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, $\tilde{L}_1$, with the following specifications: It takes as input an arbitrary basis $B = \{b_i\}_i \in \mathbb{Z}^{d \times d}$ 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 $O(d^{\beta+1+\varepsilon} \beta^{1+\varepsilon} d^3)$ where $\beta = \log \max \|b_i\|$ (for any $\varepsilon > 0$ and $\omega$ is a valid exponent for matrix multiplication). This is the first LLL-reducing algorithm with a time complexity that is quasi-linear in $\beta$ 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 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.

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 $B = \{b_i\}_i \in \mathbb{Z}^{d \times d}$. 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 $\tilde{L}_1$.

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 [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 $O(d^{\beta+1+\varepsilon} \beta^{2+\varepsilon})$ when the input basis $B = \{b_i\}_i \in \mathbb{Z}^{d \times d}$ satisfies $\max \|b_i\| \leq 2^\beta$. For the sake of simplicity, we will only consider full-rank lattices. The current best algorithm for integer multiplication is Fürlers’s, which allows one to multiply two $k$-bit long integers in time $M(k) = O(k (\log k)^{2 \log^* k})$. The analysis of $L^3$ was quickly refined by Kaltofen [11], who showed a $O(d^4 \beta^3 (d + \beta)^{2})$ complexity bound. Schnorr [25] later proposed an algorithm of bit-complexity $O(d^4 \beta (d + \beta)^{1+\varepsilon})$, using approximate computa-

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

STOC ’11, June 6–8, 2011, San Jose, California, USA.
Copyright 2011 ACM 978-1-4503-0691-1/11/06 ...$10.00.

403
tions 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 $\beta$ (for fixed $d$). More recently, Nguyen and Stehlé devised $L^2$ [21], a variant of $L^2$ with complexity $O(d^{1+\beta}(d+\beta))$. The latter bound is quadratic with respect to $\beta$ (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 $L^1$ way is $O(\beta^{1+\epsilon})$, leading to an overall $O(\beta^{1+\epsilon})$ bound for Euclid’s algorithm.
- Lehmer [15] proposed an acceleration of Euclid’s algorithm by the means of truncations. Since the $\ell$ most significant bits of the remainders provide the first $\Omega(\ell)$ bits of the sequence of quotients, one may: Truncate the remainders to precision $\ell$; 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 most significant bits of the remainders providing the first $\Omega(\ell)$ bits of the sequence of quotients.

- The time bound is $O(\ell^{1+\epsilon})$ close to the complexity bound of $O(\beta^{1+\epsilon})$. 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 $O(\beta^{1+\epsilon})$. 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 $O(\beta^{1+\epsilon}(\log \beta)^{d-1})$ for fixed $d$.

**Our Result.** We adapt the Lehmer-Knuth-Schönhage gcd framework to the case of LLL-reduction. $L^3$ taken as input a non-singular $B \in \mathbb{Z}^{d \times d}$, terminates within $O(\beta d^{1+\epsilon})$ bit operations, where $\beta = \log \max \|b_k\|$ and returns a basis of the lattice $B(L)$ spanned by $B$ which is LLL-reduced in the sense of Definition 1 given hereafter. ($L^2$ reduces bases for $Z = (3/4, 1/2, 0)$.) The time bound is obtained via an algorithm that can multiply two $d \times d$ matrices in $O(d^\omega)$ scalar operations. (We can set $\omega \approx 2.376$ [4].)

Our complexity improvement is particularly relevant for applications of LLL-reduction where $\beta$ 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 $Z = (\delta, \eta, \theta)$ 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 $B = QR$ (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 $Z$-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).

Let $Z = (\delta, \eta, \theta)$ be valid LLL-parameters for $i \in \{1, 2\}$. We say that $Z_1$ is stronger than $Z_2$ and write $Z_1 > Z_2$ 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 $(\delta, \eta, \theta)$ close to the ideal parameters $(1, 1/2, 0)$, the derived $\alpha$ tends to $2/\sqrt{3}$):

**Theorem 1** ([2, Th. 5.4]). Let $B \in \mathbb{R}^{d \times d}$ be $(\delta, \eta, \theta)$-LLL-reduced with R-factor $R$. Let $\alpha = \frac{\sqrt{3} d^2 \cdot \beta^{1+\epsilon}}{\beta - \eta^2}$.

Then, for all $i$, $r_{i,i} \leq \alpha \cdot r_{i+1,i+1}$ and $r_{i,i} \leq \|b_k\| \leq \alpha^i \cdot r_{i,i}$.

This implies that $\|b_k\| \leq \alpha^{d-1} \det B|^{1/d}$ and $\alpha^{-d} r_{i,i} \leq \lambda_i \leq \alpha^i r_{i,i}$, where $\lambda_i$ is the $i$th minimum of the lattice $L(B)$.

$L^1$ 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 $\pm 1$ 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 $B_0 := \sigma_k^{-1}B$ is reduced for some $k$ and $\sigma_k$ is the diagonal matrix diag$(2^k, 1, \ldots, 1)$. 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 $\sigma_1 B_0 U_1$ is reduced and repeat until we have $\sigma_k B_0 U_1 \cdots U_k = B(U_1 \cdots U_k)$. Each entry of $U_1$ and each entry of $U_1 \cdots U_k$ can be bounded sensitive to the shape of the lattice. Further we will illustrate that the bit-size of any entry of $U_k$ can be made $O(\ell + d)$ (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 $\max \| b_{i,j} \| < 2^{-p}$ holds for all $i$ and only the most significant $p + O(\log d)$ 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 $\Delta B$, $\Delta$ does not represent anything, i.e., the matrix $\Delta B$ is not a product of $\Delta$ 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 $\Delta B$ with $\max \| \Delta b_{i,j} \| \leq 2^{-p}$. 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 $\max \| \Delta r_{i,j} \| \leq c_0 \cdot \text{cond}(R) \cdot 2^{-p}$.

**Lemma 2** ([2, Le. 5.5]). If $B \in \mathbb{R}^{d \times d}$ with R-factor $R$ is $(\delta, \eta, \theta)$-reduced then $\text{cond}(R) \leq \frac{\delta + 1}{\delta - 1} \rho^3$, with $\rho = (1 + \eta + \theta)\alpha$, with $\alpha$ 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 $\theta$ was introduced in Definition 1, as such a property does not hold if LLL-reductions 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 $\Xi_1$-reduced $B \in \mathbb{R}^{d \times d}$ and any $\Delta B$ with $\max \| \Delta b_{i,j} \| \leq 2^{-c_1 d}$, the matrix $B + \Delta B$ is non-singular and $\Xi_2$-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-$\tilde{L}^1$.** Our quasi-linear general lattice reduction algorithm, $\tilde{L}^1$, is composed of a sequence of calls to a specialized lift-reduction algorithm, Lift-$\tilde{L}^1$. 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-$\tilde{L}^1$ algorithm in Figure 4 is a rigorous implementation of the pseudo algorithm in Figure 1: Lift-$\tilde{L}^1$ must refine current matrices more often than this pseudo algorithm to properly handle a specified reduction.

**Figure 1:** pseudo-Lift-$\tilde{L}^1$.

It could be noted that $\text{clean}$ 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 $O(d^{p+\varepsilon} + \beta)$ stems from a series of $\beta$ calls to H-LLL [20] or $L^2$ [21] on integral matrices whose entries have bit-lengths $O(d)$. 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 $O(d^2 \beta)$ (see Lemma 11). We recall that known LLL reduction algorithms perform two types of vector operations: Either transmutations 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 $O(d^{p+\varepsilon} + \beta^{1+\varepsilon})$ derives from both the HNF computation mentioned above and a series of product trees of balanced matrix multiplications whose overall product has bit-length $O(d \beta)$. Furthermore, the precise cost dependence of $\tilde{L}^1$ in $\beta$ is $\text{Poly}(d) \cdot M(\beta) \log \beta$. We also remark that the cost can be proven to be $O(d^{1+\varepsilon} \log | \det B | + d^{1+\varepsilon} + d^2 (\log | \det B |)^{1+\varepsilon}) + H(d, \beta)$, where $H(d, \beta)$ denotes the cost of computing the Hermite normal form. Finally, we may note that if the size-reduction parameter $\theta$ is not considered as a constant, then a factor $\text{Poly}(\log(1/\theta))$ is involved in the cost of the leaf calls.

**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**

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 ma-
We aim at computing a unimodular $U$ such that $\sigma_iBU$ is reduced. We will use a fast algorithm. An efficient algorithm for this is to start with a triangular lattice basis and then reduce it using a general purpose reduction algorithm. Here, we use HNF and triangularization to shift and reduce bases. We analyze the cost 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 $-h_{i,j} \leq h_{i,i} - h_{j,j}$ for all $i \neq j$. The HNF of a lattice $L$ is reduced. Let $U$ be the block-diagonal matrix $\text{diag}(I_i, U_i')$. Let $B := HNF(B)$. Then $C := \sigma_{k'}C$. Lift-reduction: Find $U'$ unimodular such that $\sigma_{k'}CU'$ is reduced.

Let $B := HNF(B)$. At the end of Step 1, the matrix $B = H$ is upper triangular, $\prod b_{i,i} = \det H \leq 2^\omega$, and the $1 \times 1$ bottom rightmost sub-matrix of $H$ is trivially \texttt{Z}-reduced. In each iteration we \texttt{Z}-reduce a lower-right sub-matrix of $B$ via lift-reduction (increasing the dimension with each iteration). This is done by augmenting the previous \texttt{Z}-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 \texttt{Z}-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 $\ell_k$ is $O(\log \det H) = O(d\beta)$. For a more detailed analysis, see \textcolor{red}{[17, 32]}.

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

1. $B := HNF(B)$.
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. $\ell_k := \log_2(b_{k,k})$, $C := \sigma_{k'}C$.
5. Lift-reduction:

Find $U'$ unimodular such that $\sigma_{k'}CU'$ is reduced.

6. Let $U := \text{block-diagonal} \text{matrix } \text{diag}(I_i, U_i')$.
7. Compute $B := B - U$.
8. Reducing row $i$ symmetrically modulo $b_{i,i}$ for $i < k$.

**Algorithm 6.** The algorithm of Figure 2 \texttt{Z}-reduces $B$ such that $\max |b_{i,i}| \leq 2^\omega$ using $O(d^{\omega+1+\varepsilon}(\beta^{1+\varepsilon} + d)) + \sum_{k=0}^{d} C_k$ bit operations, where $C_k$ is the cost of Step 5 for the specific value of $k$.
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 \( \Xi \)-reduced. The statement is valid for \( k = d \), as a non-zero matrix in dimension 1 is always reduced, and instantiating the statement with \( k = 1 \) ensures that the matrix returned by the algorithm is \( \Xi \)-reduced. The non-trivial ingredient of the proof of the statement is to show that for \( k < d \), the input of the \( \text{lift-reduction} \) of Step 5 is valid, i.e., that at the beginning of Step 5 the matrix \( C \) is \( \Xi \)-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 \( \Xi \)-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 \( r_{1,j} = h_{k,k+j-1}/2^k \), 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 \( \lambda_1(L') \geq \min_{i \geq h_i} h_i \geq 1 \). 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 \( \ell_k \), 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 \( d \) -calls to the lift-reduction by \( \sum_{k < \ell} C_k \). 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 \( \text{current value of } B \) by \( U' \). We consider separately the costs of computing the products by \( U' \) of the \( k \times (d-k+1) \) top-right submatrix \( B' \) of \( B \), and of the \((d-k) \times (d-k+1) \) bottom-right submatrix \( B'' \) of \( B \).

For \( i \leq k \), the magnitudes of the entries of the \( i \)-th row of \( B' \) are uniformly bounded by \( h_{i,i} \). By Lemma 5, if \( e,j < d-k+1 \), then \( |u'_{e,j}| \leq 2^{k+2} \cdot \frac{d}{\alpha_{e,j}} \cdot \frac{d}{\alpha_{e,j}} \) (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 \( \ell_k \), we have \( r_{1,1} \leq 1/2 \). Overall, this gives that the \( j \)-th column of \( U' \) is uniformly bounded as \( \log \|u'_j\| = \mathcal{O}(\ell_k + d + \log r_{j,j}) \). The bounds on the bit-lengths of the rows of \( B' \) 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 \( S = \log det H + d^2 + \log \alpha_{j,j} \). Since \( det H = |\det B| \) the multiplication of \( B' \) with \( U' \) can be performed within \( \mathcal{O}(d^2 M \cdot (\log \det B)/d + d + \ell_k) \) bit operations. We now consider the product \( P := BU' \). By reducedness of \( B \), we have \( \|B\| \leq \alpha^{d} r_{j,j} \) (from Theorem 1). Recall that we have \( |u'_{e,j}| \leq 2^{k+2} \cdot \frac{d}{\alpha_{e,j}} \cdot \frac{d}{\alpha_{e,j}} \). As a consequence, we can uniformly bound \( \log \|u'_j\| \) and \( \log \|p_j\| \) 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 \( \mathcal{O}(d^{k+2} M \cdot (\log \det B)/d + d + \ell_k) \) 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 \( \mathcal{O}(d^k M^2 \cdot \beta^{1+i}) \) of converting \( B \) to Hermite normal form.

We use the term \( C_k \) 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 \( \text{LLL} \)-reducing algorithm [16, 11, 21, 20] or a specifically designed lift-reducing algorithm such as \( \text{Lift-L}^2 \), described in Section 4.

It can be noted from the proof of Lemma 6 that the non-reduction costs can be refined as \( \mathcal{O}(d^{k+2} M \cdot (\log \det B) + d^k M(d) + \mathcal{H}(d, \beta)) \). 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 \( b_{i,j} \). Thus we could replace \( \mathcal{H}(d, \beta) \) by \( \mathcal{O}(d^2 \beta^{1+i}) \) for upper triangular inputs. Using the cost of \( \text{H-LLL} \) for lift-reduction, we can bound the complexity of Figure 2 by \( \text{poly}(d) \cdot \beta^2 \). This is comparable to \( L^2 \) and \( \text{H-LLL} \).

### 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 \( \sigma \cdot BU \) 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 \( \sigma \cdot B \)

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 \( \sigma \cdot BU \) 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.
Lemmas 7. Let \( \Xi_1, \Xi_2, \Xi_3 \) 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 \( B \in \mathbb{Z}^{d \times d} \) be \( \Xi_1 \)-reduced and let \( \Delta B \) be such that \( \max \left| \frac{\|b_i\|}{\|b_j\|} \right| < 2^{-\ell-c_3d} \). If \( \sigma_1(B) + \Delta B \) is \( \Xi_3 \)-reduced for some \( \Xi \), then \( \sigma_1 \) is \( \Xi_2 \)-reduced.

The above result implies that to find a \( U \) such that \( \sigma_1 \) is \( \Xi_2 \)-reduced, we can use \( U \) such that \( \sigma_1(B \cdot E) \) is \( \Xi_3 \)-reduced for some \( \Xi \), for well chosen matrices \( B \) and \( E \), outlined as follows.

Definition 2. For \( B \in \mathbb{Z}^{d \times d} \) with \( \beta = \log \max |b_i| \) and precision \( p \), we choose to store the \( p \) most significant bits of \( B \), MSB\(_p\)(\( B \)), as a matrix product \( B \cdot E \) or just the pair \((B', E)\). This pair should satisfy \( B' \in \mathbb{Z}^{d \times d} \) with \( p = \log \max \|b_i^p\| \), \( E = \text{diag}(2e_i - p) \) with \( e_i \in \Xi \) such that \( \frac{2e_i - |b_i|}{|b_j|} < 2^p \), and \( \max \|b_i - b_j\|^{e_i / p} \leq 2^{p} \).

3.2 Finding a unimodular \( U \) reducing \( \sigma_1 \) at low cost

The algorithm TrLiftLLL (a truncated lift-LLL) we propose is an adaption of the StrengthenLLL from [19], which aims at strengthening the LLL-reducibility of an already reduced basis, i.e., \( \Xi_1 \)-reduced basis with \( \Xi_1 < \Xi_2 \). One can recover a variant of StrengthenLLL by setting \( \ell = 0 \). 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 \( \ell \geq 0, B \in \mathbb{Z}^{d \times d}, E = \text{diag}(2e_i) \) with \( |b_i| \leq 2^{e_i + 1 - p} \), \( e_i \in \Xi \), and \( BE \) is \( \Xi_1 \)-reduced; It runs in time \( O(d^3\ell^2 + (d + \ell)(d + \ell + \tau) + d^2\log \max(1 + |e_i|)) \), where \( \tau = O(d^{3}(\ell + d)) \) 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. \( D = \text{diag}(2^{e_i}) \) with \( d_i \in \Xi \) satisfying \( \max |e_i - d_i| \leq c'_4(\ell + \rho_d) \).
2. \( U \) is unimodular and \( \max |u_{i,j}| \leq 2^{\ell + c'_4d} \).
3. \( D^{-1}UD \) is unimodular and \( \sigma_1(BE)(D^{-1}UD) \) is \( \Xi_2 \)-reduced.

When setting \( \ell = O(d) \), we obtain the base case of lift-LLL, 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 lift-LLL, LLLL, and HLLL algorithms perform two types of basis operations: They either subtract to a vector \( b_1 \ldots b_{d - 1} \) (translation), or they exchange \( b_{d - 1} \) and \( b_i \) (switches). Translations leave the \( r_{i,j} \)'s unchanged. Switches are never performed when the optimal Lovász condition \( r_1^2 \leq r_{2,1}^2 + r_{2,i}^2 \) is satisfied, and thus cannot increase any of the quantities \( \max_{i < l} r_{i,j} \) (for varying \( i \)), nor decrease any of the quantities \( \min_{i \geq j} r_{i,j} \). 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 LLLL ([16, p. 523]), LLLL ([21, Th. 6]) and HLLL ([20, Th. 4.3]). We choose HLLL as this currently provides the best complexity bound, although \( L^3 \) would remain quasi-linear with \( L^2 \). 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 Lemmas 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 \( \ell \). If it is not the case, we may replace \( \ell \) by \( \ell/t \), and add some more lifting at the end.

<table>
<thead>
<tr>
<th>Inputs: LLL parameters ( \Xi ), a ( \Xi )-reduced basis ( B \in \mathbb{Z}^{d \times d} ), a lifting target ( t ), a divisor ( t/\ell ). Output: A ( \Xi )-reduced basis of ( \sigma_1(B) ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Let ( \Xi_0, \Xi_1 ) be valid parameters with ( \Xi_0 &lt; \Xi &lt; \Xi_1 ), ( c_3 ) as in Lemma 7 for ((\Xi_1, \Xi_2, \Xi_3) := (\Xi, \Xi, \Xi_1)), ( c_1 ) as in Lemma 3 with ((\Xi_1, \Xi_2, \Xi_3) := (\Xi, \Xi_0)), and ( c'_4 ) as in Theorem 2 with ((\Xi_1, \Xi_2, c_4) := (\Xi_0, \Xi_2, c_4 + 2)).</td>
</tr>
<tr>
<td>2. For ( k ) from 1 to ( \ell/\ell ) do</td>
</tr>
<tr>
<td>3. ( (B', E, t) := \text{TrLiftLLL}(B', E, t) ).</td>
</tr>
<tr>
<td>4. ( (D, U) := \text{TrLiftLLL}(B', E, t) ).</td>
</tr>
<tr>
<td>5. ( B := \sigma_1(BD^{-1}UD) ).</td>
</tr>
<tr>
<td>6. Return ( B ).</td>
</tr>
</tbody>
</table>

**Figure 3:** The Lehmer-LiftLLL algorithm.

Theorem 3. Lehmer-LiftLLL is correct. Furthermore, if the input matrix \( B \) satisfies \( \max |b_i| \leq 2^p \), then its bit-complexity is \( O(d^3(\ell^{1+\epsilon} + t^{-p}(\ell + \beta))) \).

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

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-LLL, 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-LLL.

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 $\sigma_1 BU$ 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 $\ell$ 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 $\sigma_1 B$ can be transformed into a factored unimodular $U'$ which also reduces $\sigma_1 B$ and for which each entry can be stored with only $O(\ell + d)$ 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 lemma works 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 $A_{u,v,j}$ 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 $\Xi_1, \Xi_2, \Xi_3$ be valid LLL parameters. There exists a constant $c_7$ such that the following holds for all $\ell \geq 0$. Let $B \in \mathbb{R}^{d \times d}$ (with R-factor $R$) be $\Xi_1$-reduced, and $U$ be unimodular such that $\sigma_1 BU$ (with R-factor $R'$) is $\Xi_2$-reduced. If $DU \in \mathbb{Z}^{d \times d}$ satisfies $|\Delta u_{i,j}| \leq 2^{-(\ell+c_7 \cdot d)}r_{i,j}^{c_7}$ for all $i,j$ then $U + DU$ is unimodular.

**Proof.** Since $U$ is unimodular, the matrix $V = U^{-1}$ exists and has integer entries. We can thus write $U + DU = (I + DU^{-1})U$, and prove the result by showing that $U + DU$ is strictly upper triangular, i.e., that $(U^{-1} DU)_{i,j} = 0$ for $i \geq j$. We have $(U^{-1} DU)_{i,j} = \sum_{k \leq d} v_{i,k} \cdot \Delta u_{k,j}$. 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 $DU$ imply that $r_{i,j}^{c_7} \leq 2^{-(\ell+c_7 \cdot d)}$. Since $i \geq j$ and $\sigma_1 BU$ is reduced, Theorem 1 implies that $r_{i,j}^{c_7} \leq 2^{-(\ell+c_7 \cdot d)}$, for some constant $c > 0$. By using the second part of Lemma 5, we obtain that there exists $c' > 0$ such that $|v_{i,k}| \leq 2^{c' \cdot d} \cdot r_{i,j}^{c_7} \leq 2^{c+c' \cdot d}$. As $V$ is integral, setting $c_7 > c + c'$ allows us to ensure that $v_{i,k} = 0$, as desired.

**Lemma 9.** Let $\Xi_1, \Xi_2, \Xi_3$ be valid LLL parameters such that $\Xi_2 > \Xi_1$. 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 $\Xi_1$-reduced, and $U$ be unimodular such that $\sigma_1 BU$ (with R-factor $R'$) is $\Xi_2$-reduced. If $DU \in \mathbb{Z}^{d \times d}$ satisfies $|\Delta u_{i,j}| \leq 2^{-(\ell+c_8 \cdot d)}r_{i,j}^{c_8}$ for all $i,j$ then $\sigma_1 B(U + DU)$ is $\Xi_3$-reduced.

**Proof.** We proceed by showing that $|\sigma_1 B(U)|$ is columnwise small compared to $|\sigma_1 BU|$ and by applying Lemma 3. We have $|DU| \leq 2^{-(\ell+c_8 \cdot d)}\text{diag}(r_{i,j}^{-1})\text{diag}(r_{i,j})$ by assumption, where $c_{i,j} = 1$ for all $i,j$. Since $B$ is $\Xi_1$-reduced, we also have $|R| \leq \text{diag}(r_{i,j})T + \theta_1 T\text{diag}(r_{i,j})$, 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

$$|R\Delta U| \leq 2^{-(\ell+c_8 \cdot d)}(\text{diag}(r_{i,j})T\text{diag}(r_{i,j}) + \theta_1 T)\text{diag}(r_{i,j}).$$

Since $B$ is $\Xi_1$-reduced, by Theorem 1, we have $r_{i,j} \leq \sigma_1 r_{i,j}$ for all $i \leq j$, hence it follows that

$$|R\Delta U| \leq 2^{-(\ell+c_8 \cdot d)}(\sigma_1 + \theta_1)T\text{diag}(r_{i,j}).$$

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

$$\|(\sigma_1 B\Delta U)_{j}\| \leq 2^c \|(B\Delta U)_{j}\| = 2^c |(R\Delta U)_{j}| \leq 2^{c+c_8 \cdot d}r_{i,j}.$$

We complete the proof by noting that $r_{i,j}^{c_8} \leq \|(\sigma_1 BU)_{j}\|$ and by applying Lemma 3 (which requires that $c_8$ is set sufficiently large).

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

**Theorem 4.** Let $\Xi_1, \Xi_2, \Xi_3$ be valid LLL parameters with $\Xi_2 > \Xi_1$. 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 $\Xi_1$-reduced, and $U$ be unimodular such that $\sigma_1 BU$ is $\Xi_2$-reduced. Let $d_i := |\log |b_i||$ for all $i$. Let $D := \text{diag}(2^{d_i})$, $x := k + c_9 \cdot d_i$, $U := 2^{2D^{d_i}}$ and $U' := 2^{-2D^{d_i}}(U')$. We write $\text{Clean}(U, (d_i), x) := (U', D, x)$. Then $U'$ is unimodular and $\sigma_1 BU'$ is $\Xi_3$-reduced. Furthermore, the matrix $\hat{U}$ satisfies $\max |\hat{u}_{i,j}| \leq 2^{c_8+c_{10} \cdot d}$. 

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

Now, consider $\Delta U = U' - U$. Since $\Delta U = 2^{-2D^{d_i}}((U') - U)D$, we have $|\Delta u_{i,j}| \leq 2^{d_i - d_j} \cdot r_{i,j}$ for all $i,j$. Thus by Theorem 1 and Lemma 4, we have $|\Delta u_{i,j}| \leq 2^{-2x+c_8 \cdot d} \cdot r_{i,j}$ for some constant $c$. Applying Lemma 9 and showing that $U'$ is unimodular and $\sigma_1 BU'$ is $\Xi_3$-reduced (if $c_9$ is chosen sufficiently large).

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

$$|\hat{u}_{i,j}| = |u_{i,j}| 2^{x+d_i-d_j} \leq 2^{x+c_8 \cdot d} \frac{r_{i,j}}{2|\log |b_i||} \frac{2|\log |b_i||}{r_{i,j}},$$

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

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 $O(d^2(d + \ell) + d \log \max(1 + |e_{i,j}|))$ 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-L’.

**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 $D = \text{diag}(2^{d_i})$. Let $V = 2^{-y}E^{-1}V'E \in \mathbb{Z}^{d \times d}$ with
V′ ∈ Z^d×d and E = diag(2^{-v_i}). Let ℓ ∈ Z and f_i ∈ Z for i ≤ d. Then it is possible to compute the output (W′, F, z) of Clean(U · V, (f_i)_i, ℓ) (see Theorem 4) from x, y, ℓ, U′, V′, (d_i), (e_i), (f_i)_i in time O(d^2M(t + log d)), where

\[ max_{i,j}(|u'_{ij}|, |v'_{ij}|) \leq 2^\ell \]

and

\[ max_{i}(d_i - e_i, |f_i - e_i|, |\ell - (x + y)|) \leq t. \]

For short, we will write W := U ∩ V, with W = 2^{-w}F^{-1}W′F and F = diag(2^{\ell}).

Proof. We first compute m = max{|d_i - e_i|}. We have

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

where

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

Then we compute T. We multiply U'' by 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 ≤ 3t. We then multiply the latter by V'', which costs O(d^2M(t + log d)). We multiply the result from the left by (FD^{-1}) and from the right by EF^{-1}. From T, the matrix W of Theorem 4 may be computed and rounded within O(d^2t) bit operations. □

It is crucial in the complexity analysis of Lift-LLL that the cost of the merging process above is independent of the magnitude of the scalings (d_i, e_i, and f_i).

4.2 Lift-LLL algorithm

The Lift-LLL 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-LLL is correct.

Proof. When ℓ ≤ 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 ≥ c_1 is chosen so that Lemma 3 applies now and Lemma 7 will apply later in the proof. Thus BU_1 is Ξ_1-reduced and has the correct structure by definition of MSB. Step 4 works (by induction) because BU_1 satisfies the input requirements of Lift-LLL. Thus σ_{i/2}BU_1U_{R_1} is Ξ_1-reduced. Because of the selection of c_3 in Step 3 we know also that σ_{i/2}BU_1U_{R_1} is reduced (weaker than Ξ_1) using Lemma 7. Thus by Theorem 4, the matrix BU_1 is reduced (weakly) and has an appropriate format for TrLiftLLL'. By Theorem 2, the matrix σ_{i/2}BU_1R_1U_{R_2} is Ξ_2-reduced and by Theorem 4 we have that σ_{i/2}BU_1R_1U_{R_2} is Ξ_2-reduced. By choice of c_3 and Lemma 3, we know that the matrix BU_1 is Ξ_1-reduced and satisfies the input requirements of Lift-LLL'. Thus, by recursion, we know that σ_{i/2}BU_1R_1U_{R_2} is Ξ_1-reduced. By choice of c_3 and Lemma 7, the matrix σ_{i/2}BU_1R_1U_{R_2} is weakly reduced. By Theorem 4, the matrix BU_1U_{R_2} is reduced and satisfies the input requirements of TrLiftLLL'. Therefore.

Inputs: Valid LLL-parameters Ξ_1 > Ξ_2 > Ξ_3 > Ξ_4;
a lifting target ℓ;
(B', (e_i)_i) such that B = B'diag(2^{-v_i}) ∈ Ξ_1-reduced and max |b'_i| ≤ 2^{d/2 + d}. 

Output: (U', (d_i)_i, x) such that σ_iBU_1 is Ξ_1-reduced, with U = 2^{-w}diag(2^{d_i}) diag(2^{-d_i}) and max |u'_i| ≤ 2^{d-i/2 + d/2}.

1. If ℓ ≤ d, then use TrLiftLLL with lifting target ℓ.

Then it is possible to compute the output (W', F, z) of Clean(U · V, (f_i)_i, ℓ) (see Theorem 4) from x, y, ℓ, U', V', (d_i), (e_i), (f_i)_i in time O(d^2M(t + log d)), where

\[ max_{i,j}(|u'_{ij}|, |v'_{ij}|) \leq 2^\ell \]

and

\[ max_{i}(d_i - e_i, |f_i - e_i|, |\ell - (x + y)|) \leq t. \]

For short, we will write W := U ∩ V, with W = 2^{-w}F^{-1}W′F and F = diag(2^{\ell}).

Proof. We first compute m = max{|d_i - e_i|}. We have

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

where

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

Then we compute T. We multiply U'' by 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 ≤ 3t. We then multiply the latter by V'', which costs O(d^2M(t + log d)). We multiply the result from the left by (FD^{-1}) and from the right by EF^{-1}. From T, the matrix W of Theorem 4 may be computed and rounded within O(d^2t) bit operations. □

4.3 Complexity analysis

Theorem 6. Lift-LLL has bit-complexity

\[ O \left( d^{1+\epsilon}(d + \ell + \tau) + d^\omega M(\ell) \log \ell + \ell \log(\beta + \ell) \right), \]

where τ is the total number of LLL-switches performed by the calls to H-LLL (through TrLiftLLL), and max |b'_i| ≤ 2^d.

Proof. We first bound the total cost of the calls to TrLiftLLL. There are O(1 + ℓ/d) such calls, and for any of these the lifting target is O(δ). Their contribution to the cost of Lift-LLL is therefore O(d^{1+\epsilon}(d + \ell + \tau)). Also, the cost of handling the exponents in the diverse diagonal matrices is O(d(1 + ℓ/d)log(\beta + \ell)).

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

4.4 L̃ algorithm

The algorithm of Figure 4 is the Knuth-Schönage-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-LLL.
Algorithm $\bar{L}^1$ We define $\bar{L}^1$ 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 $\bar{L}^1$ provides valid inputs to Lift-$\bar{L}^1$. 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-$\bar{L}^1$ which requires a truncated basis $B'$. Just as in Lift-$\bar{L}^1$ we use a stronger reduction parameter to compensate for needing a truncation.

**Inputs:** Valid LLL parameters $\Xi_1 > \Xi$.

$C$ $\Xi$-reduced with $\beta_k = \log \max \|C\|$; a lifting target $\ell_k$.

**Output:** $U$ unimodular, such that $\sigma_C U$ is $\Xi$-reduced.

1. $C'_F := MSB_{\ell_0+d}(C)$
2. Call TrLiftLLL on $(C'_F, \Xi_1)$. Let $D^{-1} U_0 D$ be the output.
3. $B' := C'_F D^{-1} U_0$; $E := D$
4. Call Lift-$\bar{L}^1$ on $(B', E, \Xi_1)$. Let $U_{\ell_k}$ be the output.
5. Return $U := D^{-1} U_0 D U_{\ell_k}$.

**Figure 5: From Figure 2 to Lift-$\bar{L}^1$**

This processing before Lift-$\bar{L}^1$ is similar to what goes on inside of Lift-$\bar{L}^1$. 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-$\bar{L}^1$, $O(d^{d+\epsilon}(d+\ell_k+\tau_i)+d^\omega M(\ell_k) \log \ell_k + \ell_k \log (\beta_k + \ell_k))$ from Theorem 6, which we can use as $C_k$ from Lemma 6.

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

**Lemma 11. Consider the execution of Steps 2–8 of $\bar{L}^1$ (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 in the execution. Let $B$ be a basis occurring at any moment of Step 5 during the execution. Let $R$ be the R-factor of $B$ and $n_{\text{MSB}}$ be the number of times MSB has been called so far. We define the energy of $B$ as $\mathcal{E}(B, n_{\text{MSB}}) := \frac{1}{\log 1780} \sum (\log |i-j| \cdot \log r_{i,j}) + d^2 n_{\text{MSB}}$ (using the natural logarithm). Then the number of LLL-switches performed so far satisfies $t \leq \mathcal{E}(B, n_{\text{MSB}}) = O(d \cdot \log \det H)$.

**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 $\Xi_0$ is the strongest set of LLL parameters ever considered during the execution of the algorithm, each LLL switch increases the weighted sum of the $r_{i,j}$'s (see [16, (1.23)]) and hence $\mathcal{E}$ by at least 1.

We now consider truncations. Each increase of $n_{\text{MSB}}$ possibly decreases each $r_{i,j}$ (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 $\epsilon < 1$ such that each $r_{i,j}$ decreases by a factor no smaller than $(1 + \epsilon)$. Overall, the possible decrease of the weighted sum of the $r_{i,j}$'s is counterbalanced by the term $d^2 n_{\text{MSB}}$ 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 $r_{i,j}$'s but increases their weights. Since at the moment of an adjoining all log $r_{i,j}$'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 $\sigma_i$ (including those within the calls to TrLiftLLL) cannot decrease any $r_{i,j}$, 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 $\mathcal{E}(B, n_{\text{MSB}}) - \mathcal{E}((h_{d,d}), 0)$. The initial value $\mathcal{E}((h_{d,d}), 0)$ of the energy is $\geq 0$. Also, at the end of the execution, the term $\sum (i-1) \log r_{i,j}$ is $O(\log \det H)$. As there are 5 calls to MSB in the algorithm from Figure 4 (including those contained in the calls to TrLiftLLL), we can bound $d^2 n_{\text{MSB}}$ by $5d^2 \sum \ell_k/d = 5 \log \det H$. □

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 $\Xi$ and a matrix $B \in \mathbb{Z}^{d \times d}$ with $\max \|b_i\| \leq 2^\beta$, the $\bar{L}^1$ algorithm returns a $\Xi$-reduced basis of $L(B)$ within $O(d^{d+\beta} + d^{\omega+1} + d^{\beta+\epsilon})$ 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


