A survey of approaches to the shortest vector problem on lattices: the LLL algorithm and beyond

Jeremy Porter, CSCI-6101

April 18, 2011

Contents

Abstract

Lattices admit several well-known problems which have been shown to be computationally hard to solve exactly, but which allow polynomial time approximation algorithms. The roots of this research are found in computational number theory, so it is perhaps no surprise that more recent research has found exciting applications in cryptography and cryptanalysis.

We will briefly introduce the concepts behind lattices and the most commonly posed lattice problems, with our main focus being on the presentation of the famous Lenstra-Lenstra-Lovász algorithm, and several of the most crucial improvements which have been made since its discovery. As much of the recent research requires considerable familiarity with past results, and as these results are themselves not trivial to grasp, we have focused on presenting details of the fundamental material rather than of the cutting edge research. Our goal is to provide a solid base of understanding, from which a reader would feel comfortable moving to more recent and in-depth results.

1 Introduction and motivation

1.1 Number theory and cryptography

Lattice problems have long been known to have deep relations to various fields of mathematics. The original LLL algorithm paper [14] was intended as a method for lattice basis reduction in order to efficiently factor polynomials in $\mathbb{Q}[x]$. It was not until later that the more general applications of basis reduction were explored, and continue to be explored today. Basis reduction in general, and specifically the LLL algorithm is used to study a variety of other problems, including diophantine approximation (approximating $r \in \mathbb{R}$ with $r' \in \mathbb{Q}$, within some given parameters) [11], computing ideal class groups over the ring of integers of a number field [28], and factorization problems related to RSA [15].

This last use is related to perhaps the most stimulating motivation for modern research into lattice basis reduction. Lattices have strong potential for applications to cryptography, both as a tool for cryptanalysis [22, 12] of RSA and knapsack-style systems (among others), and also in the construction of wholly new cryptographic systems. Of particular interest is the equivalence of average-case/worst-case complexity on the *unique* shortest-vector lattice problem (a variation of the traditional shortest vector problem). In [1], Ajtai used lattices to describe for the first time a cryptographic primitive which admitted a polynomial-time solution to a random instance only if there were a polynomial-time solution to the worst-case instance of this particular lattice problem. Such provable security is very attractive from a cryptographic standpoint [20].

A public-key cryptosystem was later constructed around this principle [2], and while practically inefficient it served to motivate further research. Perhaps because the relative difficulty of the unique shortest vector problem is poorly understood with respect to other lattice problems, the two most well-established lattice-based public-key cryptosystems are based on the CVP and SVP lattice problems (see section 1.3); these are the GGH and NTRU systems, respectively. However, neither system has approached the level of popularity of competing, non-lattice-based systems like RSA or ECC. Still, lattice-based cryptography is an active area of research, with recently renewed interest because of its potential resilience to post-quantum cryptanalysis [18].

1.2 Lattice basics

We will attempt to present the basic essentials necessary for studying computationally hard problems on lattices. As we have already stated however, lattices are an interesting focal point of several branches of mathematics and are widely studied in their own right. For more details on lattices and their mathematical underpinnings, and for the background of the material we present here, see [7].

An n dimensional lattice L is an additive subgroup of $\mathbb{R}^n$ such that

$$L = L(b_1, \ldots, b_n) = \left\{ \sum_{i=1}^{n} a_i b_i : a_i \in \mathbb{Z} \right\} \tag{1.2.1}$$

where the b_i are linearly independent and span $\mathbb{R}^n$. In other words, a lattice is a free $\mathbb{Z}$-module of rank n, with the elements b_i acting as the basis vectors. Of course, a rank n lattice may also be thought of as living in $\mathbb{R}^m$ for $m > n$. Each rank n lattice L therefore has at least $(n-1)$ sublattices L_i for $i = 1, \ldots, (n-1)$, where L_i is the lattice formed from the first i basis vectors $b_1, \ldots, b_i$.

Geometrically, lattices form tilings of the space $\mathbb{R}^n$, dividing it into infinitely many identical *fundamental regions*. The volume of this region is intrinsically related to the lattice, so while we may choose any suitable basis to represent L, the value of $\mathrm{vol}(L)$ is independent of this choice. Another invariant value is the determinant $\det(L) = \mathrm{vol}(L)^2$, so named because lattices also admit matrix representations.

With basis vectors $b_i = (b_{i1}, b_{i2}, \ldots, b_{in})$, write

$$B = \begin{bmatrix} b_{11} & b_{12} & \ldots & b_{1n} \\ b_{21} & b_{22} & \ldots & b_{2n} \\ \vdots & & \ddots & \vdots \\ b_{n1} & b_{n2} & \ldots & b_{nn} \end{bmatrix} \tag{1.2.2}$$

where the i-th row consists of the elements of the i-th basis vector. This matrix B is then the *generator matrix* for the lattice, since any vector v on the lattice may be written as

$$v = \xi B$$

where $\xi = (\xi_1, \ldots, \xi_n)$ is a vector with elements all in $\mathbb{Z}$. This can simply be thought of as shorthand for the definition in equation (1.2.1). The matrix $G = MM^T$ is called the *Gram matrix*, and its determinant satisfies $\det(G) = \mathrm{vol}(L)^2$, which gives us the name for $\det(L)$.

We will use the usual linear algebra definitions of vector length (or norm, in this case the ℓ^2 norm) and inner products, so that $|x| = \sqrt{x_1^2 + \ldots x_n^2}$, and $\langle x, y \rangle = \sum_i x_i y_i$ respectively.

As mentioned, lattices have a natural connection with number theory. As the details are beyond our scope, we will simply sketch the basic connection. We observed that any lattice vector $x \in L$ can be written as a sum of integer multiples of the basis vectors $b_1, \ldots, b_n$, or in other words that

$$x = \xi_1 b_1 + \xi_2 b_2 + \cdots + \xi_n b_n = \xi \cdot B$$

where $\xi_i \in \mathbb{Z}$ and $\xi \in \mathbb{Z}^n$. Since B is a fixed value for the lattice, each vector x is uniquely determined by

the elements of ξ. In particular, we may compute the square of the norm $||x|| = |x|^2$ as

$$\begin{aligned} ||x|| = \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} \left(\xi_1 b_{1i} + \xi_2 b_{2i} + \cdots + \xi_n b_{ni}\right)^2 \\ &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \xi_j b_{ji} \right)^2 \end{aligned}$$

where b_{ij} is again the j-th element of the basis vector b_i. Expanding this expression and collecting like terms, we find an n variable polynomial $\sum_{i=1}^{n} c_i \xi_i = f(\xi)$ given in terms of the integers ξ_i, corresponding to the squared norm of the lattice vector determined by ξ. This degree 2 polynomial is homogeneous, so it is a *quadratic form*. Quadratic forms are central objects of study in varied fields of mathematics, which offers an immediate setting for any studies on lattices.

1.3 Lattice problems

Shortest Vector Problem (SVP)

Given a lattice L of dimension n defined by the basis elements $b_1, \ldots, b_n$, find the $\ell \in L$ of minimum, non-zero length. For arbitrary dimension n, the only algorithms which guarantee an exact solution run in exponential time. An exact solution can be found by using the AKS sieve [3], which operates in $2^{O(n)}$ time and space.

Closest Vector Problem (CVP)

Given a lattice L of dimension n defined by the basis elements $b_1, \ldots, b_n$, and a target $t \in \mathbb{R}^n$, find $\ell \in L$ which minimizes the distance between their end points. This can be solved exactly using HKZ-bases (see definition 2.6) and a bounded enumeration approach in $O(n^n s)$ arithmetic operations, where s is the bit size of the input [13]. Using dual HKZ-bases (a concept we will not explore), an exact solution is given in $n! s^{O(1)}$ operations [4].

Shortest Independent Vector Problem (SIVP)

Given a lattice L of dimension n defined by the basis elements $b_1, \ldots, b_n$, find n linearly independent $\ell_i \in L$ which minimizes $\max_i |\ell_i|$. We can find an exact solution using a polynomial time reduction between SIVP and CVP, which has the same bound $n! s^{O(1)}$ as CVP [17].

Following the polynomial time reductions given in [17], it is also possible to solve any of these three problems by a reduction to a variant of CVP called CVPP. This accepts as input the lattice, the target vector, and a polynomially sized "hint" which relies exclusively on the lattice. This "CVP with Preprocessing" variant is solved in [21] by pre-computing the Voronoi cell of the lattice as the hint. This Voronoi approach to CVPP uses $\tilde{O}(2^{2n+o(n)})$ operations, and therefore provides exact solutions to each of SVP, CVP, and SIVP within the same bound. Indeed, this is the asymptotically fastest approach to finding exact solutions to each of the problems.

SVP, CVP, and SIVP are known to be NP-hard, so the performance of the above solutions is understandable. In particular, no polynomial algorithms are expected for their exact solutions. However, polynomial time approximation algorithms do exist for SVP and CVP [16]. Finding an approximate solution to SVP using any approximation factor $\leq \sqrt{2}$ is provably NP-hard, but we will pursue much larger factors and so be able to consider polynomial time approximations. While CVP and SIVP are interesting both in their own right and for their use in cryptography, the most famous problem is SVP, because of the algorithm which lead to its first well-known approximation solution. This solution is the LLL algorithm, which takes an arbitrary lattice basis as its input, and returns a vector within a factor of $2^{O(n)}$ of the lattice's shortest overall vector.

2 The Lenstra-Lenstra-Lovasz algorithm

The Lenstra-Lenstra-Lovasz (LLL) algorithm [14] was originally presented as a method for factoring polynomials; however, the approach used was also a natural fit for approaching the SVP on lattices. It does not offer an exact solution, however it does guaranteed an approximation bound of $2^{O(n)}$. Throughout this section, we will summarize the key results for the algorithm and the underlying mathematics. Our details are based on an amalgam of the presentations in [19, 14, 6, 5].

2.1 The theory

The algorithm relies primarily on two key theorems borrowed from linear algebra, presented here as theorems 2.1 and 2.2.

Theorem 2.1 (Hadamard)**.** *For an n-dimensional lattice L with basis vectors $b_1, \ldots, b_n$ and determinant* $\det(L)$*,*

$$\det(L) \leq \prod_{i=1}^{n} |b_i|.$$

In particular, if the basis $b_1^*, \ldots, b_n^*$ is orthogonal, then this becomes the equality

$$\det(L) = \prod_{i=1}^{n} |b_i^*|.$$

Gram-Schmidt orthogonalization allows us to compute an orthogonal basis from a basis which need not be orthogonal. This is typically used to find an orthonormal basis, where the length of each orthogonal basis vector is also normalized to be of unit length; however, we omit this normalization step as it is not needed for our purposes.

Theorem 2.2 (Gram-Schmidt)**.** *Given a basis $b_1, \ldots, b_n$ for a lattice L, inductively define*

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$$

where

$$\mu_{i,j} = \frac{\langle b_i, b_j^* \rangle}{\langle b_j^*, b_j^* \rangle}.$$

Then these b_i^* form an orthogonal basis for the same lattice L.

So for instance, given $L(b_1, \ldots, b_n)$ we can compute

$$\begin{aligned} b_1^* &= b_1 \\ b_2^* &= b_2 - \frac{\langle b_2, b_1^* \rangle}{\langle b_1^*, b_1^* \rangle} b_1^* \\ b_3^* &= b_3 - \frac{\langle b_3, b_2^* \rangle}{\langle b_2^*, b_2^* \rangle} b_2^* - \frac{\langle b_3, b_1^* \rangle}{\langle b_1^*, b_1^* \rangle} b_1^* \\ &\vdots \end{aligned}$$

where the lattice $L(b_1^*, \ldots, b_n^*)$ is identical to the original, and each pair of vectors b_i^*, b_j^* with $i \neq j$ is orthogonal.

Using the same notation from these two theorems, we define two conditions on lattice bases which must be met for the LLL algorithm to operate. Collectively, we will refer to these as the "LLL conditions".

Definition 2.3. *A lattice basis is size-reduced if the Gram-Schmidt coefficients satisfy* $|\mu_{i,j}| \leq \frac{1}{2}$ *for* $1 \leq j < i \leq n$.

Phrased in terms of the Gram-Schmidt orthogonalization process, this means all coefficients used to compute the vectors b_i^* are bounded above by $\frac{1}{2}$.

Definition 2.4. *A lattice basis is* α*-reduced if*

$$|b_i^*|^2 \geq \frac{1}{\alpha} |b_{i-1}^*|^2.$$

We remark that if the lattice is already size-reduced, this is equivalent to requiring

$$|b_i^*|^2 \geq (\delta - \mu_{i,i-1}^2) |b_{i-1}^*|^2$$

for $1 < i \leq n$, where $\delta = \frac{1}{\alpha} + \frac{1}{4}$.

Definition 2.5. *A lattice basis is LLL-reduced if it is both size-reduced and 2-reduced.*

When needing the distinction, we will refer to size-reduction and 2-reduction as the first and second LLL conditions, respectively. Note that using the definition of the Euclidean norm, our notion of LLL reduction sets $\delta = \frac{3}{4}$ and can be re-written to match its form in [14] as

$$\begin{aligned} |b_i^*|^2 &\geq (\frac{3}{4} - \mu_{i,i-1}^2) |b_{i-1}^*|^2 \\ |b_i^*|^2 &\geq \frac{3}{4} |b_{i-1}^*|^2 - \mu_{i,i-1}^2 |b_{i-1}^*|^2 \\ |b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 &\geq \frac{3}{4} |b_{i-1}^*|^2. \end{aligned}$$

A stronger condition on lattice bases gives rise to an alternate notion of reduction, named for Hermite, Korkin, and Zolotarev:

Definition 2.6. *A lattice basis if HKZ-reduced if is it both size-reduced and if b_i^* is the shortest non-zero vector for the sublattice L_i.*

We will not deal with this concept directly; however, many modern approaches to hard lattice problems require HKZ-reduced lattice bases instead of LLL-reduced lattice bases, and we will make reference to these later.

With these definitions in mind, we are led to the fundamental result of the LLL algorithm.

Theorem 2.7 (Lenstra, Lenstra, & Lovasz)**.** *A lattice L with basis vectors $b_1, \ldots, b_n$ satisfying both the first and second LLL conditions (2.5) has the following properties:*

1. $\det(L) \leq \prod_{i=1}^{n} |b_i| \leq 2^{\frac{n(n-1)}{4}} \cdot \det(L)$
2. $|b_1| \leq 2^{\frac{n-1}{2}} \cdot |x|$, *for all non-zero vectors* $x \in L$

We prove each claim separately.

Proof of (1). The LHS of the first property is the inequality $\det(L) \leq \prod_{i=1}^{n} |b_i|$, which is simply the Hadamard inequality of Theorem 2.1. The RHS of this property is the inequality $\prod_{i=1}^{n} |b_i| \leq 2^{\frac{n(n-1)}{4}} \cdot \det(L)$. Recall that combining the two LLL conditions for (2.5) gives

$$\begin{aligned} |b_i^*|^2 &\geq \left(\frac{3}{4} - \left(\frac{1}{2}\right)^2\right) |b_{i-1}^*|^2 \\ &\geq \frac{1}{2}|b_{i-1}^*|^2. \end{aligned}$$

By a simple induction argument on the index i, we therefore have

$$|b_j^*|^2 \leq 2^{i-j}|b_i^*|^2 \tag{2.1.1}$$

for all $i \geq j$.

Recall too that the Gram-Schmidt process in Theorem 2.2 defines the b_i^* as

$$b_i^* = b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$$

so that

$$|b_i| = \left| b_i^* + \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \right|$$

and therefore

$$|b_i| \leq |b_i^*| + \left| \sum_{j=1}^{i-1} \mu_{i,j} b_j^* \right|$$

by the triangle inequality. This implies

$$\begin{aligned}
|b_i|^2 &\leq |b_i^*|^2 + \sum_{j=1}^{i-1} \frac{1}{4} 2^{i-j} |b_i^*|^2 \\
&= |b_i^*|^2 \left(1 + \frac{1}{4} \cdot \sum_{j=1}^{i-1} 2^j\right) \\
&= |b_i^*|^2 \left(1 + \frac{1}{4} \cdot \left[\frac{2^i - 1}{2-1} - 1\right]\right) = |b_i^*|^2 \left(1 + \frac{1}{4}\left[2^i - 2\right]\right) \\
&\leq 2^{i-1} \cdot |b_i^*|^2 \qquad (2.1.2)
\end{aligned}$$

Now since $|b_i|^2 \leq 2^{i-1} \cdot |b_i^*|^2$ and $|b_i| \leq 2^{\frac{i-1}{2}} \cdot |b_i^*|$, we can write

$$\begin{aligned}
\prod_{i=1}^{n} |b_i| &\leq \prod_{i=1}^{n} 2^{\frac{i-1}{2}} \cdot |b_i^*| \\
&= 2^{\left(\sum_{i=1}^{n} \frac{i-1}{2}\right)} \prod_{i=1}^{n} |b_i^*| \\
&= 2^{\frac{n(n-1)}{4}} \prod_{i=1}^{n} |b_i^*| \\
&= 2^{\frac{n(n-1)}{4}} \cdot \det(L)
\end{aligned}$$

which gives the desired RHS inequality. □

Proof of (2). For any non-zero vector $x \in L$, we may write

$$x = \sum_{j=1}^{i} s_j b_j = \sum_{j=1}^{i} t_j b_j^*$$

with $s_i \in \mathbb{Z},\ t_j \in \mathbb{R}$ and i being the largest index having $t_i \neq 0$. We observe that t_i and s_i must in fact be equal. To see this, consider

$$\begin{aligned}
x &= \sum_{j=1}^{i} t_j b_j^* \\
&= t_i \cdot b_i^* + t_{i-1} \cdot b_{i-1}^* + \cdots + t_1 \cdot b_1^*
\end{aligned}$$

and using the Gram-Schmidt identity of Theorem 2.2, this gives

$$\begin{aligned}
&= t_i \left(b_i - \sum_{k=1}^{i-1} \mu_{i,k} b_j^*\right) + t_{i-1}\left(\dots\right) + \dots \\
&= t_i b_i + \dots
\end{aligned}$$

where we really only care about the first term of the sum. Importantly, this coefficient t_i is the only one associated to the vector b_i in this sum, and we may easily write $x = \sum_{j=1}^{i} s_j b_j$ where s_i will be the only coefficient associated to the vector b_i. As these sums are equal, so too are the coefficients t_i and s_i.

This forces $t_i \neq 0$ to be an integer, allowing us to write

$$|x|^2 \geq t_i^2 |b_i^*|^2$$

and trivially implying $|x|^2 \geq |b_i^*|^2$ for all i. Since equation (2.1.2) in the proof of part 1 showed that $|b_j|^2 \leq 2^{i-1}|b_i^*|^2$ for $1 \leq j \leq i \leq n$, then letting $j = 1$ gives

$$\begin{aligned} |b_1|^2 &\leq 2^{i-1}|b_i^*|^2 \\ &\leq 2^{n-1}|b_i^*|^2 \quad (\text{since } i \leq n) \\ &\leq 2^{n-1}|x|^2. \end{aligned}$$

Taking the square root of both sides gives the inequality $|b_1| \leq 2^{\frac{n-1}{2}}|x|$, as desired. □

2.2 The algorithm

The LLL algorithm (Figure 1 in [14]) is an "induction"-style algorithm. The basis vectors b_i are examined in order for $i = 1$ to $i = k$, and we ensure that the LLL conditions are satisfied over this range. Then, under the assumption that the conditions are satisfied for $i \leq k$, we are able to increment k and enforce the same LLL conditions for the basis vector b_{k+1}. Beginning with $k = 2$, we terminate when $k = n$ and have satisfied the LLL conditions over all basis vectors. Operations on the basis vectors should be thought of as happening *in situ* unless otherwise specified, as their order is significant to the functioning of the algorithm.

Algorithm 1 : LLL basis reduction algorithm

```
1:  procedure LLL-REDUCE(b_1, ..., b_n)
2:      k := 2
3:      while k ≤ n do
4:          SIZE-RED(k)                          ▷ Size-reduce all basis vectors up to b_k
5:          if |b_k^*|^2 < (3/4 − μ_{k,k−1}^2) |b_{k−1}^*|^2 then   ▷ Check second LLL condition
6:              SWAP(b_k, b_{k−1})
7:              k := max(2, k − 1)               ▷ First LLL condition may be invalidated
8:          else
9:              k := k + 1
10:         end if
11:     end while
12: end procedure
```

Algorithm (1) gives the pseudocode for the LLL algorithm, with calls to the two subroutines `SIZE-RED` and `SWAP`. The subroutine `SIZE-RED`(k) is given explicitly in algorithm (2), and ensures that the basis vectors are size-reduced for all $i \leq k$. The subroutine `SWAP`(b_k, b_{k+1}) simply swaps the positions of its two input basis vectors:

$$(b_1, b_2, \ldots, b_j, b_{j+1}, \ldots, b_n)$$
$$\Big\downarrow \texttt{SWAP}(b_j, b_{j+1})$$
$$(b_1, b_2, \ldots, b_{j+1}, b_j, \ldots, b_n).$$

The progress of `LLL-REDUCE` is straightforward, except perhaps at line 7 where the loop index is potentially decremented instead of incremented. This is because after discovering the second LLL condition was not met, the algorithm swapped basis vectors b_k, b_{k+1} to compensate. However, since b_k has now been replaced with another vector, we no longer have the strong induction guarantee that all b_i are size-reduced for $i \leq k$. We must re-check the size reduction condition to re-establish the invariant.

Algorithm 2 : Size reduction algorithm

```
1: procedure SIZE-RED(k)
2:     for ℓ = (k − 1) to 1 do
3:         r := ⌈μ_{k,ℓ}⌋
4:         b_k := b_k − r · b_ℓ
5:         μ_{k,ℓ} := μ_{k,ℓ} − r
6:     end for
7: end procedure
```

We also remark that the parameter δ was chosen arbitrary for the original algorithm and in fact the only restrictions are that $\frac{1}{4} \leq \delta \leq 1$. Assuming $\delta = 1$, this would give the best possible bound of

$$\left(\frac{4}{3}\right)^{\frac{n}{2}} \tag{2.2.1}$$

on our approximation factor; however, we would then no longer have a guarantee of polynomial time termination using the same argument as given above. In the end, the selection of this "twiddle factor" does not play a very significant role, and while there is research on the topic, it is not germane to our discussion.

2.3 Correctness

Once we are convinced that both LLL conditions are met from (2.5), the correctness of the LLL algorithm is simply an appeal to theorem 2.7 which states that b_1 must be within a factor of $2^{\frac{n-1}{2}}$ of the shortest vector in L. The second LLL condition is met explicitly by the if/then statement on line 5 of `LLL-REDUCE`. However, it is not as apparent that the first LLL condition is met by the operations in `SIZE-RED`. To see this, recall the definition

$$\mu_{k,\ell} = \frac{\langle b_k, b_\ell^* \rangle}{\langle b_\ell^*, b_\ell^* \rangle}$$

from theorem 2.2. In line 4 of `SIZE-RED` we replace b_k with $b_k - r \cdot b_\ell$, so that updating the coefficient $\mu_{k,\ell}$ should give

$$\mu'_{k,\ell} = \frac{\langle b_k - r \cdot b_\ell, b_\ell^* \rangle}{\langle b_\ell^*, b_\ell^* \rangle} \;=\; \frac{\langle b_k, b_\ell^* \rangle - r \cdot \langle b_\ell^*, b_\ell \rangle}{\langle b_\ell^*, b_\ell^* \rangle}.$$

Since $\frac{\langle b_\ell^*, b_\ell \rangle}{\langle b_\ell^*, b_\ell^* \rangle} = 1$, this is just

$$\mu'_{k,\ell} = \mu_{k,\ell} - r,$$

and on line 3 we chose r so that it was at most $\frac{1}{2}$ away from $\mu_{k,\ell}$. Hence, the updated value given by line 5 satisfies $\mu'_{k,\ell} \leq \frac{1}{2}$ as desired.

Since both LLL conditions are met at the end of the algorithm, we need only show termination to know that the output will be LLL-reduced as desired.

2.4 Termination

Although the procedure `LLL-REDUCE` only consists of a single `while` loop, it is not at all obvious whether the algorithm will terminate. This is because the index variable k may either be incremented or decremented on each run through the loop. To show that it does in fact terminate, we use a potential function argument.

First, recall the definition of L_i as the sublattice of L formed from the first i basis vectors $b_1, \ldots, b_i$. Let

$$d_i = \det(L_i)^2 = \prod_{1 \leq\ j\ \leq i} |b_j^*|^2 \tag{2.4.1}$$

to be the square of the determinant of L_i, and

$$D = \prod_{i=1}^{n} d_i$$

be the product of these squared determinants for the first i sublattices. In other words,

$$D = \prod_{i=1}^{n} d_i \ = \ \prod_{i=1}^{n} \left(\prod_{j=1}^{i} |b_j^*|^2 \right).$$

We observe that the `SIZE-RED` subroutine has no effect on the value of any of the $|b_i^*|$, and it is only the calls to `SWAP` which may potentially alter these values, and hence the value of d_i. If `SWAP`$(k-1, k)$ is called and $(k-1) = i$, then the basis vectors for the lattice L_i change from being

$$\{b_1, \ldots, b_{k-2}, b_{k-1}\}$$

to being

$$\{b_1, \ldots, b_{k-2}, b_k\},$$

and the value of d_i changes accordingly. On the other hand, if $i \neq (k-1)$ then there are two possibilities, neither of which see L_i change:

- $i < (k-1)$, and so the basis vectors being swapped are not part of L_i
- $i > (k-1)$, and the lattice L_i before `SWAP` is identical to L_i afterwards, since

$$\{b_1, \ldots, b_{k-1}, b_k, \ldots, b_i\} = \{b_1, \ldots, b_k, b_{k-1}, \ldots, b_i\}$$

Having already defined D as the product of squared determinants for all sublattices L_i, let D' be the same value after a single call to `SWAP`(b_k, b_{k-1}). The ratio of these values is therefore

$$\begin{aligned}\frac{D'}{D} &= \frac{\left(\prod_{i=1}^{k-2} d_i\right) \cdot \det(L(b_1, \ldots, b_{k-2}, b_k))^2}{\left(\prod_{i=1}^{k-2} d_i\right) \cdot \det(L(b_1, \ldots, b_{k-2}, b_{k-1}))^2} \\ &= \frac{\left(\prod_{j=1}^{k-2} |b_j^*|^2\right) \cdot |b_k^*|^2}{\prod_{j=1}^{k-1} |b_j^*|^2} \\ &= \frac{|b_k^*|^2}{|b_{k-1}^*|^2}.\end{aligned}$$

Since the LLL conditions guarantee that $|\mu_{i,j}| \leq \frac{1}{2}$ for $1 \leq j < i \leq n$ and $|b_i^*|^2 \geq (\frac{3}{4} - \mu_{i,i-1}^2)|b_{i-1}^*|^2$ this implies

$$\frac{|b_i^*|^2}{|b_{i-1}^*|^2} \leq \frac{3}{4} \quad \text{and so} \quad D' \leq \frac{3}{4} \cdot D.$$

Letting $D^{(m)}$ be the value of D after m swaps, we have by induction that

$$0 \leq D^{(m)} \leq (3/4)^m \cdot D.$$

After each `SWAP` operation, the value of D is decreasing by at least a factor of $\frac{3}{4}$. From the other end, we can argue using Hermite's constant (Prop. 6.4.1 in [6]) that each d_i is bounded from below by a positive constant depending only on the sublattice L_i, so therefore D is also bounded from below by a positive constant. Since D is decreasing by a constant factor with each call to `SWAP`, but cannot decrease below a positive constant, there can only be finitely many calls to `SWAP`. And since the loop index of algorithm 1 is decremented only when `SWAP` is called, we know that the index k must reach its maximum value after finitely many iterations, thus the loop (and the algorithm) must terminate.

2.5 Complexity and analysis

In analyzing the complexity of the LLL algorithm or its later variants, we must take care on the issue of integer accuracy. In order to eliminate the need for arbitrary precision in intermediate results, a bound B is typically introduced such that $|b_i|^2 \leq B$ for all $1 \leq i \leq n$, which in turn bounds the integers in any operation to having $O(n \log B)$ bits.

There are essentially two areas of `LLL-REDUCE` in algorithm (1) where computational work is done: when `SIZE-RED` is called and when `SWAP` is called. Everything else in the procedure only contributes $O(1)$ for each call to either of these subroutines. Moreover, since the loop index k is decremented iff `SWAP` is called, the number of calls to `SIZE-RED` is equal to the number of times the index k is incremented, which can be at most $n - 1$ more times than it is decremented. Thus, if we determine the number of calls to `SWAP` we are also bounding the number of calls to `SIZE-RED` within a constant factor.

Recall the value D defined in our potential function argument above, and observe that D must be

bounded by

$$\begin{aligned} D &= \prod_{i=1}^{n} \prod_{j=1}^{i} |b_j^*|^2 \\ &\leq \prod_{i=1}^{n} \prod_{j=1}^{i} B \\ &\leq B^{\frac{n(n-1)}{2}}. \end{aligned}$$

Since D decreases by a factor of at least $\frac{3}{4}$ for each call to `SWAP`, and $D \geq 1$, there can be at most

$$\log_{\frac{3}{4}} \left(B^{\frac{n(n-1)}{2}} \right) = O\left(n^2 \log(B) \right)$$

calls to `SWAP` in total, with each call performing only $O(1)$ operations. This also gives a bound of $O(n^2 \log(B))$ calls to `SIZE-RED`, which loops over at most n vectors and updates at most n elements for each vector. Thus, the calls to `SWAP` require $O(n^2 \log(B))$ operations, and the calls to `SIZE-RED` require $O(n^4 \log(B))$ arithmetic operations. The final complexity of the LLL algorithm is therefore $O(n^4 \log(B))$, as demonstrated in both [14] and [29].

3 Improvements and variations on LLL

3.1 Schönhage, 1984 - blockwise reduction

In [27], Schönhage expanded the usual concept of 2-reduction to that of *semi-reduction*, which is when a basis $b_1, \ldots, b_n$ satisfies

$$|b_r^*|^2 \leq 2^{n+s-r} |b_s^*|^2$$

for all $1 \leq r < s \leq n$. Schönhage then made two key observations: first, that the size reduction operations (calls to `SIZE-RED`) are the most expensive part of the LLL algorithm, and second, that each 2-reduction operation (calls to `SWAP`) necessitated an extra size-reduction operation. These are the "backtrack" operations made after calls to `SWAP`, which are needed to restore the size reduction invariant $|\mu_{i,j}| \leq \frac{1}{2}$.

To limit the amount of backtrack needed, Schönhage opted to 2-reduce *blocks* of basis vectors together, instead of using `SWAP` to 2-reduce only a pair of basis vectors at a time. Blockwise 2-reducing a set of basis vectors incurs more overhead than performing a simple `SWAP`. However, Schönhage also observed that the choice of always calling `SWAP` on the non-2-reduced vector of smallest index b_k was arbitrary, and in fact other choices of index would be valid. The concept of semi-reduction was then used to make clever choices as to which blocks of basis vectors will be 2-reduced on each pass of the algorithm, and this is enough to minimize the overhead costs. The final result is an improvement to $O(n^3 \log(B))$ complexity when the basis vectors are bounded by B.

3.2 Schnorr, 1987 - "Schnorr's algorithm"

In [24], Schnorr proposed a semi block-$2k$ reduction algorithm which was one of the first serious deviations from the basic LLL framework. Even more importantly, it provided a significant reduction in the

approximation factor, allowing shortest basis vectors to be found while still guaranteeing polynomial time. Improvements to this approach along with modern summaries of the algorithm can be found in

Let

$$\pi_i : \mathbb{R}^n \to \sum_{j \geq i} b_j^* \mathbb{R}$$
$$\pi_i(b_i) \mapsto b_i^*$$

denote the mapping from an n-dimensional lattice vector onto the subspace of $\mathbb{R}^n$ orthogonal to $b_1, \ldots, b_{i-1}$. Where the LLL algorithm size-reduced all basis vectors b_i for $1 \leq n$, the block-$2k$ approach instead divides these basis vectors into n/k blocks of k basis vectors

$$\{\pi_{k(i-1)+1}(b_{k(i-1)+1}), \ldots, \pi_{k(i-1)+1}(b_{ki})\}$$

with $1 \leq i \leq \frac{n}{k}$, and treats each block as corresponding to a rank k lattice L_i^k with the same index $1 \leq j \leq \frac{n}{k}$. It also divides the basis vectors into $(n/k - 1)$ blocks of $2k$ vectors

$$\{\pi_{k(i-1)+1}(b_{k(i-1)+1}), \ldots, \pi_{k(i-1)+1}(b_{ki+k})\}$$

with $1 \leq i \leq \frac{n}{k} - 1$, and treats each block as corresponding to a rank $2k$ lattice L_i^{2k}.

Looping over i, each L_i^k is then HKZ-reduced and size-reduced. Next, a step is taken analogous to the call to `SWAP` in our version of `LLL-REDUCE`, algorithm 1: we first find the minimum j such that $|b_{jk}^*|^2 > 2|b_{jk+1}^*|^2$, and call `SWAP`$(b_{jk}^*, b_{jk+1}^*)$. We then apply HKZ reduction to the two rank k lattices L_{j-1}^k, L_j^k which contains basis vectors affected by this swap. Finally, we check the inequality

$$\det(L_i^k) > \frac{4}{3}\beta_k^k \det(L_{i+1}^k) \tag{3.2.1}$$

where β_k is defined as

$$\sup_i \left\{ \left(\frac{\det(L_i^k)}{\det(L_{i+1}^k)} \right)^{\frac{1}{k}} \right\}.$$

Exactly where this β_k constant comes from is not important for us, although we will use it again momentarily to show that Schnorr's algorithm returns a desirable result. If inequality (3.2.1) is not satisfied, we apply HKZ reduction to the rank $2k$ lattice L_{j-1}^{2k}, otherwise we exit the loop.

Following this process guarantees that each sublattice L_i^k is HKZ reduced, and because of our operations on the sublattices L_j^{2k} we are also guaranteed that

$$\left(\frac{\det(L_i^k)}{\det(L_{i+1}^k)} \right)^{\frac{1}{k}} \leq (1+\epsilon)\beta_k.$$

This, along with the upper bound $\beta_k \leq 4k^2$, leads to the approximation factor

$$\frac{|b_1|}{\min\{x \in L\}} \leq (4k^2)^{\frac{n}{2k}}.$$

This is a significant improvement on the best possible upper bound given for the LLL algorithm in equation (2.2.1), and Schnorr proved it to run in

$$O(n^4 \log(b) + b^{\frac{k}{2}+o(k)} b^2 \log(B))$$

arithmetic operations on $O(n \log(B))$-sized integers, which is still polynomial in the dimension, and essentially degrades to the complexity of the original LLL when $k = 1$

3.3 Schnorr, 1988 - deep insertions

In [25], Schnorr focused on improving the LLL approach to integer accuracy. Performing the Gram-Schmidt orthogonalization process to compute the coefficients $\mu_{i,j}$ potentially requires $O(n \log B)$ bits for each integer, whether an intermediate or final value. It is possible to use floating point arithmetic to avoid the integer arithmetic on such large values; however, it would be necessary to maintain a minimum level of integer accuracy for this variation to have the same guarantee on its output as the original LLL algorithm.

Schnorr's contribution was to propose a self-correction method that allowed for approximate rational arithmetic instead of floating point arithmetic, while still keeping the approximation error in check. This modified algorithm still used $O(n^4 \log(B))$ arithmetic operations; however, the arithmetic used integers bounded by $O(n+\log(B))$ instead of $O(n \log(B))$. Combining this approach with the semi-reduction methods of Schönhage, Schnorr was also able to reduce the number of arithmetic operations to $O(n^{3.5} \log(B))$ on integers of the same bound.

Schnorr later proposed another popular variant along with Euchnerr [26], which involved so-called "deep insertions". This modified the 2-reduction behaviour of $\texttt{SWAP}(b_k, b_{k+1})$, allowing the subroutine to move the vector b_{k+1} farther back (or "deeper") into the list of previously processed basis vectors. Specifically, the modified `SWAP` would find a minimal index $i < k$ which still decreased $|b_i^*|^2$ by the factor $\delta = \frac{1}{\alpha} + \frac{1}{4}$ (recall this is $\delta = \frac{3}{4}$ in the case of LLL). The basis vector b_k would be moved to position i, and the position of all basis vectors $b_i, \ldots, b_{k-1}$ would be increased by one. Also present in this work was another block-wise HKZ reduction algorithm or "BKZ" algorithm (distinct from the method of section 3.2), which reduces lattice bases using the stronger notion of HKZ reduction (definition 2.6) over LLL reduction. In practice, this produces much shorter vectors than the original LLL algorithm, but at the cost of guaranteed polynomial time complexity: in the worst case, the deep insertion/BKZ algorithm may take super-exponential time.

3.4 Storjohann, 1996

In [29], Storjohann re-cast the LLL algorithm's operations in terms of matrix arithmetic. Using the method of LU decomposition from linear algebra, we may uniquely factor $U = LB$ where B is the generating matrix (1.2.2) of a lattice, U is upper triangular, and L is lower triangular. In this case, the matrix U can be uniquely written as

$$U = \begin{bmatrix} d_1 & d_1\mu_{2,1} & d_1\mu_{3,1} & \ldots & d_1\mu_{n,1} \\ & d_2 & d_2\mu_{3,2} & \ldots & d_2\mu_{n,2} \\ & & d_3 & \ldots & d_3\mu_{n,3} \\ & & & \ddots & \vdots \\ & & & & d_n \end{bmatrix}$$

where the $\mu_{i,j}$ are as in theorem 2.2, and the d_i are as in equation (2.4.1) of the LLL potential function argument. Both U and the generating matrix B are maintained throughout the algorirhtm, where B initially contains the n basis vectors as its rows. Whenever `SWAP` or `SIZE-RED` are called, the rows of B are updated as usual, and then row operations are used to update U in parallel. Size-reducing the basis vector b_k involves operating only on the k-th row in A and the k-th column in U. Performing the 2-reduction step on basis vectors b_k, b_{k-1} involves interchanging the k-th and $(k-1)$-th rows of A, and a slightly more complicated set of operations on U:

- set $U_{k,j} := \dfrac{1}{d_{k-1}} \left(d_{k-2} \cdot U_{k,j} + d_{k-1}\mu_{k,k-1} \cdot U_{k-1,j} \right)$
- interchange row k with row $k-1$
- interchange column k with column $k-1$
- set $U_{k,j} := \dfrac{1}{d_{k-2}} \left(d_{k-1} \cdot U_{k,j} - d_{k-1}\mu_{k,k-1} \cdot U_{k-1,j} \right)$

In the following expanded version of the matrix U adapted from [29], we can clearly see that the above operations only affect matrix elements in three distinct regions.

$$U = \left[\begin{array}{ccc||cc||ccc} d_1 & \dots & d_1\mu_{k-2,1} & d_1\mu_{k-1,1} & d_1\mu_{k,1} & d_1\mu_{k+1,1} & \dots & d_1\mu_{n,1} \\ & \ddots & \vdots & \vdots & \vdots & \vdots & & \vdots \\ & & d_{k-2} & d_{k-2}\mu_{k-1,k-2} & d_{k-2}\mu_{k,k-2} & d_{k-2}\mu_{k+1,k-2} & \dots & d_{k-2}\mu_{n,k-2} \\ \hline\hline & & & d_{k-1} & d_{k-1}\mu_{k,k-1} & d_{k-1}\mu_{k+1,k-1} & \dots & d_{k-1}\mu_{n,k-1} \\ & & & & d_k & d_k\mu_{k+1,k} & \dots & d_k\mu_{n,k} \\ \hline\hline & & & & & d_{k+1} & \dots & d_{k+1}\mu_{n,k+1} \\ & & & & & & \ddots & \vdots \\ & & & & & & & d_n \end{array}\right]$$

Specifically, having divided the matrix into nine regions with these double lines, if we number the regions 1-9 in a left-to-right, top-to-bottom fashion, then only the elements contained in regions 2, 5, and 6 need to be altered.

The key observation for Storjohann's improvement was to then notice that the loop within `SIZE-RED` was unnecessary for the final output of the algorithm to remain correct. In fact, if we are willing to temporarily remove our size-reduction invariant, then each call to `SIZE-RED` only requires a single iteration of its loop to maintain 2-reduction of the basis. This lowers the overall workload of LLL to $O(n^2 \log B)$ arithmetic operations, at the cost of no longer having a size-reduced basis given as output. However, at the end of the algorithm we can restore the size-reduction invariant in a single pass over all basis vectors, once they are already 2-reduced. This final step adds some overhead, but it can be limited by using the above matrix representations. The final result is an LLL variant which has the same bounds on the size of its minimal output vector, but only requires $O(n^3 \log B)$ arithmetic operations, using integers with $O(n \log B)$ bits as usual.

4 Summary and conclusions

Moving beyond initial improvements

The variations proposed by Schnorr [24] are still regarded as perhaps the best version of the LLL algorithm, with Schnorr and Euchnerr's BKZ variant [26] next preferred. Despite the possibility for super-exponential worst case running time, the BKZ algorithm and its successor variants are preferred as a matter of practice, and have been experimentally verified to perform better in some instances [10].

Perhaps surprisingly, the various flavours of the LLL algorithm have remained competitive since the original algorithm was proposed in 1982. The recently proposed variant of LLL called L^2 was significant [23], as it was the first algorithm to offer LLL reduction with complexity that only grows quadratically with respect to the bounded bit size $\log B$. Yet the overall complexity was still essentially $O(n^5 \log^2(B))$, which is not too far from the original bound.

Indeed, the only polynomial time algorithms which have provably better approximation bounds are the blockwise HKZ algorithm of Schnorr [24] (later refined in [8]), and a new blockwise reduction approach presented in [9]. And although these differ sufficiently from the LLL approach to be classified as a new category, and indeed are sufficiently complicated to warrant more background than we are able to provide here, there is no denying the powerful influence of the LLL approach still visible in these modern variations. Clever basis reduction as a means to producing a "somewhat short" vector remains the core of the approach.

A significant problem arises with comparing SVP approximation approaches, in the form of distinguishing between an algorithm being "asymptotically faster" versus "faster in practice". This limits any definitive conclusions on which approach is used more often, or more successfully. The curse of dimensionality also plays a role here; a problem may become intractable because of its dimension before we begin to see asymptotic convergence to a nicer upper bound on complexity. For this reason, the literature often speaks of algorithms performing poorly or well in high dimension versus in low dimension. Rather than give a conclusive recommendation, we have attempted to outline several important variations which each provide significant asymptotic improvements in the overall operating complexity of the LLL algorithm. However, these may not be the preferred methods in practice.

References

[1] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108. ACM, 1996.

[2] M. Ajtai and C. Dwork. A public-key cryptosystem with worst-case/average-case equivalence. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293. ACM, 1997.

[3] M. Ajtai, R. Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 601–610. ACM, 2001.

[4] J. Blömer. Closest vectors, successive minima, and dual HKZ-bases of lattices. *Automata, Languages and Programming*, pages 248–259, 2000.

[5] P.B. Borwein. *Computational excursions in analysis and number theory*. Springer Verlag, 2002.

[6] H. Cohen. *A course in computational algebraic number theory*. Springer Verlag, 1993.

[7] J.H. Conway, N.J.A. Sloane, and E. Bannai. *Sphere packings, lattices, and groups*. Springer Verlag, 1999.

[8] N. Gama, N. Howgrave-Graham, H. Koy, and P. Nguyen. Rankins constant and blockwise lattice reduction. *Advances in Cryptology-CRYPTO 2006*, pages 112–130, 2006.

[9] N. Gama and P.Q. Nguyen. Finding short lattice vectors within mordell's inequality. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 207–216. ACM, 2008.

[10] N. Gama and P.Q. Nguyen. Predicting lattice reduction. In *Proceedings of the theory and applications of cryptographic techniques 27th annual international conference on Advances in cryptology*, pages 31–51. Springer-Verlag, 2008.

[11] G. Hanrot. LLL: a tool for effective diophantine approximation. In P.Q. Nguyen and B. Vallée, editors, *The LLL Algorithm*, pages 215–263. Springer, 2010.

[12] A. Joux and J. Stern. Lattice reduction: A toolbox for the cryptanalyst. *Journal of Cryptology*, 11(3):161–185, 1998.

[13] R. Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of operations research*, pages 415–440, 1987.

[14] A.K. Lenstra, H.W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[15] A. May. Using LLL-reduction for solving RSA and factorization problems. In P.Q. Nguyen and B. Vallée, editors, *The LLL Algorithm*, pages 315–348. Springer, 2010.

[16] D. MICCIANCIO. The shortest vector in a lattice is hard to approximate to within some constant. *SIAM journal on computing*, 30(6):2008–2035, 2001.

[17] D. Micciancio. Efficient reductions among lattice problems. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 84–93. Society for Industrial and Applied Mathematics, 2008.

[18] D. Micciancio. Cryptographic functions from worst-case complexity assumptions. In P.Q. Nguyen and B. Vallée, editors, *The LLL Algorithm*, pages 427–452. Springer, 2010.

[19] D. Micciancio. Lattice algorithms and applications. University Lecture Notes, April 2011.

[20] D. Micciancio and O. Regev. Lattice-based cryptography. *Post-quantum cryptography*, pages 147–191, 2009.

[21] D. Micciancio and P. Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *Proceedings of the 42nd ACM symposium on Theory of computing*, pages 351–358. ACM, 2010.

[22] P. Nguyen and J. Stern. The two faces of lattices in cryptology. *Cryptography and Lattices*, pages 146–180, 2001.

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

[24] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical computer science*, 53(2-3):201–224, 1987.

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

[26] C.P. Schnorr and M. Euchner. Lattice basis reduction: improved practical algorithms and solving subset sum problems. *Mathematical programming*, 66(1):181–199, 1994.

[27] A. Schönhage. Factorization of univariate integer polynomials by Diophantine approximation and an improved basis reduction algorithm. *Automata, languages and programming*, pages 436–447, 1984.

[28] D. Simon. Selected applications of LLL in number theory. In P.Q. Nguyen and B. Vallée, editors, *The LLL Algorithm*, pages 265–282. Springer, 2010.

[29] A. Storjohann. Faster algorithms for integer lattice basis reduction. Technical Report 249, ETH Zurich, Dept. Comp. Sc., July 1996.