



HAL
open science

An LLL-reduction algorithm with quasi-linear time complexity

Andrew Novocin, Damien Stehlé, Gilles Villard

► **To cite this version:**

Andrew Novocin, Damien Stehlé, Gilles Villard. An LLL-reduction algorithm with quasi-linear time complexity. 2010. ensl-00534899v1

HAL Id: ensl-00534899

<https://ens-lyon.hal.science/ensl-00534899v1>

Preprint submitted on 10 Nov 2010 (v1), last revised 7 Apr 2011 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

An LLL-Reduction Algorithm with Quasi-linear Time Complexity

Andrew Novocin, Damien Stehlé, and Gilles Villard

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

{andrew.novocin,damien.stehle,gilles.villard}@ens-lyon.fr

Abstract. We devise an algorithm, \tilde{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 \tilde{L}^1 is able to mimic the Knuth-Schönhage fast gcd algorithm thanks to a combination of innovative 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.

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 [25], Yap [29], and Eisenbrand [4] whose algorithm is exponential in d . Our result can be seen as a generalization of the Knuth-Schönhage quasi-linear GCD 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, 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 \tilde{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 [13]. 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 [15]). 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 [19, 3] for two recent surveys).

L^3 was originally proven to be of bit-complexity $\mathcal{O}(d^{5+\varepsilon} \beta^{2+\varepsilon})$ when the input basis $B = (\mathbf{b}_i)_i \in \mathbb{Z}^{d \times d}$ satisfies $\max \|\mathbf{b}_i\| \leq 2^\beta$. For the sake of simplicity, we will only consider full-rank lattices. Also, the ε 's in the complexity bounds stem from the fast multiplication of integers. 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 [8], who showed a $\mathcal{O}(d^5\beta^2(d+\beta)^\varepsilon)$ complexity bound. Schnorr [21] 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 [24, 27, 10, 22], but they have not lowered the cost with respect to β (for fixed d). More recently, Nguyen and Stehlé devised L^2 [18], 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 [17] 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 [12] 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 [9] and Schönhage [23] 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 [29, 25], and the resulting algorithm was then used in [4] to reduce lattices in arbitrary dimensions in quasi-linear time. Unfortunately, the cost of 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{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 the following definition. (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 $\mathcal{O}(n^\omega)$ scalar operations. (We can set $\omega \approx 2.376$ [2].)

Definition 1 ([1, 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 said 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:

Theorem 1 ([1, 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 i th minimum of the lattice $L(B)$.

\widetilde{L}^1 and its analysis rely on two recent lattice reduction techniques, whose contributions can be easily explained in the gcd framework. The efficiency of the fast gcd algorithms [9, 23] stems from two sources: Performing operations on truncated remainders is meaningful (which allows to consider remainders with smaller bit-sizes), and the obtained transformations corresponding to the quotients sequence have small bit-sizes (which allows 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. Also, the resulting unimodular transformation matrices (integral with determinant ± 1 so that the spanned lattice is preserved) may have large magnitudes, but must be stored on few bits. To solve these dilemmas, we truncate basis matrices as $B \approx B'E$ where B' has small bit-size and E is a re-scaling diagonal matrix, and store unimodular transforms as $U = D_1U'D_2$ where U' has small bit-size, and D_1 and D_2 are re-scaling diagonal matrices. Applying a U to a B (with such representations) can be performed efficiently, but, as opposed to the one-dimensional case, combining the transforms becomes more involved: to multiply a U_1 and a U_2 (with such representations) into a unimodular U_3 , we apply a so-called cleaning process to U_3 , involving truncations of coefficients (while preserving unimodularity).

GRADUAL FEEDING OF THE INPUT. Gradual feeding was introduced by Novocin, and van Hoeij [20, 7], in the context of specific lattice bases that are encountered while factoring rational polynomials (e.g., with the algorithm from [6]). 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 $\text{diag}(2^\ell, 1, \dots, 1)$. If one runs L^3 on B directly, the unimodular transformations obtained during the execution are likely to require an inefficient bit-length. To control the intermediate transformations, the matrix B can be slowly reduced: 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 well controlled. In fact 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). We call this process lift-reducing, and it can be used to provide a family of new reduction algorithms. We illustrate that the general reduction problem can be reduced to lift-reduction by performing a Hermite Normal Form (HNF) computation on B beforehand. Note that there could be other means of seeding the lift-reduction process.

TRUNCATIONS OF BASIS MATRICES. In order to work on as few bits of basis matrices as possible, we apply (column-wise) truncations. A truncation of precision p replaces a matrix B by a truncated matrix $B + \Delta B$ such that $\max \frac{\|\Delta b_i\|}{\|b_i\|} \leq 2^{-p}$ holds for all i , and only the first $\mathcal{O}(p)$ bits of every entry of $B + \Delta B$ are allowed to be non-zero. A truncation is a efficiency-motivated column-wise perturbation. The following lemmata explain why we are interested in such perturbations.

Lemma 1 ([1, Se. 2], refined from [5]). *Let $p > 0$, $B \in \mathbb{R}^{d \times d}$ non-singular with R -factor R , and ΔB with $\max \frac{\|\Delta b_i\|}{\|b_i\|} \leq 2^{-p}$. If $\text{cond}(R) = \| \|R\| R^{-1} \|_2$ (using the induced norm) satisfies $c_0 \cdot \text{cond}(R) \cdot 2^{-p} < 1$ with $c_0 = 8d^{3/2}$, then $B + \Delta B$ is non-singular and its R -factor $R + \Delta R$ satisfies $\max \frac{\|\Delta r_i\|}{\|r_i\|} \leq c_0 \cdot \text{cond}(R) \cdot 2^{-p}$.*

Lemma 2 ([1, Le. 5.5]). *If $B \in \mathbb{R}^{d \times d}$ is (δ, η, θ) -reduced with R -factor R then $\text{cond}(R) \leq \frac{\rho+1}{\rho-1} \rho^d$, 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 [26, Se. 3.1]).

Lemma 3 ([1, 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.*

PSEUDO-LIFT- \tilde{L}^1 . When we combine gradual feeding and truncation we see a difficulty which must be addressed. Namely, lift-reducing a truncation of B will not give the same transformation as lift-reducing B directly; likewise any truncation of U perturbs our reduction even further. Thus after working with truncations we must apply any transformations to a higher precision lattice and refine the result. The Lift- \tilde{L}^1 algorithm in figure 4 is a rigorous implementation of this pseudo algorithm in figure 1; Lift- \tilde{L}^1 must refine more often to properly handle a specified reduction. It

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

1. get $U_{1,\text{small}}$ from `pseudo-Lift- \tilde{L}^1 (truncate(B), $\ell/2$)`
2. $C := \sigma_{\ell/2} B U_{1,\text{small}}$
3. get U_c from `refineReduction(C)`
4. get $U_{2,\text{small}}$ from `pseudo-Lift- \tilde{L}^1 (truncate($C U_c$), $\ell/2$)`
5. $U_{\text{small}} := \text{clean}(U_{1,\text{small}} \cdot U_c \cdot U_{2,\text{small}})$
6. Return U_{small} .

Fig. 1. pseudo-Lift- \tilde{L}^1 .

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 \tilde{L}^1 . The term $\mathcal{O}(d^{5+\varepsilon} \beta)$ stems from a series of β calls to H-LLL [17] or L^2 [18] on integral matrices whose entries have bit-lengths $\mathcal{O}(d)$. These calls are unusual in that the number of LLL switches performed by each one is on average $\mathcal{O}(d^2)$ instead of the $\mathcal{O}(d^3)$ worst-case bound. 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 \tilde{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 \tilde{L}^1 in β is $\text{Poly}(d) \cdot \mathcal{M}(\beta) \log \beta$. Finally, we remark that the cost can be proven to be $\mathcal{O}(d^{4+\varepsilon} \log |\det B| + d^{5+\varepsilon} + d^\omega (\log |\det B|)^{1+\varepsilon}) + \mathcal{H}(d, \beta)$ if $\mathcal{H}(d, \beta)$ denotes the cost of computing the Hermite normal form.

ROAD-MAP. We construct \tilde{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: A reduction algorithm relying on gradual feeding

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 $\sigma_\ell = \text{diag}(2^\ell, 1, \dots, 1)$). 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 G-S norms 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 [7, Le. 4]. For the second one, notice that $\prod r'_{i,i} = |\det B'| = 2^\ell |\det B| = \prod r_{i,i}$. We now prove the third statement. Let \mathbf{v}_i and \mathbf{v}'_i represent 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 $\lambda_i \leq \max_{\|\cdot\|}(S_i) \leq \lambda'_i \leq \max_{\|\cdot\|}(S'_i) \leq 2^\ell \lambda_i$. \square

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:*

$$\forall i, j : |u_{i,j}| \leq 2d^3 \zeta_1^d \cdot \frac{r'_{j,j}}{r_{i,i}} \leq 2^{\ell+1} d^3 \zeta_1^{2d} \cdot \frac{r_{j,j}}{r_{i,i}},$$

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

$$\forall i, j : |v_{j,i}| \leq 2^{\ell+1} d^3 \zeta_2^d \cdot \frac{r_{i,i}}{r'_{j,j}} \leq 2^{\ell+1} d^3 \zeta_2^{2d} \cdot \frac{r_{i,i}}{r_{j,j}}.$$

Proof. Write $B = QR$ and $C = Q'R'$ with Q and Q' orthogonal. Then

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

with $\bar{R} = R \cdot \text{diag}(1/r_{i,i})$ and $\bar{R}' = R' \cdot \text{diag}(1/r'_{j,j})$. From the proof of [1, 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 $|\bar{R}'| \leq \alpha_2^d T$. Finally, we have $|Q|, |Q'| \leq C$, where $c_{i,j} = 1$ for all i, j . Using the triangular inequality, we obtain:

$$|U| \leq 2\zeta^d \text{diag}(r_{i,i}^{-1}) T C^2 T \text{diag}(r'_{j,j}) \leq 2d^3 \zeta^d \text{diag}(r_{i,i}^{-1}) C \text{diag}(r'_{j,j}).$$

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

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

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 . \square

LLL-REDUCTION AS A SEQUENCE OF LIFT-REDUCTIONS. The algorithm of Figure 2 efficiently reduces the task of LLL-reducing arbitrary integer non-singular matrices to a sequence of Lift-reductions. The Lift-reduction process is seeded with an HNF computation. We 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 [14, 28], it can be computed in time $O(d^{\omega+1+\varepsilon}\beta^{1+\varepsilon})$, if 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. $B := \text{HNF}(B)$.
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 $\sigma_{\ell_k} C U'$ is Ξ -reduced.
6. Let U be the block-diagonal matrix $\text{diag}(I, U')$.
7. Compute $B := B \cdot U$, reducing row i symmetrically modulo $b_{i,i}$ for $i < k$.
8. Return B .

Fig. 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, and $\prod b_{i,i} = |\det H| \leq 2^{d\beta}$. After iteration k of the loop, the bottom-right $(d-k+1)$ -dimensional submatrix is Ξ -reduced. This implies that any input to Step 5 is valid for Lift-reduction, and thus provides correctness (proved in lemma 6). 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 reduces B such that $\max \|\mathbf{b}_i\| \leq 2^\beta$ using $d^{\omega+1+\varepsilon}(\beta^{1+\varepsilon} + d) + \sum_{k=1}^{d-1} \mathcal{C}(d-k+1, \tau_k, \beta_k, \ell_k)$ bit operations where $\mathcal{C}(d-k+1, \tau_k, \beta_k, \ell_k)$ is the cost of Lift-reducing a $(d-k+1)$ -dimensional Ξ -reduced basis with column bit-size β_k , target lift ℓ_k , and which uses τ_k switches.*

Proof. We let U_H be the unimodular transformation such that $H = BU_H$; U'_k be the $(d-k+1) \times (d-k+1)$ unimodular transformation that reduces $\sigma_{\ell_k} C$ at Step 5; U''_k be the unimodular transformation that reduces rows $1 \leq i < k$ at Step 7. With input B the algorithm returns the Ξ -reduced basis $B \cdot U_H \cdot \text{diag}(I, U''_{d-1}) \cdot U''_{d-1} \cdots \text{diag}(I, U''_2) \cdot U''_2 \cdot U'_1$ hence is correct.

We claim that at the beginning of step 5 the matrix C is Ξ -reduced. If we let R be the R-factor of C and R' the R-factor of C' the bottom right $(d-k) \times (d-k)$ sub-matrix of C . We know that $r_{i,j} = r'_{i-1,j-1}$ for $j \geq i > 1$ and that C' is a Ξ -reduced basis. By the above argument we know that C' is a basis of the lattice generated by the columns of the bottom $(d-k) \times (d-k)$ of the Hermite form of B which has successive minima $\lambda_1 \geq \min\{h_{k+1,k+1}, h_{k+1,k+1}, \dots, h_{d,d}\} \geq 1$. Thus $r_{2,2}\alpha = r'_{1,1}\alpha \geq 1 \geq r_{1,1} = h_{k,k}/2^{\lceil \log h_{k,k} \rceil}$ and $|r_{1,j}| \leq r_{1,1}/2$ for $j > 1$ which proves the claim. It follows that the cost of the $d-1$ calls to H-LLL is $\sum_{k=1}^{d-1} \mathcal{C}(d-k+1, \tau_k, \beta_k, \ell_k)$ bit operations where β_k is the column bit-size of C at step 5.

For analyzing the cost of step 7 we note that we need only compute the product of the final $d-k+1$ columns of B and U' . Further we can bound $|u'_{i,j}|$ using Lemma 5. Let the product of the two be called P then $p_{i,j} = \sum_{e=1}^{d-k+1} b_{i,k+e-1} \cdot u'_{e,j}$. We let R be the R-factor of C at the beginning of step 5.

For rows $i \leq k$ of the product BU' we have $\log |b_{i,k+e-1}| \leq \log h_{i,i}$ since entries in row i are reduced symmetrically by $h_{i,i}$, and $\log |u'_{e,j}| = \mathcal{O}(\ell_k + d + \log r_{j,j})$. The latter leads to $\log |p_{i,j}| = \mathcal{O}(\log h_{i,i} + \ell_k + d + \log r_{j,j})$. The product for rows $i \leq k$ is implemented as follows. For a row integer vector \mathbf{x} and a column integer vector \mathbf{y} of dimension d whose entries have sizes

bounded respectively by $t\beta$ and $s\beta$, consider the problem of computing $\mathbf{x} \cdot \mathbf{y}$. The entries of \mathbf{x} and \mathbf{y} may be decomposed following $x_j = \sum_{i=0}^t x_j^{(i)} 2^{i\beta}$, with $\log |x_j^{(i)}| \leq \beta$, and $y_j = \sum_{i=0}^s y_j^{(i)} 2^{i\beta}$, with $\log |y_j^{(i)}| \leq \beta$. The vector product $\mathbf{x} \cdot \mathbf{y}$ can be obtained by first computing the matrix product

$$\begin{bmatrix} x_1^{(0)} & \dots & x_d^{(0)} \\ \vdots & \dots & \vdots \\ x_1^{(t)} & \dots & x_d^{(t)} \end{bmatrix} \cdot \begin{bmatrix} y_1^{(0)} & \dots & y_1^{(s)} \\ \vdots & \dots & \vdots \\ y_d^{(0)} & \dots & y_d^{(s)} \end{bmatrix}, \quad (1)$$

then by summing the resulting integer vectors by columns and by rows. If $t, s \leq d$ the bit cost is $\mathcal{O}(d^\omega \mathcal{M}(\beta))$. Using that $\sum \log h_{i,i} = \log |\det B|$ and $\sum \log r_{j,j} \leq \log |\det B|$, the latter cost bound can be applied to the product BU' for rows $i \leq k$ leading to a bit cost $\mathcal{O}(d^\omega \mathcal{M}(\ell_k + d + \log |\det B|/d))$.

For rows $i > k$ of the product BU' we have $|b_{i,k+e}| \leq \alpha^d r_{e+1,e+1}$ and $|u'_{e+1,j}| \leq 2^{\ell_k+1} d^3 \zeta_1^{2d} \cdot r_{j,j}/r_{e+1,e+1}$ hence $\log |p_{i,j}| = \mathcal{O}(d + \ell_k + \log r_{j,j})$. For a row integer vector \mathbf{x} of dimension d whose entries have bit sizes $t_j\beta$, $1 \leq j \leq d$, and \mathbf{y} as previously, we write $x_j = \sum_{i=0}^{t_j} x_j^{(i)} 2^{i\beta}$, with $\log |x_j^{(i)}| \leq \beta$. The product $\mathbf{x} \cdot \mathbf{y}$ may be obtained by applying (1) to

$$\begin{bmatrix} x_1^{(0)} & \dots & x_1^{(t_1)} & \dots & x_d^{(0)} & \dots & x_d^{(t_d)} \end{bmatrix} \cdot [y_1 \dots y_1 \dots y_d \dots y_d]^t.$$

If $\sum t_i = d$ then the cost in d remains asymptotically unchanged. Hence using that $\sum \log r_{j,j} \leq \log |\det B|$ the product BU' for rows $i > k$ can also be computed with $\mathcal{O}(d^\omega \mathcal{M}(\ell_k + d + \beta))$ bit operations. The costs for the $d - 1$ products BU' is $\sum_k \mathcal{O}(d^\omega \mathcal{M}(\ell_k + d + \log |\det B|/d))$, which is also $\mathcal{O}(d^\omega \mathcal{M}(\log |\det B|) + d^{\omega+1} \mathcal{M}(d))$. When we also consider the cost $\mathcal{O}(d^{\omega+1+\varepsilon} \beta^{1+\varepsilon})$ of converting B to Hermite normal form we complete the proof. \square

From the proof of Lemma 6 we may see that the non-reduction costs can be refined as $\mathcal{O}(d^\omega \mathcal{M}(\log |\det B|) + d^{\omega+1} \mathcal{M}(d) + \mathcal{H}(d, \beta))$. Furthermore we make no assumptions on the algorithm used in step 5, allowing lift-reduction to be a complete black-box. We also note that the HNF is only used as a triangularization, thus any triangularization of the input B will suffice, however then the user might need 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 directly we can bound the complexity of Figure 2 by $\mathcal{Poly}(d) \cdot \beta^2$. This is comparable to L^2 and H-LLL.

3 Truncating matrix entries within Lift-reduction

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 B reduced and $\ell \geq 0$, finding U such that $\sigma_\ell BU$ is reduced can be done by looking at the most significant bits of B only. 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 (not 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 $\sigma_\ell B$

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_2 such that the following holds for any $\ell \geq 0$. Let $B \in \mathbb{R}^{d \times d}$ be Ξ_1 -reduced, U such that $\sigma_\ell BU$ is Ξ_3 -reduced and ΔB with $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-\ell - c_2 \cdot d}$. Then $\sigma_\ell(B + \Delta B)U$ is Ξ_2 -reduced.*

Proof. By Lemma 5, there exists a constant c such that $|u_{i,j}| \leq 2^{c \cdot d \frac{R'_{i,i}}{R'_{j,j}}}$, where R (resp. R') is the R-factor of B (resp. $C = \sigma_\ell BU$). Let $C + \Delta C = \sigma_\ell(B + \Delta B)U$. The norm of $\Delta \mathbf{c}_i = \sum_j u_{j,i} \sigma_\ell \Delta \mathbf{b}_j$ satisfies $\Delta \mathbf{c}_i \leq \sum_j 2^{-p + \ell + c \cdot d \frac{R'_{i,i}}{R'_{j,j}}} \|\mathbf{b}_j\| \leq d \alpha^d 2^{-p + \ell + c \cdot d} R'_{i,i}$, by Theorem 1 and with p such that $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-p}$. Furthermore, we have $\|\mathbf{c}_i\| \geq R'_{i,i}$. This gives $\max \frac{\|\Delta \mathbf{c}_i\|}{\|\mathbf{c}_i\|} \leq d \alpha^d 2^{-p + \ell + c \cdot d}$. By Lemma 3 (that we apply to C and $C + \Delta C$), there exists c' such that if $p \geq \ell + c' \cdot d$, then $C + \Delta C$ is Ξ_2 -reduced. \square

By combining Lemmata 7 and 3, we have that a reducing U can be found by working on a truncation of B .

Lemma 8. *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.*

Proof. Let $\Xi_0 < \Xi_1$ be a valid set of reduction parameters. By Lemma 3, there exists a constant c such that if $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^{-c \cdot d}$, then $B + \Delta B$ is non-singular and Ξ_0 -reduced. We conclude by using Lemma 7. \square

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 such that: Each entry of $B' \in \mathbb{Z}^{d \times d}$ has bit-length $\leq p = \ell + c \cdot d$ for some constant c (depending solely on the reduction parameters); $E = \text{diag}(2^{e_i - p})$ with $e_i \in \mathbb{Z}$ such that $\frac{2^{e_i} - \|\mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq 2^d$. Note that if B is integral with $\max \|\mathbf{b}_i\| \leq 2^\beta$, then each entry of E may be stored on $\log(d + \beta)$ bits. We denote this computation of the pair (B', E) by $\text{MSB}_{(p)}(B)$.

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

The algorithm `TrLiftLLL` we describe hereafter is an adaptation of the `StrengthenLLL` algorithm from [16], 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. When setting $\ell = 0$ `TrLiftLLL` will be used 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.

When setting $\ell = \mathcal{O}(d)$, we obtain the base case of $\text{lift-}\tilde{\text{L}}^1$, the quasi-linear recursive algorithm to be introduced in the next section. The most expensive step of TrLiftLLL is a call to a LLL-type algorithm, which must be switch-based, a standard property given below. The switch property is for instance satisfied by L^3 ([13, p. 523]), L^2 ([18, Th. 6]) and H-LLL ([17, Th. 4.3]). We choose H-LLL as this currently provides the best complexity bound.

A STANDARD PROPERTY (P) SATISFIED BY LLL-REDUCING ALGORITHMS. When called on a basis matrix B with R-factor R , the above LLL-reducing algorithms perform two types of operations: They either subtract to a vector \mathbf{b}_k an integer combination of $\mathbf{b}_1, \dots, \mathbf{b}_{k-1}$ (translation), or they exchange \mathbf{b}_{k-1} and \mathbf{b}_k (switches). Translations leave the $r_{i,i}$'s unchanged. Switches 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 will prove the following theorem.

Theorem 2. *For any valid sets of reduction 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 $E = \text{diag}(2^{e_i})$ such that $\max |b_{i,j}| \leq 2^{c_4(\ell+d)}$, $e_i \in \mathbb{Z}$ 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 made to H-LLL; And it returns two matrices U and D such that:*

1. $D = \text{diag}(2^{d_i})$ 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.

The possible unbalancedness of the columns of BE (due to E), prevents us from applying H-LLL directly on $C = \sigma_\ell BE$. Indeed, even if we were dividing the full matrix by a large common power of 2, the resulting basis may have a bit-size that is arbitrarily large compared to d and ℓ . Our goal is to call H-LLL on a matrix whose entries have bit-sizes $O(d+\ell)$. To circumvent the possible unbalanced-ness of the columns of C , we find blocks of consecutive vectors whose $r_{i,i}$'s have similar magnitudes, where R is the R-factor of C , and we apply a column-scaling to re-balance C before calling H-LLL.

FINDING BLOCKS. The definition of block is motivated by property (P) above. To determine meaningful blocks, the first step is to find good approximations to the $r_{i,i}$'s. Computing the R-factor of a non-singular matrix is most often done by applying Householder's algorithm (see [5, Ch. 19]). The following lemma is a rigorous and explicit variant of standard backward stability results.

Lemma 9 ([1, Se. 6]). *Let $p \geq 0$ and $B \in \mathbb{R}^{d \times d}$ be non-singular with R-factor R . Let \hat{R} be the R-factor computed by Householder's algorithm with floating-point precision p . If $c_5 2^{-p} < 1$ with $c_5 = 80d^2$, then there exists an orthogonal \hat{Q} such that $\hat{Q}\hat{R} = B + \Delta B$ with $\max \frac{\|\Delta \mathbf{b}_i\|}{\|\mathbf{b}_i\|} \leq c_5 2^{-p}$.*

Let $B = QR$ and $\sigma_\ell B = Q'R'$ be the QR factorizations of B and $\sigma_\ell B$ respectively. We have:

$$\text{cond}(R') = \| \|R'\| \|R'^{-1}\| \| = \| \|Q'^t \sigma_\ell Q R\| \|R^{-1} Q'^t \sigma_\ell^{-1} Q'\| \|$$

$$\begin{aligned} &\leq \| |Q^t|_{\sigma_\ell} |Q| |R| |R^{-1}| |Q^t|_{\sigma_\ell^{-1}} |Q'| \| \\ &\leq d^2 2^\ell \text{cond}(R) \leq d^2 \frac{\rho + 1}{\rho - 1} \rho^{d\ell}, \end{aligned}$$

where ρ is as in Lemma 2. Therefore, Lemmata 1 and 9 imply that Householder's algorithm with precision $p = \ell + O(d)$ allows us to find \widehat{R}' with $\max \frac{|\widehat{r}'_{i,i} - r'_{i,i}|}{r'_{i,i}} \leq \frac{1}{100}$. Since $R = R'E$, we have $\max \frac{|\widehat{r}_{i,i} - r_{i,i}|}{r_{i,i}} \leq \frac{1}{100}$, with $\widehat{R} = \widehat{R}'E$. Furthermore, as the run-time of Householder's algorithm in precision p is $O(d^3 p^{1+\varepsilon})$, the computation of the $\widehat{r}_{i,i}$'s costs $O(d^3(\ell + d)^{1+\varepsilon})$.

We define the blocks of vectors of C as follows: The first block starts with $\mathbf{c}_{i_1} = \mathbf{c}_1$ and stops with \mathbf{c}_{i_2-1} where i_2 is the smallest i such that $\min_{j \geq i} \widehat{r}_{j,j} > \nu \cdot \max_{j < i} \widehat{r}_{j,j}$ (if $i_2 = d + 1$, then the process ends); The k th block starts with \mathbf{c}_{i_k} and stops with $\mathbf{c}_{i_{k+1}-1}$ where i_{k+1} is the smallest index $i > i_k$ such that $\min_{j \geq i} \widehat{r}_{j,j} > \nu \cdot \max_{j < i} \widehat{r}_{j,j}$. The purpose of the constant $\nu \geq 2$, to be set later, is to handle the inaccuracy of \widehat{R} and to ensure that the matrix $CD^{-1}UD$ eventually obtained will be size-reduced.

Let $I_k = [i_k, i_{k+1})$. Since $\nu \geq 2$, Property (P) implies that the unimodular U obtained by calling H-LLL on C will satisfy $u_{i,j} = 0$ if $i \in I_{k_1}$ and $j \in I_{k_2}$ with $k_1 < k_2$, i.e., U is (I_k) -block upper triangular. Any diagonal block-submatrix of U will be unimodular. Computing the I_k 's from $\widehat{r}_{j,j}$'s may be done in time $O(d^2(d + \ell + \log \max(1 + |e_i|)))$.

By construction of the blocks, the amplitude of $r_{i,i}$'s within a block is bounded.

Lemma 10. *We use the same notations as above. We let $(\ell_i)_i$ be such that the $r_{i,i}/\ell_i$'s are the diagonal coefficients of the R-factor of B . There exists a constant c_6 (depending on Ξ_1 and ν only) such that for any k , we have $\frac{\max_{i \in I_k} r_{i,i}}{\min_{i \in I_k} r_{i,i}} \leq 2^{c_6 |I_k|} \cdot \max_{i \in I_k} \ell_i$.*

Proof. Let $i, j \in I_k$. We are to upper-bound $\frac{r_{j,j}}{r_{i,i}}$. If $j \leq i$, the reducedness of B implies that $\frac{r_{j,j}}{\ell_j} \leq \alpha^{i-j} \frac{r_{i,i}}{\ell_i}$, for α as in Theorem 1. The fact that $\ell_i \geq 1$ (see Lemma 4) provides the result. Assume now that $j > i$. If $r_{i,i} = \max_{t \geq i} r_{t,t}$, then the bound holds. Otherwise, by definition of the blocks, there exists $i' > i$ in I_k such that $r_{i',i'} \leq 2\nu \cdot r_{i,i}$ (the factor 2 takes the inaccuracy of \widehat{R} into account). By induction, it can be shown that $r_{i'',i''} \geq (2\nu)^{|I_k|} r_{i,i}$, with $i'' = i_{k+1} - 1$. We conclude that $\frac{r_{j,j}}{r_{i,i}} \leq (2\nu)^{|I_k|} \frac{r_{j,j}}{r_{i'',i''}} \leq (2\nu\alpha)^{|I_k|} \ell_j$, by using the first part of the proof (since $j \leq i''$). \square

RE-BALANCING THE COLUMNS OF C . The blocks allow us to define the diagonal matrix D of Theorem 2. We define the gap between two blocks I_k and I_{k+1} to be $\gamma_k = \frac{\min_{j \in I_{k+1}} \widehat{r}_{j,j}}{\max_{j \in I_k} \widehat{r}_{j,j}}$. We define $D = \text{diag}(2^{d_i})$ such that the block structure is preserved, but the gaps get shrunk: For $i \in I_k$, we set $d_i = e_1 + \sum_{k' < k} \lceil \log_2 \gamma_{k'} / \sqrt{\nu} \rceil$. This ensures that $B' = BED^{-1}$ is Ξ_1 -reduced. Also, the d_i 's satisfy Property 1 of Theorem 2: Thanks to the reducedness of BE , the size condition on B , and Lemma 4, each e_i is within $O(\ell + d)$ of $\log r_{i,i}$. Thanks to Lemmata 10 and 4, the same holds for the d_i 's.

The matrix $C' = CD^{-1}$ with R-factor $R' = RD^{-1}$ admits the same block-structure as C : For any k , we have $\min_{j \in I_{k+1}} r'_{j,j} \geq \nu' \cdot \max_{j \in I_k} r'_{j,j}$, with $\nu' = \sqrt{\nu}/2$. Also, Property 1 of Theorem 2 ensures that any entry of C' may be stored on $O(\ell + d)$ bits (it is a shift of a small integer), i.e. the matrix C' is balanced.

LLL-REDUCING. We now call H-LLL on input matrix C' , with LLL-parameters $\Xi > \Xi_2$, and let $C^{(2)}$ be the output matrix. Thanks to the balancedness and bit-size bound for the entries of C' , this costs $O(d^{2+\varepsilon}(d + \ell + \tau)(d + \ell))$ bit operations (see [17, Th. 4.4]), where τ be the number of switches performed. Let U be the corresponding unimodular transform (which can be recovered

from C' and $C^{(2)}$ by a matrix inversion). Lemma 5 and the fact that B' is Ξ_1 -reduced ensure that Property 2 of Theorem 2 is satisfied. Also, since C' follows the block-structure defined by the I_k 's, Property (P) may be used to assert that U is $(I_k)_k$ -block upper triangular and that its diagonal blocks are unimodular. The coefficients of D are non-decreasing, and they are constant within any I_k . This ensures that $D^{-1}UD$ is integral and that its diagonal blocks are exactly those of U , and thus $D^{-1}UD$ is unimodular.

Let $C^{(3)} = \sigma_\ell(BE)(D^{-1}UD) = C^{(2)}D$. It remains to show that $C^{(3)}$ is Ξ_2 -reduced. Let $R^{(2)}$ (resp. $R^{(3)}$) be the R-factor of $C^{(2)}$ (resp. $C^{(3)}$). Let $\Xi = (\delta, \eta, \theta)$ and $\Xi_2 = (\delta_2, \eta_2, \theta_2)$. If i and j belong to the same I_k , then $|r_{i,j}^{(3)}| \leq \eta r_{i,i}^{(3)} + \theta r_{j,j}^{(3)}$, because this holds for $R^{(2)}$ and $\frac{r_{i,j}^{(3)}}{r_{i,j}^{(2)}} = \frac{r_{i,i}^{(3)}}{r_{i,i}^{(2)}} = \frac{r_{j,j}^{(3)}}{r_{j,j}^{(2)}} = 2^{d_{i_k}}$. Since $\eta < \eta_2$ and $\theta < \theta_2$, the size-reduction condition for (i, j) is satisfied. Similarly, the Lovász conditions are satisfied inside the I_k 's. They are also satisfied for any $i = i_k - 1$, since $c_{i_k}^{(2)}$ is multiplied by $2^{d_{i_k}} \geq 2^{d_{i_k-1}}$. It remains to check the size-reduction conditions for (i, j) with $i \in I_k$, $j \in I_{k'}$ and $k' > k$. By reducedness of $C^{(2)}$, we have $|r_{i,j}^{(2)}| \leq \eta r_{i,i}^{(2)} + \theta r_{j,j}^{(2)}$. Since it was the case for R' , by Property (P), we have that $r_{i,i}^{(2)} \leq \frac{1}{\nu'} r_{j,j}^{(2)}$ (with $\nu' = \sqrt{\nu}/2$), and thus $|r_{i,j}^{(2)}| \leq (\theta + \frac{1}{\nu'}) r_{j,j}^{(2)}$. This gives $|r_{i,j}^{(3)}| \leq (\theta + \frac{1}{\nu'}) r_{j,j}^{(3)}$. In order to ensure size-reducedness, it thus suffices to choose ν such that $\theta + \frac{1}{\nu'} \leq \theta_2$.

3.3 A Lehmer-like lift-LLL algorithm

By combining Lemma 8 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. 8 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. $(B', E) := \text{MSB}_{(t+c_3d)}(B)$ (see the explanation after Le. 8).
4. $(D, U) := \text{TrLiftLLL}(B', E, t)$.
5. $B := \sigma_t B D^{-1} U D$.
6. Return B .

Fig. 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 $O(d^3 \ell (d^{1+\varepsilon} t + t^{-1+\varepsilon} (\ell + \beta)))$.*

Proof. The correctness is provided by Lemmata 3 and 8 and by Theorem 2. At any moment throughout the execution, the matrix B is a Ξ -reduced basis of the lattice spanned by an ℓ' -lift of the input, for some $\ell' \leq \ell$. Therefore, by Theorem 1 and Lemma 4, the inequality $\max \|\mathbf{b}_i\| \leq \alpha^d \max r_{i,i} \leq 2^{c(\ell+\beta)}$ holds throughout the execution, for some constant c . The cost of Step 3 is $O[d^2(t + \log(\ell + \beta))]$. The cost of Step 4 is $O[d^{4+\varepsilon} t^2 + d^2 \log(\ell + \beta)]$. Step 5 is performed by first computing $\sigma_t B D^{-1}$, whose entries have bit-sizes $O(\ell + \beta)$, and then multiplying by U and finally by D . This costs $O(d^3(\ell + \beta)t^\varepsilon)$ bit operations. The claimed complexity bound can be obtained by summing over the ℓ/t loop iterations. \square

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, it is possible to obtain a LLL-reducing algorithm of complexity $\mathcal{P}oly(d) \cdot \beta^{1.5+\varepsilon}$.

4 A quasi-linear time Lift-LLL algorithm

We now aim at constructing a recursive variant of the **Lehmer-LiftLLL** algorithm of the previous section. Before being able to do so, we must work on lift-reducing unimodular transformations which are not produced from specific algorithms. More specifically, we need to show how to work on them at low cost, even if they are not produced by **TrLiftLLL** such as in Theorem 2. This study is done in full generality and afterwards is used to analyze a specified lift- \tilde{L}^1 algorithm.

4.1 Sanitizing lift-LLL-reducing 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 $\sigma_\ell B U$ is reduced. Furthermore, as shown in Theorem 2, the unimodular U can be found and stored on few bits. However, that U was obtained by a direct application of a well-understood LLL-reducing algorithm. In this section we show that any U which reduces $\sigma_\ell B$ can be transformed into a factored unimodular U' which also reduces $\sigma_\ell B$, each entry can be stored with only $\mathcal{O}(\ell + d)$ bits, and products of such factored matrices can be computed quickly. This analysis can be used as a general framework for studying lift-reductions.

Lemma 11. *Let Ξ_1, Ξ_2 be valid LLL parameters. There exists a constant 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 $\sigma_\ell B U$ (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 $2^{\ell+c_7 \cdot d} \leq 2^{c \cdot d} \cdot \frac{r'_{j,j}}{r_{k,k}}$. We thus have $\frac{r_{k,k}}{r'_{j,j}} \leq 2^{-(\ell+c_7 \cdot d)}$. Since $i \geq j$ and $\sigma_\ell B U$ 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 12 shows that the “clean” transformation matrix remains valid for the reduction process.

Lemma 12. *Let Ξ_1, Ξ_2, Ξ_3 be valid LLL parameters with $\Xi_2 > \Xi_3$. There exists a constant 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 $\sigma_\ell B U$ (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 column-wise small compared to $|\sigma_\ell B U|$ and by applying Lemma 3. By assumption, we have $|\Delta U| \leq 2^{-(\ell+c_8 \cdot d)} \text{diag}(r_{i,i}^{-1}) C \text{diag}(r'_{j,j})$, where $c_{i,j} = 1$ for all i, j . Since B is Ξ_1 -reduced, we have $|R| \leq \text{diag}(r_{i,i}) T + \theta_1 T \text{diag}(r_{j,j})$, where T is upper triangular with $t_{i,j} = 1$ for all $i \leq j$. Then we get

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

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

$$|R \Delta U| \leq 2^{-(\ell+c_8 \cdot d)} (\alpha_1^d + \theta_1) T C \text{diag}(r'_{j,j}).$$

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 $r'_{j,j} \leq \|(\sigma_\ell B U)_j\|$ and by applying Lemma 3 (which requires that c_8 is set sufficiently large). \square

Lemmata 11 and 12 allow us to impose 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 constants $c_9, c_{10} > 0$ 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 $\sigma_\ell B U$ is Ξ_2 -reduced. Let $d_i := \lfloor \log \|\mathbf{b}_i\| \rfloor$ for all i . Let $D := \text{diag}(2^{d_i})$, $x := \ell + c_9 \cdot d$, $\widehat{U} := 2^x D U D^{-1}$ and $U' := 2^{-x} D^{-1} \lfloor \widehat{U} \rfloor D$. We write $\text{Clean}(U, (d_i)_i, \ell) := (U', D, x)$. Then U' is unimodular and $\sigma_\ell B U'$ is Ξ_3 -reduced. Furthermore, the matrix \widehat{U} satisfies $|\widehat{u}_{i,j}| \leq 2^{2\ell+c_{10} \cdot d}$.*

Proof. We first show that $U' \in \mathbb{Z}^{d \times d}$. If $\lfloor \widehat{u}_{i,j} \rfloor = \widehat{u}_{i,j}$, 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 $\lfloor \widehat{u}_{i,j} \rfloor \in \mathbb{Z} \subseteq 2^{x+d_i-d_j} \mathbb{Z}$. We conclude that $u'_{i,j} \in \mathbb{Z}$.

Now, consider $\Delta U = U' - U$. Since $\Delta U = 2^{-x} D^{-1} (\lfloor \widehat{U} \rfloor - \widehat{U}) D$, 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 $|\Delta u_{i,j}| \leq 2^{-x+c \cdot d} \cdot \frac{r'_{j,j}}{r_{i,i}}$. Applying Lemmata 11 and 12 shows that U' is unimodular and $\sigma_\ell B U'$ is Ξ_3 -reduced (if c_9 is chosen sufficiently large).

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

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

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 $\text{TrLiftLLL}'$ the algorithm resulting from the combination of Theorems 4 and 2. $\text{TrLiftLLL}'$ is to be used as base case of the recursion process of $\text{Lift-}\widetilde{\text{L}}^1$. Secondly, the following shows how to combine lift-LLL-reducing unimodular transforms, which is the engine of the recursion.

Lemma 13. *Let $U = 2^{-x} D^{-1} U' D \in \mathbb{Z}^{d \times d}$ with $U' \in \mathbb{Z}^{d \times d}$ and $D = \text{diag}(2^{d_i})$. Let $V = 2^{-y} E^{-1} V' E \in \mathbb{Z}^{d \times d}$ be defined similarly. Let $z \in \mathbb{Z}$ and $f_i \in \mathbb{Z}$ for $i \leq d$. Then it is possible to compute the output (W', F, z) of $\text{Clean}(U \cdot V, (f_i)_i, \ell)$ (see Theorem 4) from $x, y, z, U', V', (d_i)_i, (e_i)_i, (f_i)_i$, in time $O(d^3 \mathcal{M}(t))$, where $\max_{i,j} (|u'_{i,j}|, |v'_{i,j}|) \leq 2^t$ and $\max_i (|d_i - e_i|, |f_i - e_i|, |z - x + y|) \leq t$. For short, we will write $W := U \odot V$, where $W = 2^{-z} F^{-1} W' F$.*

Proof. We first compute $m = \max |d_i - e_i|$. We have $UV = 2^{(-x-y-m)}D^{-1}U'\text{diag}(2^{d_i-e_i+m})V'E$. We first multiply U' by $\text{diag}(2^{d_i-e_i+m})$, 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 $\leq 2X$. We then multiply the latter by V' , which costs $O(d^3\mathcal{M}(X))$. The matrix \widehat{W} from Theorem 4 may be computed and rounded within $O(d^2X)$ bit operations. \square

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

4.2 $\text{Lift-}\widetilde{\text{L}}^1$ and $\widetilde{\text{L}}^1$

The algorithm of Figure 4 is the Knuth-Schönhage-like generalization of the Lehmer-like algorithm of Figure 3. We define $\widetilde{\text{L}}^1$ as the algorithm from Figure 2, where $\text{Lift-}\widetilde{\text{L}}^1$ is used to implement lift-reduction. As we will see we use the truncation process MSB described after Lemma 8 and $\text{TrLiftLLL}'$ to ensure that $\widetilde{\text{L}}^1$ provides valid inputs to $\text{Lift-}\widetilde{\text{L}}^1$.

The $\text{Lift-}\widetilde{\text{L}}^1$ algorithm relies on two recursive calls, on MSB, truncations, and on calls to $\text{TrLiftLLL}'$. 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.

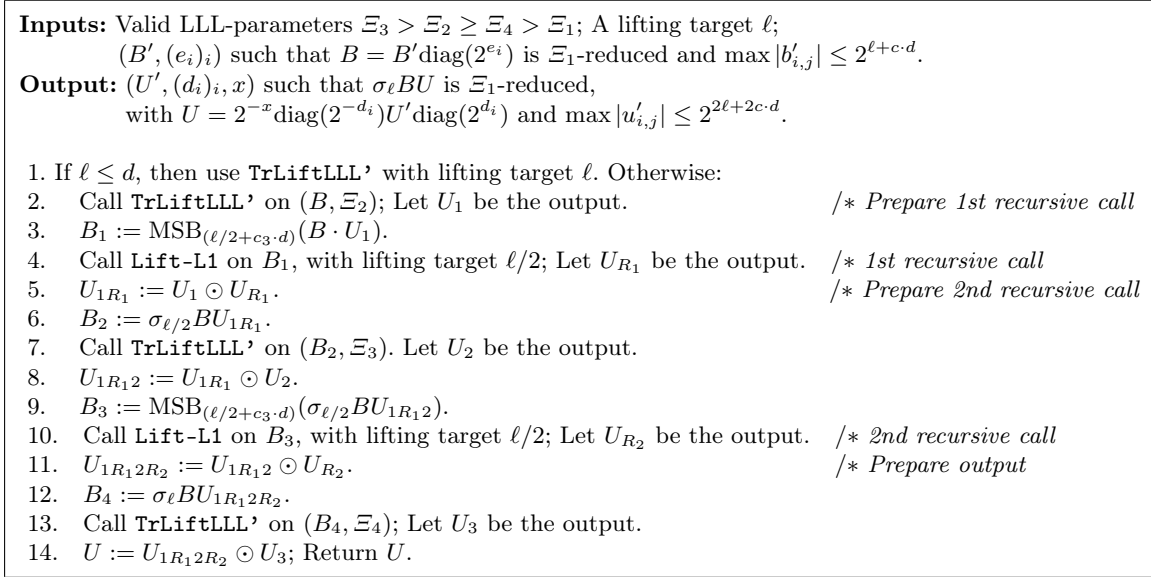


Fig. 4. The $\text{Lift-}\widetilde{\text{L}}^1$ algorithm.

Theorem 5. $\text{Lift-}\widetilde{\text{L}}^1$ 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 is chosen so that Lemma 7 applies now and Lemma 8 will apply at Step 6. 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 $\text{Lift-}\widetilde{\text{L}}^1$. Thus $\sigma_{\ell/2}B_1U_{R_1}$ is Ξ_1 -reduced. Because of the selection of c_3 in Step 3 we know also that $\sigma_{\ell/2}BU_1U_{R_1}$ is reduced (weaker than Ξ_1) using Lemma 8. Thus by Theorem 4, the matrix B_2 is reduced (weakly) and has an appropriate format for $\text{TrLiftLLL}'$. By Theorem 2, the matrix $\sigma_{\ell/2}BU_{1R_1}U_2$ is Ξ_3 -reduced and by Theorem 4 we have that $\sigma_{\ell/2}BU_{1R_12}$ is Ξ_2 -reduced.

By choice of c_3 and Lemma 7, we know that the matrix B_3 is Ξ_1 -reduced and satisfies the input requirements of $\text{Lift-}\tilde{\mathcal{L}}^1$. Thus, by recursion, we know that $\sigma_{\ell/2}B_3U_{R_2}$ is Ξ_1 -reduced. By choice of c_3 , the matrix $\sigma_\ell BU_{1R_1}2U_{R_2}$ is weakly reduced. By Theorem 4, the matrix B_4 is reduced and satisfies the input requirements of $\text{TrLiftLLL}'$. Therefore, the matrix $\sigma_\ell BU_{1R_1}2R_2$ is Ξ_4 -reduced. Theorem 4 can be used to ensure U has the correct format and $\sigma_\ell BU$ is Ξ_1 -reduced. \square

4.3 Complexity analysis

Theorem 6. $\text{Lift-}\tilde{\mathcal{L}}^1$ has bit-complexity

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

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

Now, let $\mathcal{C}(d, \ell)$ be the cost of the remaining operations performed by $\text{Lift-}\tilde{\mathcal{L}}^1$, in dimension d and with lifting target ℓ . If $\ell \leq d$, then $\mathcal{C}(d, \ell) = O(1)$ (as the cost of $\text{TrLiftLLL}'$ 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 $O(1)$ multiplications of d -dimensional integer matrices whose coefficients have bit-length $O(d + \ell)$. 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 $\mathcal{C}(d, \ell) = O(d^\omega \mathcal{M}(d + \ell) \log(d + \ell))$. \square

USING $\text{Lift-}\tilde{\mathcal{L}}^1$ AS LIFT-REDUCTION. We can now bound the cost of $\tilde{\mathcal{L}}^1$ by using the wrapper algorithm from Figure 2 where lift-reduction is implemented by $\text{Lift-}\tilde{\mathcal{L}}^1$. Some care must be taken as the input of $\text{Lift-}\tilde{\mathcal{L}}^1$ is a truncated basis $B'E$ while the input to Step 5, C , is a full-precision basis.

Thus to use $\text{Lift-}\tilde{\mathcal{L}}^1$ we choose $\Xi_1 > \Xi$ and let $C'F := \text{MSB}_{\ell_k - c_3 \cdot d}(C)$ be chosen as in lemma 8. By the construction of MSB and the choice of c_3 we know that $C'F$ is reduced and $\|(C - C'F)_j\| \leq 2^{-\ell_k + c_3 \cdot d} \|C_j\|$. Thus we can call TrLiftLLL on $(C'F, \Xi_1)$ to get $D^{-1}UD$ such that $C'FD^{-1}UD$ is Ξ_1 reduced and then we let $B' := C'FD^{-1}U$ and $E := D$. Now we may call $\text{Lift-}\tilde{\mathcal{L}}^1$ to get U_l so that $\sigma_{\ell_k} C'FD^{-1}UDU_l$ is Ξ_1 -reduced and thus by lemma 8, $\sigma_{\ell_k} CD^{-1}UDU_l$ is Ξ -reduced. On an intuitive level we see that the larger e_i the lower the impact of shifting will be and the smaller precision we need in $b'_{1,i}$ (if all e_i were larger than $\ell_k + c \cdot d$ then $\sigma_{\ell_k} C$ would already be reduced).

This pre/post-processing for $\text{Lift-}\tilde{\mathcal{L}}^1$ is similar to what goes on inside of $\text{Lift-}\tilde{\mathcal{L}}^1$, and the complexity of this processing is less than the complexity in Theorem 6 which we can use as $\mathcal{C}(d - k + 1, \tau_k, \beta_k, \ell_k)$ from lemma 6. Now we need only amortize over all calls to step 5. Using τ in the complexity of lattice reduction allows us to amortize the costs of the LLL-switches over many subsequent calls in the style of [20, 7]. We must adjust the standard Energy function to allow for the truncations which occur.

Lemma 14. *If we define the energy of B with R -factor R as $E(B, n_{\text{trunc}}) = \sum[(i-1) \cdot \log_{\alpha_1}(r_{i,i})] + 2n_{\text{trunc}} \cdot d$ where n_{trunc} is the number of times MSB has been called so far, and α_1 is the α of Ξ_1 , then the number of switches so far satisfies $\tau \leq E - E_0 = \mathcal{O}(d \log |\det B| + dn_{\text{trunc}})$ where E_0 is the initial value of E .*

Proof. Each LLL switch increases the weighted sum of the $r'_{i,i}$ s (see [13, (1.23)]) hence E by at least 1. The lemma is true if $\ell \leq d$, then each increase of n_{trunc} has the potential of decreasing each G-S norm (and again when we return from the truncation). We see in the proof of [1, Co. 5.1] that for any two reduction parameters $\Xi' < \Xi$ there exists an $\epsilon < 1$ such that each G-S norm decreases by a factor no smaller than $(1 - \epsilon)$. However we could adapt the choice of c_3 to $c_3 + 1$ to ensure that $(1 - \epsilon) > (1 - (1/2^d))$ and each decrease in G-S norm must overall not decrease E . We should also note that in Figure 1 the act of adjoining a new row does not change the previous G-S norms but increases their weights. Thus at the time of adjoining a new row E increases at most by $\log |\det C|$. Note that each product by σ_ℓ (including those within the calls to `TrLiftLLL`) cannot decrease G-S norms by lemma 4. Thus the energy never decreases and number of switches is bounded by the growth $E - E_0$. \square

We obtain our main result by combining Theorems 5 and 6, and Lemma 14 to amortize the LLL-costs in Lemma 6 with n_{trunc} bounded by $2\beta = \sum 2(\ell_k/d)$.

Theorem 7. *Given as inputs Ξ and a matrix $B \in \mathbb{Z}^{d \times d}$ with $\max \|b_j\| \leq 2^\beta$, then the \tilde{L}^1 algorithm returns a Ξ -reduced basis of $L(B)$ in time $\mathcal{O}(d^{5+\epsilon}\beta + d^{\omega+1+\epsilon}\beta^{1+\epsilon})$.*

We can refine this analysis to $\mathcal{O}(d^{5+\epsilon} + d^{4+\epsilon} \log |\det B| + d^\omega (\log |\det B|)^{1+\epsilon}) + \mathcal{H}(d, \beta)$ where $\mathcal{H}(d, \beta)$ is the cost of computing the HNF or $\mathcal{O}(d^2\beta^{1+\epsilon})$ if B is already upper-triangular. We note for instance that knapsack lattices (see for instance [11]) can be reduced in time $\mathcal{O}(d^{5+\epsilon} + d^{4+\epsilon}\beta + d^\omega\beta^{1+\epsilon})$.

References

1. X.-W. Chang, D. Stehlé, and G. Villard. Perturbation analysis of the QR Factor R in the context of LLL lattice basis reduction. HAL Report ensl-00529425, <http://prune1.ccsd.cnrs.fr/ensl-00529425/en>, École Normale Supérieure de Lyon, France, 2010.
2. D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.
3. 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.
4. 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.
5. N. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM Publications, 2002.
6. 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.
7. 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.
8. 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.
9. D. Knuth. The analysis of algorithms. In *Actes du Congrès International des Mathématiciens de 1970*, volume 3, pages 269–274. Gauthiers-Villars, 1971.
10. 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.
11. J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM*, 32:229–246, 1985.
12. D. H. Lehmer. Euclid's algorithm for large numbers. *American Mathematical Monthly*, 45:227–233, 1938.
13. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

14. K. S. McCurley and J. L. Hafner. Asymptotically fast triangularization of matrices over rings. *SIAM Journal on Computing*, 20:1068–1083, 1991.
15. D. Micciancio and S. Goldwasser. *Complexity of lattice problems: a cryptographic perspective*. Kluwer Academic Press, 2002.
16. I. Morel, D. Stehlé, and G. Villard. From an LLL-reduced basis to another. In progress.
17. 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.
18. P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM Journal on Computing*, 39(3):874–903, 2009.
19. 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.
20. A. Novocin. *Factoring Univariate Polynomials over the Rationals*. PhD thesis, Florida State University, 2008.
21. C. P. Schnorr. A more efficient algorithm for lattice basis reduction. *Journal of Algorithms*, 9(1):47–62, 1988.
22. C. P. Schnorr. Fast LLL-type lattice reduction. *Information and Computation*, 204:1–25, 2005.
23. A. Schönhage. Schnelle Berechnung von Kettenbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.
24. 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.
25. 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.
26. D. Stehlé. Floating-point LLL: theoretical and practical aspects. Chapter of [19].
27. A. Storjohann. Faster Algorithms for Integer Lattice Basis Reduction. Technical Report TR249, ETH, Dpt. Comp. Sc., Zürich, Switzerland, 1996.
28. 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.
29. 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.