Methods to Solve the Shortest Vector Problem

Summer Project 2018

Rebecca Carter

Contents

1 Introduction

The Shortest Vector Problem (SVP) is one of the most widely studied lattice problems, with key applications in the study of lattice based cryptosystems.

SVP is the problem of finding the shortest non-zero vector in a lattice $\mathcal{L}$. There are a variety of different forms of SVP, which split into exact forms, and approximate forms. In exact-SVP aim to find a non-zero vector $\mathbf{v}$ in $\mathcal{L}$ of minimal length, or equivalently the length of such a vector. There is also great interest in approximate solutions to SVP, that is finding non-zero lattice vectors which have norm at most $\alpha \cdot \det(\mathcal{L})^{1/n}$ (Hermite-SVP, $\alpha > 0$), or at most $\alpha \cdot \lambda_1(\mathcal{L})$ (Approx-SVP, $\alpha \geq 1$), for some approximation factor α.

Correspondingly there are two different types of algorithms for solving SVP: exact algorithms and approximate algorithms. Exact algorithms are expensive, with running times at least exponential in the dimension, but find an exact shortest lattice vector. In low dimensions this is typically done using an exhaustive method such as enumeration, but high running times rule out this approach beyond dimension 100. Thus in higher dimensions approximate algorithms are used, which produce so-called 'reduced' lattice bases $\mathbf{b_1}, \ldots, \mathbf{b_n}$ for the lattice. The first algorithm of this type was the LLL algorithm, from which faster and stronger methods have been developed.

The two types of algorithm are invariably used together, with exact algorithms using approximate ones to pre-process the input basis, and block-wise approximate algorithms repeatedly calling an exact algorithm in a low dimension as a subroutine.

2 Preliminaries

2.1 Lattice Basics

Definition 2.1. [1]A *lattice* $\mathcal{L}$ is a discrete subgroup of $\mathbb{R}^n$, generated by integer combinations of the vectors of some basis $B = \mathbf{b_1}, \ldots, \mathbf{b_k}$:

$$\mathcal{L} = \mathcal{L}(B) \coloneqq \left\{ \sum_{i=1}^{k} z_i \mathbf{b_i} : z_i \in \mathbb{Z} \right\}$$

Remark. The integer k is known as the *rank*, or dimension, of the lattice $\mathcal{L}$; if $k = n$ then $\mathcal{L}$ is said to be of *full rank*.

A key concept in the study of lattices is that of a fundamental region, the area of which is an essential lattice invariant, known as its *volume* or *determinant*, $\det(\mathcal{L})$.

Definition 2.2. A set $\mathcal{F} \subset \mathbb{R}^n$ is a *fundamental region* of a lattice $\mathcal{L}$ if its translates $x + \mathcal{F} = \{x + y : y \in \mathcal{F}\}$, taken over all x in $\mathcal{L}$, form a partition of $\mathbb{R}^n$.

For any basis B of $\mathcal{L}$ there is a natural fundamental region, known as the *fundamental paralleliped*:

$$\mathcal{P}(B) := B \cdot \left[-\frac{1}{2}, \frac{1}{2} \right)^n = \left\{ \sum_{i=1}^{n} c_i \mathbf{b_i} : c_i \in \left[-\frac{1}{2}, \frac{1}{2} \right) \right\}$$

It is clear from this that for any lattice $\mathcal{L}$ and any basis B, $\det(\mathcal{L}) = |\det(B)|$.

[1]For simplicity consider only integer lattices $\mathcal{L} \subset \mathbb{Z}^n$

2.2 The Shortest Vector Problem

The Shortest Vector Problem involves finding the shortest non-zero vector in a lattice, or an approximation to it, which will always exist, since lattices are discrete. The length of this is known as the *minimum distance* of the lattice, and denoted $\lambda_1(\mathcal{L})$.

In 1845 Hermite [11] proved that

$$\frac{\lambda_1(\mathcal{L})}{\det(\mathcal{L})^{1/n}} \leq (4/3)^{(n-1)/4}$$

which led to the definition of Hermite's constants

$$\gamma_n := \sup \left(\frac{\lambda_1(\mathcal{L})}{\det(\mathcal{L})^{1/n}} \right)^2$$

where the supremeum is taken over all n-dimensional lattices. The exact value of γ_n is known only for $1 \leq n \leq 8$ and $n = 24$, but it is known that γ_n grows linearly in n [6], which guarantees the existence of very short lattice vectors.

Minkowski's Convex Body Theorem gives another upper bound on the minimum distance of any lattice, and although this bound can be arbitrarily loose it is known in general that the bound cannot be improved beyond small constant factors.

Theorem 2.3 (Minkowski's First Theorem)**.** *For any lattice* $\mathcal{L}$

$$\lambda_1(\mathcal{L}) \leq \sqrt{n} \cdot \det(\mathcal{L})^{1/n}$$

The problem can also be considered heuristically, asking how long the shortest vector of a 'random' lattice $\mathcal{L}$ is expected to be. For a large radius R would expect $(\mathrm{Vol}(B_R))/(\det(\mathcal{L}))$ copies of a fundamental region $\mathcal{F}$ of $\mathcal{L}$ to fit into an n-dimensional ball B_R of radius R. This gives

Heuristic 2.4 (Gaussian Heuristic)**.** The shortest non-zero vector in a 'random' lattice $\mathcal{L} \subset \mathbb{R}^n$ has length approximately

$$\lambda_1(\mathcal{L}) = \min_{\mathbf{v} \in \mathcal{L}, \mathbf{v} \neq 0} ||\mathbf{v}|| \approx \sqrt{\frac{n}{2\pi e}} \det(\mathcal{L})^{1/n}$$

2.3 Gram-Schmidt Orthogonalisation

Gram-Schmidt Orthogonalisation is a method of orthogonalising basis vectors, which is used as a starting point in most methods for solving SVP.

For linearly independent vectors $\mathbf{b_1} \ldots \mathbf{b_k} \in \mathbb{R}^n$ the Gram-Schmidt orthogonalised vectors are defined recursively by:

$$\mathbf{b_j^*} = \mathbf{b_j} - \sum_{i<j} \mu_{j,i} \mathbf{b_i^*} \qquad \text{with } \mu_{j,i} = \frac{\langle \mathbf{b_j}, \mathbf{b_i^*} \rangle}{||\mathbf{b_i^*}||}$$

Even though the Gram-Schmidt vectors are not a basis for $\mathcal{L}(B)$, they do provide useful information about the lattice, and its minimum distance $\lambda_1(\mathcal{L})$:

Lemma 2.1. *For any lattice* $\mathcal{L}$ *with basis* $\mathbf{b_1}, \ldots, \mathbf{b_n}$*,*

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

$$\lambda_1 \geq \min_i ||\mathbf{b_i^*}||$$

Combined with Minkowski's First Theorem, this gives a bound on the minimum distance of a lattice:

Lemma 2.2. *For any lattice $\mathcal{L}(B)$ with Gram-Schmidt vectors $\mathbf{b_i^*}$*

$$\min_i ||\mathbf{b_i^*}|| \leq \lambda_1(\mathcal{L}(B)) \leq \sqrt{n} \left(\prod_{i=1}^{n} ||\mathbf{b_i^*}|| \right)^{\frac{1}{n}}$$

2.4 Lattice Reduction

While it is clear, even in two dimensions, that a lattice is unlikely to have an orthogonal basis, approximate algorithms for SVP generally aim to find a basis which is nearly orthogonal, in some precise way. Such bases are referred to as being *reduced*, and there are many specific types of reduced basis. The intuition here is that a basis of this type would have short, and nearly orthogonal, vectors, and so it is more likely that one of these basis vectors is a shortest lattice vector, or at least is close to being so.

The aim is for such a basis to have a small *approximation factor*, $||\mathbf{b_1}||/\lambda_1(\mathcal{L})$, or small *Hermite factor*, $||\mathbf{b_1}||/\det(\mathcal{L})^{1/n}$.

3 LLL

The LLL algorithm was introduced by Lenstra, Lenstra and Lovász [12], and was the first polynomial time algorithm for solving approximate-SVP. The LLL algorithm is based on finding a so-called *LLL-reduced* basis for the lattice, which has certain properties - keyly that its first vector is an approximation to SVP within a factor of $2^{\frac{n-1}{2}}$.

Definition 3.1. A lattice basis is *LLL-reduced* with factor δ if the following two conditions hold:

$$|\mu_{i,j}| \leq \frac{1}{2} \qquad 1 \leq j < i \leq n \tag{1}$$

$$||\mathbf{b_i^*} + \mu_{i,i-1}\mathbf{b_{i-1}^*}||^2 \geq \delta ||\mathbf{b_{i-1}^*}||^2 \qquad 1 < i \leq n \tag{2}$$

for some $\frac{1}{4} < \delta < 1$, and $\mathbf{b_i^*}$ and $\mu_{i,j}$ the result of Gram Schmidt orthogonalization on the basis.

Remark. A basis satisfying condition (1) is said to be *size-reduced*, and condition (2) is referred to as the *Lovász condition*.

In a size-reduced basis the projection, $\mathbf{b_i} - \mathbf{b_i^*}$, of $\mathbf{b_i}$ over the vector space spanned by $\{\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}\}$ lies within the fundamental paralleliped of the sublattice $\mathcal{L}(\{\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}\})$:

$$\mathcal{P} = \left\{ \sum_{j=1}^{i-1} z_j \mathbf{b_j} : z_j \in \left[-\frac{1}{2}, \frac{1}{2} \right] \right\}$$

This ensures that $\mathbf{b_i}$ is close to orthogonal to the linear span of $\{\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}\}$. To see this,

$$\mu_{i,j} = \frac{\langle \mathbf{b_i}, \mathbf{b_j^*} \rangle}{||\mathbf{b_j^*}||} = \frac{||\mathbf{b_i}||\,||\mathbf{b_j^*}||\cos(\theta)}{||\mathbf{b_j^*}||^2}$$

$$\implies |\cos(\theta)| \leq \frac{||\mathbf{b_j^*}||}{2||\mathbf{b_i}||}$$

where θ is the angle between $\mathbf{b_i}$ and $\mathbf{b_j^*}$. Thus size reducing the basis reduces $|\cos(\theta)|$, and so θ is closer to $\pm\pi/2$. Note that size reduction of a basis does not affect it's Gram-Schmidt orthogonalisation.

Figure 1 demonstrates size reduction for a 2 dimensional example, where $\mathbf{b_1}, \mathbf{b_2}$ are the original basis vectors, and $\mathbf{b_2'}$ the result of size reduction.

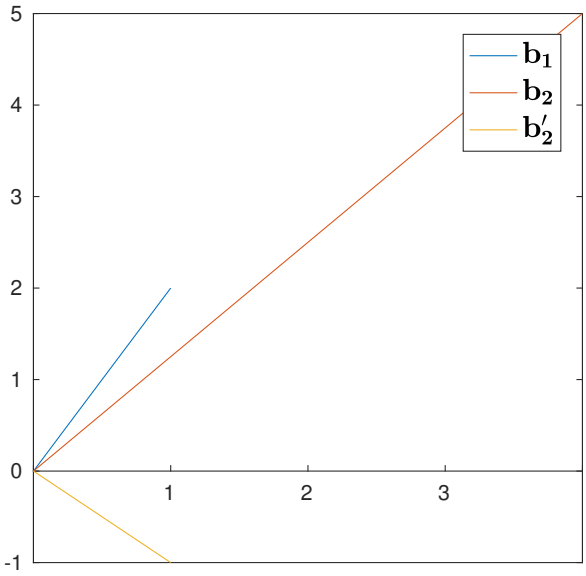

Figure 1: Size Reduction in 2 dimensions.

A method for size-reducing a basis $\mathbf{b_1}, \ldots, \mathbf{b_k}$ is given in Algorithm 1.

Algorithm 1 Size-Reduction

Compute Gram-Schmidt coefficients $\mu_{i,j}$
for $i = 2$ to k **do**
  for $j = i - 1$ to 1 **do**
    $\mathbf{b_i} \leftarrow \mathbf{b_j} - \lfloor \mu_{i,j} \rceil \mathbf{b_j}$ [2]
    for $k = 1$ to j **do**
      $\mu_{i,k} \leftarrow \mu_{i,k} - \lfloor \mu_{i,j} \rceil \mu_{j,k}$

As seen in Section 2 the Gram-Schmidt orthogonalised vectors of a basis provide an approximation to $\lambda_1(\mathcal{L})$, so to get closer to a shortest vector, aim to reduce the norms of these vectors. Since $\prod_i ||\mathbf{b_i^*}|| = \det(\mathcal{L})$ is constant, in order to overall reduce the norms of the vectors, it is necessary to make them balanced and prevent them decreasing too fast. The Gram-Schmidt orthogonalisation process is based on an ordered basis, so exchanging two vectors affects the resulting orthogonalised basis. In particular when $\mathbf{b_i}$ and $\mathbf{b_{i+1}}$ are exchanged the only Gram-Schmidt vectors to change are $\mathbf{b_i^*}$ and $\mathbf{b_{i+1}^*}$, with $\mathbf{b_i^*}$ becoming $\mathbf{b_{i+1}^*} + \mu_{i+1,i}\mathbf{b_i^*}$. Thus when the Lovász condition (2) holds, exchanging $\mathbf{b_i}$ and $\mathbf{b_{i+1}}$ only reduces $||\mathbf{b_i^*}||$ by a factor of δ, and overall the norms of the GS-vectors do not decrease 'too quickly'.

As in [12], from now on δ is taken to be $\frac{3}{4}$.

Lemma 3.1. *In an LLL-reduced basis B,* $||\mathbf{b_{i+1}^*}||^2 \geq \frac{1}{2}||\mathbf{b_i^*}||^2$ *for all* $1 \leq i < n$.

[2] $\lfloor x \rceil$ denotes the nearest integer to x

Proof. Since the Gram-Schmidt vectors are mutually orthogonal,

$$\begin{aligned}\frac{3}{4}||\mathbf{b_i^*}||^2 &\leq ||\mu_{i+1,i}\mathbf{b_i^*} + \mathbf{b_{i+1}^*}||^2 \\ &= \mu_{i+1,i}^2\,||\mathbf{b_i^*}||^2 + ||\mathbf{b_{i+1}^*}||^2 \\ &\leq \frac{1}{4}||\mathbf{b_i^*}||^2 + ||\mathbf{b_{i+1}^*}||^2 \\ \implies ||\mathbf{b_{i+1}^*}||^2 &\geq \frac{1}{2}||\mathbf{b_i^*}||^2\end{aligned}$$

□

It then follows that the first vector in an LLL-reduced basis approximates a shortest lattice vector.

Proposition 3.1. *Let* $\{\mathbf{b_1}, \ldots, \mathbf{b_n}\}$ *be an LLL-reduced basis for* $\mathcal{L}$*, then the following hold:*

$$\begin{aligned}&||\mathbf{b_j}||^2 \leq 2^{i-1}||\mathbf{b_i^*}||^2 && \text{for } 1 \leq j \leq i \leq n \\ &||\mathbf{b_1}|| \leq 2^{(n-1)/2}\lambda_1(\mathcal{L}(B))\end{aligned}$$

Proof. Since the $\mathbf{b_i^*}$ are orthogonal,

$$\begin{aligned}\frac{3}{4}||\mathbf{b_{i-1}^*}||^2 \leq ||\mathbf{b_i^*} + \mu_{i,i-1}\mathbf{b_{i-1}^*}||^2 &= ||\mathbf{b_i^*}||^2 + \mu_{i,i-1}^2||\mathbf{b_{i-1}^*}||^2 \\ \implies ||\mathbf{b_i^*}||^2 \geq \left(\frac{3}{4} - \mu_{i,i-1}^2\right)||\mathbf{b_{i-1}^*}||^2 &\geq \frac{1}{2}||\mathbf{b_{i-1}^*}||^2\end{aligned}$$

So by induction

$$||\mathbf{b_j^*}||^2 \leq 2^{i-j}||\mathbf{b_i^*}||^2 \quad \text{for } 1 \leq j \leq i \leq n$$

Using (1) gives

$$\begin{aligned}||\mathbf{b_i}||^2 &= ||\mathbf{b_i^*}||^2 + \sum_{j=1}^{i-1}\mu_{i,j}^2||\mathbf{b_j^*}||^2 \\ &\leq ||\mathbf{b_i^*}||^2 + \sum_{j=1}^{i-1}\frac{1}{4}||\mathbf{b_j^*}||^2 \\ &\leq ||\mathbf{b_i^*}||^2 + \sum_{j=1}^{i-1}\frac{1}{4}2^{i-j}||\mathbf{b_i^*}||^2 \\ &= \left(1 + \frac{1}{4}(2^i - 2)\right)||\mathbf{b_i^*}||^2 \\ &\leq 2^{i-1}||\mathbf{b_i^*}||^2\end{aligned}$$

And so $||\mathbf{b_j}||^2 \leq 2^{j-1}||\mathbf{b_j^*}||^2 \leq 2^{i-1}||\mathbf{b_i^*}||^2 \quad 1 \leq j \leq i \leq n.$
Now it follows that

$$||\mathbf{b_1}|| = ||\mathbf{b_1^*}|| \leq 2^{(i-1)/2}||\mathbf{b_i^*}|| \leq 2^{(n-1)/2}||\mathbf{b_i^*}|| \text{ for all } 1 \leq i \leq n$$

Thus by Lemma 2.2

$$||\mathbf{b_1}|| \leq 2^{(n-1)/2}\min_i ||\mathbf{b_i^*}|| \leq 2^{(n-1)/2}\lambda_1(\mathcal{L}(B))$$

□

Algorithm 2 LLL reduction

Input: $\mathbf{b_1}, \ldots, \mathbf{b_n}$ a basis for lattice $\mathcal{L}$.

Output: An LLL reduced basis $\mathbf{b_1}, \ldots, \mathbf{b_n}$ for $\mathcal{L}$

1: Apply Gram-Schmidt Orthogonalisation to find $\mathbf{b_i^*}\,(1 \leq i \leq n)$ and $\mu_{i,j}(1 \leq j < i \leq n)$
2: Let $k = 2$
3: **while** $k < n + 1$ **do**
4: **if** $k > 1$ **then**
5: **if** $|\mu_{k,k-1}| > 1/2$ **then**
6: $r \leftarrow \lfloor \mu_{k,k-1} \rceil$
7: $\mathbf{b_k} \leftarrow \mathbf{b_k} - r\mathbf{b_{k-1}}$
8: **for** $j = 1, \ldots, k-2$ **do**
9: $\mu_{k,j} \leftarrow \mu_{k,j} - r\mu_{k-1,j}$
10: $\mu_{k,k-1} \leftarrow \mu_{k,k-1} - r$
11: **if** $k \geq 2$ **and** $||\mathbf{b_k^*} + \mu_{k,k-1}\mathbf{b_{k-1}^*}||^2 < \frac{3}{4}||\mathbf{b_{k-1}^*}||^2$ **then**
12: Swap $\mathbf{b_k}$ and $\mathbf{b_{k-1}}$
13: Update $\mathbf{b_i^*}$ and $\mu_{i,j}$
14: $k \leftarrow k - 1$
15: **else**
16: **for** $l = k-2, \ldots, 1$ **do**
17: **if** $|\mu_{k,l}| > 1/2$ **then**
18: $r \leftarrow \lfloor \mu_{k,l} \rceil$
19: $\mathbf{b_k} \leftarrow \mathbf{b_k} - r\mathbf{b_l}$
20: **for** $j = 1, \ldots, l-1$ **do**
21: $\mu_{k,j} \leftarrow \mu k, j - r\mu_{l,j}$
22: $\mu_{k,l} \leftarrow \mu_{k,l} - r$
23: $k \leftarrow k + 1$

So the first vector in an LLL reduced basis is a solution to approx-SVP with factor $2^{(n-1)/2}$. In practice LLL finds much shorter vectors than guaranteed by this bound. See Section 7.

The LLL algorithm, introduced in [12], takes any basis for a lattice $\mathcal{L}$ and produces an LLL-reduced basis for the lattice.

The if loop begun on line 4 is simply ensuring that $|\mu_{k,k-1}| \leq 1/2$, and the rest of the size reduction is carried out within the else begun on line 15.

The if loop on line 11 is entered only if the Lovász condition fails at k, in which case we switch $\mathbf{b_i}$ and $\mathbf{b_{i-1}}$. This, in particular, causes $\mathbf{b_{k-1}^*}$ to be replaced by $\mathbf{b_k^*} + \mu_{k,k-1}\mathbf{b_{k-1}^*}$,so the Lovász condition now holds for k. To have reached this point must already have passed into the else section on line 15 for all $i < k$, and so the Lovász condition holds for all $i < k$ when the if loop is entered. Thus when this section of the algorithm has been completed, the Lovász condition holds at $1, \ldots, k-2, k$. It may no longer hold for $k-1$, so the value of k is reduced.

It follows that when the else loop on line 15 is entered, the Lovász condition is satisfied for all $i < k$, and so when k is incremented on line 23, the basis $\mathbf{b_1}, \ldots, \mathbf{b_k}$ is LLL-reduced.

So, as the algorithm only terminates when $k = n + 1$, it follows that it produces an LLL-reduced basis.

In order to see that the algorithm does eventually terminate it is necessary to introduce the concept of the potential of a basis:

Definition 3.2. For a basis $B = (\mathbf{b_1}, \ldots, \mathbf{b_n})$, let $\mathcal{L}_i$ be the sublattice generated by $\mathbf{b_1}, \ldots, \mathbf{b_i}$

for $1 \leq i \leq n$. The *potential* of the basis is:

$$\Phi(B) := \prod_{i=1}^{n} \det(\mathcal{L}_i) = \prod_{i=1}^{n} (||\mathbf{b_i^*}||, \ldots, ||\mathbf{b_i^*}||) = \prod_{i=1}^{n} ||\mathbf{b_i^*}||^{n-i+1}$$

$\Phi(B)$ changes only when some $\mathbf{b_i^*}$ does, which occurs only when $\mathbf{b_k}$ and $\mathbf{b_{k-1}}$ are switched. Then $||\mathbf{b_{k-1}^*}||^2$ is reduced by a factor at least $2/\sqrt{3}$, and so $\Phi(B)$ also reduced by at least a factor of $2/\sqrt{3}$.

The potential of the original basis is bounded by $\prod_{i=1}^{n} \leq \max_i ||\mathbf{b_i}||^{n^2} = 2^{\text{poly}(n,|B|)}$, so to see that the algorithm terminates it remains only to show that the potentials of the intermediate bases are bounded below.

For any basis B, since $\mathcal{L}$ is an integer lattice, B consists of integer vectors. Thus the sub-lattices $\mathcal{L}_i$ are also integer lattices, so have integer determinants. The basis vectors must be non-zero, so for all i $\det(\mathcal{L}_i) \geq 1$. Hence for any intermediate basis B, $\Phi(B) \geq 1$.

It follows that the LLL algorithm will always terminate, as $\Phi(B)$ can only decrease at most $\log_{2/\sqrt{3}}(2^{\text{poly}(n,|B|)})$ times. Since each step in the algorithm takes at most polynomial time, it follows that the LLL algorithm runs in polynomial time.

4 Improving LLL

Since the introduction of the LLL algorithm there has been much research into improving it. Research has largely focused on either improving the speed of the algorithm, while producing reduced bases of a similar quality, or in obtaining better approximation factors than LLL, at the expense of the runnning time.

Schnorr and Euchner presented in [17] three different improved algorithms for lattice reduction: a floating point version of the LLL algorithm, a variant using 'deep insertions', and a practical algorithm for block Korkin-Zolotarev (BKZ) reduction (see Section 5).

4.1 Floating Point LLL

When implemented the LLL algorithm suffers from slow subroutines, due to the long integer arithmetic required. In order to counter this it was proposed to use floating point arithmetic when working with the numbers[3] $\mu_{i,j}$ and $||\mathbf{b_i^*}||^2$. The lattice vectors themselves must be kept in their exact form, but the other values can be stored as floating points and the exact lattice vectors can be used to correct errors if necessary. Approximate versions of the basis vectors are also used within calculations to increase speed, and the exact vectors used when error correction is necessary.

The algorithm for floating point LLL, L^{3}FP, is given in Algorithm 3, where v' is the floating point(fp) value corresponding to the exact v, and $\tau \in \mathbb{Z}$ the number of precision bits in floating point arithmetic. Note that to offset small floating point errors it is necessary to take $1/2 < \delta < 1$.

To minimise errors:

1. Whenever enter stage k, the values of $\mu_{k,j}$ $(1 \leq j \leq k-1)$ and $||\mathbf{b_k^*}||^2$ are recalculated from the exact basis, correcting these values.

2. If a large reduction coefficient $|\lfloor \mu_{k,j} \rceil| > 2^{\tau/2}$ occurs during size reduction of $\mathbf{b_k}$, decrease stage to $k-1$. This will correct the coefficients $\mu_{k-1,j}$, $\mu_{k,j}$ for $1 \leq j \leq k-1$, as well as $||\mathbf{b_{k-1}^*}||^2, ||\mathbf{b_k^*}||^2, \mathbf{b_{k-1}}', \mathbf{b_k}'$

[3] After the original calculation of the Gram-Schmidt vectors it is not necessary to store the vectors $\mathbf{b_i^*}$ and it suffices to use the square of their norm

Algorithm 3 L^{3}FP Algorithm

Input: A basis $\mathbf{b_1}, \ldots, \mathbf{b_n}$ for a lattice $\mathcal{L}$.

Output: An LLL-reduced basis $\mathbf{b_1}, \ldots, \mathbf{b_n}$ for $\mathcal{L}$.

1: $k \leftarrow 2, F_c \leftarrow false$
2: **for** $i = 1, \ldots, n$ **do**
3: $\quad \mathbf{b'_i} \leftarrow (\mathbf{b_i})'$
4: **while** $k \leq n$ **do**
5: $\quad c_k \leftarrow ||\mathbf{b'_k}||^2$
6: $\quad$ **if** $k = 2$ **then**
7: $\quad\quad c_1 \leftarrow ||\mathbf{b^*_1}||^2$
8: $\quad$ **for** $j = 1, \ldots, k-1$ **do**
9: $\quad\quad$ **if** $|\langle \mathbf{b'_k}, \mathbf{b'_j} \rangle| < 2^{\tau/2} ||\mathbf{b'_k}|| \, ||\mathbf{b'_j}||$ **then**
10: $\quad\quad\quad s \leftarrow \langle \mathbf{b_k}, \mathbf{b_j} \rangle'$
11: $\quad\quad$ **else**
12: $\quad\quad\quad s \leftarrow \langle \mathbf{b'_k}, \mathbf{b'_j} \rangle$
13: $\quad\quad \mu_{k,j} \leftarrow \left(s - \sum_{i=1}^{j-1} \mu_{j,i} \mu_{k,i} c_i \right) / c_j$
14: $\quad\quad c_k \leftarrow c_k - \mu_{k,j}^2 c_j$
15: $\quad$ **for** $j = k-1, \ldots, 1$ **do**
16: $\quad\quad$ **if** $|\mu_{k,j}| > 1/2$ **then**
17: $\quad\quad\quad \mu \leftarrow \lfloor \mu_{k,j} \rceil$
18: $\quad\quad\quad$ **if** $|\mu| > 2^{\tau/2}$ **then**
19: $\quad\quad\quad\quad F_c \leftarrow true$
20: $\quad\quad\quad$ **for** $i = 1, \ldots, j-1$ **do**
21: $\quad\quad\quad\quad \mu_{k,i} \leftarrow \mu_{k,i} - \mu \mu_{j,i}$
22: $\quad\quad\quad \mu_{k,j} \leftarrow \mu_{k,j} - \mu$
23: $\quad\quad\quad \mathbf{b_k} \leftarrow \mathbf{b_k} - \mu \mathbf{b_j}$
24: $\quad\quad\quad \mathbf{b'_k} \leftarrow (\mathbf{b_k})'$
25: $\quad$ **if** F_c **then**
26: $\quad\quad F_c \leftarrow false$
27: $\quad\quad k = \max(k-1, 2)$
28: $\quad\quad$ **go to** line 4
29: $\quad$ **if** $\delta c_{k-1} > c_k + \mu_{k,k-1}^2 c_{k-1}$ **then**
30: $\quad\quad$ Swap $\mathbf{b_k}, \mathbf{b_{k-1}}$, swap $\mathbf{b'_k}, \mathbf{b'_{k-1}}$
31: $\quad\quad k = \max(k-1, 2)$
32: $\quad$ **else**
33: $\quad\quad k \leftarrow k + 1$

3. If $|\langle \mathbf{b'_k}, \mathbf{b'_j} \rangle| < 2^{-\tau/2} ||\mathbf{b'_k}|| \, ||\mathbf{b'_j}||$ then compute $\langle \mathbf{b_k}, \mathbf{b_j} \rangle'$ instead of $\langle \mathbf{b'_k}, \mathbf{b'_j} \rangle$

The number of swaps in L^{3}FP is proportional to $\log_2(B/\tau)$ times the number of swaps in the LLL algorithm, however it cannot be proved that this algorithm always terminates. B here is an upper bound of the norm of the original basis vectors.

Nguyen and Stehlé found in [15] an implementation of floating point LLL, referred to as L^2, which provably outputs LLL reduced bases in polynomial time $\mathcal{O}(d^4 n(d + \log B) \log B)$.

L^2 improves the L^{3}FP algorithm by improving the accuracy of the Gram-Schmidt computations through a systematic use of floating point arithmetic to stabilise the size reduction process. In order to improve the accuracy of the Gram-Schmidt coefficients Nguyen and Stehlé use slightly different formulae:

$$r_{i,j} = \langle \mathbf{b_i}, \mathbf{b_j} \rangle - \sum_{k=1}^{j-1} \mu_{j,k} \cdot r_{i,k} \quad \text{and} \quad \mu_{i,j} = \frac{r_{i,j}}{r_{j,j}}$$

where $r_{i,j} = \mu_{i,j} ||\mathbf{b^*_j}||^2 = \langle \mathbf{b_i}, \mathbf{b^*_j} \rangle$ for $i \geq j$ They also define $s_j = ||\mathbf{b_i}||^2 - \sum_{k=1}^{j-1} \mu_{i,k} \cdot r_{i,k}$ for $1 \leq j \leq i$, so $s_i = r_{i,i} = ||\mathbf{b^*_i}||^2$. The Lovász condition then becomes $\delta r_{k-1,k-1} \leq s_{k-1}$, so storing the s_j allows the checking of consecutive Lovász conditions without any additional cost.

The core of L^2 is an iterative floating point (fp) implementation of Babai's nearest plane algorithm. This is known to terminate with correct results, and importantly requires very few bits of the Gram-Schmidt Orthogonalisation (GSO), making L^2 efficient. The overall L^2 algorithm is given in Algorithm 4; for details of the iterative Babai nearest plane algorithm used see [2].

Algorithm 4 L^2

Input: A valid pair (δ, η) such that $1/4 < \delta < 1$ and $1/2 < \eta < \sqrt{\delta}$, a basis $(\mathbf{b_1}, \ldots, \mathbf{b_n})$ and an fp-precision l.

Output: An L^3-reduced basis with factor pair (δ, η).

Compute exactly the Gram matrix $G = G(\mathbf{b_1}, \ldots, \mathbf{b_n})$
$\delta' := (\delta + 1)/2$, $r'_{1,1} := \diamond(\langle \mathbf{b_1}, \mathbf{b_1} \rangle)$, $\kappa := 2$ [4]
while $\kappa \leq d$ **do**
    Size-reduce $\mathbf{b}_\kappa$ using iterative Babai algorithm, this updates the fp-GSO.
    $\bar{\kappa} := \kappa$
    while $\kappa \geq 2$ and $\delta' r'_{\kappa-1,\kappa-1} \geq s'_{\kappa-1}$ **do**
        $\kappa := \kappa - 1$
    for $i = 1$ *to* $\kappa - 1$ **do**
        $\mu'_{\kappa,i} := \mu'_{\bar{\kappa},i}$, $r'_{\kappa,i} := r_{\bar{\kappa},i}$, $r_{\kappa,\kappa} := s'_\kappa$
    Insert $\mathbf{b}_{\bar{\kappa}}$ right before $\mathbf{b}_\kappa$ and update G accordingly
    $\kappa := \kappa + 1$
Output $(\mathbf{b_1}, \ldots, \mathbf{b_d})$

The constant η here replaces the factor $1/2$ in the size-reduction condition of the basis, because as the GSO coefficients are known only approximately if insist on $\eta = 1/2$ the algorithm will loop forever. The factor can be chosen arbitrarily close to $1/2$ in L^2, and if need $\eta = 1/2$ can achieve this by running L^2 with $\bar{\delta} = \delta + 2(\eta - 1/2)$ and $\eta > 1/2$, and then running the LLL algorithm on the output basis. Chen and Nguyén's L^2 implementation of floating point LLL is implemented in the Magma library [1].

[4] $\diamond$ indicates the floating point rounded value of the operation

Stehlé discusses the various implementations of floating-point LLL from both a theoretical and practical point of view in [21].

4.2 Deep Insertions

Schnorr and Euchner also discussed a variant of L^{3}FP, which finds shorter lattice vectors, by extending the swap step $\mathbf{b_k} \leftrightarrow \mathbf{b_{k-1}}$. This technique is called 'deep insertion' and replaces the standard swap step with the process described in Algorithm 5. This means that, unlike in the standard LLL algorithm, basis vectors can be swapped even when not adjacent.

Algorithm 5 Deep Insertion Swap

```
1: c ← ||b'_k||^2, i ← 1
2: while i < k do
3:     if δc_i ≤ c then
4:         c ← c − μ^2_{k,i} c_i
5:         i ← i + 1
6:     else
7:         (b_1, ... b_k) ← (b_1, ··· b_{i−1}, b_k, b_i, ..., b_{k−1})
8:         rearrange b'_j accordingly
9:         k ← max(i − 1, 2)
10:        go to Algorithm 3 line 4
11: k ← k + 1
```

In the worst case L^{3}FP with deep insertions may be super-polynomial in time, but if deep insertions are only performed in the case that either $i \leq c_0$ or $k - 1 \leq c_0$, for a fixed constant c_0, then the runtime remains polynomial. In practice it tends to run quite rapidly and often returns a significantly better basis than standard L^{3}FP.

5 Block Korkin-Zolotarev Reduction

The BKZ algorithm was introduced by Schnorr and Euchner in 1994 [17], and uses the concept of Korkin-Zolotarev reduced bases [13] to reduce a lattice in blocks.

For a lattice $\mathcal{L} = \mathcal{L}(\mathbf{b_1}, \ldots, \mathbf{b_n})$ define $\pi_i : \mathbb{R}^m \to span(\mathbf{b_1}, \ldots, \mathbf{b_{i-1}})^\perp$, the orthogonal projection map, and let $B_{[j,k]}$ be the local projected block $(\pi_j(\mathbf{b_j}), \ldots, \pi_j(\mathbf{b_k}))$, and $\mathcal{L}_{[j,k]}$ the lattice spanned by $B_{[j,k]}$, whose dimension is $k - j + 1$.

Definition 5.1. An ordered basis $(\mathbf{b_1}, \ldots, \mathbf{b_n})$ of a lattice $\mathcal{L}$ is *Korkin-Zolotarev reduced* if it is size reduced (1), and:

$$||\mathbf{b_i^*}|| = \lambda_1((\pi_i(\mathcal{L})) \quad for\ i = 1, \ldots, n$$

Korkin-Zolotarev (KZ) reduced bases are very costly to find, as they are equivalent to solving SVP. In the worst case a KZ-reduced basis satisfies:

$$||\mathbf{b_i^*}|| \approx \sqrt{n - i + 1} \left(\prod_{j=i}^{n} ||\mathbf{b_j^*}|| \right)^{1/(n-i+1)} \tag{3}$$

Schnorr [19] introduced the concept of a block KZ-reduced basis, which uses KZ-reduced blocks of low dimension:

Definition 5.2. An ordered basis $\mathbf{b_1}, \ldots, \mathbf{b_n}$ is β-reduced if it is LLL reduced (with factor δ), and for each $1 \leq j \leq n$:

$$||\mathbf{b_j^*}|| = \lambda_1\left(\mathcal{L}_{[j,k]}\right) \qquad \text{where } k = \min(j + \beta - 1, n)$$

.

The local block $B_{[j,k]}$ generating $\mathcal{L}_{[j,k]}$ is called a β-block.

The β-blocks span lattices of dimension at most β, so as β increases it becomes more difficult to find a BKZ-reduced basis. The idea for BKZ reduction is to iteratively replace β-blocks in the basis by KZ-reduced bases for the local projected lattice. This ensures that the vectors $\mathbf{b_j^*}, \ldots, \mathbf{b_k^*}$ are balanced, with norms which do not reduce too quickly.

The worst case profile of a KZ-reduced basis can be seen in Figure 2, where $x_i := \log(||\mathbf{b_i^*}||)$, and Figure 3 shows how the iterative approach of BKZ reduction steadily balances the basis profile, assuming worst case KZ bases are produced at each stage.

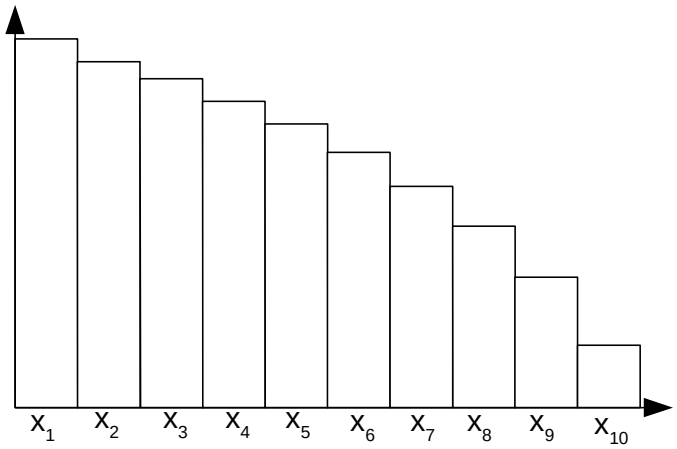

Figure 2: Worst Case HKZ profile (3)

The first practical algorithm for BKZ reduction was introduced by Schnorr and Euchner in [17], and is given in Algorithm 6. It finds a BKZ reduced basis by first applying LLL reduction to the basis, and then iteratively reducing the local block $B_{[j,k]}$ for $j = 1, \ldots, n$, to make sure the first vector of each such block is the shortest in the projected lattice.

During each iteration the lattice $\mathcal{L}_{[j,k]}$ is enumerated (see Section 6) to find a shortest vector

$$\mathbf{v} = \sum_{i=j}^{k} v_i \pi_j(\mathbf{b_i})$$

If this vector is not equal to $(1, 0, \ldots, 0)$, that is $||\mathbf{b_j^*}|| > \lambda_1(\mathcal{L}_{[j,k]})$, the vector $\mathbf{b}^{new} = \sum_{i=j}^{k} v_i \mathbf{b_i}$ is inserted into the basis, between $\mathbf{b_{j-1}}$ and $\mathbf{b_j}$. The resulting set is no longer a basis for the lattice, so the LLL algorithm is applied to $(\mathbf{b_1}, \ldots, \mathbf{b_{j-1}}, \mathbf{b}^{new}, \mathbf{b_j}, \ldots, \mathbf{b_n})$ to produce a new LLL reduced basis $(\mathbf{b_1}, \ldots, \mathbf{b_n})$. Otherwise LLL is simply called on $(\mathbf{b_1}, \ldots, \mathbf{b_h})$, where $h = \min(k+1, n)$, to ensure the next block is reduced before enumerating.

The variable z is used to keep track of the number of j for which $||\mathbf{b_j^*}|| = \lambda_1(\mathcal{L}_{[j,k]})$, so when the algorithm terminates, the basis is BKZ-reduced. Since j is always calculated modulo $n-1$, it will continue to loop over the values 1 to n until the entire basis is BKZ-reduced.

There is no known good upper bound on the complexity of BKZ, the best upper bound known for the number of calls to the enumeration subroutine is exponential [8]. In practice Gama and Nguyen showed in [5] BKZ is practical for low values of β, but the running time

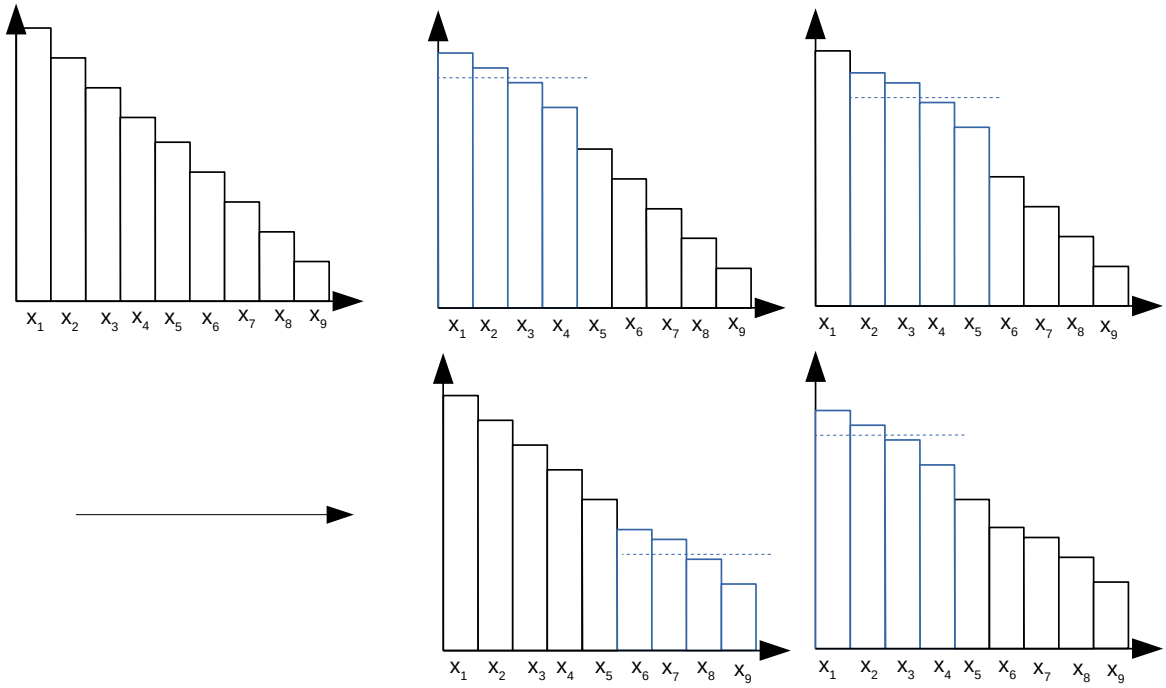

Figure 3: BKZ reduction, assuming worst case HKZ reduction.

significantly increases for $\beta \geq 25$, due to a sharp increase in the number of calls to the enumeration subroutine (see Section 7). Hanrot, Pujol and Stehlé showed in [9] that BKZ can be terminated before its completion, while still providing excellent bases.

Chen and Nguyen introduced in [2] an updated version of BKZ, called BKZ 2.0, with four improvements:

- Early abort - limit the number of calls made to the SVP oracle, provides exponential speed up over BKZ for $\beta \geq 30$
- Sound pruning [4]- speeds up enumeration by discarding subtrees in which the probability of finding the desired lattice point is too small
- Preprocessing of local blocks - cost of enumeration is strongly influenced by the quality of the local basis, so use recursive aborted-BKZ preprocessing on local basis before enumeration
- Optimizing enumeration radius R - the number of nodes at depth d of enumeration tree is proportional to R^d, so to optimize for all indexes except last 30, instead take

$$R = \min\left(\sqrt{\gamma}\, GH(\mathcal{L}_{[j,k]}), ||\mathbf{b}_\mathbf{j}^*||\right)$$

where γ is a radius parameter, and

$$GH(\mathcal{L}_{[j,k]}) = \sqrt{\frac{n}{2\pi e}}\det(\mathcal{L})^{1/n}$$

the Gaussian Heuristic. Chen and Nguyen took $\sqrt{\gamma} = \sqrt{1.01}$ in [2].

BKZ 2.0 is implemented in the the fplll lattice reduction library [3].

Algorithm 6 BKZ Algorithm

Input: A basis $\mathbf{b_1}, \ldots, \mathbf{b_n}$ for a lattice $\mathcal{L}$.
Output: A BKZ-reduced basis for $\mathcal{L}$.

$z \leftarrow 0, j \leftarrow 0$
LLL$(\mathbf{b_1}, \ldots, \mathbf{b_n}, \mu)$
while $z < n - 1$ **do**
  $j \leftarrow (j \mod (n-1)) + 1$
  $k \leftarrow \min(j + \beta - 1, n)$
  $h \leftarrow \min(k + 1, n)$
  $v \leftarrow$ **Enum** $\left(\mu_{[j,k]}, ||\mathbf{b_j^*}||^2, \ldots, ||\mathbf{b_k^*}||^2\right)$
  if $v \neq (1, 0, \ldots, 0)$ **then**
    $z \leftarrow 0$
    LLL $\left(\mathbf{b_1}, \ldots, \sum_{i=j}^{k} v_i \mathbf{b_i}, \mathbf{b_j}, \ldots \mathbf{b_n}, \mu\right)$ at stage j
  else
    $z \leftarrow z + 1$
    LLL$(\mathbf{b_1}, \ldots, \mathbf{b_n}, \mu)$ at stage $h - 1$

6 Enumeration

In low dimension, $n < 70$, SVP can be solved exactly by exhaustive search, finding all vectors $\mathbf{v} \in \mathcal{L}$ in a lattice of norm less than or equal to some radius R. This is done by enumerating all short vectors in projected lattices $\pi_i(\mathcal{L})$, where π_i the projection map orthogonal to $\mathbf{b_1}, \ldots, \mathbf{b_{i-1}}$. A depth k of the enumeration tree will contain all projected lattice points $||\pi_{n+1-k}(\mathbf{v})||$ of norm less than or equal to R, and the leaves of the tree are the sought vectors $\mathbf{v} \in \mathcal{L}$ of norm less than or equal to R. Solving SVP in this way may have complexity significantly larger than $|\mathcal{L} \cap B(\mathbf{0}, R)|$, due to 'dead-ends' in the enumeration tree.

Schnorr & Euchner's algorithm [17] (Algorithm 7) is the most efficient known practical method of enumeration, it is used as a subroutine in BKZ (see Section 5) and outperforms the theoretical algorithms of Kannan and AKS, even though they have much better theoretical complexity. It minimises the expression

$$c_j(u_j, \ldots, u_k) := ||u_j \mathbf{b_j} + \cdots + u_k \mathbf{b_k}|| = \sum_{s=j}^{k} \left(\sum_{i=s}^{k} u_k \mu_{i,s} \right)^2 c_s$$

for $(u_j, \ldots, u_k) \in \mathbb{Z}^{k-j+1} \setminus \{0\}$, where $c_s = ||\mathbf{b_s^*}||^2$ and $\mu_{s,s} = 1$. It uses depth first enumeration of all integer vectors $(\tilde{u}_t, \ldots, \tilde{u}_k)$ which satisfy $c_t(\tilde{u}_t, \ldots, \tilde{u}_k) < \bar{c}_j$, where $\bar{c}_j$ the current minimum for c_j $(t = k, \ldots, j)$.

Given a reduced basis the ENUM algorithm outputs the shortest vector in $2^{\mathcal{O}(n^2)}$ polynomial time operations. Since the cost of enumeration strongly depends on the radius R and the decrease of the $||\mathbf{b_i^*}||$, the more reduced the basis the faster the enumeration; Gama and Nguyen observed in [5] that when the basis is only LLL-reduced the running time does indeed look super exponential. Kannan observed that the cost of enumeration is so high, that even an aggressive pre-processing significantly lowers the total cost, while barely contributing to it. In practice tend to use BKZ as pre-processing.

The running time of the enumeration algorithm is N polynomial time operations, where N is the total number of nodes in the enumeration tree. Hanrot and Stehlé suggested in [10] that a good estimate of N can be derived from the Gaussian Heuristic. This predicts the number of nodes at depth k scanned by the Schnorr-Euchner enumeration to be close to

$$H_k = \frac{1}{2} \frac{V_k(R)}{\prod_{i=n+1-k}^{b} ||\mathbf{b_i^*}||}$$

Algorithm 7 ENUM

Input: Basis vectors $\mathbf{b_1}, \ldots, \mathbf{b_k}, c_i = ||\mathbf{b_i^*}||^2$ for $i = 1, \ldots, k$ and $\mu_{i,t}$ for $1 \leq t < i \leq k$
Output: the minimal $(u_1, \ldots, u_k)$, the minimum $\bar{c}_j$ of $c_j(u_j, \ldots, u_k)$ and minimal vector $\mathbf{v}$

$\bar{c} \leftarrow c,\ \tilde{\mu}_1 \leftarrow 1,\ y_1 \leftarrow \Delta_1 \leftarrow 0,\ s \leftarrow t \leftarrow 1,\ \delta_1 \leftarrow 1$
for $i = 2, \ldots, k+1$ **do**
  $\tilde{c}_i \leftarrow u_i \leftarrow \tilde{u}_i \leftarrow y_i \leftarrow \Delta_i \leftarrow 0,\ \delta_i \leftarrow 1$
while $t \leq k$ **do**
  $\tilde{c}_t \leftarrow \tilde{c}_{t+1} + (y_t + \tilde{u}_t)^2 c_t$
  if $\tilde{c}_t < \bar{c}$ **then**
    if $t > 1$ **then**
      $t \leftarrow t-1,\ y_t \leftarrow \sum_{i=t+1}^{s} \tilde{u}_i \mu_{i,t},\ \tilde{u}_t \leftarrow \nu_t \leftarrow \lfloor -y_t \rceil,\ \Delta_t \leftarrow 0$
      if $\tilde{u}_t > -y_t$ **then**
        $\delta_t \leftarrow -1$
      else
        $\delta_t \leftarrow 1$
    else
      $\bar{c} \leftarrow \tilde{c},\ u_i \leftarrow \tilde{u}_i$ for $i = j, \ldots, k$
  else
    $t \leftarrow t+1,\ s \leftarrow \max(s, t)$
    if $t < s$ **then**
      $\Delta_t \leftarrow -\Delta_t$
    if $\Delta_t \delta_t \geq 0$ **then**
      $\Delta_t \leftarrow \Delta_t + \delta_t$
    $\tilde{u}_t \leftarrow \nu_t + \Delta_t$
  $\mathbf{v} := \sum_{i=1}^{k} u_i \mathbf{b_i}$

For a reduced basis typically have $||\mathbf{b_i^*}||/||\mathbf{b_{i+1}^*}|| \approx q$ where q depends on the reduction algorithm. So, as shown in [4],

$$H_k \approx q^{(n-k)k/2} 2^{\mathcal{O}(n)} \lessapprox q^{n^2/8} 2^{\mathcal{O}(n)}$$

with the maximum being achieved for $k \approx n/2$

Since enumeration is the most practical solver of SVP, there has been much effort to optimize it. Heuristically the main approach is pruning the enumeration tree, by removing branches with low ratio between the estimated branch size and the probability of containing the desired solution. This idea was introduced in [17] and further studied in [18]. Pruning replaces the n inequalities $||\pi_{n+1-k}(\mathbf{v})|| \leq R$ by $||\pi_{n+1-k}(\mathbf{v})|| \leq R_k$ where $R_1 \leq \cdots \leq R_n = R$ are real numbers defined by the pruning strategy. As discussed in [4], can consider the 'probability' p_{succ} that the target vector is still in the tree after pruning. Gama, Nguyen and Regev introduced the idea of *extreme pruning*, in which repeatedly randomise the input basis and run enumeration with pruning which has very small p_{succ}. The advantage comes from the fact that although the success probability is exponentially small, the volume of the enumeration tree which is searched decreases by a stronger exponential factor.

7 Quality of algorithm output

Lattice reduction algorithms are somewhat mysterious, with experimental evidence suggesting that they generally behave better than their proved worst case bounds. Gama and Nguyen performed the first precise assessment of this in [5], using the NTL [20] implementations of

floating-point LLL, LLL with deep insertions (DEEP), and the BKZ algorithm. The theoretical bounds on LLL are an approximation factor of at most $(4/3)^{(n-1)/2} \approx 1.154^n$, and a Hermite factor at most $(4/3)^{(n-1)/4} \approx 1.0754^n$. LLL with deep insertions is expected to improve both of these, but no provable upper bound is known. BKZ was proved by Schnorr [19] to achieve an approximation factor less than or equal to $\gamma_\beta^{(n-1)/(\beta-1)}$ if it terminates, and similarly achieves a Hermite factor less than or equal to $\sqrt{\gamma_\beta}^{1+(n-1)/(\beta-1)}$.

7.1 Experimental values of Hermite and Approximation factors

Gama & Nguyen observed the average behaviour of LLL, BKZ and DEEP on random lattices of varying dimension, and found that when the lattice had no exceptional structure the Hermite factor of all three algorithms appeared to be exponential in the lattice dimension, with DEEP and BKZ exhibiting the same overall behaviour as LLL, but with much smaller constants.

	LLL	BKZ-20	BKZ-28	DEEP-50
c = Hermite factor$^{1/n}$	1.0219	1.0128	1.0109	1.011
Best proved upper bound	1.0754	1.0337	1.0282	1.0754

Table 1: Table showing root Hermite factor for LLL, BKZ, DEEP reproduced from [5]

Since c is always close to 1, it follows that the Hermite factor is always small, unless the lattice dimension is huge. For example in dimension 300 LLL has Hermite factor of around 665 and BKZ-20 has Hermite factor around 48, compared to upper bounds of more than 2 million and 2 thousand respectively. This suggests Hermite SVP with factor n is easy in dimension up to at least 450. In high dimensions BKZ and DEEP can achieve a Hermite factor of approximately 1.01^n, but not much less due to restrictions on block-size in BKZ.

It can be shown that linearly many applications of an algorithm with Hermite factor α can achieve an approximation factor of α^2. So results suggest that these reduction algorithms solve SVP with an approximation factor of roughly $1.01^{2n} \approx 1.02^n$ in the worst case. Gama and Nguyen showed that this worst case can occur, in the case of echelon lattices, but on average these algorithms achieve, in a reasonable time, an approximation factor less than or equal to 1.01^n. Thus approximate-SVP with factor n should be easy up to dimension at least 250 even in the worst case , and on average should be easy up to dimension at least 500.

Further experiments were performed by Schneider and Buchmann [16], which found that for low dimensions ($n < 75$) BKZ could achieve root Hermite factors below 1.01, even below 1.005, though for dimensions above 200 the Hermite factor seemed to stabilise above 1.01. From their experimental data they found a fitting function $f(\beta)$ which predicts the Hermite factor which is reachable with a chosen block-size β:

$$f(\beta) = 1.01655 - 0.000196185 \cdot \beta$$

In [2] BKZ 2.0 was shown to achieve a Hermite factor of slightly below 1.01^n, in lattice dimension up to $n = 800$, using a block-size of 90, and 18 pruned enumerations at 5%. Chen & Nguyen also implemented a simulation algorithm which allowed them to guess the approximate Hermite factor achieved, given an arbitrary block-size, their results are reproduced in Table 2. They also showed experimentally that a polynomial number of calls to the enumeration subroutine seems to suffice to obtain a Hermite factor not far from that of a full reduction, justifying the use of early abort.

Target Hermite Factor	1.01^n	1.009^n	1.008^n	1.007^n	1.006^n	1.005^n
Approximate block-size	85	106	133	168	216	286

Table 2: Approximate required block-size for high-dimensional BKZ, table reproduced from [2]

7.2 Run times

The Schnorr-Euchner enumeration routine [17] is the most efficient exhaustive method for SVP, in practice. Given a reduced basis input it outputs the shortest vector in $2^{\mathcal{O}(n^2)}$ polynomial operations, depending on the quality of the basis. For dimensions less than 60 enumeration can be used to solve SVP in under an hour, but due to the exponential runtime it is impractical in much higher dimensions.

BKZ and DEEP have no good known upper complexity bound, the best is $(n\beta)^n$ polynomial time operations, but this does not seem to be tight - with a block-size of 20 a 100-dimensional lattice can be reduced in a few seconds. For a fixed dimension the running time of BKZ increases with block-size, but the standard BKZ algorithm [17] experiences a steep increase in running time around $\beta = 20 - 25$, due to a sudden increase in the number of calls to the enumeration subroutine. The increase sharpens with the dimension, so for high lattice dimension a block-size much greater than 25 is not realistic. This was supported by the experiments in [16], which observed that for block size $\beta \geq 40$ the enumeration subroutine accounted for more than 99% of the running time, whereas in low block-sizes it accounts for less than 10% of the run time. Figures 12 and 13 in [5] show the runtimes of BKZ for varying block-sizes and dimensions.

The runtime of DEEP is much more regular than that of BKZ, with the running time growing exponentially on a regular basis; the slope of runtime increase also does not increase with the size of the lattice. This suggests that DEEP can be run in very high dimensions with larger block-size than BKZ, which may make it preferable, though it is still not expect to significantly decrease the 1.01^n prediction for the Hermite factor.

Chen and Nguyen [2] also found upper bounds on the cost of the enumeration subroutine, using extreme pruning with aborted BKZ preprocessing, by considering the number of nodes processed. For enumeration with extreme pruning at double precision, approximately 200 clock-cycles are required per node, so this can be used to approximate the run times of the BKZ 2.0 algorithms. For example with block-size 120 enumeration involves approximately 2^{53} nodes, which would take less than 30 core-years on a 1.86-GHz Xeon.

Block-size	100	110	120	130	140	150	160	170	180	190	200	250
BKZ-75 - 20%	41.4	47.1	53.1	59.8	66.8	75.2	84.1	94.1	105.8	117.6	129. 4	204.1
Simulated BKZ-90–120	40.8	45.3	50.3	56.3	63.3	69.4	79.9	89.1	99.1	103.3	111.1	175.2

Table 3: Upper bounds on the cost of the enumeration subroutine, using extreme pruning with aborted-BKZ pre-processing. Cost is given as log2 (number of nodes). Table from [2].

References

[1] W. Bosma, J. Cannon, and C. Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[2] Y. Chen and P. Q. Nguyen. BKZ 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20. Springer, 2011.

[3] The FPLLL development team. fplll, a lattice reduction library. Available at `https://github.com/fplll/fplll`, 2016.

[4] N. Gama, P. Q. Nguyen, and O. Regev. Lattice enumeration using extreme pruning. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 257–278, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] N. Gama and P. Q. Nguyen. Predicting lattice reduction. In *Proc. EUROCRYPT 2008*, pages 31–51. Springer, 2011.

[6] M. Gruber and C. G. Lekkerkerker. *Geometry of Numbers*. North Holland, 1987.

[7] G. Hanrot, X. Pujol, and D. Stehlé. Algorithms for the shortest and closest lattice vector problems. In *Coding and Cryptology*, pages 159–190. Springer Berlin Heidelberg, 2011.

[8] G. Hanrot, X. Pujol, and D. Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *Advances in Cryptology – CRYPTO 2011*, pages 447–464. Springer, 2011.

[9] G. Hanrot, X. Pujol, and D. Stehlé. Terminating BKZ. *IACR Cryptology ePrint Archive*, 2011:198, 2011.

[10] G. Hanrot and D. Stehlé. Improved analysis of kannan's shortest lattice vector algorithm. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 170–186. Springer Berlin Heidelberg, 2007.

[11] C. Hermite. Extraits de lettres de M. Hermite à M. Jacobi sur différents objets de la théorie des nombres, deuxième lettre. *J. Reine Angew. Math.*, 1850.

[12] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 1982.

[13] A. Korkine and G. Zolotarev. Sur les forms quadratiques. *Mathematische Annalen*, 1873.

[14] P. Q. Nguyen. Lattice reduction algorithms: Theory and practice. In K. G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 2–6. Springer Berlin Heidelberg, 2011.

[15] P. Q. Nguyen and D. Stehlé. Floating-point LLL revisited. In *EUROCRYPT*, 2005.

[16] M. Schneider and J. Buchmann. Extended lattice reduction experiments using the BKZ algorithm. In *Lecture Notes in Informatics (LNI), Proceedings - Series of the Gesellschaft fur Informatik (GI)*, pages 241–252, 01 2010.

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

[18] C. P. Schnorr and H. H. Hörner. Attacking the chor-rivest cryptosystem by improved lattice reduction. In L. C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, pages 1–12. Springer Berlin Heidelberg, 1995.

[19] C.P. Schnorr. A hierarchy of polynomial time lattice basis reduction algorithms. *Theoretical Computer Science*, 53(2):201 – 224, 1987.

[20] V. Shoup. Number theory C++ library (NTL). Available at `http://www.shoup.net/ntl/`.

[21] D. Stehlé. Floating-point LLL: Theoretical and practical aspects. In *The LLL Algorithm: Survey and Applications*. Springer, 2009.