Gradual sub-lattice reduction and a new complexity for factoring polynomials

Mark van Hoeij, Andrew Novocin

Gradual sub-lattice reduction and a new complexity for factoring polynomials

Mark van Hoeij[1⋆] and Andrew Novocin[2]

[1] Florida State University, 208 Love Building Tallahassee, FL 32306-4510
hoeij@math.fsu.edu – http://www.math.fsu.edu/~hoeij
[2] LIP/INRIA/ENS, 46 allée d'Italie, F-69364 Lyon Cedex 07, France
Andrew.Novocin@ens-lyon.fr – http://andy.novocin.com/pro

Abstract. We present a lattice algorithm specifically designed for some classical applications of lattice reduction. The applications are for lattice bases with a generalized knapsack-type structure, where the target vectors are boundably short. For such applications, the complexity of the algorithm improves traditional lattice reduction by replacing some dependence on the bit-length of the input vectors by some dependence on the bound for the output vectors. If the bit-length of the target vectors is unrelated to the bit-length of the input, then our algorithm is only linear in the bit-length of the input entries, which is an improvement over the quadratic complexity floating-point LLL algorithms. To illustrate the usefulness of this algorithm we show that a direct application to factoring univariate polynomials over the integers leads to the first complexity bound improvement since 1984. A second application is algebraic number reconstruction, where a new complexity bound is obtained as well.

1 Introduction

Lattice reduction algorithms are essential tools in computational number theory and cryptography. A lattice is a discrete subset of $\mathbb{R}^n$ that is also a $\mathbb{Z}$-module. The goal of lattice reduction is to find a 'nice' basis for a lattice, one which is near orthogonal and composed of short vectors. Since the publication of the 1982 Lenstra, Lenstra, Lovász [15] lattice reduction algorithm many applications have been discovered, such as polynomial factorization [15,11] and attacking several important public-key cryptosystems including knapsack cryptosystems [23], RSA under certain settings [7], and DSA and some signature schemes in particular settings [12]. One of the important features of the LLL algorithm was that it could approximate the shortest vector of a lattice in polynomial time. This is valuable because finding the exact shortest vector in a lattice is provably NP-hard [1,18]. Given a basis $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{R}^n$ which satisfies $\| \mathbf{b}_i \| \leq X \quad \forall i$, the LLL algorithm has a running time of $\mathcal{O}(d^5 n \log^3 X)$ using classical arithmetic. Recently there has been a resurgence of lattice reduction work thanks to Nguyen and Stehlé's L^2 algorithm [20,21] which performs lattice reduction in $\mathcal{O}(d^4 n \log X[d + \log X])$ CPU operations. The primary result of L^2 was that the dependence on $\log X$ is only quadratic allowing for improvement on applications using large input vectors.

The main result: Many applications of LLL (see the applications section below) involve finding a vector in a lattice whose norm is known to be small in advance. In such cases it can be more efficient to reduce a basis of a sub-lattice which contains all targeted vectors than reducing a basis of the entire lattice. In this paper we target short vectors in specific types of input lattice bases which we call knapsack-type bases. The new algorithm introduces a search parameter B which the user provides. This parameter is used to bound the norms of targeted short vectors. To be precise:

The *rows* of the following matrices represent a knapsack-type basis

⋆ Supported by NSF 0728853

$$\left(\begin{array}{ccc|ccc} 0 & \cdots & 0 & 0 & \cdots & P_N \\ 0 & \cdots & 0 & 0 & \ddots & 0 \\ 0 & \cdots & 0 & P_1 & \cdots & 0 \\ \hline 1 & \cdots & 0 & x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & x_{r,1} & \cdots & x_{r,N} \end{array}\right) \quad \text{or} \quad \left(\begin{array}{ccc|ccc} 1 & \cdots & 0 & x_{1,1} & \cdots & x_{1,N} \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & x_{r,1} & \cdots & x_{r,N} \end{array}\right).$$

The specifications of our algorithm are as follows. It takes as input a knapsack-type basis $\mathbf{b}_1, \ldots, \mathbf{b}_d \in \mathbb{Z}^n$ of a lattice L with $\| \mathbf{b}_i \| \leq X \ \forall i$ and a search parameter B; it returns a *reduced basis generating a* sub-lattice $L' \subseteq L$ such that if $\mathbf{v} \in L$ and $\| \mathbf{v} \| \leq B$ then $\mathbf{v} \in L'$.
Our algorithm has the following complexity bounds for various input:

No P_i	$\mathcal{O}(d^2(n+d^2)(d+\log B)[\log X + n(d+\log B)])$
No restriction on P_i	$\mathcal{O}(d^4(d+\log B)[\log X + d(d+\log B)])$
Many P_i large w.r.t. B	$\mathcal{O}(dr^3(r+\log B)[\log X + d(r+\log B)])$

These complexity bounds have several distinct parameters, so a comparison with other algorithms is a bit subtle. The most significant parameter to explore is B, the search parameter. If one selects $B = X$ then our algorithm will return a reduced basis of $L' = L$ in $\mathcal{O}(d^2 n(n+d^2)[d^2 + \log^2 X])$. This is an interesting result because our algorithm, like the original LLL and the L^2 algorithms, uses switches and size-reductions of the vectors to arrive at a reduced basis. The fact that we return a reduced basis with a complexity so similar to L^2 implies that there are alternative orderings on the switches which lead to similar performance.
When using a smaller value of B than X the algorithm will return either:

- A reduced basis of a sub-lattice L' which contains all vectors of norm $\leq B$. This sub-lattice may be different than the sub-lattice, L'', generated by all vectors of norm $\leq B$, and we do have $L'' \subseteq L' \subseteq L$. Also, because the basis of L' is reduced, we have an approximation of the shortest non-zero vector of L.
- The empty set, in which case the algorithm has proved that no non-zero vector of norm $\leq B$ exists in L.

We offer the following complexity comparison with L^2 [20] for some values of B on square input lattices (with P_j's). When a column has a non-zero P_j we can reduce the $x_{i,j}$ modulo P_j. Thus, without loss of generality, we may assume that P_j is the largest element in its column. Note that $r = d - N$.

L^2	$\mathcal{O}(d^6 \log X + d^5 \log^2 X)$
$B = \mathcal{O}(X)$	$\mathcal{O}(d^7 + d^5 \log^2 X)$
$B = \mathcal{O}(X^{1/d})$	$\mathcal{O}(d^2 r^5 + r^3 \log^2 X)$
$B = 2^{\mathcal{O}(d)}$	$\mathcal{O}(d^4 r^3 + d^2 r^3 \log X)$

It should be noted that [20] explores running times of L^2 on knapsack lattices with $N = 1$ (such lattice bases are used in [9]). In this case, L^2 will have complexity $\mathcal{O}(d^5 \log X + d^4 \log^2 X)$.
Our approach: We reduce the basis gradually, using many separate calls to another lattice reduction algorithm. To get the above complexity results we chose H-LLL [19] but there are many

suitable lattice reduction algorithms we could use instead such as [13,15,20,24]. For more details on why we made this decision see the discussion in section 5.

There are three important features to our approach. First, we approach the problem column by column. Beginning with the $r \times r$ identity and with each iteration of the algorithm we expand our scope to include one more column of the $x_{i,j}$. Next, within each column iteration, we reduce the new entries bit by bit, starting with a reduction using only the most significant bits, then gradual including more and more bits of data. Third, we allow for the removal of vectors which have become too large. This allows us to always work on small entries, but restricts us to a sub-lattice.

The proof of the algorithm's complexity is essentially a study of two quantities, the product of the Gram-Schmidt lengths of the current vectors which we call the active determinant and an energy function which we call progress. We amortize all of the lattice reduction costs using progress, and we bound the number of iterations and number of vectors using the active determinant. Neither of these quantities is impacted by the choice of lattice reduction algorithm.

Applications of the algorithm: As evidence for the usefulness of this new approach we show two new complexity results based on applications of the main algorithm. The first result is a new complexity for the classical problem of factoring polynomials in $\mathbb{Z}[x]$. If the polynomial has degree N, coefficients smaller than $\log(A)$, and when reduced modulo a prime p has r irreducible factors then we prove a complexity of $\mathcal{O}(N^3r^4 + N^2r^4 \log A)$ for the lattice reduction costs using classical arithmetic. One must also add the cost of multi-factor Hensel lifting which is $\mathcal{O}(N^6 + N^4 \log^2 A)$ ignoring the small terms $\log(r)$ and $\log^2 p$ (see [8] for details). This is the first improvement over the Schönhage bound given in 1984 [25] of $\mathcal{O}(N^8 + N^5 \log^3 A)$.

The second new complexity result comes in the problem of reconstructing a minimal polynomial from a complex approximation of the algebraic number. In this application we know $\mathcal{O}(d^2 + d \log H)$ bits of an approximation of some complex root of an unknown polynomial $h(x)$ with degree d and with maximal coefficient of absolute value $\leq H$. Then our algorithm can be used to find the coefficients of $h(x)$ in $\mathcal{O}(d^7 + d^5 \log^2 H)$ CPU operations.

Other problems of common interest which might be impacted by our algorithm include integer relation finding (where $N = 1$) and simultaneous Diophantine approximation of several real numbers [10,6] (where $r = 1$).

Notations: All costs are given for the bit-complexity model. A standard row vector will be denoted $\mathbf{v}$, $\mathbf{v}[i]$ represents the i^{th} entry of $\mathbf{v}$, $\mathbf{v}[i, \ldots, j]$ a vector consisting of all entries of $\mathbf{v}$ from the i^{th} entry to the j^{th} entry, and $\mathbf{v}[-1]$ the final entry of $\mathbf{v}$. Also we will use $\|\mathbf{w}\|_\infty$ as the max-norm or the largest absolute value of an entry in the vector $\mathbf{w}$, $\| \mathbf{w} \| := \sqrt{\sum(\mathbf{w}[i])^2}$ which we call the norm of $\mathbf{w}$, and $\mathbf{w}^T$ as the transpose of $\mathbf{w}$. The scalar product will be denoted $\mathbf{v} \cdot \mathbf{w} := \sum \mathbf{v}[i] \cdot \mathbf{w}[i]$. For a matrix M we will use $M[1, \ldots, k]$ to denote the first k columns of M. The n by n identity matrix will be denoted $I_{n \times n}$. For a real number x we use $\lceil x \rceil$ and $\lfloor x \rfloor$ to denote the closest integer $\geq x$ and $\leq x$ respectively.

Road map: In section 2 we give a brief introduction to lattice reduction algorithms. In section 3 we present the central algorithm of the paper and prove its correctness. In section 4 we prove several important features by studying quasi-invariants we call the active determinant and progress. In this section we treat lattice reduction as a black-box algorithm. In section 5 we prove the overall complexity and other important claims about the new algorithm by fixing a choice for a standard lattice reduction algorithm. In section 6 we offer new complexity results for factoring polynomials in $\mathbb{Z}[x]$ and algebraic number reconstruction.

2 Background on lattice reduction

The purpose of this section is to present some facts from [15] that will be needed throughout the paper. For a more general treatment of lattice reduction see [17].

A lattice, L, is a discrete subset of $\mathbb{R}^n$ that is also a $\mathbb{Z}$-module. Let $\mathbf{b}_1, \ldots, \mathbf{b}_d \in L$ be a basis of L and denote $\mathbf{b}_1^*, \ldots, \mathbf{b}_d^* \in \mathbb{R}^n$ as the Gram-Schmidt orthogonalization over $\mathbb{R}$ of $\mathbf{b}_1, \ldots, \mathbf{b}_d$. Let $\delta \in (1/4, 1]$ and $\eta \in [1/2, \sqrt{\delta})$. Let $l_i = \log_{1/\delta} \| \mathbf{b}_i^* \|^2$, and denote $\mu_{i,j} = \frac{\mathbf{b}_i \cdot \mathbf{b}_j^*}{\mathbf{b}_j^* \cdot \mathbf{b}_j^*}$. Note that $\mathbf{b}_i, \mathbf{b}_i^*, l_i, \mu_{i,j}$ will change throughout the algorithm sketched below.

Definition 1. $\mathbf{b}_1, \ldots, \mathbf{b}_d$ *is* LLL-reduced *if* $\| \mathbf{b}_i^* \|^2 \leq \frac{1}{\delta - \mu_{i+1,i}^2} \| \mathbf{b}_{i+1}^* \|^2$ *for* $1 \leq i < d$ *and* $|\mu_{i,j}| \leq \eta$ *for* $1 \leq j < i \leq d$.

In the original paper the values for (δ, η) were chosen as $(3/4, 1/2)$ so that $\frac{1}{\delta - \eta^2}$ would simply be 2.

Algorithm 1 (Rough sketch of LLL-type algorithms)
Input: *A basis* $\mathbf{b}_1, \ldots, \mathbf{b}_d$ *of a lattice* L.
Output: *An LLL-reduced basis of* L.

A - $\kappa := 2$
B - ***while*** $\kappa \leq d$ ***do:***
- *1* - (Gram-Schmidt over $\mathbb{Z}$). *By subtracting suitable* $\mathbb{Z}$*-linear combinations of* $\mathbf{b}_1, \ldots, \mathbf{b}_{\kappa-1}$ *from* $\mathbf{b}_\kappa$ *make sure that* $|\mu_{i,\kappa}| \leq \eta$ *for* $i < \kappa$.
- *2* - (LLL Switch). *If interchanging* $\mathbf{b}_{\kappa-1}$ *and* $\mathbf{b}_\kappa$ *will decrease* $l_{\kappa-1}$ *by at least 1 then do so.*
- *3* - (Repeat). *If not switched* $\kappa := \kappa + 1$, *if switched* $\kappa = max(\kappa - 1, 2)$.

That the above algorithm terminates, and that the output is LLL-reduced was shown in [15]. Step B1 has no effect on the l_i. In step B2 the only l_i that change are $l_{\kappa-1}$ and l_κ. The following lemmas present some standard facts which we will need.

Lemma 1. *An LLL switch can not increase* $\max(l_1, \ldots, l_d)$, *nor can it decrease* $\min(l_1, \ldots, l_d)$.

Lemma 2. *If* $\| \mathbf{b}_d^* \| > B$ *then any vector in* L *with norm* $\leq B$ *is a* $\mathbb{Z}$*-linear combination of* $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$.

In other words, if the current basis of the lattice is $\mathbf{b}_1, \ldots, \mathbf{b}_d$ and if the last vector has sufficiently large G-S length then, provided the user is only interested in elements of L with norm $\leq B$, the last basis element can be removed.

Lemma 2 follows from the proof of [15, Eq. (1.11)], and is true regardless of whether $\mathbf{b}_1, \ldots, \mathbf{b}_d$ is LLL-reduced or not. However, if one chooses an arbitrary basis $\mathbf{b}_1, \ldots, \mathbf{b}_d$ of some lattice L, then it is unlikely that the last vector has large G-S length (after all, $\| \mathbf{b}_d{}^* \|$ is the norm of $\mathbf{b}_d$ reduced modulo $\mathbf{b}_1, \ldots, \mathbf{b}_{d-1}$ over $\mathbb{R}$). The effect of LLL reduction is to move G-S length towards later vectors.

3 Main algorithm

In this section we present the central algorithm of the paper and a proof of its correctness. Our algorithm is a kind of wrapper for other standard lattice reduction algorithms. We try to present it as independently as possible of the choice of lattice reduction algorithm. In order to be general we must first outline the features that we require of the chosen lattice reduction algorithm. Our first requirement is that the output satisfy the following slightly weakened version of LLL-reduction.

Definition 2. *Let $L \subseteq \mathbb{R}^n$ be a lattice and $\mathbf{b}_1, \ldots, \mathbf{b}_s \in L$ be $\mathbb{R}$-linearly independent. We call $\mathbf{b}_1, \ldots, \mathbf{b}_s$ an α-reduced basis of L if 1,2, and 3a hold, and an (α, B)-reduced sequence (basis of a sub-lattice) if 1,2, and 3b hold:*

1. $\| \mathbf{b}_i^* \| \leq \alpha \| \mathbf{b}_{i+1}^* \|$ *for* $i = 1 \ldots s-1$.
2. $\| \mathbf{b}_i^* \| \leq \| \mathbf{b}_i \| \leq \alpha^{i-1} \| \mathbf{b}_i^* \|$ *for* $i = 1 \ldots s$.
3. *(a)* $L = \mathbb{Z}\mathbf{b}_1 + \cdots + \mathbb{Z}\mathbf{b}_s$.
   (b) $\| \mathbf{b}_s^* \| \leq B$ *and for every* $\mathbf{v} \in L$ *with* $\| \mathbf{v} \| \leq B$ *we have* $\mathbf{v} \in \mathbb{Z}\mathbf{b}_1 + \cdots + \mathbb{Z}\mathbf{b}_s$.

The original LLL algorithm from [15] returns output with $\alpha = \sqrt{2}$, L^2 from [20] with $\alpha = \sqrt{\frac{1}{\delta - \eta^2}}$ for appropriate choices of (δ, η), and H-LLL from [19] reduced with $\alpha = \frac{\theta\eta + \sqrt{(1+\theta^2)\delta - \eta^2}}{\delta - \eta^2}$ for appropriate (δ, η, θ). We may now also make a useful observation about an (α, B)-reduced sequence.

Lemma 3. *If the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_s$ form an (α, B)-reduced sequence and we let $\mathbf{b}_1^*, \ldots, \mathbf{b}_s^*$ represent the GSO, then the following properties are true:*

- $\| \mathbf{b}_i^* \| \leq \alpha^{s-i} B$ *for all* i.
- $\| \mathbf{b}_i \| \leq \alpha^{s-1} B$ *for all* i.

We use the concept of α-reduction as a means of making proofs which are largely independent of which lattice reduction algorithm a user might choose. For a basis which is α-reduced, a small value of α implies a strong reduction. In our algorithm we use the variable α as the worst-case guarantee of reduction quality. We make our proofs (specifically Lemma 8 and Theorem 3) assuming an $\alpha \geq \sqrt{4/3}$. This value is chosen because [15,20,19] cannot guarantee a stronger reduction. An (α, B)-reduced bases is typically made from an α-reduced basis by removing trailing vectors with large G-S length. The introduction of (α, B)-reduction does not require creating new lattice reduction algorithms, just the minor adjustment of detecting and removing vectors above a given G-S length.

Algorithm 2 ***LLL_with_removals***

Input: $\mathbf{b}_1, \ldots, \mathbf{b}_s \in \mathbb{R}^n$ *and* $B \in \mathbb{R}$.

Output: $\mathbf{b'}_1, \ldots, \mathbf{b'}_{s'} \in \mathbb{R}^n$ (α, B)*-reduced,* $s' \leq s$.

Procedure: *Use any lattice reduction procedure which returns an α-reduced basis and follows Assumption 1. However, when it is discovered that the final vector has G-S length provably $> B$ remove that final vector (deal with it no further).*

Assumption 1 *The lattice reduction algorithm chosen for LLL_with_removals must use switches of consecutive vectors during its reduction process. These switches must have the following properties:*

1. *There exists a number $\gamma > 1$ such that every switch of vectors $\mathbf{b}_i$ and $\mathbf{b}_{i+1}$ increases $\| \mathbf{b}_{i+1}^* \|^2$ by a factor provably $\geq \gamma$.*

2. *The quantity max*$\{\| \mathbf{b}_i^* \|, \| \mathbf{b}_{i+1}^* \|\}$ *cannot be increased by switching* $\mathbf{b}_i$ *and* $\mathbf{b}_{i+1}$.
3. *No steps other than switches can affect G-S norms* $\| \mathbf{b}_1^* \|, \ldots, \| \mathbf{b}_s^* \|$.

Assumption 1 is not very strong as [15,20,19,24,27] and the sketch in Algorithm 1 all conform to these assumptions. We do not allow for the extreme case where $\gamma = 1$, although running times have been studied in [2,16]. It should also be noted that in the floating point lattice reduction algorithms $\| \mathbf{b}_s^* \|$ is only known approximately. In this case one must only remove vectors whose approximate G-S length is sufficiently large to ensure that the exact G-S length is $\geq B$.

The format of the input matrices was given in section 1. A search parameter B is given to bound the norm of the target vectors. The algorithm performs its best when B is small compared to the bit-length of the entries in the input matrix, although B need not be small for the algorithm to work.

Definition 3. *We say the* P_j *are* large enough *if:*

$$|P_j| \geq 2\alpha^{4r+4k+2}B^2 \text{ for all but } k = \mathcal{O}(r) \text{ values of } j. \tag{1}$$

Note that if $N = \mathcal{O}(r)$ then the P_j are trivially large enough. However, for applications where N is potentially much larger than r this becomes a non-trivial condition. In this case having B close to X means that the P_j's are not large enough.

In the following algorithm we will gradually reduce the input basis. This will be done one column at a time, similar to the experiments in [3,6]. The current basis vectors are denoted $\mathbf{b}_i$ and we will use M to represent the matrix whose rows are the $\mathbf{b}_i$. We will use the notation $\mathbf{x}_j$ to represent the column vector $(x_{1,j}, \ldots, x_{r,j})^T$.

The matrix M will begin as $I_{r\times r}$, and we will adjoin $\mathbf{x}_1$ and a new row $(\mathbf{0}, P_1)$ if appropriate. Each time we add a column $\mathbf{x}_j$ we will need to calculate the effects of prior lattice reductions on the new $\mathbf{x}_j$. We use $\mathbf{y}_j$ to represent a new column of entries which will be adjoined to M. In fact $\mathbf{y}_j = M[1, \ldots, r] \cdot \mathbf{x}_j$. Before adjoining the entries we also scale them by a power of 2, to have smaller absolute values. This keeps the entries in M at a uniform absolute value. The central loop of the algorithm is the process of gradually using more and more bits of $\mathbf{y}_j$ until every entry in M is again an integer. No rounding is performed: we use rational arithmetic on the last column of each row. Throughout the algorithm the number of rows of M will be changing. We let s be the current number of rows of M. If (1) is satisfied for some $k = \mathcal{O}(r)$ then we can actually bound s by $2r + 2k + 1$. We use c as an apriori upper bound on s, either $c := 2r + 2k + 1$ or $c := r + N$. The algorithm has better performance when c is small. We let L represent the lattice generated by the rows of A.

Algorithm 3 Gradual_LLL

Input: *A search parameter,* $B \geq \sqrt{5} \in \mathbb{Q}$*, an integer knapsack-type matrix,* A*, and an* $\alpha \geq \sqrt{4/3}$.

Output: *An* (α, B)*-reduced basis* $\mathbf{b}_1, \ldots, \mathbf{b}_s$ *of a sub-lattice* L' *in* L *with the property that if* $\mathbf{v} \in L$ *and* $\| \mathbf{v} \| \leq B$ *then* $\mathbf{v} \in L'$.

The Main Algorithm:

1 - **if** (1) holds set $c := \min(2r + 2k + 1, r + N)$
2 - $s := r; M := I_{r\times r}$

3 - **for** $j = 1 \dots N$ **do**:
  a - $\mathbf{y}_j := M[1, \dots, r] \cdot \mathbf{x}_j$; $\ell := \lfloor \log_2 (\max\{|P_j|, \|\mathbf{y}_j\|_\infty, 2\}) \rfloor$
  b - $M := \begin{bmatrix} 0 & \lfloor P_j/2^\ell \rfloor \\ M & \lfloor \mathbf{y}_j/2^\ell \rfloor \end{bmatrix}$; **if** $P_j \neq 0$ **then** $s := s + 1$ **else** remove zero row
  c - **while** $(\ell \neq 0)$ **do:**
    i - $\mathbf{y}_j := 2^\ell \cdot M \cdot [0, \cdots, 0, 1]^T$; $\ell := \max\{0, \lceil \log_2 (\frac{\|\mathbf{y}_j\|_\infty}{\alpha^{2c} B^2}) \rceil\}$
    ii - $M := \left[M[1, \dots, r + j - 1] \middle| \mathbf{y}_j/2^\ell \right]$
    iii - Call LLL_with_removals on M and set M to output; adjust s
4 - **return** M

First we will prove the correctness of the algorithm. We need to show that the Gram-Schmidt lengths are never decreased by scaling the final entry or adding a new entry.

Lemma 4. *Let $\mathbf{b}_1, \dots, \mathbf{b}_s \in \mathbb{R}^n$ be the basis of a lattice and $\mathbf{b}_1^*, \dots, \mathbf{b}_s^*$ its GSO. Let $\sigma : \mathbb{R}^n \to \mathbb{R}^n$ scale up the last entry by some factor $\beta > 1$, then we have $\| \mathbf{b}_i^* \| \leq \| \sigma(\mathbf{b}_i)^* \|$. In other words, scaling the final entry of each vector by the same scalar $\beta > 1$ cannot decrease $\|\mathbf{b}_i^*\|$ for any i.*

Lemma 5. *Let $\mathbf{b}_1, \dots, \mathbf{b}_s \in \mathbb{R}^n$ and let $\mathbf{b}_1^*, \dots, \mathbf{b}_s^* \in \mathbb{R}^n$ be their GSO. The act of adjoining an $(n+1)^{st}$ entry to each vector and re-evaluating the GSO cannot decrease $\| \mathbf{b}_i^* \|$ for any i (assuming that the new entry is in $\mathbb{R}$).*

The proofs of these lemmas are quite similar and can be found in the appendix. Now we are ready to prove the first theorem, asserting the correctness of algorithm 3's output.

Theorem 1. *Algorithm 3 correctly returns an α-reduced basis of a sub-lattice, L', in L such that if $\mathbf{v} \in L$ and $\| \mathbf{v} \| \leq B$ then $\mathbf{v} \in L'$.*

Proof. When the algorithm terminates all entries are unscaled and each vector in the output is inside of L as it is a linear combination of the original input vectors. Thus the output is a basis of a sub-lattice L' inside L. Further, the algorithm terminates after a final call to step 3(c)iii so returns an (α, B)-reduced sequence.

Now we show that if $\mathbf{v} \in L$ and $\| \mathbf{v} \| \leq B$ then $\mathbf{v} \in L'$. The removed vectors correspond to vectors $\tilde{\mathbf{b}_i} \in L$ that, by lemmas 4 and 5, have G-S length at least as large as those of $\mathbf{b}_i$. The claim then follows from lemmas 1 and 2.

4 Two invariants of the algorithm

Here we present the important proofs about the set-up of our algorithm. All proofs in this section and the next allow for a black-box lattice reduction algorithm up to satisfying assumption 1. Each proof in this section involves the study of an invariant. The two invariants which we use are:

- The Active Determinant, AD(M), which is the product of the G-S lengths of the active vectors. This remains constant under standard lattice reduction algorithms, and allows us to bound many features of the proofs.
- The Progress, $PF = \sum_{i=1}^{s} (i-1) \log \| \mathbf{b}_i^* \|^2 + n_{\text{rm}} r \log(4\alpha^{4c} B^4)$, where n_{rm} is the total number of vectors which have been removed so far. This function is an energy function which never decreases, and is increased by ≥ 1 for each switch made in the lattice reduction algorithm.

A study of the active determinant

Definition 4. *We call the active determinant of the vectors $\mathbf{b}_1, \ldots, \mathbf{b}_s$ the product of their Gram-Schmidt lengths. For notation we use, AD or $AD(\{\mathbf{b}_i\}) := \prod_{i=1}^{s} \| \mathbf{b}_i^* \|$. For a matrix M with the i^{th} row denoted by $M[i]$, we use AD or $AD(M) = AD(\{M[1], \ldots, M[s]\})$.*

For an (α, B)-reduced sequence we can nicely bound the AD. We have such a sequence after each execution of step 3(c)iii.

Lemma 6. *If $\mathbf{b}_1, \ldots, \mathbf{b}_s$ are an (α, B)-reduced sequence then $AD \leq (\alpha^{s-1} B^2)^{s/2}$.*

We now want to attack two problems, bounding the norm of each vector just before lattice reduction, and bounding the number of vectors throughout the algorithm.

Lemma 7. *If $s \leq c$ then just before step 3(c)iii we have $\| \mathbf{b}_i \|^2 \leq 2\alpha^{4c} B^4$ for $i = 1 \ldots s$.*

The full details of this proof can be found in the appendix. The following theorem holds trivially when there is no condition on the P_j or if $N = 0$. When $N > r$ and B is at least a bit smaller than X we can show that not all of the extra vectors stay in the lattice. In other words, if there is enough of a difference between B and X then the sub-lattice aspect of the algorithm begins to allow for some slight additional savings. Here the primary result of this theorem is allowing $\mathcal{O}(r)$ vectors with a relatively weak condition on the P_j.

Theorem 2. *Throughout the algorithm we have $s \leq c$.*

Proof. If $c = r + N$ then $s \leq c$ is vacuously true. So assume $c = 2(r + k) + 1$ and all but $k = \mathcal{O}(r)$ of the P_j satisfy $|P_j| \geq 2\alpha^{4r+4k+2} B^2$. When the algorithm begins, AD $= 1$ and $s = r$. For s to increase step 3 must finish without removing a vector. If this happens during iteration j then the AD has increased by a factor $|P_j|$. The LLL-switches inside of step 3(c)iii do not alter the AD by Assumption 1. Each vector which is removed during step 3(c)iii has G-S length $\leq 2\alpha^{4r+4k+2} B^2$ by Lemmas 7 and 1. After iteration j we have $n_{\text{rm}} = r + j - s$ as the total number of removed vectors. All but k of the P_i have larger norm than any removed vector. Therefore the smallest AD can be after iteration j is $\geq (2\alpha^{(4r+4k+2)} B^2)^{j-k-n_{\text{rm}}}$. Rearranging we get AD $\geq (2\alpha^{4r+4k+2} B^2)^{s-r-k}$. This contradicts Lemma 6 when s reaches $2r + 2k$ for the first time because $(2\alpha^{4r+4k+2} B^2)^{r+k} \geq (\alpha^{2r+2k-1} B^2)^{r+k}$.

Corollary 1. *Throughout the algorithm we have $\| \mathbf{b}_i^* \| \leq 2\alpha^{2c} B^2$.*

We also use the active determinant to bound the number of iterations of the main loop, i.e. step 3c. First we show in the appendix that AD is increased by every scaling which does not end the main loop.

Lemma 8. *Every execution of step 3(c)ii either increases the AD by a factor $\geq \frac{\alpha^c B}{2}$ or sets $\ell = 0$.*

Now we are ready to prove that the number of iterations of the main loop is $\mathcal{O}(r + N)$. This is important because it means that, although we look at all of the information in the lattice, the number of times we have to call lattice reduction is unrelated to $\log X$.

Theorem 3. *The number of iterations of step 3c is $\mathcal{O}(r + N)$.*

The strategy of this proof is to show that each succesful scaling increases the active determinant and to bound the number of iterations using Lemma 6 and Corollary 1. For space constraints this proof is provided in the appendix.

A study of the progress function We will now amortize the costs of lattice reduction over each of the $\mathcal{O}(r+N)$ calls to step 3(c)iii. We do this by counting switches, using Progress PF (defined below). In order to mimic the proof from [15] for our algorithm we introduce a type of Energy function which we can use over many calls to LLL (not only a single call).

Definition 5. *Let $\mathbf{b}_1, \ldots, \mathbf{b}_s$ be the current basis at any point in our algorithm, let $\mathbf{b}_1^*, \ldots, \mathbf{b}_s^*$ be their GSO, and $l_i := \log_\gamma \| \mathbf{b}_i^* \|^2$ for all $i = 1 \ldots s$. We let n_{rm} be the number of vectors which have been removed so far in the algorithm. Then we define the progress function PF to be:*

$$PF := 0 \cdot l_0 + \cdots + (s-1) \cdot l_s + n_{rm} \cdot c \cdot \log_\gamma (4\alpha^{4c} B^4).$$

This function is designed to effectively bound the largest number of switches which can have occurred so far. To prove that it serves this purpose we must prove the following lemma:

Lemma 9. *After step 2 Progress PF has value 0. No step in our algorithm can cause the progress PF to decrease. Further, every switch which takes place in step 3(c)iii must increase PF by at least 1.*

Theorem 4. *Throughout our algorithm the total number of switches used by all calls to step 3(c)iii is $\mathcal{O}((r+N)c(c+\log B))$ with P_j and $\mathcal{O}(c^2(c+\log B))$ with no P_j.*

Proof. Since Lemma 9 shows us that PF never decreases and every switch increases PF by at least 1, then the number of switches is bounded by PF. However PF is bounded by Lemma 7 which bounds $l_i \leq \log_\gamma (\alpha^{4r} B^4)$, Theorem 2 which bounds $s \leq c$, and the fact that we cannot remove more vectors than are given which implies $n_{\mathrm{rm}} \leq r+N$. Further we can see that $(s-1)l_s \leq (c-1)\log_\gamma (4\alpha^{4c} B^4)$ so PF is maximized by making $n_{\mathrm{rm}} = (r+N)$ (or c if no vectors added) and $s = 0$. In which case we have number of switches $\leq PF \leq (r+N)(c-1)(\log_\gamma (4\alpha^{4c} B^4) = \mathcal{O}((r+N)c(c+\log B))$. Also if there are no P_j, we can replace $r+N$ by c.

5 Complexity bound of main algorithm

In this section we wish to prove a bound for the overall bit-complexity of algorithm 3. The complexity bound must rely on the complexity bound of the lattice reduction algorithm we choose for step 3(c)iii. The results in the previous sections have not relied on this choice. We will present our complexity bound using the H-LLL algorithm from [19]. We choose H-LLL for this result because of its favorable complexity bound and because the analysis of our necessary adaptations is relatively simple. See [19] for more details on H-LLL.

We make some minor adjustments to the H-LLL algorithm and its analysis. The changes to the algorithm are the following:

- We have a single non-integer entry in each vector of bit-length $\mathcal{O}(c + \log X)$.
- Whenever the final vector has G-S length sufficiently larger than B, it is removed. This has no impact on the complexity analysis.

We use τ as the number of switches used in a single call to H-LLL. This allows the analysis of progress PF to be applied directly. The following theorem is an adaptation of the main theorem in [19] adapted to reflect our adjustments.

Theorem 5. *If a single call to step 3(c)iii, with H-LLL [19] as the chosen variation of LLL, uses τ switches then the CPU cost is bounded by $\mathcal{O}((\tau + c + \log B)c^2[(r+N)(c+\log B) + \log X])$ bit-operations.*

Now we are ready to complete the complexity analysis of the our algorithm.

Theorem 6. *The cost of executing algorithm 3 with H-LLL [19] as the variant of LLL in step 3(c)iii is*

$$\mathcal{O}((r+N)c^3(c+\log B)[\log X + (r+N)(c+\log B)])$$

CPU operations, where B is a search parameter chosen by the user, $|A[i,j]| \leq X$ for all i,j, and $c = r+N$ or $c = \mathcal{O}(r)$ (see definition 3 for details). If there are no P_j's then the cost is

$$\mathcal{O}((r+N+c^2)(c+\log B)c^2[\log X + (r+N)(c+\log B)]).$$

Proof. Steps 2, 3b, 3(c)i, and 3(c)ii have negligible costs in comparison to the rest of the algorithm. Step 3a is called N times, each call performs s inner products. While each inner product performs r multiplications each of the form $\mathbf{b}_i[m] \cdot x_{m,j}$ appealing to Corollary 1 we bound the cost of each multiplication by $\mathcal{O}((c+\log B)\log X)$. Since Theorem 2 gives $s \leq c$ we know that the total cost of all calls to step 3a is $\mathcal{O}(Ncr(c+\log B)\log X)$. Let $k = \mathcal{O}(r+N)$ be the number of iterations of the main loop. Let τ_i be the number of LLL switches used in the i^{th} iteration. Theorem 5 gives the cost of the i^{th} call to step 3(c)iii as $= \mathcal{O}((\tau_i + c + \log B)c^2[(r+N)(c+\log B) + \log X])$. Theorem 4 implies that $\tau_1 + \cdots + \tau_k = \mathcal{O}((r+N)c(c+\log B))$ (or $\mathcal{O}(c^2(c+\log B))$ when there are no P_j's). The total cost of all calls to step 3(c)iii is then $\mathcal{O}([k(c+\log B) + \tau_1 + \cdots + \tau_k]c^2[(r+N)(c+\log B) + \log X])$. The term $[k(c+\log B) + \tau_1 + \cdots + \tau_k]$ can be replaced by $\mathcal{O}((r+N)c(c+\log B))$ (if no P_j then $\mathcal{O}((r+N+c^2)(c+\log B))$). The complete cost of is now $\mathcal{O}(Nrc(c+\log B)\log X + (r+N)c^3[c + \log B)(\log X + (r+N)(c+\log B)])$. The first term is absorbed by the cost of the second term, proving the theorem. If there are no P_j then we get $\mathcal{O}((r+N+c^2)(c+\log B)c^2[\log X + (r+N)(c+\log B)])$.

6 New complexities for applications of main algorithm

Our algorithm has been designed for some applications of lattice reduction. In this section we justify the importance of this algorithm by directly applying it to two classical applications of lattice reduction.

New complexity bound for factoring in $\mathbb{Z}[x]$ In [4] it is shown that the problem of factoring a polynomial, $f \in \mathbb{Z}[x]$, can be accomplished by the reduction of a large knapsack-type lattice. In this subsection we merely apply our algorithm to the lattice suggested in [4].

Reminders from [4]. Let $f \in \mathbb{Z}[x]$ be a polynomial of degree N. Let A be a bound on the absolute value of the coefficients of f. Let p be a prime such that $f \equiv l_f f_1 \cdots f_r \bmod p^a$ a separable irreducible factorization of f in the p-adics lifted to precision a, the f_i are monic, and l_f is the leading coefficient of f. For our purposes we choose $B := \sqrt{r+1}$.

We will make some minor changes to the All-Coefficients matrix defined in [4] to produce a matrix that looks like:

$$\begin{pmatrix} & & & & p^{a-b_N} \\ & & & \iddots & \\ & & p^{a-b_1} & & \\ 1 & & x_{1,1} & \cdots & x_{1,N} \\ & \ddots & \vdots & \ddots & \vdots \\ & & 1 \; x_{r,1} & \cdots & x_{r,N} \end{pmatrix}.$$

Here $x_{i,j}$ is the j^{th} coefficient of $f_i' \cdot f/f_i$ mods p^a divided by p^{b_j} and p^{b_j} represents $\sqrt{N}$ times a bound on the j^{th} coefficient of $g' \cdot f/g$ for any true factor $g \in \mathbb{Z}[x]$ of f. In this way the target vectors will be quite small. An empty spot in this matrix represents a zero entry. This matrix has $p^{a-b_j} > 2^{N^2+N\log(A)} > 2\alpha^{4r+2}B^2$ for all j. An (α, B)-reduction of this matrix will solve the recombination problem by a similar argument to the one presented in [4] and refined in [22]. Now we look at the computational complexity of making and reducing this matrix which gives the new result for factoring inside $\mathbb{Z}[x]$.

Theorem 7. *Using algorithm 3 on the All-Coefficients matrix above provides a complete irreducible factorization of a polynomial f of degree N, coefficients of bit-length $\leq \log A$, and r irreducible factors when reduced modulo a prime p in*

$$\mathcal{O}(N^2 r^4 [N + \log A])$$

CPU operations. The cost of creating the All-Coefficients matrix adds $\mathcal{O}(N^4[N^2 + \log^2 A])$ CPU operations using classical arithmetic (suppressing small factors $\log r$ and $\log^2 p$) to the complexity bound.

The following chart gives a complexity bound comparison of our algorithm with the factorization algorithm presented by Schönhage in [25] we estimate both bounds using classical arithmetic and fast FFT-based arithmetic [5]. We also suppress all $\log N$, $\log r$, $\log p$, and $\log\log A$ terms.

Classical Gradual_LLL	$\mathcal{O}(N^3 r^4 + N^2 r^4 \log A + N^6 + N^4 \log^2 A)$
Classical Schönhage	$\mathcal{O}(N^8 + N^5 \log^3 A)$
Fast Gradual_LLL	$\mathcal{O}(N^3 r^3 + N^2 r^3 \log A)$
Fast Schönhage	$\mathcal{O}(N^6 + N^4 \log^2 A)$

The Schönhage algorithm is not widely implemented because of its impracticality. For most polynomials, r is much smaller than N. Our main algorithm will reduce the All-Coefficients matrix with a competitive practical running time, but constructing the matrix itself will require more Hensel lifting than seems necessary in practice. In [22] a similar switch-complexity bound to section 4 is given on a more practical factoring algorithm.

Algebraic number reconstruction The problem of finding a minimal polynomial from an approximation of a complex root was attacked in [14] using lattice reduction techniques using knapsack-type bases. For an extensive treatment see [17].

Theorem 8. *Suppose we know $\mathcal{O}(d^2 + d\log H)$ bits of precision of a complex root α of an unknown irreducible polynomial, $h(x)$, where the degree of h is d and its maximal coefficient has absolute value $\leq H$. Algorithm 3 can be used to find $h(x)$ in $\mathcal{O}(d^7 + d^5 \log^2 H)$ CPU operations.*

This new complexity is an improvement over the L^2 algorithm which would use $\mathcal{O}(d^9+d^7\log^2 H)$ CPU operations to reduce the same lattice. Although, one can prove a better switch-complexity with a two-column knapsack matrix by using [10, Lem. 2] to bound the determinant of the lattice as $\mathcal{O}(X^2)$ and thus the potential function from [15] is $\mathcal{O}(X^{2d})$, leading to a switch complexity of $\mathcal{O}(d\log X)$ (posed as an open question in [26, sec. 5.3]). Using this argument the complexity for L^2 is reduced to $\mathcal{O}(d^8+d^6\log^2 H)$.

Acknowledgements. We thank Damien Stehlé, Nicolas Brisebarre, and Valérie Berthé for many helpful discussions. Also Ivan Morel for introducing us to H-LLL. This work was partially funded by the LaRedA project of the Agence Nationale de la Recherche, it was also supported in part by a grant from the National Science Foundation. It was initiated while the second author was hosted by the Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier (LIRMM).

References

1. M. Ajtai *The shortest vector problem in L2 is NP-hard for randomized reductions*, STOC 1998, pp. 10–19.
2. A. Akhavi, *The optimal LLL algorithm is still polynomial in fixed dimension*, Theor. Comp. Sci. vol. 297 iss. 1-3 Mar. 2003, LATIN, pp. 3–23.
3. K. Belabas *A relative van Hoeij algorithm over number fields*, J. Symb. Comp. **37** 2004, pp. 641–668.
4. K. Belabas, M. van Hoeij, J. Klüners, and A. Steel, *Factoring polynomials over global fields*, preprint arXiv:math/0409510v1 (2004).
5. D. Bernstein, *Multiprecision Multiplication for Mathematicians*, accepted by Advances in Applied Mathematics find at http://cr.yp.to/papers.html#m3, 2001.
6. C. Bright, *Vector Rational Number Reconstruction*, Masters Thesis, University of Waterloo, 2009.
7. D. Coppersmith, *Small solutions to polynomial equations, and low exponent RSA vulnerabilities*, J. of Cryptology 10, pp. 233–260, 1997.
8. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.
9. D. Goldstein and A. Mayer, *On the equidistribution of Hecke points*, Forum Mathematicum **15** 2003, pp. 165–189.
10. G. Hanrot, *LLL: a tool for effective diophantine approximation*, LLL+25 2007, pp. 81–118.
11. M. van Hoeij, *Factoring polynomials and the knapsack problem*, J. Num. The., **95** 2002, pp. 167–189.
12. N. A. Howgrave-Graham and N. P. Smart, *Lattice attacks on digital signature schemes*, Design, Codes and Cryptography, 23, pp. 283–290, 2001.
13. E. Kaltofen,*On the complexity of finding short vectors in integer lattices*, EUROCAL'83,LNCSv.162,pp.235–244
14. R. Kannan, A. K. Lenstra, and L. Lovász, *Polynomial Factorization and Nonrandomness of Bits of Algebraic and Some Transcendental Numbers*, STOC 1984, pp. 191–200.
15. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann. **261** 1982, pp. 515–534.
16. H. W. Lenstra, *Flags and lattice basis reduction*, Euro. Cong. Math. v. 1 2001, Verlag, Basel.
17. L. Lovász, *An Algorithmic Theory of Numbers, Graphs and Convexity*, SIAM 1986
18. D. Micciancio, *The Shortest Vector Problem is NP-hard to approximate to within some constant*, SIAM journal on Computing v. 30 num. 6, pp. 2008–2035, 2001.
19. I. Morel, D. Stehlé, and G. Villard, *H-LLL: Using Householder Inside LLL*, ISSAC'09, pp. 271–278.
20. P. Nguyen and D. Stehlé, *Floating-point LLL revisited*, Eurocrypt 2005, v. 3494 LNCS, pp. 215–233.
21. P. Nguyen and D. Stehlé, *An LLL Algorithm with Quadratic Complexity*, J. Cmp. v.39n.3 2009, pp. 874–903.
22. A. Novocin, *Factoring Univariate Polynomials over the Rationals*, Ph.D. Flor. St. Univ. 2008.
23. A. Odlyzko, *The rise and fall of knapsack cryptosystems*, Crytology and Computational Number Theory, vol. 42 of Proc. of Symposia in Applied Mathematics, A.M.S., pp. 75–88, 1990.
24. C. P. Schnorr, *A more efficient algorithm for lattice basis reduction*, J. of Algo. **9** 1988, pp. 47–62.
25. A. Schönhage, *Factorization of univariate integer polynomials by Diophantine approximation and an improved basis reduction algorithm*, ICALP 84, LNCS **172**, pp. 436–447.
26. D. Stehlé, *Floating-point LLL: Theoretical and Practical Aspects*,LLL+25 2007, pp. 33–61.
27. A. Storjohann, *Faster algorithms for integer lattice basis reduction*, Technical report 1996, ETH Zürich.