i.e., that at the beginning of Step 5 the matrix C is Ξ-reduced. Let R be the R-factor of C. Let C' be the bottom-right $(d-k) \times (d-k)$ submatrix of C. By induction, we know that C' is Ξ-reduced. It thus remains to show that the first row of R satisfies the size-reducedness condition, and that Lovász' condition between the first two rows is satisfied. We have $r_{1,j} = h_{k,k+j-1}/2^{\ell_k}$, for $j \leq d-k+1$, thus ensuring the size-reducedness condition. Furthermore, by the shape of the unimodular transformations applied so far, we know that C' is a basis of the lattice L' generated by the columns of the bottom-right $(d-k)$-dimensional submatrix of H, which has first minimum $\lambda_1(L') \geq \min_{i>k} h_{i,i} \geq 1$. As $r_{2,2}$ is the norm of the first vector of C', we have $r_{2,2} \geq \lambda_1(L') \geq 1$. Independently, by choice of ℓ_k, we have $r_{1,1} \leq 1$. This ensures that Lovász' condition is satisfied, and completes the proof of correctness.

We now bound the cost of the algorithm of Figure 2. We bound the overall cost of the $d-1$ calls to lift-reduction by $\sum_{k<d} \mathcal{C}_k$. It remains to bound the contribution of Step 7 to the cost. The cost dominating component of Step 7 is the computation of the product of the last $d-k+1$ columns of (the current value of) B by U'. We consider separately the costs of computing the products by U' of the $k \times (d-k+1)$ top-right submatrix $\overline{B}$ of B, and of the $(d-k) \times (d-k+1)$ bottom-right submatrix $\underline{B}$ of B

For $i \leq k$, the magnitudes of the entries of the i-th row of $\overline{B}$ are uniformly bounded by $h_{i,i}$. By Lemma 5, if $e, j < d-k+1$, then $|u'_{e,j}| \leq 2^{\ell_k+2} d^3 \zeta_1^d \cdot \frac{r_{j,j}}{r_{e,e}}$ (recall that R is the R-factor of C at the beginning of Step 5). As we saw above, we have $r_{2,2} \geq 1$, and, by reducedness, we have $r_{e,e} \geq \alpha^{-e}$ for any $e \geq 2$ (using Theorem 1). Also, by choice of ℓ_k, we have $r_{1,1} \geq 1/2$. Overall, this gives that the jth column of U' is uniformly bounded as $\log \|\mathbf{u}'_j\| = \mathcal{O}(\ell_k + d + \log r_{j,j})$. The bounds on the bit-lengths of the rows of $\overline{B}$ and the bounds on the bit-lengths of the columns of U' may be very unbalanced. We do not perform matrix multiplication naively, as this unbalancedness may lead to too large a cost (the maxima of row and column bounds may be much larger than the averages). To circumvent this difficulty we use Recipe 1 of [24] with "$S = \log \det H + d^2 + d\ell_k$". Since $\det H = |\det B|$ the multiplication of $\overline{B}$ with U' can be performed within $\mathcal{O}(d^\omega \mathcal{M}((\log|\det B|)/d + d + \ell_k))$ bit operations.

We now consider the product $P := \underline{B}U'$. By reducedness of $\underline{B}$, we have $\|\underline{\mathbf{b}}_j\| \leq \alpha^d r_{j,j}$ (from Theorem 1). Recall that we have $|u'_{e,j}| \leq 2^{\ell_k+2} d^3 \zeta_1^d \cdot \frac{r_{j,j}}{r_{e,e}}$. As a consequence, we can uniformly bound $\log \|\mathbf{u}'_j\|$ and $\log \|\mathbf{p}_j\|$ by $\mathcal{O}(\ell_k + d + \log r_{j,j})$ for any j. We can thus use Recipe 3 of [24] to compute P, with "$S = \mathcal{O}(\log \det H + d^2 + d\ell_k)$" using $\mathcal{O}(d^{\omega+\varepsilon} \mathcal{M}((\log|\det B|)/d + d + \ell_k))$ bit operations.

The proof can be completed by noting that the above matrix products are performed $d-1$ times during the execution of the algorithm and by also considering the cost $\mathcal{O}(d^{\omega+1+\varepsilon} \beta^{1+\varepsilon})$ of converting B to Hermite normal form. $\square$

We use the term $\mathcal{C}_k$ in order to amortize over the loop iterations the costs of the calls to the lift-reducing algorithm. In the algorithm of Figure 2 and in Lemma 6, the lift-reducing algorithm is not specified. It may be a general-purpose LLL-reducing algorithm [16, 11, 21, 20] or a specifically designed lift-reducing algorithm such as Lift-$\widetilde{\mathrm{L}}^1$, described in Section 4.

It can be noted from the proof of Lemma 6 that the non-reduction costs can be refined as $\mathcal{O}(d^{\omega+\varepsilon} \mathcal{M}(\log|\det B|) + d^{\omega+1+\varepsilon} \mathcal{M}(d)) + \mathcal{H}(d, \beta)$. We note that the HNF is only used as a triangularization, thus any triangularization of the input B will suffice, however then it may be needed to perform d^2 reductions of entries $b_{i,j}$ modulo $b_{i,i}$. Thus we could replace $\mathcal{H}(d, \beta)$ by $\mathcal{O}(d^2 \beta^{1+\varepsilon})$ for upper triangular inputs. Using the cost of H-LLL for lift-reduction, we can bound the complexity of Figure 2 by $\mathcal{P}oly(d) \cdot \beta^2$. This is comparable to L^2 and H-LLL.

3. TRUNCATING MATRIX ENTRIES

We will now focus on improving the lift-reduction step introduced in the previous section. In this section we show how to truncate the "remainder" matrix and we give an efficient factorization for the "quotient" matrices encountered in the process. This way the unimodular transformations can be found and stored at low cost. In the first part of this section, we show that given any B reduced and $\ell \geq 0$, finding U such that $\sigma_\ell BU$ is reduced can be done by looking at only the most significant bits of each column of B. In the context of gcd algorithms, this is equivalent to saying that the quotients can be computed by looking at the most significant bits of the remainders only. In the gcd case, using only the most significant bits of the remainders allows one to efficiently compute the quotients. Unfortunately, this is where the gcd analogy stops as a lift-reduction transformation U may still have entries that are much larger than the number of bits kept of B. In particular, if the diagonal coefficients of the R-factor of B are very unbalanced, then Lemma 5 does not prevent some entries of U from being as large as the magnitudes of the entries of B (as opposed to just the precision kept). The second part of this section is devoted to showing how to make the bit-size of U and the cost of computing it essentially independent of these magnitudes. In this framework we can then describe and analyze a Lehmer-like lift-reduction algorithm.

3.1 The most significant bits of B suffice for reducing $\sigma_\ell B$

It is a natural strategy to reduce a truncation of B rather than B, but in general it is unclear if some U which reduces a truncation of B would also reduce B even in a weaker sense. However, with lift-reduction we can control the size of U which allows us to overcome this problem. In this section we aim at computing a unimodular U such that $\sigma_\ell BU$ is reduced, when B is reduced, by working on a truncation of B. We use the bounds of Lemma 5 on the magnitude of U to show that a column-wise truncation precision of $\ell + \mathcal{O}(d)$ bits suffices for that purpose.

LEMMA 7. *Let Ξ_1, Ξ_2, Ξ_3 be valid reduction parameters with $\Xi_3 > \Xi_2$. There exists a constant c_3 such that the following holds for any $\ell \geq 0$. Let $B \in \mathbb{R}^{d\times d}$ be Ξ_1-reduced and ΔB be such that $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-\ell - c_3 \cdot d}$. If $\sigma_\ell(B + \Delta B)U$ is Ξ_3-reduced for some U, then $\sigma_\ell BU$ is Ξ_2-reduced.*

The above result implies that to find a U such that $\sigma_\ell BU$ is reduced, it suffices to find U such that $\sigma_\ell(B' \cdot E)U$ is reduced (for a stronger Ξ), for well chosen matrices B' and E, outlined as follows.

DEFINITION 2. *For $B \in \mathbb{Z}^{d\times d}$ with $\beta = \log \max \|\mathbf{b}_j\|$ and precision p, we chose to store the p most significant bits of B, $\mathrm{MSB}_p(B)$, as a matrix product $B'E$ or just the pair (B', E). This pair should satisfy $B' \in \mathbb{Z}^{d\times d}$ with $p = \log \max \|\mathbf{b}'_j\|$, $E = \mathrm{diag}(2^{e_i - p})$ with $e_i \in \mathbb{Z}$ such that $\frac{2^{e_i} - \|\mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^d$, and $\max \frac{\|(\mathbf{b}_j - \mathbf{b}'_j \cdot 2^{e_i - p}\|}{\|\mathbf{b}_j\|} \leq 2^{-p}$.*

3.2 Finding a unimodular U reducing $\sigma_\ell B$ at low cost

The algorithm `TrLiftLLL` (a truncated lift-LLL) we propose is an adaptation of the `StrengthenLLL` from [19], which aims at strengthening the LLL-reducedness of an already reduced basis, i.e., Ξ_2-reducing a Ξ_1-reduced basis with $\Xi_1 < \Xi_2$. One can recover a variant of `StrengthenLLL` by setting $\ell = 0$ below. We refer the reader to [24] for a complete description of `TrLiftLLL`.

THEOREM 2. *For any valid parameters $\Xi_1 < \Xi_2$ and constant c_4, there exists a constant c'_4 and an algorithm `TrLiftLLL` with the following specifications. It takes as inputs $\ell \geq 0$, $B \in \mathbb{Z}^{d\times d}$ and $E = \mathrm{diag}(2^{e_i})$ with $\max \|\mathbf{b}_i\| \leq 2^{c_4(\ell + d)}$, $e_i \in \mathbb{Z}$ and BE is Ξ_1-reduced; It runs in time $O(d^{2+\varepsilon}(d+\ell)(d+\ell+\tau) + d^2 \log \max(1 + |e_i|))$, where $\tau = O(d^2(\ell + d))$ is the number of switches performed during the single call it makes to H-LLL; And it returns two matrices U and D such that:*

1. *$D = \mathrm{diag}(2^{d_i})$ with $d_i \in \mathbb{Z}$ satisfying $\max |e_i - d_i| \leq c'_4(\ell + d)$,*
2. *U is unimodular and $\max |u_{i,j}| \leq 2^{\ell + c'_4 \cdot d}$,*
3. *$D^{-1}UD$ is unimodular and $\sigma_\ell(BE)(D^{-1}UD)$ is Ξ_2-reduced.*

When setting $\ell = \mathcal{O}(d)$, we obtain the base case of lift-$\widetilde{\mathrm{L}}^1$, the quasi-linear time recursive algorithm to be introduced in the next section. The most expensive step of `TrLiftLLL` is a call to an LLL-type algorithm, which must satisfy a standard property that we identify hereafter.

When called on a basis matrix B with R-factor R, the L^3, L^2 and H-LLL algorithms perform two types of basis operations: They either subtract to a vector $\mathbf{b}_k$ an integer combination of $\mathbf{b}_1, \ldots, \mathbf{b}_{k-1}$ (translation), or they exchange $\mathbf{b}_{k-1}$ and $\mathbf{b}_k$ (switches). Translations leave the $r_{i,i}$'s unchanged. Switches are never perfomed when the optimal Lovász condition $r_{i,i}^2 \leq r_{i,i+1}^2 + r_{i+1,i+1}^2$ is satisfied, and thus cannot increase any of the quantities $\max_{j\leq i} r_{j,j}$ (for varying i), nor decrease any of the quantities $\min_{j\geq i} r_{j,j}$. This implies that if we have $\max_{i<k} r_{i,i} < \min_{i\geq k} r_{i,i}$ for some k at the beginning of the execution, then the computed matrix U will be such that $u_{i,j} = 0$ for any (i,j) such that $i \geq k$ and $j < k$.

We say that a LLL-reducing algorithm satisfies Property (P) if for any k such that $\max_{i<k} r_{i,i} < \min_{i\geq k} r_{i,i}$ holds at the beginning of the execution, then it also holds at the end of the execution.

Property (P) is for instance satisfied by L^3 ([16, p. 523]), L^2 ([21, Th. 6]) and H-LLL ([20, Th. 4.3]). We choose H-LLL as this currently provides the best complexity bound, although $\widetilde{\mathrm{L}}^1$ would remain quasi-linear with L^3 or L^2.

`TrLiftLLL` will also be used with $\ell = 0$ in the recursive algorithm for strengthening the reduction parameters. Such refinement is needed after the truncation of bases and transformation matrices which we will need to ensure that the recursive calls get valid inputs.

3.3 A Lehmer-like lift-LLL algorithm

By combining Lemma 7 and Theorem 2, we obtain a Lehmer-like Lift-LLL algorithm, given in Figure 3. In the input, we assume the base-case lifting target t divides ℓ. If it is not the case, we may replace ℓ by $t\lfloor \ell/t \rfloor$, and add some more lifting at the end.

Inputs: LLL parameters Ξ; a Ξ-reduced matrix $B \in \mathbb{Z}^{d\times d}$; a lifting target ℓ; a divisor t of ℓ.
Output: A Ξ-reduced basis of $\sigma_\ell B$.

1. Let Ξ_0, Ξ_1 be valid parameters with $\Xi_0 < \Xi < \Xi_1$, c_3 as in Le. 7 for "$(\Xi_1, \Xi_2, \Xi_3) := (\Xi, \Xi, \Xi_1)$", c_1 as in Le. 3 with "$(\Xi_1, \Xi_2) := (\Xi, \Xi_0)$", and c'_4 as in Th. 2 with "$(\Xi_1, \Xi_2, c_4) := (\Xi_0, \Xi_1, c_3 + 2)$".
2. For k from 1 to ℓ/t do
3. $\quad (B', E) := \mathrm{MSB}_{(t + c_3 d)}(B)$.
4. $\quad (D, U) :=$ `TrLiftLLL`(B', E, t).
5. $\quad B := \sigma_t B D^{-1} U D$.
6. Return B.

Figure 3: The `Lehmer-LiftLLL` algorithm.

THEOREM 3. `Lehmer-LiftLLL` *is correct. Furthermore, if the input matrix B satisfies $\max \|\mathbf{b}_i\| \leq 2^\beta$, then its bit-complexity is $O(d^3 \ell (d^{1+\varepsilon} t + t^{-1+\varepsilon}(\ell + \beta)))$.*

Note that if ℓ is sufficiently large with respect to d, then we may choose $t = \ell^a$ for $a \in (0, 1)$, to get a complexity bound that is subquadratic with respect to ℓ. By using `Lehmer-LiftLLL` at Step 5 of the algorithm of Figure 2 (with $t = \ell^{.5}$), it is possible to obtain an LLL-reducing algorithm of complexity $\mathcal{P}oly(d) \cdot \beta^{1.5+\varepsilon}$.

4. QUASI-LINEAR ALGORITHM

We now aim at constructing a recursive variant of the `Lehmer-LiftLLL` algorithm of the previous section. Because the lift-reducing unimodular transformations will be produced by recursive calls, we have little control over their structure (as opposed to those produced by `TrLiftLLL`). Before describing Lift-$\widetilde{\mathrm{L}}^1$, we thus study lift-reducing unimodular transformations, without considering how they were computed. In particular, we are interested in how to work on them at low cost. This study is robust and fully general, and afterwards is used to analyze lift-$\widetilde{\mathrm{L}}^1$.

4.1 Sanitizing unimodular transforms

In the previous section we have seen that working on the most significant bits of the input matrix B suffices to find a matrix U such that $\sigma_\ell BU$ is reduced. Furthermore, as shown in Theorem 2, the unimodular U can be found and stored on few bits. Since the complexity of Theorem 2 is quadratic in ℓ we will use it only for small lift-reductions (the leaves of our recursive tree) and repairing reduction quality (when $\ell = 0$). For large lifts we will use recursive lift-reduction. However, that means we no longer have a direct application of a well-understood LLL-reducing algorithm which was what allowed such efficient unimodular transforms to be found. Thus, in this section we show how any U which reduces $\sigma_\ell B$ can be transformed into a factored unimodular U' which also reduces $\sigma_\ell B$ and for which each entry can be stored with only $\mathcal{O}(\ell + d)$ bits. We also explain how to quickly compute the products of such factored matrices. This analysis can be used as a general framework for studying lift-reductions.

The following lemmata work because lift-reducing transforms have a special structure which we gave in Lemma 5. Here we show a class of additive perturbations which, when viewed as a transformations, are in fact unimodular transformations themselves. Note that these entry-wise perturbations are stronger than mere truncations since $\Delta u_{i,j}$ could be larger than $u_{i,j}$. Lemma 8 shows that a sufficiently small perturbation of a unimodular lift-reducing matrix remains unimodular.

LEMMA 8. *Let Ξ_1, Ξ_2 be valid LLL parameters. There exists a contant c_7 such that the following holds for any $\ell \geq 0$. Let $B \in \mathbb{R}^{d\times d}$ (with R-factor R) be Ξ_1-reduced, and U be unimodular such that $\sigma_\ell BU$ (with R-factor R') is Ξ_2-reduced. If $\Delta U \in \mathbb{Z}^{d\times d}$ satisfies $|\Delta u_{i,j}| \leq 2^{-(\ell+c_7\cdot d)} \cdot \frac{r'_{j,j}}{r_{i,i}}$ for all i, j, then $U + \Delta U$ is unimodular.*

PROOF. Since U is unimodular, the matrix $V = U^{-1}$ exists and has integer entries. We can thus write $U + \Delta U = U(I + U^{-1}\Delta U)$, and prove the result by showing that $U^{-1}\Delta U$ is strictly upper triangular, i.e., that $(U^{-1}\Delta U)_{i,j} = 0$ for $i \geq j$. We have $(U^{-1}\Delta U)_{i,j} = \sum_{k\leq d} v_{i,k} \cdot \Delta u_{k,j}$. We now show that if $\Delta u_{k,j} \neq 0$ and $i \geq j$, then we must have $v_{i,k} = 0$ (for a large enough c_7).

The inequality $\Delta u_{k,j} \neq 0$ and the hypothesis on ΔU imply that $\frac{r_{k,k}}{r'_{j,j}} \leq 2^{-(\ell+c_7\cdot d)}$. Since $i \geq j$ and $\sigma_\ell BU$ is reduced, Theorem 1 implies that $\frac{r_{k,k}}{r'_{i,i}} \leq 2^{-\ell+(c-c_7)d}$, for some constant $c > 0$. By using the second part of Lemma 5, we obtain that there exists $c' > 0$ such that $|v_{i,k}| \leq 2^{\ell+c'\cdot d} \cdot \frac{r_{k,k}}{r'_{i,i}} \leq 2^{(c+c'-c_7)d}$. As V is integral, setting $c_7 > c + c'$ allows us to ensure that $v_{i,k} = 0$, as desired. $\square$

Lemma 9 shows that a sufficiently small perturbation of a unimodular lift-reducing matrix remains lift-reducing.

LEMMA 9. *Let Ξ_1, Ξ_2, Ξ_3 be valid LLL parameters such that $\Xi_2 > \Xi_3$. There exists a contant c_8 such that the following holds for any $\ell \geq 0$. Let $B \in \mathbb{R}^{d\times d}$ (with R-factor R) be Ξ_1-reduced, and U be unimodular such that $\sigma_\ell BU$ (with R-factor R') is Ξ_2-reduced. If $\Delta U \in \mathbb{Z}^{d\times d}$ satisfies $|\Delta u_{i,j}| \leq 2^{-(\ell+c_8\cdot d)} \cdot \frac{r'_{j,j}}{r_{i,i}}$ for all i, j, then $\sigma_\ell B(U + \Delta U)$ is Ξ_3-reduced.*

PROOF. We proceed by showing that $|\sigma_\ell B\Delta U|$ is columnwise small compared to $|\sigma_\ell BU|$ and by applying Lemma 3. We have $|\Delta U| \leq 2^{-(\ell+c_8\cdot d)}\mathrm{diag}(r_{i,i}^{-1})C\mathrm{diag}(r'_{j,j})$ by assumption, where $c_{i,j} = 1$ for all i, j. Since B is Ξ_1-reduced, we also have $|R| \leq \mathrm{diag}(r_{i,i})T + \theta_1 T\mathrm{diag}(r_{j,j})$, where T is upper triangular with $t_{i,j} = 1$ for all $i \leq j$. Then using $|R\Delta U| \leq |R||\Delta U|$ we get

$$|R\Delta U| \leq 2^{-(\ell+c_8\cdot d)}\big(\mathrm{diag}(r_{i,i})T\mathrm{diag}(r_{j,j}^{-1}) + \theta_1 T\big)C\mathrm{diag}(r'_{j,j}).$$

Since B is Ξ_1-reduced, by Theorem 1, we have $r_{i,i} \leq \alpha_1^d r_{j,j}$ for all $i \leq j$, hence it follows that

$$|R\Delta U| \leq 2^{-(\ell+c_8\cdot d)}(\alpha_1^d + \theta_1)TC\mathrm{diag}(r'_{j,j}).$$

As a consequence, there exists a constant $c > 0$ such that for any j:

$$\|(\sigma_\ell B\Delta U)_j\| \leq 2^\ell\|(B\Delta U)_j\| = 2^\ell\|(R\Delta U)_j\| \leq 2^{(c-c_8)d}r'_{j,j}.$$

We complete the proof by noting that $r'_{j,j} \leq \|(\sigma_\ell BU)_j\|$ and by applying Lemma 3 (which requires that c_8 is set sufficiently large). $\square$

Lemmata 8 and 9 allow us to design an algorithmically efficient representation for lift-reducing unimodular transforms.

THEOREM 4. *Let Ξ_1, Ξ_2, Ξ_3 be valid LLL parameters with $\Xi_2 > \Xi_3$. There exist contants $c_9, c_{10} > 0$ such that the following holds for any $\ell \geq 0$. Let $B \in \mathbb{R}^{d\times d}$ be Ξ_1-reduced, and U be unimodular such that $\sigma_\ell BU$ is Ξ_2-reduced. Let $d_i := \lfloor \log \|\mathbf{b}_i\| \rfloor$ for all i. Let $D := \mathrm{diag}(2^{d_i})$, $x := \ell + c_9 \cdot d$, $\widehat{U} := 2^x DUD^{-1}$ and $U' := 2^{-x}D^{-1}\lfloor \widehat{U} \rceil D$. We write $\texttt{Clean}(U, (d_i)_i, \ell) := (U', D, x)$. Then U' is unimodular and $\sigma_\ell BU'$ is Ξ_3-reduced. Furthermore, the matrix $\widehat{U}$ satisfies $\max |\widehat{u}_{i,j}| \leq 2^{2\ell+c_{10}\cdot d}$.*

PROOF. We first show that U' is integral. If $\lfloor \widehat{u}_{i,j} \rceil = \widehat{u}_{i,j}$, then $u'_{i,j} = u_{i,j} \in \mathbb{Z}$. Otherwise, we have $\widehat{u}_{i,j} \notin \mathbb{Z}$, and thus $x + d_i - d_j \leq 0$. This gives that $\lfloor \widehat{u}_{i,j} \rceil \in \mathbb{Z} \subseteq 2^{x+d_i-d_j}\mathbb{Z}$. We conclude that $u'_{i,j} \in \mathbb{Z}$.

Now, consider $\Delta U = U' - U$. Since $\Delta U = 2^{-x}D^{-1}(\lfloor \widehat{U} \rceil - \widehat{U})D$, we have $|\Delta u_{i,j}| \leq 2^{d_j-d_i-x}$, for all i, j. Thus by Theorem 1 and Lemma 4, we have $|\Delta u_{i,j}| \leq 2^{-x+c\cdot d} \cdot \frac{r_{j,j}}{r_{i,i}}$ for some constant c. Applying Lemmata 8 and 9 shows that U' is unimodular and $\sigma_\ell BU'$ is Ξ_3-reduced (if c_9 is chosen sufficiently large).

By Lemma 5, we have for all i, j:

$$|\widehat{u}_{i,j}| = |u_{i,j}|2^{x+d_i-d_j} \leq 2^{x+\ell+c'd} \cdot \frac{r_{j,j}}{2^{\lfloor \log \|\mathbf{b}_j\| \rfloor}} \frac{2^{\lfloor \log \|\mathbf{b}_i\| \rfloor}}{r_{i,i}},$$

for some constant c'. Theorem 1 then provides the result. $\square$

The above representation of lift-reducing transforms is computationally powerful. Firstly, it can be efficiently combined with Theorem 2: Applying the process described in Theorem 4 to the unimodular matrix produced by `TrLiftLLL` may be performed in $O(d^2(d + \ell) + d\log\max(1 + |e_i|))$ bit operations, which is negligible comparable to the cost bound of `TrLiftLLL`. We call `TrLiftLLL'` the algorithm resulting from the combination of Theorems 2 and 4. `TrLiftLLL'` is to be used as base case of the recursion process of `Lift`-$\widetilde{\mathrm{L}}^1$. Secondly, the following result shows how to combine lift-LLL-reducing unimodular transforms. This is an engine of the recursion process of `Lift`-$\widetilde{\mathrm{L}}^1$.

LEMMA 10. *Let $U = 2^{-x}D^{-1}U'D \in \mathbb{Z}^{d\times d}$ with $U' \in \mathbb{Z}^{d\times d}$ and $D = \mathrm{diag}(2^{d_i})$. Let $V = 2^{-y}E^{-1}V'E \in \mathbb{Z}^{d\times d}$ with*

$V' \in \mathbb{Z}^{d\times d}$ *and* $E = \mathrm{diag}(2^{e_i})$. *Let* $\ell \in \mathbb{Z}$ *and* $f_i \in \mathbb{Z}$ *for* $i \leq d$. *Then it is possible to compute the output* (W', F, z) *of* `Clean`$(U \cdot V, (f_i)_i, \ell)$ *(see Theorem 4) from* x, y, ℓ, U', V', $(d_i)_i$, $(e_i)_i$, $(f_i)_i$, *in time* $O(d^\omega \mathcal{M}(t + \log d))$, *where*

$$\max_{i,j} \max(|u'_{i,j}|, |v'_{i,j}|) \leq 2^t$$

and

$$\max_i \max(|d_i - e_i|, |f_i - e_i|, |\ell - (x+y)|) \leq t.$$

For short, we will write $W := U \odot V$, *with* $W = 2^{-z} F^{-1} W' F$ *and* $F = \mathrm{diag}(2^{f_i})$.

Proof. We first compute $m = \max |d_i - e_i|$. We have

$$UV = 2^{(-x-y-m)} \cdot F^{-1} T \cdot F,$$

where

$$T = (FD^{-1}) U' \mathrm{diag}(2^{d_i - e_i + m}) V' (EF^{-1}).$$

Then we compute T. We multiply U' by $\mathrm{diag}(2^{d_i - e_i + m})$, which is a mere multiplication by a non-negative power of 2 of each column of U'. This gives an integral matrix with coefficients of bit-sizes $\leq 3t$. We then multiply the latter by V', which costs $\mathcal{O}(d^\omega \mathcal{M}(t + \log d))$. We multiply the result from the left by (FD^{-1}) and from the right by EF^{-1}. From T, the matrix $\widehat{W}$ of Theorem 4 may be computed and rounded within $\mathcal{O}(d^2 t)$ bit operations. $\square$

It is crucial in the complexity analysis of `Lift-`$\widetilde{\mathrm{L}}^1$ that the cost of the merging process above is independent of the magnitude scalings (d_i, e_i and f_i).

4.2 `Lift-`$\widetilde{\mathrm{L}}^1$ algorithm

The `Lift-`$\widetilde{\mathrm{L}}^1$ algorithm relies on two recursive calls, on MSB, truncations, and on calls to `TrLiftLLL'`. The latter is used as base case of the recursion, and also to strengthen the reducedness parameters (to ensure that the recursive calls get valid inputs). When strengthening, the lifting target is always 0, and we do not specify it explicitly in Figure 4.

Theorem 5. `Lift-`$\widetilde{\mathrm{L}}^1$ *is correct.*

Proof. When $\ell \leq d$ the output is correct by Theorems 2 and 4. In Step 2, Theorems 2 and 4 give that BU_1 is Ξ_2-reduced and that U_1 has the desired format. In Step 3, the constant $c_3 \geq c_1$ is chosen so that Lemma 3 applies now and Lemma 7 will apply later in the proof. Thus B_1 is Ξ_1-reduced and has the correct structure by definition of MSB. Step 4 works (by induction) because B_1 satisfies the input requirements of `Lift-`$\widetilde{\mathrm{L}}^1$. Thus $\sigma_{\ell/2} B_1 U_{R_1}$ is Ξ_1-reduced. Because of the selection of c_3 in Step 3 we know also that $\sigma_{\ell/2} B U_1 U_{R_1}$ is reduced (weaker than Ξ_1) using Lemma 7. Thus by Theorem 4, the matrix B_2 is reduced (weakly) and has an appropriate format for `TrLiftLLL'`. By Theorem 2, the matrix $\sigma_{\ell/2} B U_{1R_1} U_2$ is Ξ_3-reduced and by Theorem 4 we have that $\sigma_{\ell/2} B U_{1R_12}$ is Ξ_2-reduced. By choice of c_3 and Lemma 3, we know that the matrix B_3 is Ξ_1-reduced and satisfies the input requirements of `Lift-`$\widetilde{\mathrm{L}}^1$. Thus, by recursion, we know that $\sigma_{\ell/2} B_3 U_{R_2}$ is Ξ_1-reduced. By choice of c_3 and Lemma 7, the matrix $\sigma_\ell B U_{1R_12} U_{R_2}$ is weakly reduced. By Theorem 4, the matrix B_4 is reduced and satisfies the input requirements of `TrLiftLLL'`. Therefore,

Inputs: Valid LLL-parameters $\Xi_3 > \Xi_2 \geq \Xi_4 > \Xi_1$;
a lifting target ℓ;
$(B', (e_i)_i)$ such that $B = B' \mathrm{diag}(2^{e_i})$ is Ξ_1-reduced
and $\max |b'_{i,j}| \leq 2^{\ell + c \cdot d}$.
Output: $(U', (d_i)_i, x)$ such that $\sigma_\ell B U$ is Ξ_1-reduced,
with $U = 2^{-x} \mathrm{diag}(2^{-d_i}) U' \mathrm{diag}(2^{d_i})$
and $\max |u'_{i,j}| \leq 2^{2\ell + 2c\cdot d}$.

1. If $\ell \leq d$, then use `TrLiftLLL'` with lifting target ℓ. Otherwise:
2. */* Prepare 1st recursive call */* Call `TrLiftLLL'` on (B, Ξ_2); Let U_1 be the output.
3. $B_1 := \mathrm{MSB}_{(\ell/2 + c_3 \cdot d)}(B \cdot U_1)$.
4. */* 1st recursive call */* Call `Lift-L1` on B_1, with lifting target $\ell/2$; Let U_{R_1} be the output.
5. */* Prepare 2nd recursive call */* $U_{1R_1} := U_1 \odot U_{R_1}$.
6. $B_2 := \sigma_{\ell/2} B U_{1R_1}$.
7. Call `TrLiftLLL'` on (B_2, Ξ_3). Let U_2 be the output.
8. $U_{1R_12} := U_{1R_1} \odot U_2$.
9. $B_3 := \mathrm{MSB}_{(\ell/2 + c_3 \cdot d)}(\sigma_{\ell/2} B U_{1R_12})$.
10. */* 2nd recursive call */* Call `Lift-L1` on B_3, with lifting target $\ell/2$; Let U_{R_2} be the output.
11. */* Prepare output */* $U_{1R_12R_2} := U_{1R_12} \odot U_{R_2}$.
12. $B_4 := \sigma_\ell B U_{1R_12R_2}$.
13. Call `TrLiftLLL'` on (B_4, Ξ_4); Let U_3 be the output.
14. $U := U_{1R_12R_2} \odot U_3$; Return U.

Figure 4: The `Lift-`$\widetilde{\mathrm{L}}^1$ algorithm.

the matrix $\sigma_\ell B U_{1R_12R_2}$ is Ξ_4-reduced. Theorem 4 can be used to ensure U has the correct format and $\sigma_\ell B U$ is Ξ_1-reduced. $\square$

4.3 Complexity analysis

Theorem 6. `Lift-`$\widetilde{\mathrm{L}}^1$ *has bit-complexity*

$$\mathcal{O}\left(d^{3+\varepsilon}(d + \ell + \tau) + d^\omega \mathcal{M}(\ell) \log \ell + \ell \log(\beta + \ell)\right),$$

where τ *is the total number of LLL-switches performed by the calls to H-LLL (through* `TrLiftLLL`*), and* $\max |b_{i,j}| \leq 2^\beta$.

Proof. We first bound the total cost of the calls to `TrLiftLLL'`. There are $\mathcal{O}(1 + \ell/d)$ such calls, and for any of these the lifting target is $\mathcal{O}(d)$. Their contribution to the cost of `Lift-`$\widetilde{\mathrm{L}}^1$ is therefore $\mathcal{O}(d^{3+\varepsilon}(d + \ell + \tau))$. Also, the cost of handling the exponents in the diverse diagonal matrices is $\mathcal{O}(d(1 + \ell/d) \log(\beta + \ell))$.

Now, let $\mathcal{C}(d, \ell)$ be the cost of the remaining operations performed by `Lift-`$\widetilde{\mathrm{L}}^1$, in dimension d and with lifting target ℓ. If $\ell \leq d$, then $\mathcal{C}(d, \ell) = \mathcal{O}(1)$ (as the cost of `TrLiftLLL'` has been put aside). Assume now that $\ell > d$. The operations to be taken into account include two recursive calls (each of them costing $\mathcal{C}(d, \ell/2)$), and $\mathcal{O}(1)$ multiplications of d-dimensional integer matrices whose coefficients have bit-length $\mathcal{O}(d + \ell)$. This leads to the inequality $\mathcal{C}(d, \ell) \leq 2\mathcal{C}(d, \ell/2) + K \cdot d^\omega \mathcal{M}(d + \ell)$, for some absolute constant K. This leads to $\mathcal{C}(d, \ell) = \mathcal{O}(d^\omega \mathcal{M}(d + \ell) \log(d + \ell))$. $\square$

4.4 $\widetilde{\mathrm{L}}^1$ algorithm

The algorithm of Figure 4 is the Knuth-Schönhage-like generalization of the Lehmer-like algorithm of Figure 3. Now we are ready to analyze a general lattice reduction algorithm by creating a wrapper for Lift-$\widetilde{\mathrm{L}}^1$.

ALGORITHM $\widetilde{\mathrm{L}}^1$: We define $\widetilde{\mathrm{L}}^1$ as the algorithm from Figure 2, where Figure 5 is used to implement lift-reduction.

As we will see Figure 5 uses the truncation process MSB described in Definition 2 and `TrLiftLLL` to ensure that $\widetilde{\mathrm{L}}^1$ provides valid inputs to `Lift`-$\widetilde{\mathrm{L}}^1$. Its function is to process the input C from Step 5 of Figure 2 (the lift-reduction step) which is a full-precision basis with no special format into a valid input of `Lift`-$\widetilde{\mathrm{L}}^1$ which requires a truncated basis $B' \cdot E$. Just as in `Lift`-$\widetilde{\mathrm{L}}^1$ we use a stronger reduction parameter to compensate for needing a truncation.

Inputs: Valid LLL parameters $\Xi_1 > \Xi$;
C Ξ-reduced with $\beta_k = \log \max \|C\|$;
a lifting target ℓ_k;
Output: U unimodular, such that $\sigma_\ell C U$ is Ξ-reduced

1. $C'F := \mathrm{MSB}_{\ell_k + c_3 d}(C)$
2. Call `TrLiftLLL` on $(C'F, \Xi_1)$. Let $D^{-1}U_0 D$ be the output.
3. $B' := C'FD^{-1}U_0$; $E := D$
4. Call `Lift`-$\widetilde{\mathrm{L}}^1$ on (B', E, Ξ_1). Let U_{ℓ_k} be the output.
5. Return $U := D^{-1}U_0 D U_{\ell_k}$.

Figure 5: From Figure 2 to Lift-$\widetilde{\mathrm{L}}^1$

This processing before `Lift`-$\widetilde{\mathrm{L}}^1$ is similar to what goes on inside of `Lift`-$\widetilde{\mathrm{L}}^1$. The accuracy follows from Lemma 3, Theorem 2, Theorem 5, and Lemma 7. While the complexity of this processing is necessarily less than the bit-complexity of `Lift`-$\widetilde{\mathrm{L}}^1$, $\mathcal{O}(d^{3+\varepsilon}(d+\ell_k+\tau_k)+d^\omega \mathcal{M}(\ell_k)\log \ell_k + \ell_k \log(\beta_k + \ell_k))$ from Theorem 6, which we can use as $\mathcal{C}_k$ from Lemma 6.

We now amortize the costs of all calls to Step 5 using Figure 5. More precisely, we bound $\sum_k \ell_k$ and $\sum_k \tau_k$ more tightly than using a generic bound for the ℓ_k's (resp. τ_k's). For the ℓ_k's, we have $\sum_k \ell_k \leq \log \det H \leq d\beta$. To handle the τ_k's, we adjust the standard LLL energy/potential analysis to allow for the small perturbations of $r_{i,i}$'s due to the various truncations.

LEMMA 11. *Consider the execution of Steps 2–8 of* $\widetilde{\mathrm{L}}^1$ *(Figure 2). Let* $H \in \mathbb{Z}^{d\times d}$ *be the initial Hermite Normal Form. Let* $\Xi_0 = (\delta_0, \eta_0, \theta_0)$ *be the strongest set of LLL-parameters used within the execution. Let* B *be a basis occuring at any moment of Step 5 during the execution. Let* R *be the R-factor of* B *and* n_{MSB} *be the number of times* MSB *has been called so far. We define the energy of* B *as* $\mathcal{E}(B, n_{\mathrm{MSB}}) := \frac{1}{\log 1/\delta_0}\left(\sum_i [(i-1)\cdot \log r_{i,i}] + d^2 n_{\mathrm{MSB}}\right)$ *(using the natural logarithm). Then the number of LLL-switches performed so far satisfies* $\tau \leq \mathcal{E}(B, n_{\mathrm{MSB}}) = \mathcal{O}(d \cdot \log \det H)$.

PROOF. The basis operations modifying the energy function are the LLL switches, the truncations (and returns from truncations), the adjunctions of a vector at Steps 3–4 of the algorithm from Figure 2 and the lifts. We show that any of these operations cannot decrease the energy function.

As Ξ_0 is the strongest set of LLL parameters ever considered during the execution of the algorithm, each LLL switch increases the weighted sum of the $r_{i,i}$'s (see [16, (1.23)]) and hence $\mathcal{E}$ by at least 1.

We now consider truncations. Each increase of n_{MSB} possibly decreases each $r_{i,i}$ (and again when we return from the truncation). We see from Lemma 1 and our choices of precisions p that for any two LLL parameters $\Xi' < \Xi$ there exists an $\varepsilon < 1$ such that each $r_{i,i}$ decreases by a factor no smaller than $(1+\varepsilon)$. Overall, the possible decrease of the weighted sum of the $r_{i,i}$'s is counterbalanced by the term "$d^2 n_{\mathrm{MSB}}$" from the energy function, and hence $\mathcal{E}$ cannot decrease.

Now, the act of adjoining a new row in Figure 2 does not change the previous $r_{i,i}$'s but increases their weights. Since at the moment of an adjoining all $\log r_{i,i}$'s except possibly the first one are non-negative and since the weight of the first one is zero, Steps 3–4 cannot decrease $\mathcal{E}$.

Finally, each product by σ_ℓ (including those within the calls to `TrLiftLLL'`) cannot decrease any $r_{i,i}$, by Lemma 4.

To conclude, the energy never decreases and any switch increases it by at least 1. This implies that the number of switches is bounded by the growth $\mathcal{E}(B, n_{\mathrm{MSB}}) - \mathcal{E}((h_{d,d}), 0)$. The initial value $\mathcal{E}((h_{d,d}), 0)$ of the energy is ≥ 0. Also, at the end of the execution, the term $\sum[(i-1)\log r_{i,i}]$ is $\mathcal{O}(\log \det H)$. As there are 5 calls to MSB in the algorithm from Figure 4 (including those contained in the calls to `TrLiftLLL'`), we can bound $d^2 n_{\mathrm{MSB}}$ by $5d^2 \sum_k (\ell_k/d) = 5 \log \det H$. $\square$

We obtain our main result by combining Theorems 5 and 6, and Lemma 11 to amortize the LLL-costs in Lemma 6 (we bound $\log \det H$ by $d\beta$).

THEOREM 7. *Given as inputs* Ξ *and a matrix* $B \in \mathbb{Z}^{d\times d}$ *with* $\max \|\mathbf{b}_j\| \leq 2^\beta$, *the* $\widetilde{\mathrm{L}}^1$ *algorithm returns a* Ξ*-reduced basis of* $L(B)$ *within* $\mathcal{O}(d^{5+\varepsilon}\beta + d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$ *bit operations.*

Acknowledgements

Andrew Novocin and Damien Stehlé were partly funded by the LaRedA ANR project. Gilles Villard was partly funded by the Gecko ANR project and by a CNRS research collaboration grant to visit the MAGMA computational algebra group of the University of Sydney. Part of this work was done while Damien Stehlé was hosted by Macquarie University and the University of Sydney, whose hospitalities are gratefully acknowledged.

5. REFERENCES

[1] Karim Belabas. A relative van Hoeij algorithm over number fields. *Journal of Symbolic Computation*, 37(5):641–668, 2004.

[2] X.-W. Chang, D. Stehlé, and G. Villard. Perturbation analysis of the QR Factor R in the context of LLL lattice basis reduction. To appear in Mathematics of Computation. HAL Report ensl-00529425, `http://prunel.ccsd.cnrs.fr/ensl-00529425/en`, École Normale Supérieure de Lyon, France, 2010.

[3] D. Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997.

[4] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[5] F. Eisenbrand. Short vectors of planar lattices via continued fractions. *Inf. Process. Lett.*, 79(3):121–126, 2001.

[6] F. Eisenbrand. *50 Years of Integer Programming 1958-2008, From the Early Years to the State-of-the-Art*, chapter Integer Programming and Algorithmic Geometry of Numbers. Springer-Verlag, 2009.

[7] F. Eisenbrand and G. Rote. Fast reduction of ternary quadratic forms. In *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 32–44. Springer-Verlag, 2001.

[8] N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Publications, 2002.

[9] M. van Hoeij. Factoring polynomials and 0-1 vectors. In *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 45–50. Springer-Verlag, 2001.

[10] M. van Hoeij and A. Novocin. Gradual sub-lattice reduction and a new complexity for factoring polynomials. In *Proceedings of the 9th Latin American Theoretical Informatics Symposium LATIN 2010*, volume 6034 of *Lecture Notes in Computer Science*, pages 539–553. Springer-Verlag, 2010.

[11] E. Kaltofen. On the complexity of finding short vectors in integer lattices. In *Proceedings of EUROCAL'83*, volume 162 of *Lecture Notes in Computer Science*, pages 236–244. Springer-Verlag, 1983.

[12] R. Kannan, A. K. Lenstra, and L. Lovász. Polynomial factorization and nonrandomness of bits of algebraic and some transcendental numbers. In *Proceedings of STOC 1984*, pages 191–200. ACM Press, 1984.

[13] D. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens (Nice, 1970)*, volume 3, pages 269–274. Gauthiers-Villars, 1971.

[14] H. Koy and C. P. Schnorr. Segment LLL-reduction of lattice bases. In *Proceedings of the 2001 Cryptography and Lattices Conference (CALC'01)*, volume 2146 of *Lecture Notes in Computer Science*, pages 67–80. Springer-Verlag, 2001.

[15] D. H. Lehmer. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.

[16] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[17] K. S. McCurley and J. L. Hafner. Asymptotically fast triangularization of matrices over rings. *SIAM Journal on Computing*, 20:1068–1083, 1991.

[18] D. Micciancio and S. Goldwasser. *Complexity of lattice problems: a cryptographic perspective*. Kluwer Academic Press, 2002.

[19] I. Morel, D. Stehlé, and G. Villard. From an LLL-reduced basis to another. In progress.

[20] I. Morel, D. Stehlé, and G. Villard. H-LLL: using Householder inside LLL. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation (ISSAC'09)*, pages 271–278. ACM Press, 2009.

[21] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.

[22] P. Q. Nguyen and B. Vallée (editors). *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer-Verlag, 2009. Published after the LLL25 conference held in Caen in June 2007, in honour of the 25-th anniversary of the LLL algorithm.

[23] A. Novocin. *Factoring Univariate Polynomials over the Rationals*. PhD thesis, Florida State University, 2008.

[24] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. HAL Report ensl-00534899, `http://prunel.ccsd.cnrs.fr/ensl-00534899/en`, École Normale Supérieure de Lyon, France, 2010.

[25] C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9(1):47–62, 1988.

[26] C. P. Schnorr. Fast LLL-type lattice reduction. *Information and Computation*, 204:1–25, 2005.

[27] A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.

[28] A. Schönhage. Factorization of univariate integer polynomials by Diophantine approximation and improved basis reduction algorithm. In *Proceedings of the 1984 International Colloquium on Automata, Languages and Programming (ICALP 1984)*, volume 172 of *Lecture Notes in Computer Science*, pages 436–447. Springer-Verlag, 1984.

[29] A. Schönhage. Fast reduction and composition of binary quadratic forms. In *Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation (ISSAC'91)*, pages 128–133. ACM Press, 1991.

[30] D. Stehlé. Floating-point LLL: theoretical and practical aspects. Chapter of [22].

[31] A. Storjohann. Faster Algorithms for Integer Lattice Basis Reduction. Technical Report TR 249, ETH, Dpt. Comp. Sc., Zürich, Switzerland, 1996.

[32] A. Storjohann and G. Labahn. Asymptotically fast computation of Hermite normal forms of integer matrices. In *Proceedings of the 1996 international symposium on Symbolic and algebraic computation (ISSAC'96)*, pages 259–266. ACM Press, 1996.

[33] C. K. Yap. Fast unimodular reduction: planar integer lattices. In *Proceedings of the 1992 Symposium on the Foundations of Computer Science (FOCS 1992)*, pages 437–446. IEEE Computer Society Press, 1992.