Faster LLL-type reduction of lattice bases

Arnold Neumaier
Universität Wien, Austria
Fakultät für Mathematik
Arnold.Neumaier@univie.ac.at

Damien Stehlé
ENS de Lyon, France
Laboratoire LIP
(U. Lyon, CNRS, ENSL, INRIA, UCBL)
damien.stehle@ens-lyon.fr

ABSTRACT

We describe an asymptotically fast variant of the LLL lattice reduction algorithm. It takes as input a basis $\mathbf{B} \in \mathbb{Z}^{n\times n}$ and returns a (reduced) basis $\mathbf{C}$ of the Euclidean lattice L spanned by $\mathbf{B}$, whose first vector satisfies $\|\mathbf{c}_1\| \leq (1+c)(4/3)^{(n-1)/4}(\det L)^{1/n}$ for any fixed $c > 0$. It terminates within $O(n^{4+\varepsilon}\beta^{1+\varepsilon})$ bit operations for any $\varepsilon > 0$, with $\beta = \log \max_i \|\mathbf{b}_i\|$. It does rely on fast integer arithmetic but does not make use of fast matrix multiplication.

Keywords

Lattice reduction; LLL; blocking

1. INTRODUCTION

A Euclidean lattice is a set $L = \mathbf{B}\mathbb{Z}^n$ of all integer linear combinations of the columns of a full column rank matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$. In this setup, the columns $\mathbf{b}_i$ of $\mathbf{B}$ are said to form a basis of the lattice. Any lattice of dimension $n \geq 2$ admits infinitely many lattice bases and two basis matrices $\mathbf{B}, \mathbf{C}$ span the same lattice if and only if there exists $\mathbf{U} \in \mathbb{Z}^{n\times n}$ of determinant ± 1 such that $\mathbf{B} = \mathbf{C}\mathbf{U}$. Such a matrix $\mathbf{U}$ is said unimodular. Note that this implies that the lattice determinant $\det L = \sqrt{\det(\mathbf{B}^T\mathbf{B})}$ does not depend on the choice of the basis $\mathbf{B}$ of L.

Many interesting computational problems can be reduced to finding a basis $\mathbf{C}$ of a lattice L consisting of short vectors, from an arbitrary input basis $\mathbf{B}$ of L (see, e.g., [12]). This task, called lattice reduction, comes in different flavors. The most standard variant is LLL reduction [6], as it can be performed in polynomial time (if the input basis $\mathbf{B}$ is integral) and provides reasonable output guarantees. In particular, the first vector of a LLL-reduced basis $\mathbf{C}$ satisfies:

$$\|\mathbf{c}_1\| \leq \alpha^{(n-1)/4}(\det L)^{1/n}, \tag{1}$$

where α can be any constant greater than $4/3$.

Assume now that the input basis $\mathbf{B}$ is integral. Let $\beta = \log \max_i \|\mathbf{b}_i\|$. Observe that we may assume that $\beta \geq \Omega(n)$, as else (1) is already reached by the input basis $\mathbf{B}$. For the sake of simplicity, we further assume that $m = n$. The original LLL algorithm terminates within $O(n^5\beta^{2+\varepsilon})$ bit operations [6, 4], where $\varepsilon > 0$ is arbitrary. The $\widetilde{L}^1$ algorithm [13] produces bases satisfying (1) in time $O(n^{5+\varepsilon}\beta^{1+\varepsilon})$. If $\beta = O(n^c)$ for some $c < 2 - 1/(5-\omega) \approx 1.619$, then $\widetilde{L}^1$ is outperformed by Storjohann's algorithm [18], whose run time is $O(n^{3+1/(5-\omega)+\varepsilon}\beta^{2+\varepsilon})$. Here $\omega < 2.373$ refers to the matrix multiplication exponent.

Our result. We present an algorithm that takes as input a basis matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$ of an arbitrary integer lattice L and returns a basis matrix $\mathbf{C}$ of L whose first vector satisfies

$$\|\mathbf{c}_1\| \leq (1+c)(4/3)^{(n-1)/4}(\det L)^{1/n}. \tag{1a}$$

Here $c > 0$ is an algorithm parameter that can be set arbitrarily close to 0 (and can even tend to 0). The algorithm terminates within $O(n^{4+\varepsilon}\beta^{1+\varepsilon})$ bit operations, for any $\varepsilon > 0$. Note that (1a) is a little stronger than (1). Further, the cost increase when $c > 0$ gets lower is quite limited. The algorithm is not only the (asymptotically) fastest known today, but is also arguably simpler than those of [13, 18]: it does not rely on basis roundings as [13] or fast matrix multiplication as [18].

Techniques. A classical approach to decrease the cost dependency of LLL-type algorithms with respect to n is to rely on blocking techniques. This was first proposed by Schönhage [15] and further investigated in [18, 5]. We use a recursive blocking technique inspired from that of [5]: the Gram-Schmidt orthogonalisation (GSO) of the basis is split into large blocks such that consecutive diagonal blocks overlap by half, and the reduction algorithm makes recursive calls on these blocks. Unfortunately, the algorithm of [5] suffers from several drawbacks. First, the guaranteed output quality is worse than (1): the first output vector is only known to satisfy $\|\mathbf{c}_1\| \leq 2^{O(n\log n)}(\det L)^{1/n}$. Second, its cost bound $O(n^{3+\varepsilon}\beta^{2+\varepsilon})$ is (more than) quadratic in β: this is because there are $\Omega(\beta)$ recursion leaves, and the cost of each one is $\Omega(\beta)$.

We overcome both limitations by changing the strategy used at each recursion level for selecting the inputs to the recursive calls. Instead of a local (LLL-like) strategy consisting in choosing the first (or an arbitrary) block for which some progress can be made, we use a "rolling pin" strategy *à la* Block-Korkin-Zolotarev [14, 3]: BKZ reduces the blocks by increasing value of their starting index, and repeats such a "rolling pin tour" sufficiently many times for the basis to be reduced. This global strategy allows to avoid spending

ISSAC '16 July 20–22, 2016, Waterloo, Ontario, CANADA

DOI: 10.475/123_4

time making little local progress on the GSO and instead ensuring some significant global progress at every tour. More concretely, it was shown in [3] that $O(\log \beta)$ block calls suffice for a fixed value of n, rather than $O(\beta)$, to obtain bases that are well reduced. Note that BKZ was introduced in a different context: it aims at finding much shorter lattice vectors than guaranteed by (1), by using for each block an HKZ-reduction algorithm such as the one from [8]. The cost of BKZ grows as $2^{O(s)}(n\beta)^{O(1)}$, if s denotes the block-size. The convergence analysis from [3] can be adapted to recursive blocking for LLL-type reduction (rather than non-recursive calls to an HKZ-reduction). We adapt the simpler analysis by Neumaier in [10] of the SDBKZ variant of BKZ recently proposed by Micciancio and Walter [9]. The fact we obtain (1a) rather than (1) is a side-product of using a global ("rolling pin") strategy. This phenomenon was already noted in [3].

Our "rolling pin" strategy differs from that of BKZ in two ways. First, our blocks overlap by half (like [5]), instead of $s-1$ dimensions out of s, in order to decrease the cost dependence in n. Note that considering blocks overlapping by half was also considered in [1], in the context of finding much shorter vectors than provided by (1). Second, the "rolling pin tours" proceed backwards (from the end of the diagonal to the start) rather than forwards, in order to prove that the first vector of the output basis satisfies (1a). Note that for block-size $s = 2$, the BKZ and SDBKZ algorithms produce short bases satisfying (1a), but the analyses from [3, 10] lead to a run-time bound of $O(n^{6+\varepsilon}\beta^{1+\varepsilon})$.

Extensions. The algorithm also works for rational input matrices $\mathbf{B} \in \mathbb{Q}^{n\times n}$. If β denotes the maximum over all entries of $\mathbf{B}$ of the sum of the bit-sizes of the numerator and denominator, then the cost can be bounded as $O(n^{5+\varepsilon}\beta^{1+\varepsilon})$. If the input matrix has real entries, then it may be rounded and our algorithm can be used to compute a reducing unimodular transformation. We refer the reader to [2] for a sufficient bound on the rounding precision to guarantee reducedness of the final basis.

The algorithm may be adapted to take as input the Gram matrix $\mathbf{B}^T\mathbf{B}$ rather than the basis matrix $\mathbf{B}$. If $\mathbf{B}^T\mathbf{B}$ is integral, then the time bound $O(n^{4+\varepsilon}\beta^{1+\varepsilon})$ still applies, and the algorithm returns an equivalent Gram matrix $\mathbf{C}^T\mathbf{C} = \mathbf{U}^T\mathbf{C}^T\mathbf{C}\mathbf{U}$ and the associated unimodular transformation $\mathbf{U} \in \mathbb{Z}^{n\times n}$ such that $\|\mathbf{c}_1\|$ is bounded as in (1a). If $\mathbf{B} \in \mathbb{Z}^{m\times n}$ is not square, one may first compute $\mathbf{B}^T\mathbf{B} \in \mathbb{Z}^{n\times n}$, run the latter algorithm on $\mathbf{B}^T\mathbf{B}$ and apply to $\mathbf{B}$ every unimodular operation made. This results in an algorithm whose run time is $O(mn^{\omega+\varepsilon}\beta^{1+\varepsilon} + n^{4+\varepsilon}\beta^{1+\varepsilon})$.

2. REMINDERS

In this section, we give the background that is necessary for presenting our algorithm. We refer to [7, 12] for introductions on lattices and to the LLL lattice reduction algorithm.

Magnitudes of reducing matrices. Assume $\mathbf{B}, \mathbf{C} \in \mathbb{Q}^{n\times n}$ are two basis matrices of the same lattice. Let $\mathbf{U} = \mathbf{B}^{-1}\mathbf{C}$. As mentioned above, matrix $\mathbf{U}$ is integral and has determinant ± 1. We may bound the magnitude of every entry of $\mathbf{U}$ as

$$\max_{i,j} |u_{i,j}| \leq (\max_i \|\mathbf{b}_i\|)^{n-1}(\max_i \|\mathbf{c}_i\|)/|\det \mathbf{B}|. \quad (2)$$

Note that if $\mathbf{B}$ is integral, then the latter is bounded from above by $(\max_i \|\mathbf{b}_i\|)^{n-1}(\max_i \|\mathbf{c}_i\|)$.

Lattice reduction aims at finding $\mathbf{U}$ such that $\mathbf{BU}$ has nice properties, such as (1). In our algorithm, such a matrix will be computed by progressively updating a $\mathbf{U}$ initially set to $\mathbf{Id}$. We will use the latter bound to update $\mathbf{U}$ modulo a large integer during the execution of our algorithm: this is to master the arithmetic cost of updating $\mathbf{U}$, while making sure that the final value of $\mathbf{U}$ is correct as it is guaranteed to be smaller than half that modulus. Note that to use such a bound, we need to possess an a priori bound on the norms of the columns of the output matrix $\mathbf{C} = \mathbf{BU}$. As proved in Subsection 3.1, we have an a priori bound $\max_i \|\mathbf{c}_i\| \leq \sqrt{n}\max_i \|\mathbf{b}_i\|$.

Gram-Schmidt orthogonalisation. Let $(\mathbf{b}_1|\dots|\mathbf{b}_n) \in \mathbb{Z}^{n\times n}$ be non-singular. We recall that the Gram-Schmidt orthogonalisation (GSO) $(\mathbf{b}_1^*|\dots|\mathbf{b}_n^*) \in \mathbb{Q}^{n\times n}$ is defined as follows (for $1 \leq j < i \leq n$):

$$\mathbf{b}_i^* := \mathbf{b}_i - \sum_{j<i} \mu_{i,j}\mathbf{b}_j^*, \quad \text{with} \quad \mu_{i,j} := \frac{\langle \mathbf{b}_i, \mathbf{b}_j^* \rangle}{\|\mathbf{b}_j^*\|^2}.$$

Our algorithm will work on $\mathbf{B} = (\mathbf{b}_1|\dots|\mathbf{b}_n)$ only implicitly, by instead manipulating the pair $(\mathbf{d}, \mathbf{M}) \in \mathbb{Q}^n \times \mathbb{Q}^{n\times n}$ with $\mathbf{d} := (\|\mathbf{b}_i^*\|^2)_{i\leq n}$ and $m_{i,j} = \mu_{i,j}$ if $j < i$ and $m_{i,j} = 0$ otherwise. We call this pair the GSO coefficients of $\mathbf{B}$.

We recall that the numerators and denominators of the GSO coefficients $(\mathbf{d}, \mathbf{M})$ are of magnitudes bounded from above by

$$\begin{aligned} &\big(\max_{i\leq n} \det\left((\mathbf{b}_1|\dots|\mathbf{b}_i)^T(\mathbf{b}_1|\dots|\mathbf{b}_i)\right)\big)\big(\max_i \|\mathbf{b}_i\|^2\big) \\ &= \big(\max_{i\leq n} \prod_{j\leq i} d_j\big)\big(\max_i (d_i + \sum_{j<i} d_j m_{i,j}^2)\big) \end{aligned} \quad (3)$$

(see, e.g., [6, Proof of (1.26)]). Applying a transformation matrix $\mathbf{U}$ on $\mathbf{B}$ incurs an update of the GSO coefficients which can be computed from $\mathbf{U}$ and the GSO coefficients only. Further, when this transformation corresponds to a swap of two consecutive columns of $\mathbf{B}$ or the subtraction of an integer multiple of column to another column, the update of the GSO may be performed on $O(n)$ arithmetic operations.

Size-reduction. A basis matrix $\mathbf{B}$ is said size-reduced if its GSO coefficients satisfy $|\mu_{i,j}| \leq 1/2$ for all $i > j$. A size-reduction algorithm is described in [6, Eq. (21)]: for a given index i, it makes all $|\mu_{i,j}|$'s smaller or equal to $1/2$ by adding to $\mathbf{b}_i$ appropriate integer multiples of $\mathbf{b}_j$ for $j < i$. Using this algorithm for $i = 2, 3, \dots, n$ leads to a size-reduction algorithm for basis matrices. For a given column, the algorithm performs $O(n^2)$ arithmetic operations on rational numbers whose numerators and denominators stay bounded from above by

$$2^n \left(\max_{i\leq n} \prod_{j\leq i} d_j\right)\left(\max_i d_i + \sum_{j<i} d_j m_{i,j}^2\right), \quad (4)$$

where $(\mathbf{d}, \mathbf{M})$ are the GSO coefficients of $\mathbf{B}$ (see [6, Proof of (1.26)]). Size-reduction can be performed using and updating the GSO coefficients only. The unimodular transformation obtained by size-reducing a column has $O(n)$ non-trivial coefficients which are easily obtained during the execution of the algorithm.

REDUCTION OF 2-DIMENSIONAL LATTICES. Schönhage's algorithm [16] takes as input the GSO coefficients of a basis $\mathbf{B} \in \mathbb{Q}^{2\times 2}$ and returns a unimodular $\mathbf{U} \in \mathbb{Z}^{2\times 2}$ such that matrix $\mathbf{C} = \mathbf{BU}$ has smallest possible $\|\mathbf{c}_1\|$. This implies, by Minkowski's theorem, that $\|\mathbf{c}_1\| \leq (4/3)^{1/4}|\det \mathbf{B}|^{1/2}$. Its run time is $O(\mathcal{M}(k)\log k)$, where k denotes the (total) bit-size of the input GSO coefficients, and $\mathcal{M}(\ell)$ denotes the time required to multiply two ℓ-bit long integers.

We observe that Yap's algorithm [19] allows one to reduce 2-dimensional lattice bases in quasi-linear time. We use Schönhage's algorithm instead as we have GSO coefficients as inputs rather than a basis matrix.

3. THE ALGORITHM(S)

Algorithm `Reduce`, presented in Figure 1, is our main algorithm. However, the complexity gain over previous works stems from `RecursiveReduce`, described in Figure 2, which `Reduce` may call several times. Algorithm `Reduce` is needed as `RecursiveReduce` works only in specific dimensions, because of its recursive nature. If the dimension n of the input of `Reduce` is one of those specific dimensions, then `Reduce` may call `RecursiveReduce` only once. Else, the number of calls to `Reduce` is $(n\beta)^{o(1)}$.

The sequence of specific dimensions is driven by the correctness and cost analyses of `RecursiveReduce`. It is defined recursively: the base case is $n_1 = 2$, and, for $k \geq 2$, we define n_k as the largest multiple of n_{k-1} that is $\leq \sqrt{2}^{4-k} L^{k(k-1)/4}$. Here $L = O(\log(n\beta/\eta))$, where $\eta > 0$ is an algorithm parameter that drives the strength of the reduction (the smaller $\eta > 0$, the more reduced). We refer to Lemma 9 for the precise definition of L. Note that each n_k is even, and, assuming that L is sufficiently large, we have $n_k \in [\sqrt{2}^{3-k} L^{k(k-1)/4}, \sqrt{2}^{4-k} L^{k(k-1)/4}]$ for all k. As a result, the number of recursion levels r of `RecursiveReduce` satisfies

$$r = O(\sqrt{\log n / \log L}) = O(\sqrt{\log n / \log\log(n\beta/\eta)}).$$

Algorithms `Reduce` and `RecursiveReduce` use the GSO coefficients to define the lower-dimensional inputs of the recursive calls. Given the GSO $(\mathbf{d}, \mathbf{M}) \in \mathbb{Q}^n \times \mathbb{Q}^{n\times n}$, we define the GSO block of starting index $t+1$ and dimension s as $(\mathbf{d}', \mathbf{M}') \in \mathbb{Q}^s \times \mathbb{Q}^{s\times s}$, with $d'_i = d_{t+i}$ for $1 \leq i \leq s$ and $m'_{i,j} = m_{t+i,t+j}$ for $1 \leq i, j \leq s$. Note that $(\mathbf{d}', \mathbf{M}')$ are exactly the GSO coefficients of the vectors $\mathbf{b}^{(t)}_{t+1}, \ldots, \mathbf{b}^{(t)}_{t+s}$ obtained by projecting $\mathbf{b}_{t+1}, \ldots, \mathbf{b}_{t+s}$ orthogonally to the span of $\mathbf{b}_1, \ldots, \mathbf{b}_t$.

A call to `RecursiveReduce` on $(\mathbf{d}', \mathbf{M}')$ provides an s-dimensional unimodular matrix $\mathbf{U}'$ that reduces the basis $\mathbf{b}^{(t)}_{t+1}, \ldots, \mathbf{b}^{(t)}_{t+s}$. This matrix may be extended into an n-dimensional matrix $\mathbf{U}''$ so that when applied on the matrix $\mathbf{B}$ corresponding to $(\mathbf{d}, \mathbf{M})$, it does not change columns $\mathbf{b}_i$ for $i \leq t$ and $i > t+s$, and modifies vectors $\mathbf{b}_{t+1}, \ldots, \mathbf{b}_{t+s}$ exactly by applying $\mathbf{U}'$ to them. This is the purpose of Step 9 of `Reduce` and Step 6 of `RecursiveReduce`. The transformation may be applied to $(\mathbf{d}, \mathbf{M})$ directly. This may modify d_i for $i \in [t+1, t+s]$, and $m_{i,j}$ for $i \in [t+1, t+s]$ and $j > i$, and for $j \in [t+1, t+s]$ and $i < j$. Updating $(\mathbf{d}, \mathbf{M})$ costs $O(ns^2)$ arithmetic operations. This is performed at Step 10 of `Reduce` and Step 8 of `RecursiveReduce`.

In `Reduce`, the first four steps are meant to prepare the actual computation. `Reduce` calls `RecursiveReduce` on n_r-dimensional GSO blocks, with starting indices $k = 0, \ldots, n - n_r$. Note that the blocks overlap by $n_r - 1$ dimensions. `Reduce` may be viewed as BKZ with block-size n_r and calls to HKZ reduction within the blocks replaced by calls to `RecursiveReduce`. The analysis from [3] could be adapted from BKZ to `RecursiveReduce`. We chose to adapt the analysis of SDBKZ from [10], as it is more compact (see Section 5).

Inputs: A basis matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$ of a lattice L.
A reduction parameter $\eta \in (0, 1]$.
Output: A reduced basis matrix of L.

1. Let n_r be the largest specific dimension that is $\leq n$.
2. Compute the GSO coefficients $(\mathbf{d}, \mathbf{M})$ of $\mathbf{B}$.
3. Size-reduce $\mathbf{B}$ while updating $(\mathbf{d}, \mathbf{M})$ accordingly.
4. Define $\overline{\beta} = 2 + \lceil \log_2(\sqrt{n}(\max_{i\leq n} \|\mathbf{b}_i\|)^n) \rceil$.
5. Repeat the following $(n/n_r)^2 \lceil \ln(\overline{\beta}/\eta) \rceil$ times, unless $n = n_r$, in which case once suffices.
6. For $k = 0$ to $n - n_r$, do:
7. Let $(\mathbf{d}', \mathbf{M}')$ be the GSO block of starting index $k+1$ and dimension n_r.
8. Call `RecursiveReduce` on inputs $(\mathbf{d}', \mathbf{M}')$, η, $\overline{\beta}$. Let $\mathbf{U} \in \mathbb{Z}^{n_r \times n_r}$ be the output.
9. Let $\mathbf{U}' \in \mathbb{Z}^{n\times n}$ be the identity matrix, except that $u'_{i,j} = u_{i-k,j-k}$ for $i, j \in [k+1, k+n_r]$.
10. Replace $\mathbf{B}$ by $\mathbf{BU}'$, updating $(\mathbf{d}, \mathbf{M})$ accordingly.
11. Size-reduce $\mathbf{B}$, updating $(\mathbf{d}, \mathbf{M})$ accordingly.
12. Return $\mathbf{B}$.

Figure 1: Algorithm `Reduce`.

Inputs: GSO coefficients $(\mathbf{d}, \mathbf{M}) \in \mathbb{Q}^n \times \mathbb{Q}^{n\times n}$.
A reduction parameter $\eta \in (0, 1]$ and $\overline{\beta} \in \mathbb{Z}$.
Output: A unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n\times n}$.

1. If $n = 2$, return the output of Schönhage's algorithm.
2. Let $\mathbf{U} = \mathbf{Id}$.
Repeat the following $(2n_r/n_{r-1})^2 \lceil \ln(8\overline{\beta}/\eta) \rceil$ times.
3. For k from $2(n_r/n_{r-1} - 1)$ down to 0, do:
4. Let $(\mathbf{d}', \mathbf{M}')$ be the GSO block of starting index $t+1 = kn_{r-1}/2 + 1$ and dimension n_{r-1}.
5. Call `RecursiveReduce` on inputs $(\mathbf{d}', \mathbf{M}')$, η, $\overline{\beta}$. Let $\mathbf{U}' \in \mathbb{Z}^{n_{r-1}\times n_{r-1}}$ be the output.
6. Let $\mathbf{U}'' \in \mathbb{Z}^{n_r \times n_r}$ be the identity matrix, except that $u''_{i,j} = u'_{i-t,j-t}$ for $i, j \in [t+1, t+n_{r-1}]$.
7. $\mathbf{U} := \mathbf{UU}'' \bmod 2^{\overline{\beta}}$.
8. Update $(\mathbf{d}, \mathbf{M})$ according to transformation $\mathbf{U}''$.
9. Size-reduce $(\mathbf{d}, \mathbf{M})$ from indices $t+1$ to $t+n_{r-1}$, and update $\mathbf{U}$ modulo $2^{\overline{\beta}}$ accordingly.
10. Return $\mathbf{U}$.

Figure 2: Algorithm `RecursiveReduce`.

In `RecursiveReduce`, the input n_r-dimensional GSO is split into $2n_r/n_{r-1} - 1$ blocks of dimension n_{r-1} and starting indices $kn_{r-1}/2 + 1$ for $k = 0, 1, \ldots, 2(n_r/n_{r-1} - 1)$. Note that these blocks overlap by half. The n_r-dimensional call makes backwards sweeps of recursive calls on these blocks: it considers each block by decreasing value of k, and when it reaches $k = 0$, it starts a new tour with $k = 2(n_r/n_{r-1} - 1)$. The number of tours $(2n_r/n_{r-1})^2 \lceil \log_2(8\overline{\beta}/\eta) \rceil$ is dictated by the convergence towards reduced bases analysis of the successive-backwards-sweeps strategy (see Section 4). Moving backwards rather than forwards is important for proving (1a): indeed, at the end, the block corresponding to

$k = 0$ is recursively reduced, and so is its first sub-block, and so is its first sub-sub-block, etc. The leaves of the recursion correspond to reductions of 2-dimensional lattice bases. These are performed using Schönhage's algorithm.

THEOREM 1. *Let $c > 0$. Given as input a basis matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$ of a lattice L and parameters $\eta = c/(3n)$, r and $(n_i)_{i\leq r}$, Algorithm* `Reduce` *returns a basis $\mathbf{C}$ of L which is size-reduced and satisfies:*

$$\|\mathbf{c}_1\| \leq (1+c)\big(4/3\big)^{(n-1)/4}(\det L)^{1/n}.$$

THEOREM 2. *Given $\mathbf{B} \in \mathbb{Z}^{n\times n}$ and $\eta = \Omega(1/n^2)$ as inputs, Algorithm* `Reduce` *terminates within*

$$O\left(kn^3\mathcal{M}(n\beta)\right) \quad \textit{bit operations,}$$

with $\beta := \log \max_i \|\mathbf{b}_i\|$ and

$$k := 2^{O\left(\sqrt{(\log n)(\log\log(n\beta))}\right)}\log^2(n\beta).$$

As k is $(n\beta)^{o(1)}$, the latter cost bound is $\leq O(n^{4+\varepsilon}\beta^{1+\varepsilon})$ for all $\varepsilon > 0$.

3.1 Elementary properties

We start with a few basic observations. The first one states that the algorithms are invariant under scaling of $\mathbf{d}$.

LEMMA 1. *Consider two distinct inputs $(\mathbf{d}, \mathbf{M})$, $(\mathbf{d}', \mathbf{M}')$ of* `RecursiveReduce` *(with all other inputs identical). Let $\mathbf{U}$ and $\mathbf{U}'$ be the respective outputs. Assume $\mathbf{M} = \mathbf{M}'$ and that there exists $t > 0$ such that $\mathbf{d}' = t\mathbf{d}$. Then $\mathbf{U} = \mathbf{U}'$. The same property holds for* `Reduce`*.*

The following lemma is useful to bound the bit-sizes of all rationals occurring during the algorithm, and also the magnitude of the output unimodular matrix $\mathbf{U}$.

LEMMA 2. *During the execution of* `Reduce`*, the only operation that may change the value of $\max_i d_i$ is an execution of Schönhage's algorithm. Further, the value of this quantity after such an execution of Schönhage's algorithm cannot be larger than its value before this execution of Schönhage's algorithm.*

PROOF. When considering basis $\mathbf{B}$, the only basis/GSO operations that are performed during the execution of `Reduce` are either a replacement of some basis vector $\mathbf{b}_i$ by $\mathbf{b}_i + \sum_{j<i} x_j\mathbf{b}_j$, or an execution of Schönhage's algorithm on vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ orthogonalized against $\mathbf{b}_1, \ldots, \mathbf{b}_{i-1}$. Only executions of Schönhage's algorithm can change $\mathbf{d}$.

As Schönhage's algorithm applied to 2-dimensional $(\mathbf{d}, \mathbf{M})$ produces a $(\mathbf{d}', \mathbf{M}')$ with smallest d'_1, we must have $d'_1 \leq d_1 \leq \max(d_1, d_2)$. Further, Pythagoras' theorem and the fact that we use integer transformations imply that $d'_1 \geq \min(d_1, d_2)$. Now, note that $d'_1 d'_2 = d_1 d_2$ (the determinant is preserved), which implies that $\min(d_1, d_2) \leq d'_2 \leq \max(d_1, d_2)$. Hence we have $\max(d_1, d_2) \leq \max(d'_1, d'_2)$. □

By combining Lemma 2 and the discussion at the start of Section 2, we obtain the following.

LEMMA 3. *The matrix $\mathbf{U}$ returned at the end of the execution of* `RecursiveReduce` *is exactly the unimodular matrix that is the product of all unimodular matrices that have been successively applied to the GSO coefficients.*

PROOF. The only source of discrepancy between the output matrix $\mathbf{U}$ and the product of all unimodular matrices that have been successively applied to the GSO coefficients is the update of $\mathbf{U}$ modulo $2^{\overline{\beta}}$ at Steps 7 and 9. Let $\mathbf{B}$ and $\mathbf{C}$ respectively denote the bases corresponding to the input GSO coefficients, and to the final value of the GSO coefficients. We want to show that $\mathbf{U} = \mathbf{C}^{-1}\mathbf{B}$. For this, it suffices to show that each coefficient of $\mathbf{C}^{-1}\mathbf{B}$ has magnitude $< 2^{\overline{\beta}-1}$ so that reductions modulo $2^{\overline{\beta}}$ in the process of computing $\mathbf{U}$ have no impact on the output.

By Lemma 2 and the fact that $\mathbf{C}$ is size-reduced, we have that $\|\mathbf{c}_i\| \leq \sqrt{n}\max_j \|\mathbf{b}^*_j\| \leq \sqrt{n}\max_j \|\mathbf{b}_j\|$ for all $i \leq n$. By (2), this implies that each coefficient of $\mathbf{C}^{-1}\mathbf{B}$ has magnitude $\leq \sqrt{n}(\max_j \|\mathbf{b}_j\|)^n$. This is $< 2^{\overline{\beta}-1}$, by design of $\overline{\beta}$. □

The latter result implies that `Reduce` indeed returns a basis of the input lattice. Together with Section 2, it also implies that all rationals and integers that occur during the execution of the algorithm have bit-sizes $O(n\beta)$. The cost of `Reduce` can hence be bounded by the number of arithmetic operations multiplied by $O(\mathcal{M}(n\beta))$.

4. CORRECTNESS OF `RECURSIVEREDUCE`

This section is devoted to proving Theorem 1 for `RecursiveReduce`. The main proof component is the analysis of the convergence towards reduced bases of the repeated backwards sweeps of recursive calls.

4.1 Convergence of `RecursiveReduce`

Suppose we are at a certain recursion level r. Algorithm `RecursiveReduce` reduces the n_r-dimensional basis by making recursive calls in dimension n_{r-1}. These are grouped in tours, and each tour consists in recursive calls on GSO blocks with starting indices $1 + kn_{r-1}/2$ for k from $2(n_r/n_{r-1} - 1)$ down to 0.

We define α_r as the maximum, over all input n_r-dimensional GSO coefficients $(\mathbf{d}, \mathbf{M})$, of the quantity

$$\Big(\prod_{i=1}^{n_r/2} d'_i\Big)^2 / (\prod_{i=1}^{n_r} d'_i), \tag{5}$$

where $(\mathbf{d}', \mathbf{M}')$ are the GSO coefficients at the end of the execution of `RecursiveReduce`. This quantity measures how small the first half of the GSO coefficients are, relative to the second half. As seen in Section 2, we have $\alpha_1 \leq 4/3$.

The following lemma, inspired from [10, Sec. 3.3], quantifies the progress made during a tour.

LEMMA 4. *Let $\mathbf{d}$ and $\mathbf{d}'$ respectively denote the values of the GSO coefficients at the start and the end of an iteration of the loop starting at Step 3 of* `RecursiveReduce`*. We define (for $1 \leq k \leq \ell$)*

$$g_k := \ln \prod_{i\leq kn_{r-1}/2} d_i, \quad g'_k := \ln \prod_{i\leq kn_{r-1}/2} d'_i,$$

with $\ell = 2n_r/n_{r-1}$. We further define:

$$\begin{aligned} \mu &:= \max_{k<\ell} \frac{1}{\ell - k}\left(\frac{g_k}{k} - \frac{g_\ell}{\ell}\right), \\ \mu' &:= \max_{k<\ell} \frac{1}{\ell - k}\left(\frac{g'_k}{k} - \frac{g'_\ell}{\ell}\right), \\ \mu^* &:= \frac{\ln \alpha_{r-1}}{2}. \end{aligned}$$

Then we have

$$\mu' \leq \mu^* \quad or \quad \mu' \leq \mu^* + (1 - \frac{1}{\ell^2})(\mu - \mu^*).$$

This results states that across tours, the sequence of successive μ's lies below an affine dynamical system. Convergence is geometric: every ℓ^2 tours, the quantity $\mu - \mu^*$ either becomes negative, or it decreases by a constant factor. Note that the result is qualitatively similar to those of [3, 10] for dimension $\ell := 2n_r/n_{r-1}$ and "block size" 2.

PROOF. By dividing each d_i by $(\prod_{j\leq n} d_j)^{1/n}$, we see that we may assume without loss of generality that $g_\ell = g'_\ell = 0$ (by Lemma 1). Further, we define $g_0 = g'_0 = 0$.

Because of the backward ordering of the recursive calls during a tour, and thanks to (5), we have, for $k = \ell - 1$ down to $k = 1$:

$$2(g'_k - g_{k-1}) - (g'_{k+1} - g_{k-1}) \leq \ln \alpha_{r-1}.$$

This may be rewritten as

$$g'_k \leq \frac{1}{2}(g'_{k+1} + g_{k-1}) + \mu^*. \tag{6}$$

Assume first that $\mu \leq \mu^*$. We prove by induction on $k = \ell$ down to $k = 1$ that $g'_k \leq (\ell - k)k\mu^*$, which implies that $\mu' \leq \mu^*$. The case $k = \ell$ follows by preservation of the determinant. We now assume that $k < \ell$. Then, by induction and (6), we have

$$\begin{aligned} g'_k &\leq \frac{1}{2}((\ell - k - 1)(k + 1) + (\ell - k + 1)(k - 1))\mu^* + \mu^* \\ &= (\ell - k)k\mu^*. \end{aligned}$$

We now assume that $\mu > \mu^*$. We prove by induction on $k = \ell$ down to $k = 1$ that

$$g'_k \leq (\ell - k)k\Big(\mu^* + (1 - \frac{1}{\ell^2})(\mu - \mu^*)\Big).$$

The case $k = \ell$ follows by preservation of the determinant. We now assume that $k < \ell$. Let $q := (\ell - (k - 1))(k - 1)/2$. Then $(\ell - (k+1))(k+1) = 2((\ell - k)k - 1 - q)$ and $(\ell - (k - 1))(k - 1) = 2q$. Hence, by induction and (6), we have:

$$\begin{aligned} g'_k &\leq ((\ell - k)k - 1 - q)\Big(\mu^* + (1 - \frac{1}{\ell^2})(\mu - \mu^*)\Big) + q\mu + \mu^* \\ &= (q + (1 - \frac{1}{\ell^2})((\ell - k)k - 1 - q))(\mu - \mu^*) + (\ell - k)k\mu^*. \end{aligned}$$

Now, observe that $q \leq \ell^2 - 1$, hence $q/(1+q) \leq (\ell^2 - 1)/\ell^2 = 1 - 1/\ell^2$. This gives (using the positivity of $\mu - \mu^*$):

$$g'_k \leq (1 - \frac{1}{\ell^2})(\ell - k)k(\mu - \mu^*) + (\ell - k)k\mu^*,$$

which completes the proof. □

We now use Lemma 4 to show that at the end of the $\ell^2\lceil \ln(8\overline{\beta}/\eta)\rceil$ tours, the basis is quite close to being reduced.

LEMMA 5. *At the end of the execution of* `RecursiveReduce`*, we have:*

$$\max_{k<\ell} \frac{1}{\ell - k}\left(\frac{g_k}{k} - \frac{g_\ell}{\ell}\right) \leq \frac{\ln \alpha_{r-1}}{2} + \frac{\eta}{8}.$$

PROOF. Let μ and μ' respectively denote the values of $\max_{k<\ell} \frac{1}{\ell-k}(\frac{g_k}{k} - \frac{g_\ell}{\ell})$ at the start and at the end of the execution of `RecursiveReduce`. By Lemma 4, we have that $\mu' \leq (\ln \alpha_{r-1})/2$ or:

$$\begin{aligned} \mu' - \frac{\ln \alpha_{r-1}}{2} &\leq (1 - \frac{1}{\ell^2})^{\ell^2\lceil \ln(8\overline{\beta}/\eta)\rceil}(\mu - \frac{\ln \alpha_{r-1}}{2}) \\ &\leq \exp(-\lceil \ln(8\overline{\beta}/\eta)\rceil)\mu \\ &\leq \frac{\eta}{8}\frac{\mu}{\overline{\beta}}. \end{aligned}$$

Thanks to Lemma 2, we know that $\mu \leq \overline{\beta}$ and hence we obtain that $\mu' \leq \frac{\ln \alpha_{r-1}}{2} + \frac{\eta}{8}$. □

4.2 Proof of Theorem 1 for `RecursiveReduce`

Thanks to Lemma 3, it only remains to prove the bound on $\|\mathbf{c}_1\|$ of the output basis $\mathbf{C}$ of `RecursiveReduce`. For this, we use the results from the previous subsection on the convergence of `RecursiveReduce`.

LEMMA 6. *For all $r \geq 1$, we have $\alpha_r \leq ((4/3)e^{2\eta})^{n_r^2/4}$.*

PROOF. By Lemma 5 with $k = \ell/2$, we have (using the same notations as above):

$$\frac{2}{\ell^2}(2g_{\ell/2} - g_\ell) \leq \mu' \leq \frac{\ln \alpha_{r-1}}{2} + \frac{\eta}{8}.$$

We obtain (using the fact that $\ln \alpha_r$ is the largest possible value for $2g_{\ell/2} - g_\ell$)

$$\ln \alpha_r \leq \frac{\ell^2}{2}\Big(\frac{\ln \alpha_{r-1}}{2} + \frac{\eta}{8}\Big) = \frac{n_r^2}{n_{r-1}^2}\Big(\ln \alpha_{r-1} + \frac{\eta}{4}\Big).$$

We can then derive

$$\begin{aligned} \frac{1}{n_r^2}\ln \alpha_r &\leq \frac{1}{n_{r-1}^2}\Big(\ln \alpha_{r-1} + \frac{\eta}{4}\Big) \\ &\leq \frac{1}{n_{r-2}^2}\Big(\ln \alpha_{r-2} + \frac{\eta}{4}\Big) + \frac{1}{n_{r-1}^2}\frac{\eta}{4} \\ &\leq \cdots \\ &\leq \frac{1}{4}\ln \alpha_1 + \frac{\eta}{4}\sum_{i\leq r}\frac{1}{n_i^2}. \end{aligned}$$

The result follows by bounding α_1 by $4/3$ and $\sum_{i\leq r} 1/n_i^2$ by $\pi^2/6$. □

By Lemma 5 (with $k = 2$) and Lemma 6, we have, letting $\mathbf{d}'$ denote the values of the GSO coefficients at the end of the execution of `RecursiveReduce`:

$$\begin{aligned} \prod_{i=1}^{n_{r-1}} d'_i &\leq \Big((4/3)e^{2\eta}\Big)^{\frac{n_{r-1}^2}{4}(\frac{2n_r}{n_{r-1}} - 2)}(e^{\frac{\eta}{4}})^{\frac{2n_r}{n_{r-1}} - 2}\Big(\prod_{i=1}^{n_r} d'_i\Big)^{\frac{n_{r-1}}{n_r}} \\ &\leq \Big((4/3)e^{4\eta}\Big)^{\frac{n_{r-1}}{2}(n_r - n_{r-1})}\Big(\prod_{i=1}^{n_r} d'_i\Big)^{\frac{n_{r-1}}{n_r}}. \end{aligned}$$

By raising to the power or $1/n_{r-1}$, we obtain:

$$\Big(\prod_{i=1}^{n_{r-1}} d'_i\Big)^{\frac{1}{n_{r-1}}} \leq \Big((4/3)^{1/2}e^{2\eta}\Big)^{n_r - n_{r-1}}\Big(\prod_{i=1}^{n_r} d'_i\Big)^{\frac{1}{n_r}}.$$

By applying the latter to all values of r, we obtain:

$$\Big(d'_1 d'_2\Big)^{\frac{1}{2}} \leq \Big((4/3)^{1/2}e^{2\eta}\Big)^{n_r - 2}\Big(\prod_{i=1}^{n_r} d'_i\Big)^{\frac{1}{n_r}}.$$

Now, as the first two output vectors are reduced (by correctness of Schönhage's algorithm), we obtain:

$$d'_1 \leq \Big((4/3)^{1/2}\mathrm{e}^{2\eta}\Big)^{n_r-1}\Big(\prod_{i=1}^{n_r} d'_i\Big)^{\frac{1}{n_r}}. \quad (7)$$

To complete the proof of Theorem 1 for `RecursiveReduce`, note that for $\eta = c/(3n_r)$, we have $\mathrm{e}^{\eta(n_r-1)} \leq 1+c$.

5. CORRECTNESS OF REDUCE

We now extend the correctness of `RecursiveReduce` to that of `Reduce`. This is direct when $n = n_r$, as in that case `Reduce` makes a single call to `RecursiveReduce`. In the rest of this section, we assume that $n > n_r$. Then `Reduce` calls `RecursiveReduce` many times, in a BKZ fashion. The convergence analysis is adapted from [10].

The convergence analysis does not need to consider the inner workings of `RecursiveReduce`: it only relies on the fact that if $(\mathbf{d}', \mathbf{M}')$ are the GSO coefficients at the end of the execution of `RecursiveReduce`, then we have, by (7):

$$(d'_1)^{n_r}/(\prod_{i=1}^{n_r} d'_i) \leq \overline{\alpha}_1^{(n_r-1)n_r}, \quad (8)$$

with $\overline{\alpha}_1 = (4/3)^{1/2}\mathrm{e}^{2\eta}$.

LEMMA 7. *Let $\mathbf{d}$ and $\mathbf{d}'$ respectively denote the values of the GSO coefficients at the start and the end of an iteration of the loop starting at Step 6 of* `Reduce`*. We define (for $1 \leq k \leq n$)*

$$\overline{g}_k := \ln \prod_{i\leq k} d_i, \qquad \overline{g}'_k := \ln \prod_{i\leq k} d'_i.$$

We further define:

$$\begin{aligned} \overline{\mu} &:= \max_{k\leq n-n_r+1} \frac{1}{n-k}\left(\frac{\overline{g}_k}{k} - \frac{\overline{g}_n}{n}\right), \\ \overline{\mu}' &:= \max_{k\leq n-n_r+1} \frac{1}{n-k}\left(\frac{\overline{g}'_k}{k} - \frac{\overline{g}'_n}{n}\right), \\ \overline{\mu}^* &:= \ln \overline{\alpha}_1. \end{aligned}$$

Then we have

$$\overline{\mu}' \leq \overline{\mu}^* \quad or \quad \overline{\mu}' \leq \overline{\mu}^* + (1-a)(\overline{\mu} - \overline{\mu}^*),$$

with $a = \frac{n_r-1}{n-1}$ if $n_r \geq n/2$ and $a = \frac{n_r^2-n_r}{n^2/4+n_r^2-n_r}$ otherwise. Note that in both cases we have $a \geq n_r^2/n^2$.

PROOF. Without loss of generality, we assume that that $\overline{g}_n = \overline{g}'_n = 0$. Further, we define $\overline{g}_0 = \overline{g}'_0 = 0$.

Because of the forward ordering of the calls to `RecursiveReduce` during a tour, and thanks to (8), we have, for $k = 1$ to $k = n - n_r + 1$:

$$\overline{g}'_k - \overline{g}'_{k-1} \leq (n_r-1)\overline{\mu}^* + \frac{1}{n_r}(\overline{g}_{k+n_r-1} - \overline{g}'_{k-1}).$$

This may be rewritten as

$$\overline{g}'_k \leq \frac{1}{n_r}\overline{g}_{k+n_r-1} + (1-\frac{1}{n_r})\overline{g}'_{k-1} + (n_r-1)\overline{\mu}^*. \quad (9)$$

Assume first that $\overline{\mu} \leq \overline{\mu}^*$. We prove by induction on k that $\overline{g}'_k \leq (n-k)k\overline{\mu}^*$, which implies that $\overline{\mu}' \leq \overline{\mu}^*$. The case $k = 0$ follows from $\overline{g}'_0 = 0$. We now assume that $k > 0$. Then, by induction and (9), we have (grouping the multiples of $1/n_r$ together):

$$\begin{aligned} \overline{g}'_k &\leq \big((n-k+1)(k-1) + (n-n_r-2k+2) + (n_r-1)\big)\overline{\mu}^* \\ &= (n-k)k\overline{\mu}^*. \end{aligned}$$

We now assume that $\overline{\mu} > \overline{\mu}^*$. We prove by induction on k that

$$\overline{g}'_k \leq (n-k)k\Big(\overline{\mu}^* + (1-a)(\overline{\mu} - \overline{\mu}^*)\Big).$$

The case $k = 0$ follows from $\overline{g}'_0 = 0$. We now assume that $k > 0$. By induction and (9), we have:

$$\begin{aligned} \overline{g}'_k &\leq (n_r-1)\overline{\mu}^* + \frac{1}{n_r}(n-k-n_r+1)(k+n_r-1)\overline{\mu} \\ &\quad + (1-\frac{1}{n_r})(n-k+1)(k-1)\Big(\overline{\mu}^* + (1-a)(\overline{\mu} - \overline{\mu}^*)\Big) \\ &= \frac{X}{n_r}(\overline{\mu} - \overline{\mu}^*) + (n-k)k\overline{\mu}^*, \end{aligned}$$

with

$$X = (n-k-n_r+1)(k+n_r-1) + (n_r-1)(n-k+1)(k-1)(1-a).$$

Since $\overline{\mu} - \overline{\mu}^* > 0$, it suffices to prove that for all k, we have $X \leq n_r(n-k)k(1-a)$. By subtracting the latter from X and differentiating with respect to k, we obtain that the inequality is the strongest for $k = 1$ when $n_r \geq n/2$ and for $k = n/2 - n_r + 1$ otherwise. The fact that $a \leq (n_r-1)/(n-1)$ allows to handle the first case, whereas $a \leq (n_r^2-n_r)/(n^2/4+n_r^2-n_r)$ allows to handle the second one. This completes the proof. □

Lemma 7 may be used to show that after $O(n^2/n_r^2)$ tours, the basis is quite close to being reduced.

LEMMA 8. *At the end of the execution of* `Reduce`*, we have:*

$$\max_{k\leq n-n_r+1} \frac{1}{n-k}\left(\frac{\overline{g}_k}{k} - \frac{\overline{g}_n}{n}\right) \leq \ln \overline{\alpha}_1 + \eta.$$

PROOF. Let $\overline{\mu}$ and $\overline{\mu}'$ respectively denote the values of $\max_{k\leq n-n_r+1} \frac{1}{n-k}(\frac{\overline{g}_k}{k} - \frac{\overline{g}_n}{n})$ at the start and at the end of the execution of `Reduce`. By Lemma 7, we have that $\overline{\mu}' \leq \ln \overline{\alpha}_1$ or:

$$\overline{\mu}' - \ln \overline{\alpha}_1 \leq (1 - \frac{n_r^2}{n^2})^{(n/n_r)^2 \lceil \ln(\overline{\beta}/\eta)\rceil}(\overline{\mu} - \ln \overline{\alpha}_1).$$

Once may proceed exactly as in the proof of Lemma 5 to show that the latter is $\leq \ln \overline{\alpha}_1 + \eta$. □

To prove Theorem 1, we instantiate Lemma 8 with $k = 1$. This gives, using the definition of $\overline{\alpha}_1$:

$$d'_1 \leq \Big((4/3)^{1/2}\mathrm{e}^{3\eta}\Big)^{n-1}\Big(\prod_{i=1}^{n} d'_i\Big)^{1/n},$$

where $\mathbf{d}'$ denotes the values of the GSO coefficients at the end of the execution of `Reduce`. To complete the proof of Theorem 1, note that for $\eta = c/(3n)$, we have $\mathrm{e}^{3\eta(n-1)/2} \leq 1+c$.

6. COST ANALYSIS

This section is devoted to proving Theorem 2. We first bound the bit-sizes of all integers and rationals occurring during the executions of `Reduce` and `RecursiveReduce`.

By Lemma 2, we know that the quantity $\max_i d_i$ cannot increase. Therefore, the integer $2^{\overline{\beta}}$ is always larger than the bound (3) for any basis during the execution of the algorithm, except possibly during Steps 3, 10 and 11 of `Reduce` and Steps 8 and 9 of `RecursiveReduce`. During these steps, the bound (4) is $\leq 2^{n+\overline{\beta}}$. This implies that all GSO coefficients appearing during the execution of `Reduce` and those of `RecursiveReduce` have bit-sizes $O(n+\overline{\beta})$, which is itself $O(n\beta)$. Finally, by construction, all (integer) entries of unimodular transformations are manipulated modulo $2^{\overline{\beta}}$.

Overall, each arithmetic operation occurring during the execution involves $O(\mathcal{M}(n\beta))$ bit operations.

The following lemma provides a recursive cost bound. The result follows from [16] for $r = 1$. It follows from Subsection 4.2 and inspection of algorithm `RecursiveReduce` for $r > 1$.

LEMMA 9. *We consider the execution of* `RecursiveReduce` *given as inputs $\overline{\beta}, \eta$ and any n_r-dimensional GSO coefficients $(\mathbf{d}, \mathbf{M})$ of a basis $\mathbf{B}$ with $n_r \log \|\mathbf{B}\| \leq \overline{\beta}$. Let C_r denote the maximum number of bit operations performed during the execution, over all possible $(\mathbf{d}, \mathbf{M})$. Then, we have, for some absolute constant $K > 0$,*

$$C_1 \leq K\mathcal{M}(\overline{\beta})\log(\overline{\beta}),$$

and, for $r > 1$,

$$C_r \leq \frac{n_r^3}{n_{r-1}^3} L\left(C_{r-1} + n_{r-1}n_r^2\mathcal{M}(\overline{\beta})\right),$$

with $L = K\ln(8\overline{\beta}/\eta)$.

Now, we define $T_r := C_r/(n_r^3 L\mathcal{M}(\overline{\beta}))$. From Lemma 9, we have $T_1 \leq 1$ and, for $r > 1$,

$$T_r \leq LT_{r-1} + \frac{n_r^2}{n_{r-1}^2}.$$

By design of the sequence of specific dimensions $(n_k)_{k\geq 1}$, we have:

$$T_r \leq \exp\left(O(\sqrt{(\log n)(\log L)})\right).$$

Therefore:

$$C_r \leq \exp\left(O(\sqrt{(\log n)(\log\log(n\beta))})\right)\log(n\beta)n_r^3\mathcal{M}(n\beta).$$

During the execution of `Reduce`, algorithm `RecursiveReduce` is called $O((n^2/n_r^2)\log(n\beta))$ times. The cost of `Reduce` is hence bounded by:

$$O\left(\frac{n^2}{n_r^2}\log(n\beta)\Big(C_r + n^3\mathcal{M}(n\beta)\Big)\right).$$

By choice of r, we have

$$\frac{n}{n_r} \leq \exp\left(O(\sqrt{(\log n)(\log\log(n\beta))})\right).$$

This allows us to complete the proof of Theorem 2.

7. CONCLUSION

We have presented and analyzed the asymptotically fastest LLL-type reduction algorithm known so far. The dominating component of the cost stems from the leaves of the recursion, which correspond to 2-dimensional lattices. As a consequence, using fast matrix multiplication and Storjohann's weak size-reduction (see [18, Lemma 13]) in the naive way only affects lower order terms, and hence does not allow us to lower the complexity upper bound.

We did not implement the algorithm as – for dimensions of practical interest – it is very unlikely to be competitive with the L^2 algorithm from [11]. Indeed, the (arithmetic) term $\mathcal{M}(n\beta)$ of our complexity bound will asymptotically be reached from above, whereas the complexity bound $O(n^3\mathcal{M}(n)(n+\beta)\beta)$ of L^2 is experimentally pessimistic (see [17]).

Acknowledgments

The authors thank Paul Kirchner and Gilles Villard for interesting discussions. The second author was supported by the ERC Starting Grant ERC-2013-StG-335086-LATTAC.

8. REFERENCES

[1] N. Gama, N. Howgrave-Graham, H. Koy, and P. Q. Nguyen. Rankin's Constant and Blockwise Lattice Reduction. In *Proc. of CRYPTO*, volume 4117 of *LNCS*, pages 112–130. Springer, 2006.

[2] Saruchi, I. Morel, D. Stehlé, and G. Villard. LLL reducing with the most significant bits. In *Proc. of ISSAC*, pages 367–374. ACM, 2014.

[3] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Proc. of CRYPTO*, volume 6841 of *LNCS*, pages 447–464. Springer, 2011.

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

[5] H. Koy and C.-P. Schnorr. Segment LLL-reduction of lattice bases. In *Proc. of CALC*, volume 2146 of *LNCS*, pages 67–80. Springer, 2001.

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

[7] L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*. SIAM, 1986. CBMS-NSF Regional Conference Series in Applied Mathematics.

[8] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on Voronoi cell computations. In *Proc. of STOC*, pages 351–358. ACM, 2010.

[9] D. Micciancio and M. Walter. Practical, predictable lattice basis reduction. In *Proc. of EUROCRYPT*, volume 9665 of *LNCS*, page 1123. Springer, 2016.

[10] A. Neumaier. Bounding basis reduction properties. *IACR Cryptology ePrint Archive*, 2016:4, 2016.

[11] P. Q. Nguyen and D. Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput*, 39(3):874–903, 2009.

[12] P. Q. Nguyen and B. Vallée. *The LLL Algorithm: Survey and Applications*. Information Security and Cryptography. Springer, 2009.

[13] A. Novocin, D. Stehlé, and G. Villard. An LLL-reduction algorithm with quasi-linear time complexity. In *Proc. of STOC*, pages 403–412. ACM, 2011.

[14] C.-P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving

subset sum problems. *Mathematics of Programming*, 66:181–199, 1994.

[15] A. Schönhage. Factorization of univariate integer polynomials by Diophantine approximation and improved basis reduction algorithm. In *Proc. of ICALP*, volume 172 of *LNCS*, pages 436–447. Springer, 1984.

[16] A. Schönhage. Fast reduction and composition of binary quadratic forms. In *Proc. of ISSAC*, pages 128–133. ACM, 1991.

[17] D. Stehlé. *Floating-Point LLL: Theoretical and Practical Aspects*. 2010. Chapter of [12].

[18] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical report, ETH Zürich, 1996.

[19] C. K. Yap. Fast unimodular reduction: planar integer lattices. In *Proc. of FOCS*, pages 437–446. IEEE Computer Society Press, 1992.